Leçon 11

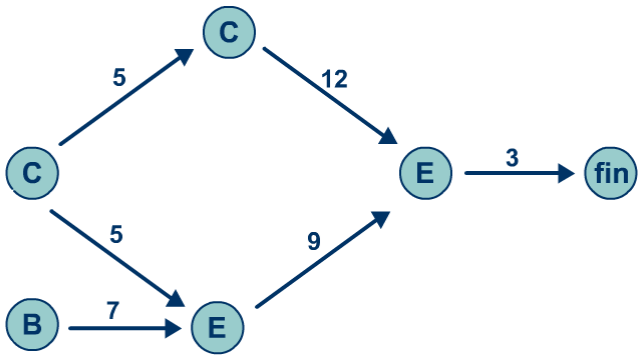
PROBLEMES D'ORDONNANCEMENT AVEC RESSOURCES

Dans cette leçon, nous retrouvons le problème d'ordonnancement déjà vu mais en ajoutant la prise en compte de contraintes portant sur les ressources.
Après un exemple d'introduction, nous définirons le problème, et aborderons le cas où des tâches doivent être effectuées par un seul opérateur puis par deux opérateurs successifs.
Après avoir souligné la difficulté de la grande majorité des problèmes d'ordonnancement, nous présenterons des méthodes de résolution approchées.
Nous terminerons par la présentation du problème particulier du bin packing.

I Introduction

Les problèmes d'ordonnancement font partie de notre vie quotidienne !
Qui ne s'est jamais trouvé avec une liste de "choses à faire" et ne pas savoir par quel bout les prendre ?
Ces problèmes concernent bien évidemment aussi de nombreuses activités des entreprises.
Le problème central de l'ordonnancement consiste à déterminer le calendrier d'exécution de tâches liées entre elles par des contraintes "temporelles", entre autres de succession, de manière à minimiser la durée totale.

Considérons par exemple la situation correspondant au graphe potentiel tâche suivant :



On peut sans difficulté déterminer le planning au plus tôt de ces tâches.
Supposons maintenant que chaque tâche soit confiée à certaines personnes et que, par exemple, C et D soient réalisées par monsieur Dupont et que B et C fassent appel à l'équipe de Durand composée de 5 personnes, chacune de ces tâches mobilisant 3 de ces personnes.
On en déduit facilement que B et C ne peuvent plus être exécutées simultanément, pas davantage que C et D.
On a donc des couples de tâches en conflit. Pour chacun d'eux il faut déterminer l'ordre dans lequel on va faire les tâches correspondantes. Il est bien sûr hors de question d'examiner toutes les possibilités ; en effet avec n couples de tâches en conflit il y a 2^n ordres possibles.

II Définition d'un problème d'ordonnancement avec ressources

Les problèmes d'ordonnancement sont très variés ; nous donnons ici les principaux éléments les définissant.
Un problème d'ordonnancement est caractérisé par un ensemble de tâches et un ensemble de ressources matérielles, humaines, financières …
Les ressources utilisées par les tâches peuvent être de nature diverse.
On peut distinguer entre elles :

- les **ressources consommables** : elles ne servent qu'une fois, par exemple les ressources financières…
- les **ressources restituables** : elles peuvent être réutilisées dès qu'elles sont libérées. C'est par exemple le cas des machines. On distinguera les **ressources cumulatives** (on dispose d'une certaine quantité de ressources) et les **ressources disjonctives** (on dispose d'une seule unité non fractionnable).

On se place dans le contexte où, pour chaque tâche, on connaît :

- sa durée
- éventuellement la date à laquelle elle doit être achevée
- le cas échéant les contraintes entre les dates de début de ces tâches
- les ressources qu'elle utilise.

Le problème consiste à déterminer à quel moment, et avec quelles ressources, les tâches sont exécutées.
Différents critères peuvent être envisagés :

- le temps total d'exécution
- le respect de certains délais

III Problème d'ordonnancement à un poste

On considère n tâches indépendantes, qui peuvent donc être exécutées dans un ordre quelconque.
Dans le problème d'ordonnancement à un poste, chacune des tâches doit être réalisée par une seule machine ou poste de travail, qui ne peut faire qu'une tâche à la fois.
On connaît le temps d'exécution p_i de chaque tâche.

Par exemple, soient 5 taches dont les temps d'exécution sont donnés dans le tableau suivant :

Tâche i	1	2	3	4	5
Temps de traitement p_i	4	3	7	2	2

Il y a 5 ordres de passage possibles soit 120.
Considérons les 2 ordres suivants : 1, 2, 3, 4, 5 et 4, 5, 2, 1, 3

Si les tâches sont exécutées sans interruption entre elles, la durée totale d'exécution est égale à 18 (4 + 3 + 7 + 2 + 2). Elle est évidemment indépendante de l'ordre
Cependant, ces 2 ordonnancements ne sont pas équivalents.
Les graphiques suivant permettent de visualiser la pile des tâches en attente. Le deuxième ordonnancement conduit, a priori, à un temps d'attente total sur les 5 tâches plus petit que le premier.

Figure 1 : ordre de passage 1, 2, 3, 4, 5

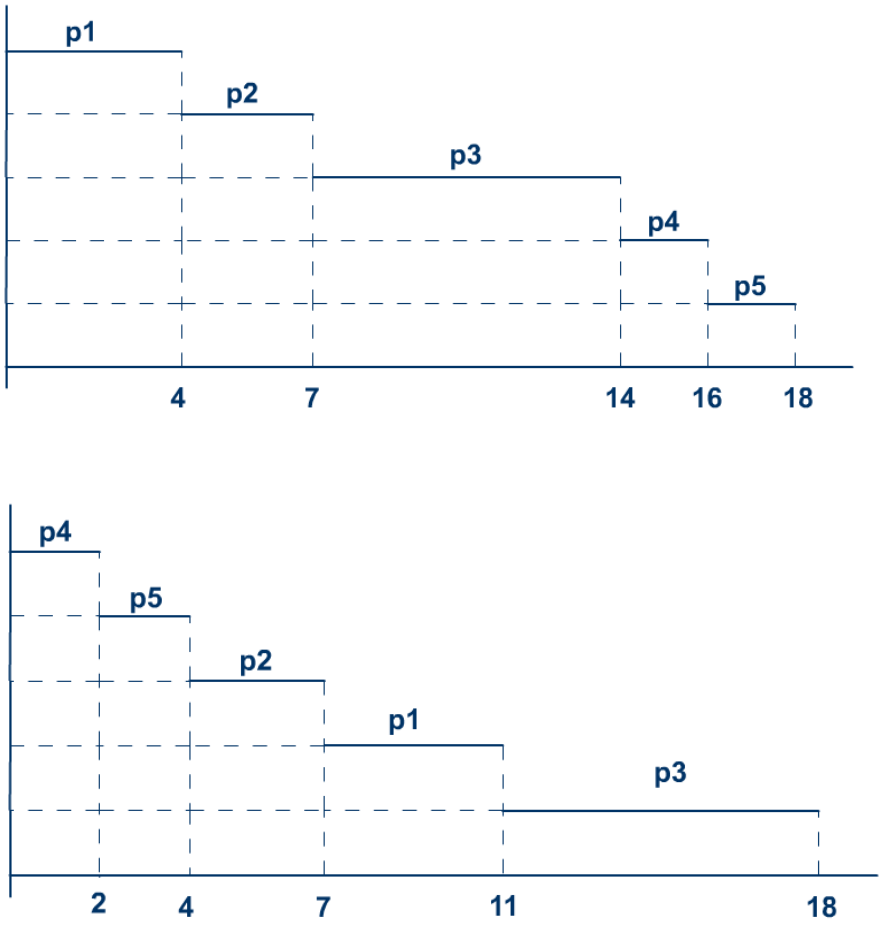


Figure 2 : ordre de passage 4, 5, 2, 1, 3

Minimisation du temps de séjour

On va s'intéresser à la recherche de l'ordonnancement qui minimise le temps de séjour total.
Le temps de séjour d'une tâche est le temps pendant lequel la tâche est en attente ou en train d'être exécutée.

Dans le cas de la figure 1, le temps de séjour est égal à 4 pour la tâche 1 (exécutée en premier) puis 7 pour la tâche 2 (qui doit attendre la fin de 1 pour être exécutée), etc.…soit un temps de séjour total égal à 59.

Il s'agit de trouver l'ordre qui minimise le temps de séjour total
Les graphiques précédents suggèrent de commencer par les tâches les plus courtes.
On a effectivement le résultat suivant :

Proposition
Etant données n tâches indépendantes, de durée p_i, en attente d'exécution sur une seule machine, l'ordonnancement qui minimise le temps de séjour de l'ensembles des tâches est obtenu en classant les tâches par durée d'exécution croissante.

Cette règle est appelée règle SPT pour Shortest Processing Times.

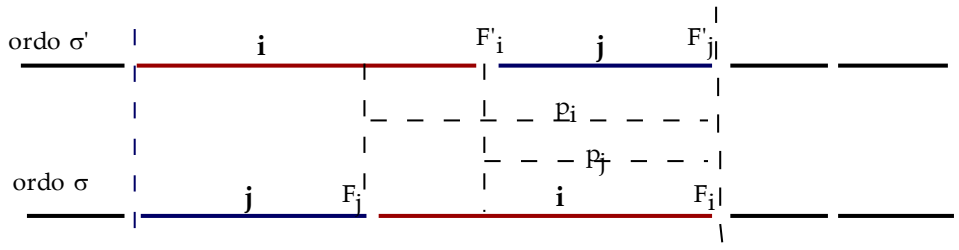
Démonstration

1 - Soit F_i le temps de séjour de la tâche i

Si on suppose que les tâches sont dans l'ordre 1, 2, ..., n, on a :
$F_1 = p_1$
$F_2 = p_1 + p_2$
.....
$F_i = p_1 + p_2 + ...+ p_i$
..
$F_n = p_1 + p_2 + ...+ p_i + ...+ p_n$
Donc $\sum_{i=1}^{n} F_i = n\,p_1 + (n-1)\,p_2 + ...+ (n - i + 1)\,p_i + p_n$
Pour minimiser cette quantité, il est donc "normal" de commencer par la tâche dont le temps de traitement est la plus faible (elle est multipliée par n dans l'expression précédente) et de terminer par celle dont le temps de traitement est le plus important qui n'est multiplié que par 1.

2 - Montrons qu'un tel ordonnancement est bien optimal.
Rappelons que l'on peut passer d'une permutation σ de 1... n à une autre permutation quelconque σ ' par une suite de transpositions d'éléments consécutifs.
Par exemple : à partir de 1,2,5,4,3 on peut arriver à 1, 2, 3, 4, 5 en procèdant à une suite de transpositions.
5, 4 → 4, 5 1, 2, 4, 5, 3
5, 3 → 3, 5 1, 2, 4, 3, 5
4, 3 → 3, 4 1, 2, 3, 4, 5

Soient σ et σ ' deux ordonnancements.
Evaluons l'impact d'une transposition d'éléments consécutifs sur la valeur de l'objectif (durée totale de séjour). L'ordonnancement σ est déduit de σ ' par permutation des 2 tâches j et i.
Soient f(σ) et f(σ ') la durée de ces 2 ordonnancements.
On veut calculer la variation $\Delta = f(\sigma) - f(\sigma')$ de la fonction objectif suite à cette transposition.



La date de fin des tâches précédant i n'est pas modifiée. Il en est de même de celle des tâches suivant j.
On doit évaluer :
$\Delta = (F_i + F_j) - (F'_i + F'_j) = F_i - F'_j + (F_j - F'_i)$
(Δ représente la variation de l'objectif quand on passe de σ ' à σ)

Or on a : $F'_j = F_i$
donc $\Delta = F_j - F'_i = p_j - p_i$
Si la durée de la tâche j est plus courte que celle de la tâche i, on a $\Delta \leq 0$, l'ordonnancement σ est meilleur que σ '.
($\Delta < 0 \rightarrow f(\sigma) < f(\sigma')$)

Soit alors $\sigma = \{i_1, i_2, ... i_n\}$ la permutation des n tâches obtenue en les ordonnant par durée croissante.
$p_{i1} \leq p_{i2} \leq .....\leq p_{in}$

Soit σ ' un autre ordonnancement quelconque différent de σ. On peut passer de σ ' à σ par une succession de transpositions d'éléments consécutifs i et j, j suivant immédiatement i et tels que $p_j \leq p_i$ (j est "mal placé" par rapport à i).
D'après le résultat précédent, chacune de ces transpositions conduit à une variation de l'objectif égale à $p_j - p_i \leq 0$.
On passe donc de σ ' à σ par une suite de transpositions qui ne peuvent que faire diminuer l'objectif.
On en déduit que le temps de séjour de l'ordonnancement σ est inférieur au temps de séjour de n'importe quel ordonnancement.
L'ordonnancement σ obtenu en classant les tâches par ordre de durée d'exécution croissante est donc optimal.

Un simple tri suffit pour trouver l'ordonnancement optimal.
Ici l'ordonnancement optimal correspond au cas de figure 2 avec les tâches dans l'ordre 4, 5, 2, 1, 3.
Si on veut avoir l'impression que la pile diminue vite, il faut donc commencer par les tâches les plus courtes !

Prise en compte des délais de livraison
Le critère du temps de séjour privilégie l'aspect minimisation des stocks d'en-cours.
Reprenons le même problème, mais ajoutons comme information la date à laquelle chaque tâche est censée être terminée.

Tâche i	*1*	*2*	*3*	*4*	*5*
Temps de traitement p_i	4	3	7	2	2
Délai de livraison	5	6	6	8	17

Suivant leur date d'exécution, et donc l'ordre de passage sur la machine, les tâches sont ou non en retard.
Par exemple, si les tâches sont dans l'ordre 1, 2, 3, 4, 5, les tâches 1 et 2 ne sont pas en retard alors que le retard des tâches 3 et 4 est de 8 et celui de la tâche 5 est de 1.
Si on note R_i le retard de la tâche i : $R_1 = R_2 = 0$, $R_3 = R_4 = 8$, $R_5 = 1$
Parmi tous les ordonnancements, on recherche celui (ou ceux) qui minimise les retards.
On peut envisager 2 types de critères : la minimisation du retard moyen sur l'ensemble des tâches ou celle du plus grand des retards.
Sur cet exemple, on a un retard moyen de 3,7 ((0 + 0 + 8 + 8 + 2) / 5) alors que le retard maximal est de 8.

Ces 2 problèmes sont de nature très différente en ce sens que la minimisation du plus grand des retards est un problème "facile" alors que la minimisation de la somme des retards est un problème "difficile" !

Minimisation du retard maximal
Proposition
Etant données n tâches indépendantes, de durée p_i, de délai de livraison d_i, en attente d'exécution sur une seule machine, l'ordonnancement qui minimise le plus grand des retards sur l'ensemble des tâches est obtenu en classant les tâches par délai de livraison croissant.

Cette règle est appelée règle EDD pour Earliest Due Date.

Démonstration
La démonstration se fait de manière analogue à la précédente en étudiant les conséquences de l'échange de 2 tâches consécutives i et j.
Supposons que $d_j > d_i$ et montrons que si on permute les tâches i et j l'ordonnancement σ est "meilleur" que l'ordonnancement σ '.
Soient R_i et R_j le retard des tâches i et j dans l'ordonnancement σ et R'_i et R'_j le retard des tâches i et j dans l'ordonnancement σ '.

Si $F_i < d_i$ (la date de fin de la tâche est avant la date de livraison) : $R_i = 0$

Sinon $R_i = F_i - d_i$.
On peut écrire que $R_i = \text{Max}(0, F_i - d_i)$, de même pour les autres tâches.
Comme la tâche j est avancée, on a $R_j \leq R'_j$ (le retard de j ne peut être que diminué).
Montrons que l'on a aussi $R_j \leq R'_j$
$R_i = \text{Max}(0, F_i - d_i)$ $R'_j = \text{Max}(0, F'_j - d_j)$
On a : $F_i = F'_j$ et $d_i < d_j$
On en déduit que $R_i \leq R'_j$ donc le max des retards de σ est inférieur au max des retards de σ '.
Le même raisonnement que précédemment permet d'affirmer que l'ordonnancement obtenu en triant les tâches par ordre de délai de livraison croissant est optimal.
Dans l'exemple précédent, l'application de la règle EDD conduit à l'ordonnancement 1, 2, 3, 4, 5.
Tout autre ordre conduira à l'existence d'une tâche dont le retard est au moins égal à 8.

Si on prend comme critère la minimisation de la somme des retards, le problème devient difficile : on ne connaît pas d'algorithme permettant de le résoudre en un temps "raisonnable", c'est-à-dire avec une croissance polynomiale, quand le nombre de tâches devient très grand.

IV Ordonnancement sur 2 postes successifs

Les exemples précédents se limitaient à l'utilisation d'un seul poste : machine ou individu. Considérons maintenant le cas où les tâches nécessitent 2 opérations successives sur 2 postes différents, par exemple, la fabrication d'une pièce suivie d'une opération de contrôle.

Toutes les tâches sont prêtes à être exécutées et on dispose pour chaque tâche de son temps de traitement p_{i1} et p_{i2} pour chacune des 2 opérations.
Il suffit de déterminer l'ordre de passage sur la première machine, celui sur la deuxième machine sera identique.
Il y a donc en général n possibilités parmi lesquelles on recherche celle(s) conduisant à la durée totale minimale.

Ce problème est résolu facilement par l'algorithme suivant :

Algorithme de Johnson

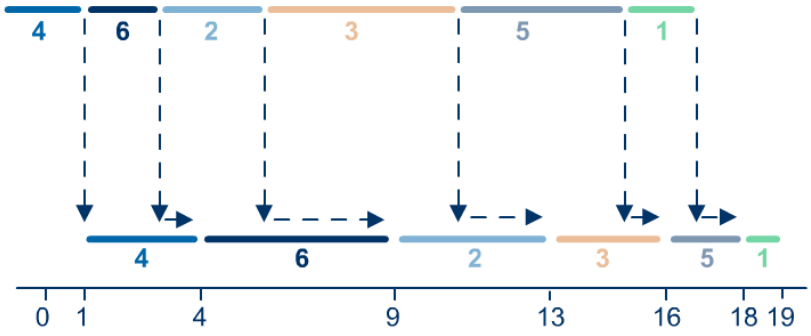
Déterminer S_1 = ensemble des tâches pour lesquelles $p_{i1} \leq p_{i2}$
Déterminer S_2 = ensemble des tâches pour lesquelles $p_{i2} < p_{i1}$
Ordonner les tâches de S_1 dans l'ordre des p_{i1} croissants
Placer les tâches correspondantes dans cet ordre sur le poste 1
Ordonner les tâches de S_2 dans l'ordre des p_{i2} décroissants
Placer les tâches correspondantes dans cet ordre à la suite des tâches précédentes sur le poste 1
Placer dans le même ordre les tâches sur le poste 2 dès que le poste 2 est disponible.

Considérons par exemple les 6 tâches suivantes.

i	*1*	*2*	*3*	*4*	*5*	*6*
p_{i1}	2	3	5	1	4	2
p_{i2}	1	4	3	3	2	5

$S_1 = \{2, 4, 6\}$
$S_2 = \{1, 3, 5\}$
On trie S_1 dans l'ordre des p_{i1} croissants : 4 6 2
On trie S_2 dans l'ordre des p_{i2} décroissants : 3 5 1

On obtient ainsi l'ordonnancement : 4 6 2 3 5 1 qui conduit à un temps total de 19.



On peut, dans ce cas, facilement se convaincre qu'on a ainsi obtenu la meilleure solution. Il n'y a pas de temps d'attente sur le poste 2 (ce qui n'est pas toujours le cas) et la tâche placée en tête sur le poste 1 a le plus petit temps de traitement possible.

Les problèmes précédents, ordonnancement sur un poste ou sur 2 postes, sont un des rares cas où il est possible d'obtenir facilement une solution optimale.
Dans la grande majorité des cas, les problèmes d'ordonnancement avec ressources sont des problèmes que l'on ne sait pas résoudre en un temps raisonnable pour des gros problèmes. Par exemple, avec 3 opérations successives le problème devient difficile. On fait donc appel à des méthodes approchées.

V Ordonnancement : cas général

On considère un problème d'ordonnancement quelconque avec des tâches, éventuellement liées entre elles par des contraintes de succession, et utilisant différentes ressources disjonctives ou cumulatives.
Les méthodes approchées reposent sur des algorithmes de liste dont le principe est très simple.

Principe d'un algorithme de liste
Les tâches sont placées progressivement.
Une tâche ne peut être placée que si elle est exécutable : les tâches qui doivent la précéder sont terminées et on dispose des ressources nécessaires.
En cas de conflit, on place les tâches selon un **ordre de priorité** prédéfini.

Tout repose donc sur le choix des règles de priorité.

De très nombreuses règles de priorité peuvent être mises en oeuvre. Elle privilégie généralement un objectif mais de manière implicite.

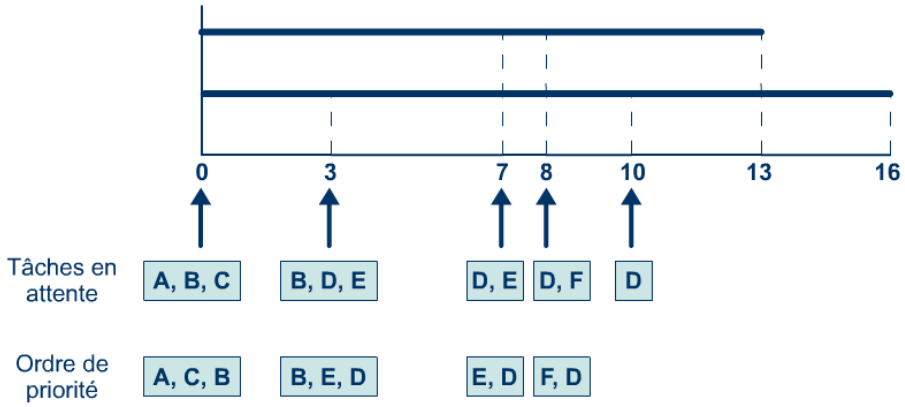
- La règle la plus simple est de prendre les tâches dans l'ordre de leur arrivée : c'est la règle dite FIFO. Les opérations attendent a priori le moins possible avant leur exécution.

- On peut aussi sélectionner l'opération qui a le plus petit temps d'exécution. Dans ce cas, c'est plutôt la minimisation des en cours de fabrication qui est visée.
- Si, parmi les tâches exécutables, la priorité est donnée à celle pour lequel le temps nécessaire à la réalisation des opérations qui doivent lui succéder est le plus long (temps total restant), cela sous-entend que le respect des délais est privilégié.
Ce dernier critère revient à classer les tâches en fonction de leur date de début au plus tard.

La validation de la performance des différentes règles de priorité vis-à-vis d'un critère donné ne peut généralement être faite que par simulation à partir des données du problème traité.

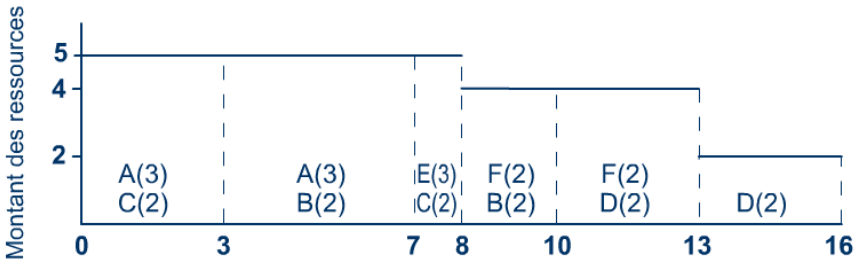
Exemple
On considère ici 6 tâches utilisant une ressource pour laquelle on dispose de 5 unités. Pour chacune de ces tâches, on connaît : la durée, les tâches qui doivent la précéder, sa date de début au plus tard, et le montant des ressources qu'elles mobilisent. Le critère de priorité est la date de début au plus tard.

Tâche	A	B	C	D	E	F
Durée	7	7	3	6	1	5
Prédécesseurs	//	//	//	C	C	A , E
Date au plus tard	0	5	1	6	4	7
Montant de la ressource	3	2	2	2	3	2



A la date t = 0, les 3 tâches A, B et C sont en attente d'affectation de ressources.
D'après la règle de priorité, on place d'abord A et C, mais il n'y a plus assez de ressources pour B.
En t = 3 la tâche C libère 2 unités de ressources. Les 3 tâches B, D, E sont maintenant en attente. La plus prioritaire est B, compte tenu de sa date au plus tard.
On attend la fin de A pour pouvoir placer une autre tâche. Ainsi de suite, dés qu'une tâche se termine, on examine les tâches en attente et, si on dispose des ressources nécessaires, on les place selon l'ordre de priorité retenu.

On peut visualiser la courbe de charge de l'utilisation des ressources.



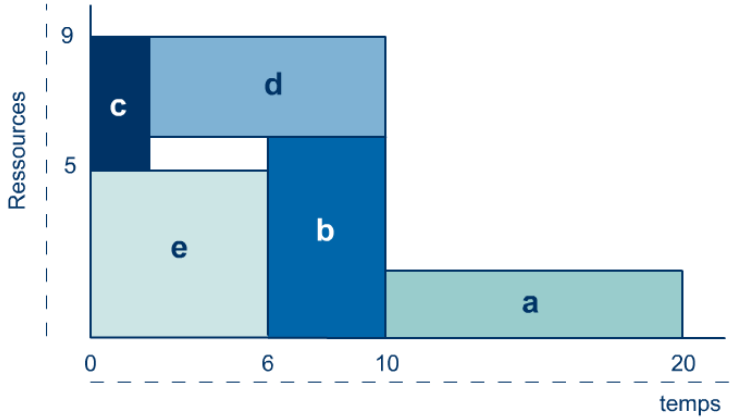
Les exemples suivants illustrent la difficulté des problèmes d'ordonnancement.

Exemple 1

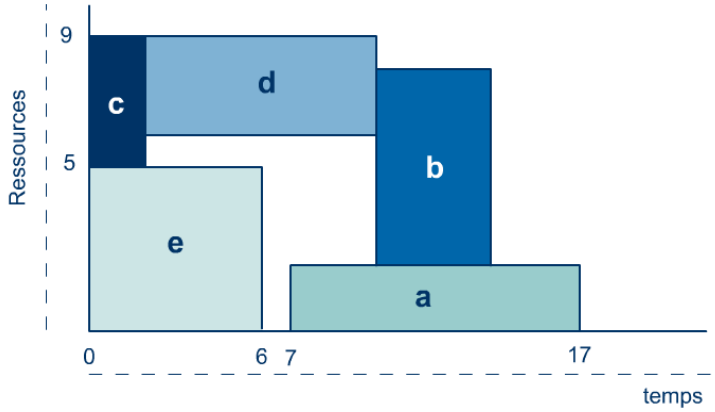
Tâche	a	b	c	d	e
Durée	10	4	2	8	6
Date de disponibilité	7	2	0	2	0
Ressource nécessaire	2	6	4	3	5

On dispose de 9 unités de ressources (ressources cumulatives).
Si on résout le problème d'ordonnancement de ces tâches en une durée minimale par un algorithme de liste, **quel que soit l'ordre de priorité** retenu, on obtient la solution suivante :

En t= 0, seules les tâches c et e sont disponibles, elles sont mises en oeuvre.
En t = 2, 4 unités de ressources sont libérées par la tâche c, les tâches b et d sont disponibles, mais on n'a de ressources que pour la tâche d ; on l'exécute.
En t= 6, la tâche e est finie, on exécute alors la tâche b.
En t= 10, les tâches b et d s'achèvent, on peut alors exécuter la tâche a.
Les tâches sont terminées en t = 20.



En fait, la solution optimale conduit à une durée de 17 :

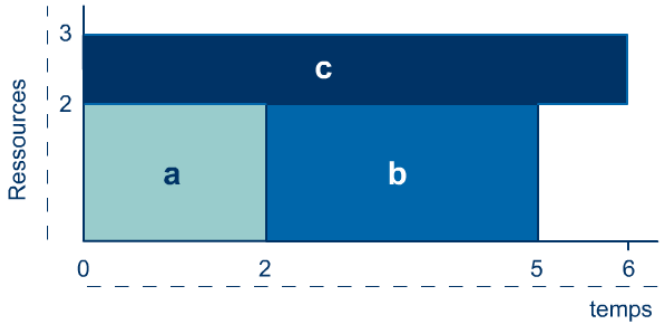


Exemple 2

Tâches	a	b	c
Durée	2	3	6
Ressources	2	2	1

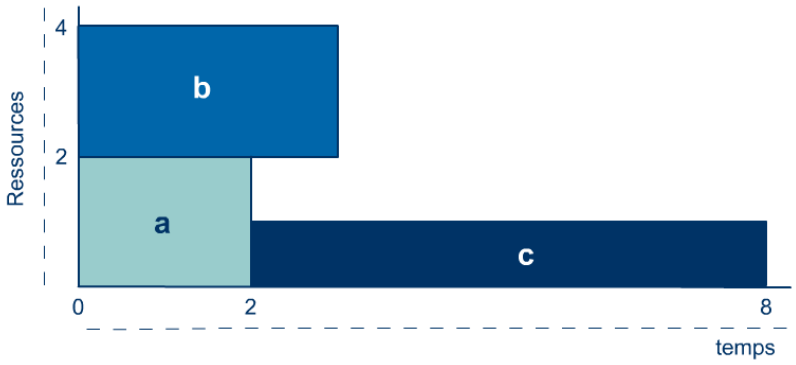
Toutes les tâches sont prêtes à être exécutées en t=0, l'ordre de priorité retenu est a b c.

1er cas : on dispose de 3 unités de ressources.



En t = 0 on exécute la tâche a ; la tâche b, bien que prioritaire par rapport à c, ne peut être faite, on lance donc c.
b est exécutée lorsque a se termine.
L'ordonnancement obtenu a une durée de 6.

2ème cas : on dispose de 4 unités de ressources.



En t = 0 on peut maintenant faire a et b
La tâche c est exécutée lorsque a se termine.
L'ordonnancement obtenu a une durée de ....8.
Avec davantage de ressources, on a dégradé la solution !

VI Le problème du bin packing

Le problème suivant semble nous éloigner de la problématique de l'ordonnancement.
On doit, dans des planches de longueur donnée, découper différents morceaux de longueurs différentes. Il s'agit de minimiser la chute ce qui revient à minimiser le nombre de planches à découper.
En fait, ce problème est strictement de même nature que celui de l'ordonnancement de tâches qui doivent être réalisées sur un intervalle de temps fixé avec le minimum de personnes.

On va modéliser ces problèmes par le problème dit du bin packing.

Définition du problème de bin packing
On doit ranger des objets mono dimensionnels dans des boîtes de taille donnée, en utilisant le moins de boîtes possible.
Les problèmes de découpe et d'ordonnancement précédents se ramènent très simplement au problème du bin packing.
Dans un cas, les morceaux correspondent évidemment aux objets et les planches aux paquets. Dans l'autre ce sont les tâches qui représentent les objets et les personnes les paquets.
Il existe de nombreuses heuristiques pour ce problème.
Leur principe général est simple : on prend les objets dans un certain ordre et, soit on les met dans un paquet que l'on choisit, soit on ouvre un nouveau paquet.
Les différentes méthodes diffèrent par l'ordre dans lequel les objets sont rangés et par la manière de choisir le paquet.

Heuristique First Fit
(First fit = le premier qui convient)

On prend les objets dans l'ordre où ils se présentent et on les place dans le premier paquet qui peut les contenir.

Exemple
Soient 10 objets de taille : 5 1 3 4 7 2 6 6 1 5
On doit les ranger dans des paquets de taille 10.
Combien de paquets faut-il au minimum ?
L'application de l'heuristique précédente donne :

taille : 5 1 3 4 7 2 6 6 1 5

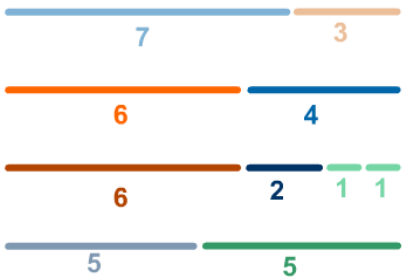
On place les 3 premiers objets dans le paquet 1 puis on ouvre un deuxième paquet dans lequel on met l'objet de taille 4.
Pour l'objet suivant, de taille 7, il faut encore un nouveau paquet.
L'objet suivant de taille 2 tient dans le paquet 2.
Pour les 2 objets de taille 6, il faut ouvrir 2 nouveaux paquets.
L'avant dernier, de taille 1, est alors mis dans le premier paquet, mais il faut ouvrir un sixième paquet pour le dernier objet de taille 5.

Cette heuristique peut s'appliquer même si les objets arrivent au fur et à mesure, comme sur un tapis de supermarché par exemple ! Une variante est de choisir le paquet qui a le plus de place disponible, c'est l'heuristique dite "best fit".

Si on suppose que les objets sont tous disponibles initialement, on peut les trier préalablement. On obtient alors l'heuristique FFD – Fist Fit Decreasing.

Heuristique First Fit Decreasing
On procède comme précédemment mais en triant au préalable les objets par taille décroissante.

Exemple
On reprend le même exemple.
Les objets sont pris dans l'ordre de leur taille : 7 6 6 5 5 4 3 2 1 1



La solution obtenue ne comporte plus que 4 paquets ; elle est évidemment optimale !

Cette leçon nous a permis de voir la diversité des problèmes d'ordonnancement avec ressources, mais aussi la complexité. Quelques rares problèmes sont résolus facilement de manière optimale (c'est-à-dire avec un temps de calcul qui croit de manière polynomiale avec la taille du problème) mais la grande majorité tombe dans la catégorie de problèmes à un million de dollars ! Leur résolution fait donc l'objet de méthodes approchées y compris dans des logiciels comme Ms-Project de Microsoft.