

# CHAPITRE I

## Le problème d'ordonnancement d'atelier

### 1.1. Introduction

La théorie de l'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales de tâches. Pour cela, il est très souvent nécessaire d'affecter en même temps les ressources nécessaires à l'exécution de ces tâches.

La façon naturelle du problème d'ordonnancement est formulé comme des modèles de programmation mathématiques, l'utilisation de la programmation en nombres entiers dans la résolution des problèmes d'ordonnancement en 1959, Où Wagner [WAG59] a été le premier à la formation d'une variété de problèmes d'ordonnancement comme des programmes en nombres entiers. Et il est discuté par Conway [CON74] et Barker [BAK67].

### 1.2. Les problèmes d'ordonnancement

#### 1.2.1 Définition des problèmes d'atelier

Nous donnons dans cette section une définition des problèmes d'atelier et introduisons une classification avant d'illustrer ces problèmes par quelques applications réelles.

#### 1.2.2 Définition des problèmes de base

Dans un problème d'atelier, une pièce doit être usinée ou assemblée sur différentes machines. Chaque machine est une ressource disjonctive, c'est-à-dire qu'elle ne peut exécuter qu'une tâche à la fois, et les tâches sont liées exclusivement par des contraintes d'enchaînement. Plus précisément, les tâches sont regroupées en  $n$  entités appelées travaux ou lots. Chaque lot est constitué de  $m$  tâches à exécuter sur  $m$  machines distinctes. Il existe trois types de problèmes d'atelier, selon la nature des contraintes liant les tâches d'un même lot. Lorsque l'ordre de passage de chaque lot est fixe et commun à tous les lots, on parle d'atelier à cheminement unique (flow-shop). Si cet ordre est fixe mais propre à chaque lot, il s'agit d'un atelier à cheminements multiples (job-shop). Enfin, si le séquençement des tâches des travaux n'est pas imposé, on parle d'atelier à cheminements libres (open-shop). Un critère d'optimalité souvent étudié est la minimisation du délai total de l'ordonnancement (makespan). Ce critère est particulièrement intéressant puisque les ordonnancements au plus tôt constituent des sous-ensembles dominants 1 pour de nombreux critères réguliers. De plus, l'analyse des ordonnancements au plus tôt permet de déterminer des chemins critiques, c'est-à-dire des chemins sur lesquels tout retard a des conséquences sur toute la chaîne, ou des

goulots d'étranglement, c'est-à-dire les étapes qui vont limiter la production de tout l'atelier. Nous supposons que les durées d'exécution des tâches sont données par une matrice entière  $P : m \times n$ , dans laquelle  $p_{ij} \geq 0$  est la durée d'exécution de la tâche  $T_{ij} \in T$  du lot  $J_j$  réalisée sur la machine  $M_i$ . Une borne inférieure classique pour les problèmes d'atelier, notée  $C_{max}^{LB}$ , est égale au maximum de la charge des machines et des durées des lots :

$$C_{max}^{LB} = \max \left( \max_{1 \leq i \leq m} \left( \sum_{j=1}^n p_{ij} \right), \max_{1 \leq j \leq n} \left( \sum_{i=1}^m p_{ij} \right) \right)$$

Dans un problème d'ordonnancement interviennent deux notions fondamentales : les tâches et les ressources. Une ressource est un moyen, technique ou humain, dont la disponibilité limitée ou non est connue à priori. Une tâche est un travail dont la réalisation nécessite un certain nombre d'unités de temps, sa durée, et d'unités de chaque ressource.

Les problèmes d'ordonnancement sont caractérisés par trois ensembles.

L'ensemble de "n" tâches  $T = \{T_1, T_2, \dots, T_n\}$ , l'ensemble de "m" machines  $P = \{P_1, P_2, \dots, P_m\}$  et l'ensemble de "s" ressources  $R = \{R_1, R_2, \dots, R_s\}$ . Ordonnancer c'est assigner les machines de l'ensemble P et des ressources de l'ensemble R aux tâches de l'ensemble T dans l'ordre de compléter toutes les tâches sous des contraintes imposées.

Il y a deux types de contraintes classiques. En tout instant chaque tâche est à exécuter au plus sur une machine et chaque machine n'est capable d'exécuter qu'une tâche à la fois.

### 1.2.3. Caractérisation des machines

Elles peuvent être en " *parallèles* ", faisant la même fonction, ou spécialisées dans l'exécution de certaines tâches ("*dédiées* ", dedicated). Dans ce cas, les machines sont disposées, en général, en séries. On distingue trois types de machines parallèles dépendant de leur vitesse. Si toutes les machines ont la même vitesse d'exécution des tâches, les machines sont appelées " *identiques* " est le problème est noté (P). Si les machines différentes par leur vitesse d'exécution et la vitesse  $b_i$  de chaque machine est constante et ne dépend pas de l'ensemble des tâches, elles sont dites " *uniformes* " (Q). Si les vitesses d'exécution des machines dépendent des tâches et sont différentes alors elles sont dites " *quelconques* " ou " *différentes* " ( unrelated, R ). Dans le cas de machines " *dédiées* " , le plus souvent sont en série, il y a trois modèles ou types d'exécution de tâches: le flow shop, l'open shop et le job shop. Si dans un atelier donné il y a "M" machines et N travaux, si les opérations élémentaires ou tâches ne sont pas liées par un ordre particulier, on parle de problème d'open shop. Dans ces problèmes, le nombre d'opérations de chaque tâche est égal au nombre de machines.

Chaque opération utilisant une machine différente. Les opérations d'une même tâche ne sont pas soumises à des contraintes de précédence, par contre, il ne peut y avoir de chevauchements entre ces opérations. Dans un flow shop, le nombre d'opérations de chaque tâche est égal au nombre de machines. Seulement il y a des contraintes de précédence de type chaîne entre les opérations d'une tâche :

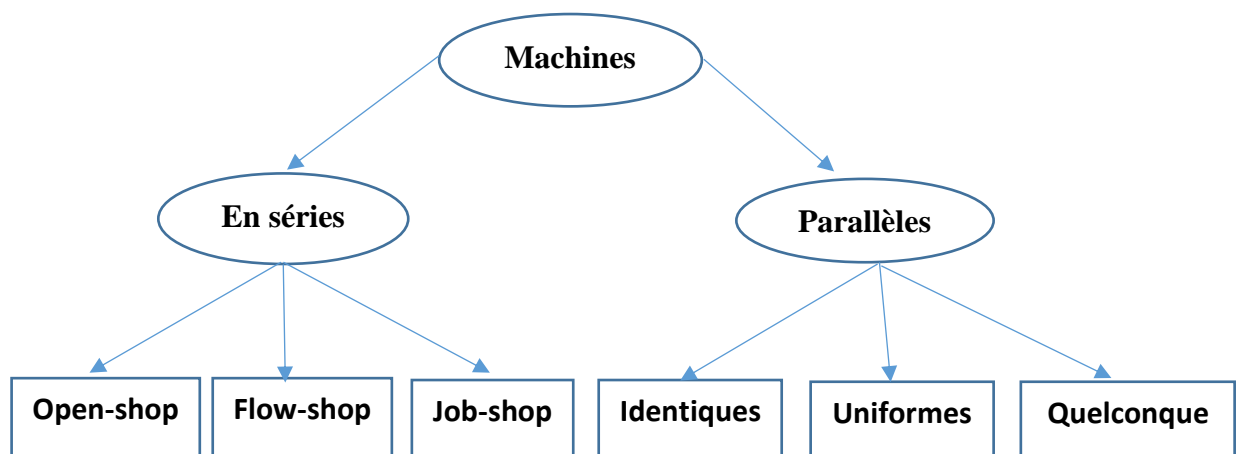
L'opération  $(i, j + 1)$  ne peut commencer avant la fin de l'opération  $(i, j)$ .

Dans le cas du Job shop, il existe des contraintes de précédence entre les opérations d'une même tâche. Par contre, l'ordre d'utilisation de machines n'est pas le même pour toutes les tâches. En général le nombre d'opérations est plus grand que le nombre de machines.

Un autre type d'ordonnancement existe, le flow shop hybride où des machines parallèles sont en série. C'est une organisation du processus de production en série. Les produits fabriqués passent dans un premier temps dans une première cellule (des machines en parallèles), puis dans une seconde, etc.

En général dans de tel système les espaces tampons ou Buffers entre les machines sont supposés de capacité illimitée et un job après sa fin d'exécution sur une machine doit attendre avant que son exécution ne commence sur une autre machine. Si les espaces tampons sont de capacité nulle, les jobs ne peuvent pas attendre entre deux machines consécutives et la propriété de " sans attente " est admise.

Le diagramme de la figure ci-dessous schématise les différents types de machines :



**Figure (1.1) :** Types des machines.

### 1.2.4. Les Tâches

A chaque tâche  $T_j \in T$ , on associe les paramètres suivants :

1. La *durée d'exécution*  $p_{ij}$  de la tâche  $T_j$  sur la machine  $M_i$ . Dans le cas de machines identiques, le temps d'exécution d'une tâche ne dépend pas de la machine, d'où  $p_{ij} = p_j$ ,  $i = 1..m$ . Si les machines sont uniformes alors  $p_{ij} = \frac{p_j}{b_i}$ ,  $i = 1..m$  où  $p_j$  est une durée d'exécution de référence, mesurée usuellement sur la machine la moins rapide et  $b_i$  est le facteur vitesse d'exécution de la machine  $M_i$ .
2. La *date d'arrivée* ou *date de disponibilité* (*release date*)  $r_j$ . La date où la tâche  $T_j$  est prête pour l'exécution. Si les dates d'arrivée sont les mêmes pour toutes les tâches, on suppose que  $r_j = 0, j = 1..n$ .
3. La *date de fin d'exécution au plutard* (*due date*)  $d_j$ . Appelée aussi *date de fin échue* ou *date de fin souhaitée*. Si la tâche termine son exécution après cette date, elle encourt une pénalité.
4. La *date de fin impérative*  $\tilde{d}_j$  (*deadline*). Si la tâche termine son exécution après cette date, elle ne risque pas seulement une pénalité mais des problèmes surgiront, soit l'atelier est bloqué ou la machine tombe en panne, etc.
5. Le *poids* ou la *priorité*  $w_j$  (*weight*). Il exprime une urgence dans l'exécution de la tâche  $T_j$ .
6. La *date de fin d'exécution*  $C_j$ . En général, c'est une variable à déterminer.
7. Le *décalage*  $L_j = C_j - d_j$  (*lateness*). Le temps total durant lequel, la tâche est autorisée à rester dans l'atelier.
8. Le *retard*  $D_j = \max \{C_j - d_j, 0\}$  (*tardiness*).
9. L'*avance*  $E_i = \max (0, -L_i)$  (*earliness*).
10. L'*indicateur de retard*  $U_i = 0$  si  $C_i \leq d_i$  et  $U_i = 1$  sinon.

### 1.2.5. Les contraintes :

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décisions. Les contraintes auxquelles sont soumises les diverses tâches qui concourent à la réalisation de problèmes d'ordonnancement sont de divers types. On distingue :

#### 1.2.5.1. Les contraintes potentielles

Elles peuvent être de deux sortes : les contraintes d'antériorité et plus généralement contraintes de cohérence technologique selon laquelle une tâche  $j$  ne peut commencer avant une tâche  $i$  ne soit terminée, par exemple la construction des piliers suit les fondations, et les contraintes de localisation temporelle qui impliquent qu'une tâche donnée  $i$  ne peut débuter

avant une date imposée appelée sa date de disponibilité  $r_i$ , ou une date avant laquelle  $i$  doit être achevée sa date d'échue  $d_i$ .

**1.2.5.2. Les contraintes de ressources :** on peut citer :

- Contraintes d'utilisation de ressources qui expriment la nature et la quantité des moyens utilisés par les tâches, ainsi que les caractéristiques d'utilisation de ces moyens.
- Contraintes de disponibilité des ressources qui précisent la nature et la quantité des moyens disponibles au cours du temps.

Lorsque dans un problème, certaines tâches nécessitent l'intervention simultanée de plusieurs types de ressources, on parle de contraintes multi-ressources.

**1.2.5.3. Les contraintes cumulatives :**

Interdisent la réalisation simultanée d'un nombre trop important de tâches compte tenu de la disponibilité maximale de la ressource à chaque instant et des quantités requises individuellement par les tâches. Ceci apparaît lorsque par exemple quatre machines sont disponibles pour la réalisation de cinq tâches simultanément, donc il faut savoir à l'avance laquelle des tâches sera retardée.

**1.2.5.4. Les Contraintes disjonctives :**

Obligent à réaliser toute paire de tâches sur des intervalles de temps disjoints. Ceci apparaît lorsque deux tâches utilisent la même machine, donc elles ne pourront pas s'exécuter simultanément (interdiction de réalisation simultanée liée, par exemple, à des raisons de sécurité ou de manque de place pour exécuter plusieurs tâches à la fois en un même endroit).

**Remarque :**

Le problème d'ordonnancement avec des contraintes potentielles uniquement est appelé problème central d'ordonnancement.

**1.2.6. Les critères :**

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques permettant d'apprécier la qualité des solutions. En cherchera à minimiser ou maximiser de tels critères. On note par exemple :

- liés au temps : le temps total d'exécution ou le temps d'achèvement d'un ensemble de tâches. Les retards (maximum, moyen, somme, nombre, ...etc) par rapport aux dates limites fixées.
- liés aux ressources : la quantité maximale, moyenne ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches.

- liés aux couts : de lancement, de production, de transport, de stockage...etc, mais aussi aux revenus, aux retours d'investissement, ...etc.

### 1.2.7. Les ressources :

Une ressource  $k$  est un moyen technique ou humain destiné être utilisé pour la réalisation d'une tâche et disponible en quantité limité. On distingue deux types de ressources : les ressources renouvelables et les ressources consommables.

Une ressource est consommable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches suivantes. Le cas pour l'argent, la matière première.

Une ressource est renouvelable si, après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité, le cas pour les hommes, les machines les processeurs, les fichiers et l'équipements en générale.

On distingue par ailleurs, principalement dans le cas de ressources renouvelable, les ressources disjonctives et les ressources cumulatives.

Une ressource disjonctive : qui ne peut exécuter qu'une tâche à la fois c'est une ressource non partageable (machin, rebot, ...etc).

Une ressource cumulative : qui peut être utilisée par plusieurs tâches en même temps c'est une ressource partageable (équipe d'ouvriers, poste de travail, ...etc).

### 1.2.8. Fonctions objectives d'un problème d'ordonnancement

Les critères d'optimalité les plus utilisées font intervenir la durée totale de l'ordonnancement, le délai d'exécution, les retards de l'ordonnancement et le coût des stocks d'encours. La durée totale de l'ordonnancement notée  $C_{max}$  est égale à la date d'achèvement de la tâche la plus tardive :

$C_{max} = \max C_j$ . C'est la longueur de l'ordonnancement (Schedule length ou makespan).

Le Flow time moyen  $\bar{F} = \frac{1}{n} \sum_{j=1}^n F_j$  ou le flow time moyen pondéré  $F_W = \frac{\sum_{j=1}^n w_j F_j}{\sum_{j=1}^n w_j}$

Le critère, flow time,  $\sum_{i=1}^n w_i C_i$  permet d'estimer le coût des stocks d'encours.

En effet la tâche " $i$ " est présente dans l'atelier entre les instants  $r_i$  et  $C_i$ , et donc les stocks dont elle a besoin doivent être disponibles entre ces deux dates, d'où le coût

$\sum_{i=1}^n w_i (C_i - r_i)$  est égale à un constant pré à  $\sum_{i=1}^n w_i C_i$

Dans beaucoup de problèmes, il faut respecter les délais, donc les dates au plus tard  $d_i$ ; on peut chercher à minimiser le plus grand retard  $T_{max} = \max T_i$ , ou bien la somme des retards

$\sum_{i=1}^n T_i$ , ou encore la somme pondérée des tâches en retard  $\sum_{i=1}^n w_i T_i$ .

Le décalage maximum  $L_{max} = \max \{ L_j \}$ .

D'autres critères peuvent être utilisés

Le retard moyen  $D_{moy} = \sum_{j=1}^n D_j$ . Le retard moyen pondéré  $D_w = \sum_{j=1}^n w_j D_j / \sum_{j=1}^n w_j$ .

Le nombre de tâches en retard  $U = \sum_{j=1}^n U_j$ , où  $U_j = 1$  si  $C_j > d_j$  et 0 sinon.

Le nombre de tâches en retard pondéré  $U_w = \sum_{j=1}^n w_j U_j$ .

### 1.2.9. Les types de problèmes d'ordonnancement

Il existe plusieurs types de problèmes d'ordonnancement, les plus connus sont :

#### 1.2.9.1. Le problème Job-Shop

Le problème Job-Shop est un des problèmes fameux d'optimisation combinatoires NP-Difficiles [APP91]. Il peut être expliqué comme suit: Étant donné un ensemble de  $n$  jobs et un ensemble de  $m$  machines. Chaque job  $j$  est composé d'un ensemble ordonné de  $m$  opérations  $O_1, O_2, O_3, \dots, O_n$ . L'ordre des opérations ne peut pas être modifié, elles sont organisées par un ordre donné propre à chaque job. En outre, l'opération  $O_i$  doit être réalisée par une machine donnée durant un temps  $T_i$  sans interruption (i.e. lorsque l'opération commence, elle ne peut pas être interrompue ou stoppée temporairement jusqu'à ce qu'elle s'achève complètement). D'autre part, chaque machine peut exécuter un seul job à la fois (i.e. dans une période donnée) et chaque job peut être exécuté par une seule machine pendant une période de temps. Le problème consiste à trouver une allocation faisable (ou l'allocation optimale) de toutes les opérations à des intervalles de temps et des machines disponibles en respectant les contraintes du problème. L'objectif final est de minimiser le temps d'exécution du dernier job. Il existe des variantes du problème Job-Shop comme celle nommée : Multi Purpose Machine Job-Shop [PON09].

#### 1.2.9.2. Le problème Flow-Shop

Le problème Flow-Shop est un des problèmes d'optimisation combinatoires difficiles à résoudre [KOU11], il ressemble au problème Job-Shop, sauf que dans le problème Flow-Shop on considère un ensemble de  $n$  jobs qui doivent être réalisés dans un ordre identique par  $m$  machines (i.e. tous les jobs passent par les  $m$  machines dans le même ordre). Idem que le

problème Job-Shop, Il existe des variantes du problème Flow-Shop comme la variante nommée : no-Wait Flow-Shop [LI06] et la variante nommée Assembly Flow-Shop [ALL06].

### 1.2.9.3. Le problème Open-Shop

Le problème Open-Shop est un problème d'optimisation combinatoire difficile [MON07]. Il représente un cas spécial du problème Job-Shop. Dans le problème Open-Shop, les opérations d'un job peuvent être exécutées dans n'importe quel ordre. Autrement dit, les machines réalisant les jobs sont parcourues dans un ordre quelconque.

## 1.3. Propriétés des ordonnancements

Les ordonnancements peuvent être classés en au moins quatre catégories : les admissibles, les semi-actifs, les actifs et les sans délai ou sans retard.

*Ordonnements admissibles* : Un ordonnancement est dit admissible s'il respecte toutes les contraintes du problèmes à savoir les dates limites, les précédences, la limitation des ressources etc.

On parle parfois de *glissement à gauche local* lorsqu'on avance le début d'une tâche sans remettre en cause l'ordre relatif entre les tâches.

On parle aussi de *glissement à gauche global* lorsqu'on avance le début d'une tâche en modifiant l'ordre relatif entre au moins deux tâches.

*Ordonnement semi-actif* : si aucun travail, ne peut avancer sur la ressource où il se trouve, compte tenu des contraintes de gamme et de précedence. La longueur de l'ordonnement correspond au chemin le plus long dans le graphe potentiel-tâche.

Aucun glissement à gauche local n'est possible : on ne peut avancer une tâche sans modifier la séquence sur la ressource.

*Ordonnement actif* : si aucune opération d'un travail  $i$  ne peut débiter son exécution plus tôt sans déplacer au minimum une autre opération.

Aucun glissement à gauche, local ou global, n'est possible. Aucune tâche ne peut être commencée plus tôt sans reporter le début d'une autre.

*Ordonnement sans délai* : lorsqu'aucune machine n'est laissée inoccupée, ceci alors qu'une file d'attente contient au moins un travail susceptible de débiter son exécution sur cette machine. Lorsqu'une méthode parcourt l'ensemble des ordonnancements sans délai, elle n'est pas capable de trouver la solution optimale, on ne laisse pas une machine inoccupée alors qu'une file contient des tâches.

On ne doit pas retarder l'exécution d'une tâche si celle-ci est en attente et si la ressource est disponible.



### 1.4. Variantes et classification de Lawler

Les variantes des problèmes d'atelier imposent généralement des restrictions supplémentaires sur l'ordonnancement (Contraintes temporelles, contraintes de ressource) ou autorisent la préemption des tâches. Le problème d'open-shop ne contient pas, comme le job-shop ou le flow-shop, de contraintes de précédence entre les tâches d'un même lot. Cependant, il peut exister des précédences entre les lots. Dans le cas préemptif, l'exécution d'une opération  $T_{ij}$  peut être interrompue par l'exécution d'une autre tâche. Elle sera alors terminée plus tard, éventuellement après d'autres interruptions. Notons que dans l'open-shop, la tâche préemptée peut appartenir au même lot, ce qui n'est pas permis dans les autres problèmes d'atelier à cause des contraintes de précédence. Enfin, dans certains problèmes appelés problèmes d'atelier sans attente (no-wait problèmes), les lots doivent être exécutés d'une seule traite, sans attente entre les machines. En d'autres termes, il n'y a pas de préemption des lots. Cette contrainte peut être due à l'absence de capacité de stockage intermédiaire ou au processus de fabrication lui-même. Nous utiliserons la classification des problèmes d'ordonnancement suivant la notation  $\alpha|\beta|\gamma$  proposée par Graham et al [GRA79]. Et étendue par Lawler et al. [LAW93]. Notons  $o$  le symbole vide. Par souci de clarté, nous négligerons la double indexation des tâches  $T_{ij}$  dans un atelier.

- $\alpha$  permet de spécifier l'environnement machine :  $\alpha = \alpha_1; \alpha_2$ 
  - $\alpha_1 = O$  pour l'open-shop,  $F$  pour le flow-shop et  $J$  pour le job-shop.
  - $\alpha_2$  correspond au nombre de machines,  $o$  si ce nombre n'est pas fixe.
- $\beta$  décrit les caractéristiques des tâches :  $\beta = \beta_1; \beta_2; \beta_3; \beta_4; \beta_5; \beta_6$ .
  - $\beta_1 = pmtn$  dans le cas d'un problème préemptif,  $o$  sinon.
  - $\beta_2 = prec$  si des contraintes de précédence existent entre les lots,  $tree$  si le graphe de précédences est une arborescence,  $o$  s'il n'existe pas de contraintes de précédence.
  - $\beta_3 = r_j$  si des dates de disponibilité sont associées aux tâches,  $o$  sinon.
  - $\beta_4 = d_j$  si des échéances sont associées aux tâches,  $o$  sinon.
  - $\beta_5 = (p_j = p)$  si les tâches ont des durées identiques,  $(p_j = 1)$  si les tâches ont des durées unitaires,  $o$  sinon.
  - $\beta_6 = no-wait$  pour les problèmes d'atelier sans attente,  $o$  sinon.
- $\gamma$  est le critère d'optimalité du problème. Notons  $C_j$  les dates d'achèvement des tâches.
  - $\gamma = f_{max}(= \max f_j)$  ou  $f_j(t)$  est la fonction de cout de la tâche  $j$  en fonction de sa date d'achèvement  $t$ . Toutes les fonctions de couts étudiées sont régulières, c'est-à-dire  $f_j$  est une fonction non décroissante de  $t$  pour  $1 \leq i \leq n$ .

- $\gamma = C_{max}$  ( $= \max C_j$ ) si l'on cherche à minimiser le délai total ou date d'achèvement maximale (Makespan).
- $\gamma = \sum C_j$  si l'on veut minimiser le délai moyen (flow time).
- $\gamma = \sum w_j C_j$  si l'on veut minimiser le délai moyen pondère (weighted flow time).
- $\gamma = L_{max}$  ( $= \max (C_j - d_j)$ ) si l'on veut minimiser le retard algébrique maximal (maximum Lateness).
- $\gamma = \sum U_j$  si l'on veut minimiser le nombre de tâches en retard (number of late jobs). La fonction  $U_j$  indique si la tâche  $j$  est en retard (la valeur de  $U_j$  est 1 si la tâche  $j$  est en retard ( $C_j > d_j$ ) et égale à 0 autrement).

Une fonction de cout  $f_j$  est dite additive si  $f_j(t + \Delta) = f_j(t) + f_j(\Delta)$  pour tout  $\Delta$  et incrémentale si  $f_j(t + \Delta) = f_j(t) + \Delta$ . Le délai moyen pondère d'une tâche est une fonction additive, alors que son retard algébrique est une fonction incrémentale.

### 1.5. Applications

Dans la réalité, un lot correspond généralement à un produit qui doit subir des opérations sur différentes machines. Un exemple classique est le problème d'ordonnancement d'un garage dans lequel des voitures (lots) doivent passer sur différents postes de travail (machines) comme la vidange, le graissage, le lavage . . . On retrouve aussi des problèmes de ce type dans l'accomplissement de formalités administratives (passage à différents guichets), dans l'organisation d'exams médicaux de patients hospitalisés, dans la maintenance d'avion lors d'une escale, en télécommunications . . . Ces applications peuvent comporter des contraintes de séquençement ou pas : un examen radiologique doit être programme avant une consultation avec un spécialiste ; un étudiant doit récupérer une convention de stage et un relevé de notes dans un ordre quelconque. Beaucoup de problèmes d'emploi du temps peuvent être vus comme des problèmes d'open-shop avec des contraintes de partage de ressource supplémentaires.

### 1.6. Résultats de complexité :

De nombreux travaux ont été effectués sur la complexité des problèmes d'atelier. Très peu de problèmes d'atelier peuvent être résolus en temps polynomial. Dans la quasi-totalité des cas polynomiaux, les algorithmes sont conçus pour construire des ordonnancements de durée  $C_{max}^{LB}$ . Les cas les plus connus sont les problèmes à deux machines ou préemptifs. Nous allons résumer les principaux résultats connus à ce jour. Nous nous focaliserons sur le délai total  $C_{max}$ . Nous donnerons cependant quelques résultats concernant d'autres critères.

**Exemples :**

$P // C_{max}$  signifie : ordonnancement non-préemptif de tâches indépendantes de temps de service arbitraire, arrivant aux instants 0, sur des machines parallèles, identiques dans l'ordre de minimiser la longueur de l'ordonnancement.

$O3 / mtn, rj / \sum_{j=1}^n C_j$  : signifie ordonnancement préemptif de longueurs de tâches arbitraires dans un open shop à trois machines, les tâches arrivent en des instants différents et l'objectif est de minimiser le flow time moyen

**1.6.1. Problèmes à une seule machine****Exemple :**

De nombreux problèmes d'ordonnancement classés NP-difficiles peuvent être résolus par des procédures par séparation et évaluation. Nous présentons des problèmes d'ordonnancement à une seule machine et un job shop.

Un atelier comporte une seule machine toujours disponible. " $n$ " tâches doivent y être exécutées les unes à la suite des autres. Elles sont toutes disponibles dès l'instant zéro. La tâche  $i$  dure  $p_i$  unités de temps et doit être terminée avant l'instant  $d_i$ , sinon il faut payer une pénalité de retard  $W_i$  par unité de temps de retard. Il s'agit de trouver le meilleur ordre de passage des tâches sur la machine de manière à minimiser la somme des pénalités de retard à payer.

Ensemble des solutions réalisables  $S_0$  est formé des  $n!$  permutations possibles des  $n$  tâches.

La séparation se fera sur la tâche que l'on fixe en dernier, puis en avant dernier, ...

Pour le calcul des bornes ou minorant, on prend en considération la somme des pénalités des tâches placées en queue. L'élimination des sous-ensembles se fera sur le sommet ayant le plus petit minorant. Le problème suivant est à 1 machine et 3 tâches.

| Les tâches | 1 | 2 | 3 |
|------------|---|---|---|
| $p_i$      | 2 | 4 | 1 |
| $d_i$      | 3 | 5 | 5 |
| $w_i$      | 4 | 2 | 3 |

**1.6.2. Problèmes à deux machines**

Parmi les problèmes d'open-shop non préemptifs, le problème  $O2||L_{max}$  dans lequel on cherche à minimiser le retard algébrique maximal des tâches est un problème NP-difficile au sens fort [LAW81], de même que le problème  $O2||\sum C_j$  [ACH82]. Par contre, le problème  $O2||C_{max}$  admet un algorithme linéaire  $O(n)$  proposé par Gonzalez et Sahni [GON76] qui

construit un ordonnancement optimal sans temps mort de durée  $C_{max}^{LB}$ . Il construit séparément deux ordonnancements pour une partition des lots correspondant à la relation  $p_{1i} \geq p_{2i}$  avant de concaténer ces deux solutions partielles pour former un ordonnancement optimal. Remarquons que cet algorithme est aussi valide dans le cas préemptif. Lawler et al [LAW81] proposent un algorithme en  $O(n)$  pour résoudre  $O2|pmtn; r_j|C_{max}$ .

Un ordonnancement optimal pour le problème  $F2||C_{max}$  peut-être recherché parmi les ordonnancements de permutation (la séquence d'exécution des lots est identique sur toutes les machines). On applique la condition suffisante d'optimalité donnée par la règle de Johnson [JOH54] : le lot  $i$  précède le lot  $j$  dans la séquence optimale si  $\min(p_{i1}, p_{j2}) \leq \min(p_{i2}, p_{j1})$ . L'algorithme de Johnson construit un tel ordonnancement en temps polynomial. L'algorithme de Jackson [JAC56] base sur la règle de Johnson résout le problème  $J2||C_{max}$  en temps polynomial. En revanche,  $J2|pmtn|C_{max}$  est NP-difficile, même si  $F2|pmtn|C_{max}$  peut être résolu en temps polynomial [GON78].

### 1.6.3. Problèmes à trois machines : la frontière

Peu de problèmes non préemptifs avec un nombre de machines supérieur à 2 sont polynomiaux. Gonzalez et Sahni [GON76] démontrent que le problème  $O3||C_{max}$  est NP-difficile au sens faible. Cependant, certains cas particuliers sont polynomiaux, Adiri et Aizikowitz [ADI89] exploitent l'existence d'une relation de dominance pour construire un ordonnancement optimal à deux machines avant de placer les tâches sur la machine dominée sans créer de conflits.

Un ordonnancement optimal du problème  $F3||C_{max}$  peut encore être recherché parmi les ordonnancements de permutation. La règle de Johnson peut être étendue lorsque la seconde machine n'est pas un goulet d'étranglement, c'est-à-dire qu'elle est complètement dominée par une des deux autres machines. Le problème est alors résolu après transformation en un problème à deux machines. Par contre, le problème préemptif  $F3|pmtn|C_{max}$  est NP-difficile.[GON78]

### 1.6.4. Cas général

Graham et al. [GRA79] ont démontré la NP-difficulté au sens fort de  $O||C_{max}$ . Le problème devient polynomial lorsque la plus longue tâche est assez courte par rapport à la charge maximale des machines [FIA83]. de Werra [D.WE90], de Werra et Solot [D.WE93] construisent des ordonnancements optimaux de durée quand la structure du graphe biparti pondéré associé à l'instance permet le calcul d'une coloration en temps polynomial. Les problèmes  $F||C_{max}$  et  $J||C_{max}$  sont aussi NP-difficiles au sens fort.

Les problèmes préemptifs sont, en général, plus faciles à résoudre que les non préemptifs car on peut faire appel aux mathématiques du continu. Cho et Sahni [CHO81] utilisent la programmation linéaire pour résoudre le problème  $O|pmtn|L_{max}$ . Lawler et al. [LAW81] proposent un algorithme en  $O(n)$  pour résoudre  $O2|pmtn ; r_j |C_{max}$ . Gonzalez et Sahni [GON76] proposent un algorithme basé sur la construction de couplages de cardinal maximal dans un graphe biparti pour  $O|pmtn|C_{max}$ . En revanche, nous avons vu précédemment que les problèmes préemptifs de job-shop et flow-shop sont tous deux NP-difficiles pour  $m \geq 3$ .

Le problème  $O|r_j ; dj |C_{max}$  est NP-difficile dans le cas général, même si les tâches ont des durées unitaires sauf dans certains cas particuliers. Graham et al. [GRA79] ont démontré que  $O|tree|C_{max}$  est NP-difficile au sens fort pour  $m \geq 2$ , sauf dans certains cas particuliers où les tâches ont des durées unitaires. Pour tous les critères usuels, les problèmes d'open-shop sans attente sont NP-difficiles au sens fort, même les problèmes ne comportant que deux machines [SAH79]. Ces problèmes restent NP-difficiles au sens fort lorsque toutes les tâches ont des durées nulles ou identiques [GON82]. Il est évident que les problèmes d'atelier avec ressources supplémentaires (c'est-à-dire autres que les machines) ne sont pas plus faciles que les mêmes problèmes sans ressource. Ainsi, tous les résultats de NP-difficulté vus précédemment restent valables pour les problèmes avec ressources. Pire, l'introduction de ressources complique le cas préemptif. De Werra et al. [D.WE91], ont démontré que le problème  $O|pmtn|C_{max}$  avec une ressource renouvelable ou consommable est NP-difficile au sens fort.

## 1.7. Conclusion

La résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales : la classe de méthodes exactes et la classe de méthodes approchées.

Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire. C'est la raison pour laquelle, elles sont beaucoup plus utilisées pour la résolution de problèmes faciles. Mais avec beaucoup de développements et d'études d'autres problèmes

sont apparus plus complexes de sorte que l'on ne peut pas résoudre ces méthodes exactes, de sorte que vous pouvez recourir à des méthodes approchées.