République Algérienne démocratique et populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École supérieure en sciences appliquées

–Tlemcen-

«Algorithmes, Modèles et Principes de base d'ordonnancement de la production»

Cours et exercices

Dr SOUIER Mehdi

Ce document est un support de cours et recueil d'exercices et de problèmes d'ordonnancement déterministe des systèmes de production. Il est dédié aux étudiants de la troisième année management industriel et logistique au niveau de l'école supérieur en sciences appliquées de Tlemcen. Il peut aussi servir aux étudiants de la deuxième année de la filière nationale en génie industriel, ainsi qu'aux étudiants et chercheurs qui s'intéressent au pilotage des systèmes de production.

Son intitulé est « Algorithmes, Modèles et Principes de base d'ordonnancement de la production ». Il est le fruit d'enseignement du module ordonnancement de la production I à l'école supérieur en sciences appliquées de Tlemcen et au département du génie électrique et électronique de l'université de Tlemcen.

Tous les étudiants de cette école bénéficient d'une solide culture et de larges connaissances en algorithmique, programmation, optimisation ainsi qu'aux mathématiques. L'objectif de ce cours est donc de s'appuyer sur ces connaissances et ce quelles recèlent d'intuition pour fournir aux problèmes d'ordonnancement des solutions optimales ou satisfaisantes si l'optimalité n'est pas atteinte.

Ce polycopié est composé de cinq chapitres :

Le premier chapitre (chapitre 1) concerne des généralités sur l'ordonnancement. Il reprend les notions essentielles et fondamentales d'ordonnancement d'atelier.

Dans les autres chapitres, nous nous intéressons de manière approfondie aux différents problèmes d'ordonnancement dans plusieurs types d'ateliers qui sont:

- ✓ Les problèmes à une seule machine (chapitre 2).
- ✓ Les problèmes à machines en parallèles (chapitre 3).
- ✓ Les problèmes à machines en série ou de type flow shop (chapitre 4).
- ✓ Les problèmes de type job shop et open shop (chapitre 5).

Dans ce polycopié, les chapitres comprennent des exercices complémentaires. Mais, les étudiants sont invités à chercher eux mêmes les solutions de ces exercices.

Nous mentionnons aussi que ce polycopié est un résumé fait à partir de la bibliographie insérée à la fin de ce document.

Table des matières

Chapitre 1
Généralités sur l'ordonnancement

1. Introduction :

L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de production. Les champs d'application de la théorie d'ordonnancement sont divers, notamment la gestion des systèmes de production, l'organisation des différentes tâches et processus et la gestion de la mémoire dans les systèmes informatiques, la planification de différents projets, l'organisation des activités de services, le transport, l'administration…

À cet effet, les problèmes d'ordonnancement ont été largement étudiés par les informaticiens, les décideurs et les industriels …, depuis de nombreuses décennies. Cette étude est également d'un intérêt théorique toujours renouvelé pour les chercheurs, à cause de la nature fortement combinatoire de ces problèmes et le manque de méthodes de résolutions générales offrant des solutions de qualité et à faible coût.

2. Définitions :

Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.

L'ordonnancement est la programmation dans le temps de l'exécution d'une suite de tâches (activités, opérations) sur un ensemble de ressources physiques (humaines et techniques), en cherchant à optimiser certains critères, financiers ou technologiques et en respectant les contraintes de fabrication et d'organisation.

L'ordonnancement concerne l'affectation de ressources limitées aux tâches dans le temps. C'est un processus de prise de décision dont le but est d'optimiser un ou plusieurs objectifs.

Cette procédure s'effectue en deux étapes dans le but d'atteindre un ou plusieurs objectifs, la première consiste à déterminer la séquence des opérations dans le temps, la deuxième consiste à répartir leur exécution sur les ressources, tout en respectant les contraintes.

Une tâche est une entité élémentaire localisée dans le temps par une date de début et de fin, dont la réalisation nécessite un certain temps, et qui consomme certaines ressources. En ordonnancement d'atelier, on parle aussi d'opération.

La ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche. Les tâches sont en concurrence pour obtenir une ou plusieurs ressources. Ces ressources ne sont disponibles qu'en quantité limitée.

3. Les contraintes

Les contraintes expriment des restrictions sur les valeurs pouvant être prises par les variables de décisions. Ce sont donc des conditions à respecter dans la construction de l'ordonnancement pour qu'il soit réalisable. Il existe plusieurs classifications possibles de contraintes.

Suivant la disponibilité des ressources et l'évolution temporelle. Une telle classification distingue deux types de contraintes en ordonnancement, les contraintes temporelles et les contraintes de ressources.

- ✓ Les contraintes de ressources : comprennent les contraintes de capacité et d'affectation, les premières peuvent distinguer deux types de ressources, les ressources disjonctives qui ne peuvent exécuter qu'une seule tâche à la fois et les ressources cumulatives permettant la réalisation simultanée de plusieurs tâches. Les contraintes d'affectation qui existent dans les systèmes flexibles englobent deux types de contraintes, les contraintes de domaine qui représentent l'ensemble des ressources candidates pour l'exécution d'une tâche et les contraintes de différence qui imposent l'utilisation de ressources différentes pour la réalisation d'un certain nombre de tâches. Lorsque la quantité d'une ressource diminue au fur et à mesure de son utilisation (matières premières, financement d'un projet, ...), elle est dite consommable (consumable). Lorsqu'elle demeure disponible en même quantité (équipe, machine d'un atelier), elle est dite renouvelable ou non-partageable
- ✓ Les contraintes temporaires peuvent être classées en deux catégories, les contraintes de temps absolu et de temps relatif, les contraintes de temps absolu permettent de représenter la limitation des valeurs possibles pour les dates des tâches. Par exemple date de début au plus tôt, date de fin au plus tard… les contraintes de temps relatif sont les contraintes relatives.

Une autre classification consiste à classer les contraintes selon leur lien avec le système de production. Ces contraintes peuvent être classées en deux types : endogènes et exogènes.

- ✓ Les contraintes endogènes constituent les contraintes liées directement au système de production et à ses performances telles que :
  - ✓ les dates de disponibilité des machines et des moyens de transport,
  - ✓ les capacités des machines et des moyens de transport,
  - ✓ les séquences des actions à effectuer ou les gammes des produits.
- ✓ Les contraintes exogènes sont des contraintes imposées extérieurement, et donc indépendantes du système de production; on distingue :
  - ✓ les dates de fin de fabrication au plus tard du produit imposées, généralement par les commandes,
  - ✓ les priorités de quelques commandes et de quelques clients,
  - ✓ les retards possibles accordés pour certains produits.

On peut aussi distinguer les contraintes suivant qu'elles soient strictes ou pas. Les contraintes strictes sont des exigences à respecter alors que les contraintes dites «relâchables» (contraintes de préférences) peuvent éventuellement ne pas être satisfaites.

4. Les objectifs de l'ordonnancement

Un problème d'ordonnancement n'est pas nécessairement exprimé comme un problème d'optimisation. Néanmoins, la notion de critère d'optimisation est toujours présente, cette notion est nécessaire pour juger la pertinence d'un ordonnancement satisfaisant ses contraintes du point de vue exploitation. Dans certains cas, où les objectifs de l'entreprise sont multiples, on cherche à optimiser (ou rapprocher de) plusieurs fonctions objectifs à la fois. Dans ce cas, le problème d'ordonnancement n'est plus un problème simple mais devient un problème multicritères. Ces objectifs sont classés en plusieurs classes :

- ✓ Les objectifs liés au temps : la minimisation du temps total d'exécution, du temps de cycle, des durées totales de réglage ou des retards par rapport aux dates de livraison…
- ✓ Les objectifs liés aux ressources : la maximisation du taux d'utilisation d'une ressource ou la minimisation du nombre de ressources à employer…
- ✓ Les objectifs liés au coût : ces objectifs sont généralement de minimiser les coûts de lancement, de production, de stockage, de transport, de retard, de non occupation des machines…

5. Formulation d'un problème d'ordonnancement

5.1.Notations

Un problème d'ordonnancement est défini par l'ensemble des opérations à réaliser, les machines disponibles pour les exécuter, les contraintes relatives aux opérations et aux machines ainsi que les fonctions objectifs à optimiser.

Le passage d'un job sur une machine est appelé opération. Cette dernière possède des caractéristiques temporelles : sa durée opératoire (processing time) (durée d'exécution), sa date de disponibilité (release date) (date à laquelle elle peut être exécutée au plus tôt), qui lorsqu'elle existe est impérative, sa date de fin au plus tard souhaitée (due date) ou date d'échéance, les contraintes de précédence (precedence constraints), et qui représente un ordre partiel des opérations. La date d'échéance peut souvent être violée au prix de pénalités diverses ; si elle est impérative, elle est appelée deadline. L'opération est soit non préemptive (non preemptive) si elle doit être réalisée sans interruption, soit préemptive (preemptive) si elle peut être exécutée par morceaux.

Généralement, nous considérons un problème $V = \{J_1 , J_2 , ..., J_n \}$ de n travaux à ordonnancer sur m machines $\{m_1 , m_2 , ..., m_m \}$. Si nous utilisons l'indice j pour désigner le travail et l'indice i pour désigner la machine, alors les principales notations des données relatives à ce problème sont :

- ✓ La durée opératoire (processing time en anglais) $p_{i,j}$: c'est le temps d'exécution du travail j sur la machine i. Il sera noté p_j si le travail doit être exécuté sur une seule machine.
- ✓ La date de disponibilité (release date en anglais) r_j : c'est la date d'arrivée du travail j dans le système, elle représente aussi sa date de début au plus tôt.
- ✓ La date d'échéance (due date en anglais) d_j : c'est la date d'exécution préférentielle du travail j (date de livraison promise de j pour le client); lorsque cette date d'échéance doit être respectée, on parlera de date d'échéance stricte (deadline en anglais).
- ✓ Le poids (weight en anglais) w_j : c'est un facteur de priorité qui dénote l'importance du travail j relativement aux autres.
- ✓ C_j : est la date effective de fin de fabrication (completion date) du job j
- ✓ Une date de début t_j,
- ✓ Retard algébrique (Lateness) du job j : $L_j=C_j - d_j$, si L_j est négative sa valeur indique l'avance du job j par rapport à sa date de fin au plus tard, sinon, L_j indique le retard du job j.
- ✓ E_j : avance (earliness) du job J_j . $E_j = \max(d_j - C_j , 0)$,
- ✓ T_j : retard (tardiness) du job J_j . $T_j = \max(C_j - d_j , 0)$,

- ✓ U_j : indicateur de retard (unit penalty) du job J_j. $U_j = 1$ si $T_j > 0$, sinon $U_j = 0$
- ✓ O_{ij} : j ème opèration du job i,
- ✓ t_{ij}: date de début (starting date) de O_{ij},
- ✓ p_{ij}: durée opératoire (processing time) de O_{ij},
- ✓ C_{ij}: date de fin (completion date) de O_{ij},

5.2.Classification des problèmes d'ordonnancement

Comme il existe beaucoup de variantes des problèmes d'ordonnancement, un effort de classification est fait dans la littérature. Cette classification permet de décrire simplement et efficacement le problème d'ordonnancement. Grâce à cette classification, il est facile d'identifier un problème d'ordonnancement sans ambiguïté.

La notation la plus utilisée en ordonnancement introduite par Graham et al en 1979, décrit les problèmes d'ordonnancement en trois champs $\alpha \mid \beta \mid \gamma$. Le champ α décrit la catégorie du problème d'ordonnancement (le nombre de machines et leur organisation). Le champ β décrit les contraintes supplémentaires. Le champ γ décrit l'objectif à optimiser.

Le champ α se divise en deux sous champs concaténés α_1 et α_2. Le champ α_2, optionnel, indique le nombre de machines dans le problème. Si ce champ est vide, le nombre de machines est une variable du problème. Les différents environnements machines possibles (voir figure 1.1), spécifiés dans le champs α_1, sont :

- ✓ Ressource unique : c'est le cas le plus simple où une seule ressource disjonctive (ne pouvant traiter qu'une opération à la fois) exécute les tâches. Dans ce cas, le champ α_2 est absent.
- ✓ Machines parallèles : il existe m machines parallèles. Chaque opération doit être exécutée sur une machine à sélectionner dans l'ensemble des m machines parallèles. Les machines peuvent être alors identiques, uniformes ou indépendantes. Le champs α_1 prend ainsi respectivement les valeurs P (machines identiques), Q (un environnement avec machines uniformes où chaque machine a sa propre vitesse du traitement) ou R (il y'a des machines différentes, la vitesse du traitement dépend des machines et des jobs).
- ✓ Les ateliers à cheminement unique ou "flow shop" : il existe m machines en série. Tous les travaux ont la même gamme opératoire, et l'ordre de visite des travaux est le même pour chaque machine. Le champ $\alpha_1 = F$.
- ✓ Les ateliers à cheminement multiples ou "job shop" : c'est un cas plus général que le flow shop, où chaque travail a sa propre gamme opératoire. Le champ $\alpha_1 = J$.

- ✓ Les ateliers à cheminements libres ou " open shop" : il existe *m* machines. L'ordre de passage sur les machines est libre pour tous les travaux, i.e., il n'existe aucune relation de précédence entre les opérations d'un même travail. Le champ $\alpha_1 = O$.

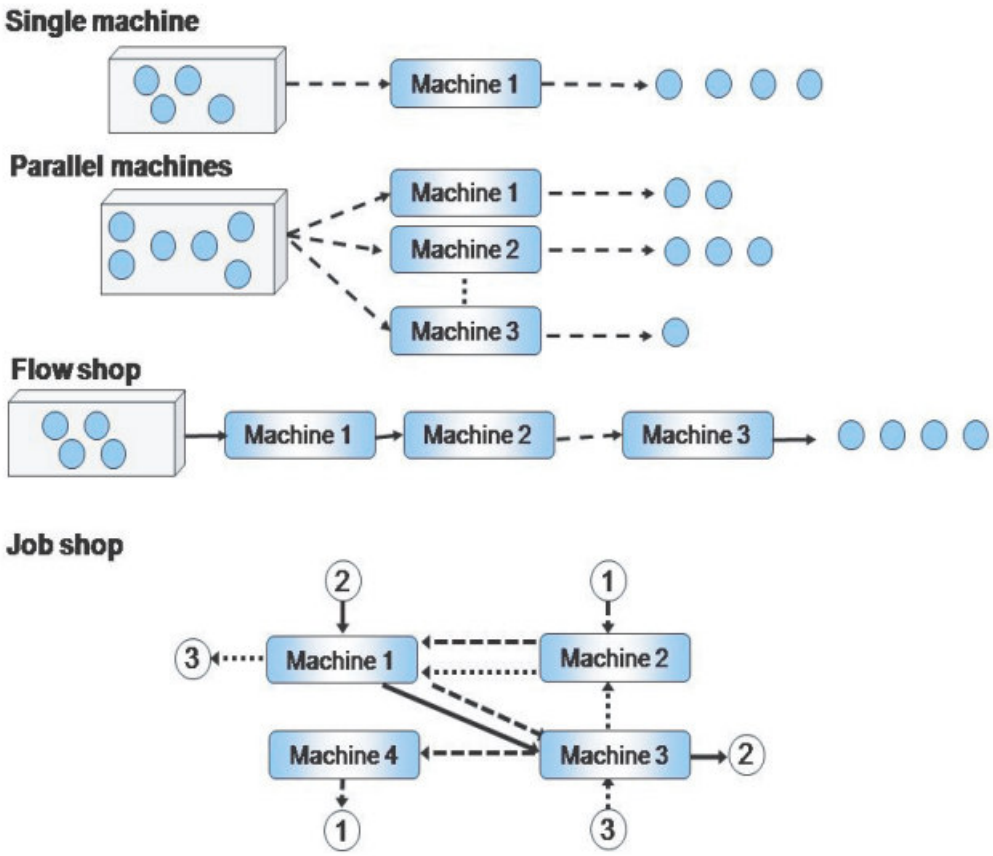


Figure 1.1: Exemples de problèmes d'atelier

Le champ β permet de décrire des contraintes additionnelles au problème. Il est constitué d'une suite de sous champs séparés par des virgules. Les principales valeurs de ces sous-champs sont:

- ✓ *prmp* : préemption : (les opérations peuvent être exécutées par morceaux.).
- ✓ *prec* : précédence : existence de contraintes générales de précédence entre les opérations (certaines tâches doivent être exécutées avant d'autres.).
- ✓ *brkdwn* : Pannes : les ressources (machines) ne sont pas disponibles en permanence.
- ✓ *prmu*: permutation : l'ordre selon lequel les jobs passent sur la première machine est maintenu à travers le système. Il n'y a pas de permutation possible entre les tâches dans un flow shop puisque les files d'attente suivent en générale la règle FIFO.
- ✓ *Block* : Blocage : le job complété doit rester sur la machine en amont prévenant ou bloquant cette machine d'effectuer un autre job.

- ✓ *nwt* : no-wait : implique que les jobs ne peuvent attendre entre deux machines successives.
- ✓ *recrc* : recirculation : implique qu'un job peut passer sur une machine ou un pool plus d'une fois (On peut le voir sur un environnement job shop ou flexible job shop).
- ✓ M_j : machine eligibility restriction : restriction d'admissibilité des machines. Le job J ne peut pas être effectué par n'importe quelle machine. L'ensemble de machines M_j détermine l'ensemble des machines qui peuvent traiter le job J.
- ✓ $S_{ii'}$: temps de préparation (changement) dépendant de la séquence entre les jobs J_i et $J_{i'}$.
- ✓ $p_{ij} = p_j$: toutes les durées opératoires sont égales à p_j.
- ✓ r_j : chaque job J possède une date de disponibilité.
- ✓ d_j : chaque job J possède une date échue (due date).
- ✓ S_{ri} : temps de préparation de la machine M_r pour le job J_i.
- ✓ M_r^k : la machine M_r possède k périodes d'indisponibilité.

γ : représente la fonction objectif à optimiser

- ✓ $C_{max} = \max\{C_i, i = 1,\ldots, n\}$: date de fin de tous les jobs ou Makespan. Il correspond à la date de fin de la dernière opération de l'ordonnancement. Un Makespan minimum implique usuellement une haute utilisation des machines (productivité).
- ✓ $\sum_i C_i$: .somme des dates de fin des opérations. On le réfère comme flow time. Ainsi, la somme pondérée des dates de fin $\sum_i w_i C_i$ est désigné comme le flow time pondéré. Cela donne une indication sur le coût d'exploitation et d'inventaire induits par l'ordonnancement (minimisation des encours).
- ✓ $L_{max} = \max\{L_i, i = 1, \ldots, n\}$: retard algébrique maximum. Il mesure la pire violation des dates échues.
- ✓ $T_{max} = \max\{T_i, i = 1, \ldots, n\}$: retard maximum.
- ✓ $\sum_i T_i$: .somme des retards sur les dates d'achèvement des jobs.
- ✓ $\sum_i w_i T_i$: Somme pondérée des retards.
- ✓ $E_{max} = \max\{E_i, i = 1, \ldots, n\}$: maximum des avances.
- ✓ $\sum_i U_i$. nombre de jobs en retard.
- ✓ $\sum_i w_i U_i$: nombre pondéré de jobs en retard.

5.3. Diagramme de Gantt

Le diagramme de Gantt (par H. Gantt) permet de montrer les séquences de traitement sur chaque machine et les dates de début et de fin des jobs. En effet, il se compose de lignes horizontales désignant les machines; les opérations y sont représentées, en fonction des machines correspondantes, à partir de leurs dates de début d'exécution, sous forme de barres ayant des longueurs proportionnelles à leurs durées opératoires. La figure 1.2 montre un exemple d'un digramme de Gantt.

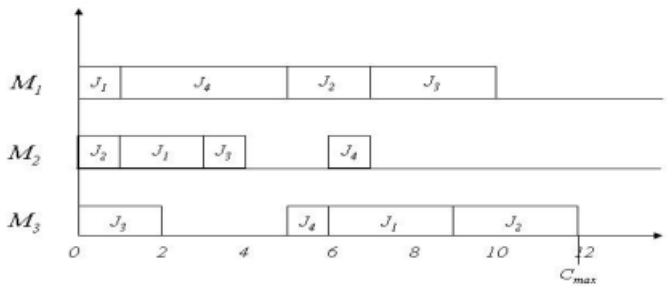


Figure 1.2 : un exemple d'un digramme de Gantt

6. Types d'ordonnancement

Il existe différents types d'ordonnancement définis comme suit (voir figure 1.3):

Un ordonnancement est semi-actif (semi-active) lorsqu'il est impossible d'avancer une opération sans modifier la séquence des opérations sur la ressource.

Il est dit actif (active) s'il est impossible d'avancer une opération sans reporter le début d'une autre opération.

Il est dit sans retard ou sans délai (non-delay) si et seulement si aucune opération n'est mise en attente lorsqu'une machine est disponible pour l'exécuter.

- ✓ Pour la majorité des problèmes, il existe des ordonnancements optimaux sans délais.
- ✓ Pour tout les problèmes avec préemention, les ordonnancements optimaux sont sans délais.
- ✓ Dans certains problèmes non préemptifs, il est intéressant de provoquer des arrêts forcés des ressources

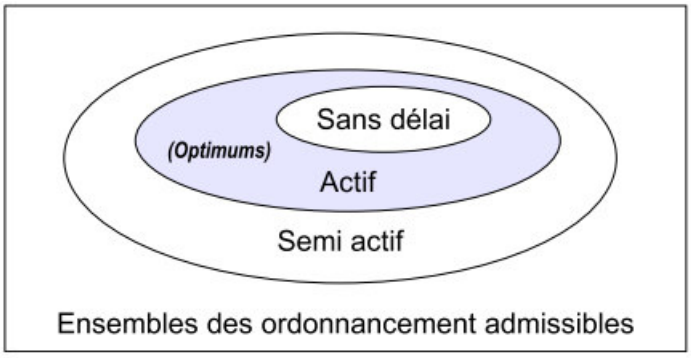


Figure 1.3. Les relations entre les différentes types d'ordonnancement

Les relations entre les différentes classes d'ordonnancement sont illustrées par le diagramme de Venn présenté sur la Figure 1.3. Celle ci montre que chaque ordonnancement sans délai est actif et chaque ordonnancement actif est semi actif.

Un ordonnancement actif n'est pas toujours sans délai, l'exemple suivant décrit un ordonnancement actif mais n'est pas sans délai:

Exemple: Considérons un atelier avec trois machines et deux jobs. Le travail 1 nécessite une unité de temps sur la machine 1 et 3 unités de temps sur la machine 2. Job 2 nécessite 2 unités de temps sur la machine 3 et 3 unités de temps sur la machine 2. Les deux tâches doivent être traitées en dernier sur la machine 2.

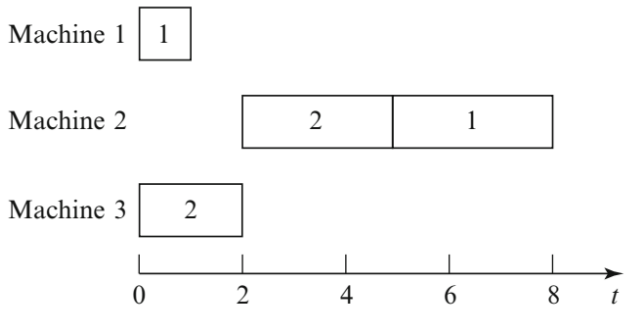


Figure 1.4. Un ordonnancement actif qui n'est pas sans délai

Considérons le planning qui traite le travail 2 sur la machine 2 avant le travail 1 (voir Figure 1.4). Il est clair que cet ordonnancement est actif; l'inversion de la séquence des deux travaux sur la machine 2 retarde le traitement du travail 2. Cependant, il n'est pas sans délai car la machine 2 reste inactive jusqu'à l'instant 2, alors qu'une tâche est déjà disponible pour être traitée à l'instant 1.

Un exemple d'un ordonnancement semi actif qui n'est pas sans délai peut être décrit comme suit:

Considérons l'exemple précédent mais cette fois les temps de traitement du job 1 sur les machines 1 et 2 sont égaux à 1 et les temps de traitement du travail 2 sur les machines 2 et 3 sont égaux à 2.

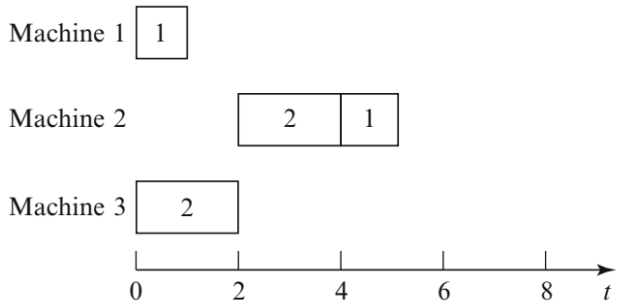


Figure 1.5. Un ordonnancement semi actif qui n'est pas actif

Considérons le calendrier dans lequel le travail 2 est traité sur la machine 2 avant le travail 1 (voir figure 1.5). Cela implique que le travail 2 commence son traitement sur la machine 2 à l'instant 2 et que le job 1 commence son traitement sur la machine 2 à l'instant 4. Cet ordonnancement est semi actif. Cependant, il n'est pas actif, car le travail 1 peut être traité sur la machine 2 sans retarder le traitement du travail 2 sur la machine 2.

7. Complexité des problèmes d'ordonnancement

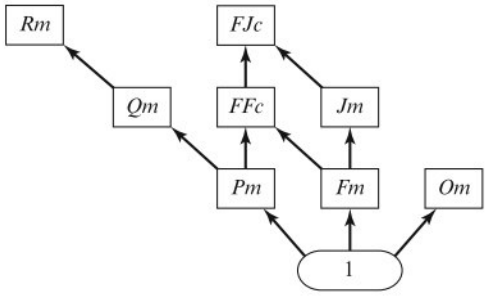
La théorie de la complexité s'intéresse à l'étude formelle de la difficulté théorique des problèmes en informatique ou d'un problème par rapport à un autre et de l'analyse de la complexité des programmes et des algorithmes. Concrètement, on cherche à savoir si le problème étudié est plutôt « facile » ou plutôt « difficile » à résoudre en se basant sur une estimation (théorique) des temps de calcul et des besoins en mémoire informatique.

Les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire. Pour résoudre un problème d'ordonnancement, nous devons toujours chercher à établir sa complexité, car cela détermine la nature de l'algorithme à mettre en œuvre.

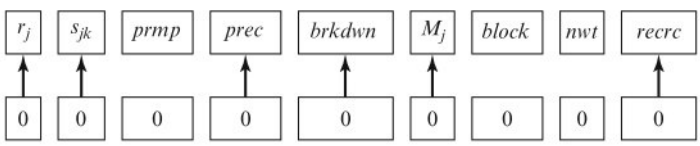
En optimisation, la complexité d'un problème est équivalente à celle du meilleur algorithme proposé pour le résoudre, c'est-à-dire un problème est facile s'il existe un algorithme capable de le résoudre en un temps polynomial, sinon le problème est considéré comme difficile ou intraitable. Mais, la théorie de complexité est basée sur les problèmes de décision et non d'optimisation. Donc, il y'a une distinction entre les deux problèmes, un problème de décision est un problème qui admet toujours une réponse oui ou non. Cependant, un problème d'optimisation peut toujours être réduit à un problème de décision, en transformant par exemple la fonction objectif en une fonction binaire.

La théorie de complexité qui a été définie dans le cadre des problèmes de décision plutôt que dans celui des problèmes d'optimisation, a classé les problèmes de décision en plusieurs classes dénommées classes de complexité, chacune d'entre elles représente un ensemble de problèmes qui peuvent être résolus en utilisant une quantité donnée de ressources de calcul. Voici quelques notions en ce qui concernent ces classes :

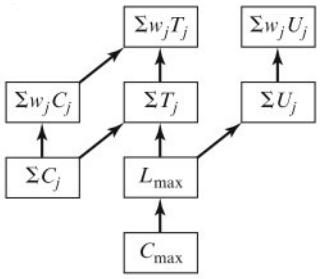
- ✓ La classe de complexité polynomiale P : regroupe l'ensemble des problèmes de décisions traités par des algorithmes déterministes en un temps polynomial.
- ✓ La classe des problèmes NP (Non deterministic Polynomial) est la classe des problèmes de décision pouvant être résolus par un algorithme polynomial non déterministe. Ce type contient un ou plusieurs points de choix dans lequel de multiples suites différentes sont possibles. Ces problèmes peuvent être résolus en énumérant l'ensemble des solutions possibles qui peuvent être évaluées à l'aide d'un algorithme polynomial.
- ✓ Un problème de décision est NP-complet quand tous les problèmes appartenant à la classe NP lui sont réductibles. Ainsi, s'il existe un algorithme polynomial pour un problème NP-complet alors tous les problèmes de classe NP peuvent être résolus en un temps polynomial.
- ✓ Un problème d'optimisation est dit NP-difficile (NP-hard) si le problème de décision qui lui correspond est NP-complet. Ces problèmes nécessitent un temps exponentiel pour être résolus.



a) L'environnement du problème d'ordonnancement



b) Restrictions et contraintes de traitement



c) Fonction objectif

Figure 1.4: Hiérarchies de complexité des problèmes de planification déterministes

Par conséquent, si le problème d'ordonnancement étudié appartient à la classe P, nous savons d'avance qu'un algorithme polynomial existe pour le résoudre. Dans le cas contraire, si le problème est NP-difficile, deux approches sont possibles.

La première est de proposer un algorithme approché, donc une heuristique, qui calcule en temps polynomial une solution s'approchant au mieux de la solution optimale. Une alternative est de proposer un algorithme qui calcule la solution optimale du problème, mais pour lequel la complexité dans le pire des cas est exponentielle. Dans ce cas, le défi est de concevoir un algorithme qui peut résoudre en temps acceptable les problèmes de la plus grande taille possible.

Souvent, un algorithme pour un problème d'ordonnancement peut également être appliqué à un autre. Par exemple, $1||\sum C_j$ est un cas particulier du problème $1||\sum W_j C_j$ et un algorithme pour le problème $1||\sum W_j C_j$ peut être utilisé pour résoudre $1||\sum C_j$. Dans la théorie de complexité, on dit alors que $1||\sum C_j$ se réduit à $1||\sum W_j C_j$. Ceci est généralement indiqué par $1||\sum C_j \propto 1||\sum W_j C_j$

Sur la base de ce concept, une chaîne de réductions peut être établie.

Par exemple: $1||\sum C_j \propto 1||\sum W_j C_j \propto P_m||\sum W_j C_j \propto Q_m|Prec|\sum W_j C_j$

Bien sûr, il y a aussi beaucoup de problèmes qui ne sont pas comparables entre eux. Par exemple $P_m||\sum W_j T_j$ n'est pas comparable à $J_m||C_{max}$

Un effort considérable a été fait pour établir une hiérarchie de problèmes décrivant les relations entre les centaines de problèmes d'ordonnancement. Dans les comparaisons entre les complexités des différents problèmes d'ordonnancement, il est intéressant de savoir comment la modification d'un seul élément de la classification d'un problème affecte sa complexité. Dans la figure 1.4, certain graphes sont présentés pour aider à déterminer la hiérarchie de complexité des problèmes d'ordonnancement déterministes.

Exercices complémentaires

Exercice 1

1. Trouvez un ordonnancement actif qui ne soit pas sans délai.
2. Trouvez un ordonnancement semi actif qui ne soit pas actif.

Exercice 2

1. Trouvez la hiérarchie de la complexité pour les problèmes suivants:

- $1 \mid\mid C_{max}$,
- $P_2 \mid\mid C_{max}$,
- $F_2 \mid\mid C_{max}$,
- $J_m \mid\mid C_{max}$,
- $FFc \mid\mid C_{max}$.

Exercice 3

1. Trouvez la hiérarchie de la complexité pour les problèmes suivants:

- $1 \mid\mid L_{max}$,
- $1 \mid prmp \mid L_{max}$,
- $1 \mid r_j \mid L_{max}$,
- $1 \mid r_j, prmp \mid L_{max}$,
- $P_m \mid\mid L_{max}$.

Chapitre 2
Ordonnancement de problèmes à une seule machine

1. Introduction :

L'étude du problème avec machine unique (chaque job ne comprend qu'une seule opération à laquelle il est assimilé) sert de base pour développer des outils de gestion des environnements multi-machines plus réalistes bien que plus complexes.

Ces problèmes tirent l'attention pour plusieurs raisons. Les problèmes d'ordonnancement avec une seule machine sont simples et représentent des cas particuliers de tous les environnements. Ces types de problèmes sont importants à la fois en raison de leurs propres valeurs intrinsèques, ainsi que leur rôle de composantes de base pour former et résoudre des problèmes plus généralisés et complexes. Leurs modèles ont donc des propriétés qu'aucun des autres environnements ne possède. Leurs résultats fournissent des bases pour les environnements complexes. En outre, les problèmes complexes peuvent être décomposés en problèmes à une seule machine…

Dans ce contexte, pour bien comprendre le comportement d'un système complexe, il est essentiel de comprendre ses composants, très souvent le problème de la machine unique apparaît comme partie d'un problème d'ordonnancement plus important. Parfois, il peut même être possible de résoudre le problème de la machine unique intégrée de manière indépendante, puis d'intégrer le résultat dans l'environnement à plusieurs machines. Par exemple, dans les processus à opérations multiples, une étape de goulot d'étranglement peut exister. Dans ce cas, les ordonnanceurs à ressource unique peuvent être utilisés dans des goulots d'étranglement ou pour organiser l'affectation de tâches à une ressource coûteuse. D'autre part, une ligne de production entière peut, parfois, être traitée comme un seul processeur. Également, comparés à plusieurs problèmes d'ordonnancement, les problèmes à machine unique sont mathématiquement plus faciles à gérer. Par

conséquent, différentes classes de ce type de problèmes peuvent être résolues en un temps polynomial. De plus, une plus grande variété de paramètres de modèle, tels que différents types de fonctions de coût, ou une introduction de coût de changement, peuvent être analysés.

La simplicité relative de l'ordonnancement à une seule ressource ainsi que son caractère fondamental pour les problèmes d'ordonnancement multi ressources sont les facteurs de motivation d'une étude profonde de ce type de problèmes. Dans ce chapitre, différents modèles à machine unique sont analysés en détail.

En raison de l'importance de ces problèmes, de nombreux travaux existent pour les modéliser et les résoudre. Pour notre part, puisque nous ne pouvons pas décrire tous ces problèmes, nous nous limitons à quelques cas particuliers dont l'optimalité est prouvée et peut être atteinte avec des algorithmes polynomiaux. La prochaine section consiste à décrire les différentes hypothèses qui forment les problèmes à une seule machine, et les autres sections sont réservées aux différentes approches de résolution des problèmes concernés par notre étude. Le temps d'achèvement total (The total completion time) et le temps d'achèvement total pondéré (The total weighted completion time) sont considérés en premier, suivis de plusieurs objectifs liés à la date d'échéance, y compris le retard maximal, le nombre des travaux (jobs) tardifs.

2. Hypothèses inhérentes à un problème à une machine:

Dans un modèle à une seule machine; Outre la limitation de la capacité de machine unique, le problème de base peut être décrit par les hypothèses suivantes:

- ✓ Il y a *n* jobs à une seule opération simultanément disponibles pour le traitement à l'instant zéro.
- ✓ La machine ne peut traiter qu'une seule opération à la fois.
- ✓ Les temps d'installation des jobs sont indépendants de la séquence des jobs et sont inclus dans les temps de traitement.
- ✓ Les attributs des jobs sont déterministes et connus à l'avance.
- ✓ Les machines sont disponibles en permanence (sans présence de pannes).
- ✓ La machine ne reste jamais libre lorsqu'il y'a des opérations en attente du traitement.
- ✓ Une fois le traitement d'une opération commence, il se déroule sans interruption.

Dans ces conditions, il existe une séquence des *n* travaux et une permutation des indices 1, .., *n*. Le nombre total de solutions distinctes du problème de base d'une seule machine est donc *n!* qui représente le nombre de différentes séquences de *n* éléments. Chaque fois qu'un programme

d'ordonnancement peut être complètement caractérisé par une permutation d'entiers, cela s'appelle un ordonnancement de permutation (a permutation schedule).

3. The Total Completion Time ($1||\sum C_j$)

Parfois, les coûts associés aux décisions d'ordonnancement impliquent un service aux clients, comme le temps passé d'un produit dans le système, l'exécution rapide du calendrier d'ordonnancement, ...

Le temps passé par un travail (job j) dans le système est son temps d'écoulement ou d'achèvement (Completion date C_j) , et l'objectif d'exécution rapide peut être interprété comme une réduction du temps d'achèvement total ($\sum C_j$). Ceci a une relation avec d'autres objectifs qui consistent à minimiser:

- ✓ La date de fin moyenne des tâches.
- ✓ L'attente moyenne des clients dans une file.
- ✓ Les stocks d'encours devant la ressource en réduisant au minimum le nombre moyen de travaux dans le système.

Plusieurs règles de séquencement ou de priorité (Dispatching rules) sont proposées pour la résolution des problèmes d'ordonnancement. Le séquencement des tâches dans l'ordre non décroissant des temps de traitement est connu sous le nom de séquencement du temps de traitement le plus court (Shortest Processing Time (SPT)). Il est également connu par une variété de noms, tels que le temps de fonctionnement le plus court (Shortest Operation Time (SOT)) et la plus courte opération imminente (Shortest Imminent Operation (SIO)).

Le théorème 2.1 formalise l'optimalité de la règle SPT pour résoudre les problèmes de type $1||\sum C_j$ et sa preuve illustre une technique utile, appelée la méthode d'échange par paires adjacentes.

Théorème 2.1:

Le Total flow time ou Total Completion Time (le temps d'achèvement total) dans un problème $1||\sum C_j$ est minimisé en appliquant la règle SPT (en ordonnançant les tâches dans l'ordre croissant des temps d'exécution ou de traitement (processing time or operating time)).

Preuve du théorème 2.1:

Considérons une séquence S qui n'est pas la séquence SPT. Autrement dit, quelque part dans S, il doit exister une paire de travaux adjacents, i et j, avec j suivant i, tel que $p_i>p_j$. Construisons maintenant une nouvelle séquence, S', dans laquelle les travaux i et j sont échangés en séquence et tous les autres travaux finissent en même temps que dans S. Ces deux cas sont illustrés dans la

figure 2.1, où *B* désigne l'ensemble des travaux précédant les travaux *i* et *j* dans les deux séquences, et *A* désigne l'ensemble des travaux suivant *i* et *j* dans les deux cas.

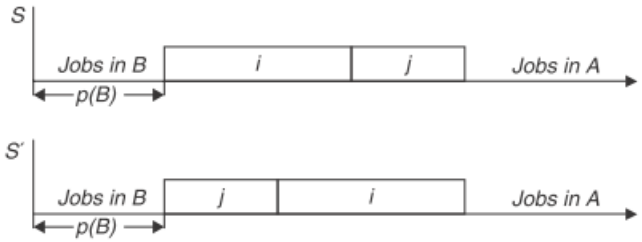


Figure 2.1: Un échange par paire de travaux adjacents.

Nous utilisons la notation $k \in A$ lorsque le travail *k* est membre de l'ensemble *A*. En plus, *p(B)* indique le temps de traitement total pour les travaux dans l'ensemble *B* qui représente le moment où le job *i* commence dans *S* et le job *j* commence dans *S′*. En outre, nous adoptons temporairement la notation $F_k(S)$ pour représenter le temps d'écoulement du travail *k* dans le programme *S*.

Nous montrons d'abord que $\sum_{j=1}^{n} F_k$ est plus petit sous *S′* que sous *S*.

$$\sum_{j=1}^{n} F_k(S) = \sum_{k\epsilon B} F_k(S) + F_i(S) + F_j(S) + \sum_{k\epsilon A} F_k(S)$$

$$= \sum_{k\epsilon B} F_k(S) + (p(B)+p_i) + (p(B)+p_i+p_j) + \sum_{k\epsilon A} F_k(S) \qquad (1)$$

$$\sum_{j=1}^{n} F_k(S') = \sum_{k\epsilon B} F_k(S') + F_j(S') + F_i(S') + \sum_{k\epsilon A} F_k(S')$$

$$= \sum_{k\epsilon B} F_k(S') + (p(B)+p_j) + (p(B)+p_j+p_i) + \sum_{k\epsilon A} F_k(S') \qquad (2)$$

Puisque les séquences *A* et *B* sont les mêmes dans les deux cas, alors:

$(1)-(2) = (p(B)+p_i)+(p(B)+p_i+p_j) - [(p(B)+p_j)+(p(B)+p_j+p_i)] = p_i-p_j>0$

En d'autres termes, l'échange des jobs *i* et *j* réduit la valeur de *F*. Par conséquent, toute séquence qui n'est pas une séquence SPT peut être améliorée par rapport à *F* en échangeant une paire de tâches adjacentes. Il en résulte que la séquence SPT elle-même doit être optimale.

L'essence de cet argument est une preuve par contradiction. Tout d'abord, nous supposons qu'une séquence non-SPT est optimale. Nous montrons ensuite avec un échange par paire de tâches adjacentes qu'une amélioration stricte peut être apportée à cette séquence optimale. Par conséquent, nous concluons qu'il est impossible pour une séquence non SPT d'être optimale.

Il est également instructif d'interpréter cette logique comme une preuve par construction:

1. Commencez avec n'importe quelle séquence non-SPT.

2. Trouvez une paire de travaux adjacents *i* et *j*, avec *j* suivant *i*, tels que $p_i > p_j$.
3. Échangez les tâches *i* et *j* en séquence, ce qui améliore ainsi la mesure de la performance.
4. Revenez à l'étape 2 de manière itérative, en améliorant la mesure de la performance à chaque fois, jusqu'à ce que la séquence SPT soit construite.

La validité de l'un ou de l'autre argument n'est pas affectée par l'existence d'une paire de travaux avec $p_i = p_j$. De plus, la méthode d'échange par paire adjacente est utile dans d'autres situations, comme nous le verrons plus tard.

4. The Total Weighted Completion Time

4.1 Le problème $1||\sum W_jC_j$.

Dans de nombreux problèmes concernant le temps d'achèvement total, les tâches n'ont pas la même importance. Une façon de distinguer les jobs consiste à attribuer une valeur ou un poids, w_j, à chaque travail *j* et à incorporer ces poids dans la mesure des performances. Dans ce cas, la version pondérée du problème présenté dans la section précédente est appelée le temps d'achèvement total pondéré (the total weighted completion time), et est classée par: $1||\sum W_jC_j$.

Le poids W_j de chaque job *j* peut être considéré comme un facteur d'importance; il peut représenter soit un coût de possession par unité de temps ou la valeur déjà ajoutée au travail *j*. Ayant vu que la règle optimale pour minimiser le temps d'achèvement total est la règle SPT. Nous devrions nous attendre à ce que la règle optimale pour le temps total pondéré soit une version pondérée de cette règle. Ce problème donne lieu à l'une des règles les plus connues de la théorie d'ordonnancement, appelée Weighted Shortest Processing Time first (WSPT) rule. Selon cette règle, les travaux sont classés par ordre décroissant de w_j / p_j.

Théorème 2.2:

La règle WSPT est optimale pour le problème $1||\sum W_jC_j$.

Preuve du théorème 2.2:

Une preuve par la méthode de l'échange par paires adjacentes est analogue à la preuve du théorème 2.1. Par contradiction, supposons qu'une séquence *S* qui n'est pas WSPT est optimale.

Dans ce calendrier, il doit y avoir au moins deux emplois adjacents, par exemple, le travail *j* suivi du travail *k*, de telle sorte que:

$$\frac{W_j}{P_j} < \frac{W_k}{P_k}$$

Supposons que le travail j commence son traitement à l'instant t. Puis, effectuons un échange par paires adjacentes sur les travaux j et k pour obtenir une séquence S'. Alors que sous le planning d'origine S, le travail j commence son traitement à l'instant t et est suivi par le travail k, sous le nouveau programme S', le travail k commence son traitement à l'instant t et est suivi par le travail j.

Les autres jobs restent dans leurs positions. Donc, le temps d'achèvement total pondéré des travaux traités avant ou après les travaux j et k n'est pas affecté par l'échange. Ainsi, la différence dans les valeurs des performances des séquences S et S' n'est due qu'aux travaux j et k (Voir figure 2.2). Sous S le temps total d'achèvement pondéré des travaux j et k est :

$$(t+p_j)w_j + (t+p_j+p_k)w_k$$

tandis que sous S' c'est: $(t+p_k)w_k + (t+p_k+p_j)w_j$

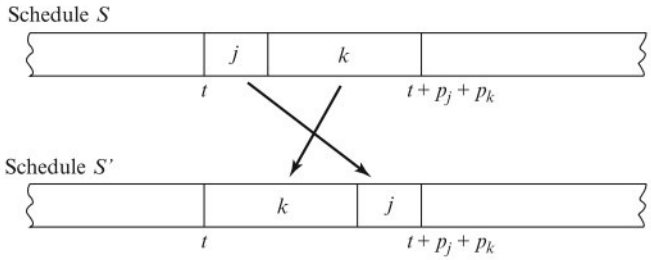


Figure 2.2: Un échange par paire de travaux pondérés et adjacents *j* et *k*

On peut facilement vérifier que si $\frac{W_j}{P_j} < \frac{W_k}{P_k}$, la somme des deux temps d'achèvement pondérés sous S' est strictement inférieure à celle de S. Cela contredit l'optimalité de S et complète la preuve du théorème.

4.2 Le problème $1|Chain|\sum W_jC_j$.

Considérons le problème du temps d'achèvement total pondéré avec présence des contraintes de précédence qui prennent la forme de chaînes parallèles (Voir figure 2.3). Ce problème peut être résolu par un algorithme relativement simple et très efficace (temps polynomial). Cet algorithme est basé sur certaines propriétés fondamentales des contraintes de précédences.

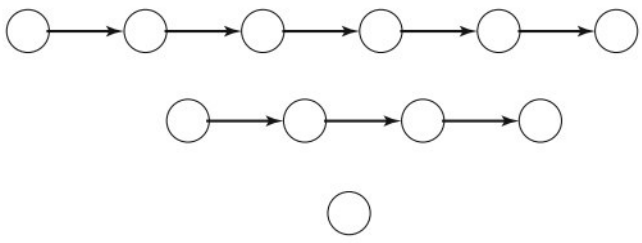


Figure 2.3: Les contraintes de précédences sous formes de chaînes

Considérons deux chaînes de travaux. Une chaîne I est constituée des travaux 1, ..., k et l'autre chaîne, par exemple chaîne II, comprend les travaux k + 1, ..., n. Les contraintes de précédences sont comme suit:

$$1 \rightarrow 2 \rightarrow \cdots \rightarrow k$$

et

$$k+1 \rightarrow k+2 \rightarrow \cdots \rightarrow n$$

Le lemme suivant est basé sur l'hypothèse que si l'ordonnanceur décide de commencer à traiter des tâches d'une chaîne, il doit terminer la chaîne entière avant de pouvoir travailler sur des tâches de l'autre chaîne.

Lemme 2.1:

Si: $\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > (<) \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$, il est alors optimal de traiter la chaîne des travaux 1, ..., k avant (après) la chaîne des travaux k +1, ..., n.

Preuve du Lemme 2.1:

Le lemme peut être prouvé en appliquant une démonstration par contradiction et un échange entre deux chaînes de travaux adjacentes qui est généralement appelé un échange de séquences adjacentes. Un tel échange est une généralisation d'un échange par paires adjacentes.

Selon le Lemme 2.1, la règle d'ordonnancement WSPT par rapport **aux chaînes à exécution sans interruption** est optimale. Donc, les chaînes doivent être classées de manière décroissante par rapport à $\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j}$

Dans ce qui suit, nous allons répondre à la question: Comment la minimisation du temps d'exécution total pondéré est-elle affectée par des contraintes de précédence dans des **chaînes à exécution avec interruption** ?

Supposons maintenant que le planificateur ne soit pas obligé de terminer tous les travaux d'une chaîne avant de pouvoir travailler sur une autre chaîne. Il peut traiter certaines tâches d'une chaîne (tout en respectant les contraintes de précédence), basculer vers une autre chaîne et, ultérieurement, revenir à la première chaîne.

Chaque chaîne ($1 \rightarrow 2 \rightarrow \cdots \rightarrow k$) peut avoir une caractéristique importante qui peut être définie comme suit: posez l^* tel que:

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = max_{1 \le l \le k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j}\right)$$

Le rapport du côté gauche est appelé facteur ρ de la chaîne 1, ..., k et est noté ρ (1, ..., k). Le travail l^* est appelé le travail qui détermine le facteur ρ de la chaîne.

Lemme 2.2:

Si le travail l^* détermine ρ (1, ..., k), alors il existe une séquence optimale qui traite les travaux 1, ..., l^* l'un après l'autre sans interruption par des travaux d'autres chaînes.

Les deux lemmes précédents (Lemmes 2.2 et 2.3) contiennent la base d'un algorithme simple qui minimise le temps d'achèvement total pondéré lorsque les contraintes de priorité prennent la forme de **chaînes avec interruption** . Cet algorithme peut être décrit par la procédure suivante:

- ✓ Lorsque la machine est libérée, sélectionnez parmi les chaînes restantes celle dont le facteur ρ est le plus élevé.
- ✓ Traitez cette chaîne sans interruption jusqu'à l'achèvement du travail qui détermine son facteur ρ.

Remarque:

D'autres variantes de la règle WSPT peuvent être utilisées pour résoudre d'autres types de problèmes tel que **total weighted discounted completion time ($1||\sum W_j(1-e^{-rC_j})$.)** qui donne lieu à une règle de priorité différente, qui programme les travaux dans l'ordre décroissant de $\frac{w_j e^{-rp_j}}{1-e^{-rp_j}}$

Cette règle est appelée Weighted Discounted Shortest Processing Time (WDSPT) rule.

5. The Maximum Lateness (le retard maximum)

5.1 Le problème $1|prec|h_{max}$

Les objectifs examinés dans le reste de ce chapitre sont liés à la date d'échéance (la date due). Le premier modèle relatif à la date d'échéance présente un caractère plutôt général, à savoir le problème $1 \mid prec \mid h_{max}$, où $h_{max} = \max(h_1(C_1),...,h_n(C_n))$ avec h_j, $j = 1, ..., n$, étant des fonctions de coût non décroissantes.

Ce problème nécessite un algorithme de programmation dynamique rétroactif efficace même lorsque les travaux sont soumis à des contraintes de précédence arbitraires. Il est clair que l'achèvement du dernier travail donne lieu au Makespan $C_{max} = \sum p_j$. Cet algorithme peut être décrit comme suit:

Supposons que :

J est l'ensemble des tâches déjà ordonnancées dans l'intervalle: $[C_{max} - \sum_{j \epsilon J} p_j, C_{max}]$.

Le complément de l'ensemble J, appelé J_c est l'ensemble des tâches restantes

J' qui est un sous ensemble de J_c représente l'ensemble des tâches ordonnançables immédiatement après les tâches dans *J* c'est-à-dire l'ensemble des tâches dont tous les successeurs sont dans *J*.

L'algorithme suivant, appelé algorithme LCL (Lowest Cost Last) conduit à un ordonnancement optimal.

- Etape 1 : poser $J=\emptyset$; J^c= toutes les tâches

  et J'=toutes les tâches sans successeurs

- Etape 2 : soit j^* tel que :

$$h_{j^*}(\sum_{k\in j^c} p_k) = min_{j\in J'}\,(h_j\left(\sum_{k\in J^c} p_k\right))$$

  - ✓ ajouter j^* à J
  - ✓ supprimer j^* de J^c
  - ✓ modifier J'

- Etape 3 : répéter l'étape 2 jusqu'à ce que $J^c=\emptyset$

L'exemple suivant illustre l'application de cet algorithme:

Exemple 2.1:

Considérons trois Jobs avec les données présentées dans le tableau suivant:

Jobs	1	2	3
P_j	2	3	5
$h_j(C_j)$	$1+C_1$	$1.2C_2$	10

Le Makespan C_{max} = 10 and $h_3(10) < h_1(10) < h_2(10)$ (car 10 < 11 < 12). Le travail 3 est donc programmé en dernier et doit commencer son traitement au moment 5 (C_{max}-P_3). Pour déterminer quel travail doit être traité avant le travail 3, $h_2(5)$ doit être comparé à $h_1(5)$. Le travail 1 ou le travail 2 peuvent être traités avant le travail 3 dans la séquence optimale car $h_1(5)=h_2(5)=6$. Donc deux séquences sont optimales 1, 2, 3 et 2, 1, 3.

5.2 Le problème $1||\sum L_j$ (Total Lateness)

Rappelez-vous que les retards de travail sont définis par $L_j = C_j - d_j$, où l'écart entre la date d'échéance d'un travail et son délai d'achèvement. Un résultat assez remarquable est que le retard total est atteint par le SPT.

Théorème 2.3:

Le retard total ($\sum L_j$) est minimisé par le séquençage SPT.

Preuve du théorème 2.3:

On pose $L = \sum L_j$ et $F = \sum C_j$

$L = \sum_{j=1}^{n} L_j = \sum_{j=1}^{n} (C_j - d_j)$

$= \sum_{j=1}^{n} C_j - \sum_{j=1}^{n} d_j$

$= F - \sum_{j=1}^{n} d_j$

Le dernier terme est la somme des dates d'échéance données qui est une constante. Puisque, L diffère de F par une constante indépendante de la séquence, la séquence qui minimise L ($\sum L_j$) doit être la séquence qui minimise F ($\sum C_j$). Selon le théorème 2.1, cette séquence est donnée par la règle SPT.

5.3 Le problème $1||L_{max}$ (Maximum Lateness) et $1||T_{max}$ (Maximum Tardiness)

Le problème $1 \mid\mid L_{max}$ est le cas spécial le plus connu du problème $1 \mid prec \mid h_{max}$. La fonction h_j est définie par $C_j - d_j$ et pour le problème $1 \mid\mid L_{max}$, la fonction T_{max} est définie par $\max(L_{max}, 0)$.

La règle Earliest Due Date (EDD) first fournit une solution simple et élégante à ce problème. Dans cette règle, les tâches sont organisées dans l'ordre croissant des dates d'échéance.

Le séquencement des tâches selon la règle (EDD) ne peut toutefois pas garantir que la fonction ($\sum L_j$) sera minimisée, car seul SPT qui peut le garantir. Cependant, nous pouvons montrer que le séquencement EDD minimise le retard maximal dans un problème $1 \mid\mid L_{max}$.

Théorème 2.4:

Le retard maximum (Maximum Lateness) et le retard algébrique maximum (Maximum Tardiness) sont minimisés par la règle EDD (Earliest Due Date).

Preuve du théorème 2.3:

L'optimalité de cette règle peut être prouvée par un simple argument d'échange (voir figure 2.2).

Considérons une séquence S qui n'est pas une séquence EDD. Cherchons dans S une paire de travaux adjacents, i et j, avec j suivant i tel que $d_i > d_j$.

Construisons maintenant une nouvelle séquence, S' dans lequel les travaux i et j sont interchangés et tous les autres travaux sont terminés en même temps que dans S.

Alors:

On dans la séquence S:

$L_i\ (S) = p(B) + p_i - d_i$

$L_j\ (S) = p(B) + p_i + p_j - d_j$

Et dans la séquence S':

$L_j\ (S') = p(B) + p_j - d_j$

$L_i\ (S') = p(B) + p_j + p_i - d_i$

D'où il en résulte que $L_j\ (S) > L_i(S')$ et $L_j\ (S) > L_j(S')$.

Par conséquent: $L_j(S) > \max\{L_i(S'), L_j(S')\}$

On pose: $L_{AB} = \max\{L_k \mid k \in A \text{ or } k \in B\}$, et notez que L_{AB} est le même sous S et S'

Alors,

$L_{max}\ (S) = \max\{L_{AB}\ , L_i\ (S), L_j\ (S)\} \geq \max\{L_{AB}\ , L_i(S'), L_j(S')\} = L_{max}(S')$

En d'autres termes, l'échange des tâches i et j n'augmente pas la valeur de L_{max}, et peut même la réduire (l'améliorer). Par conséquent, une séquence optimale peut être construite comme suit:

1. Commencez par une séquence arbitraire non-EDD.
2. Trouvez une paire de travaux adjacents i et j, avec j suivant i, tels que $d_i > d_j$.
3. Echangez i et j
4. Revenez à l'étape 2 de manière itérative jusqu'à ce qu'une séquence EDD soit construite. A chaque itération, L_{max} reste le même ou il est réduit.

Encore une fois, les liens ne perturbent pas la logique.

Un argument similaire établit que EDD minimise T_{max}, en commençant par l'inégalité suivante:

$T_{max}(S) = \max\{0, L_{max}(S)\} \geq \max\{0, L_{max}(S')\} = T_{max}(S')$.

5.4 La détermination des dates d'échéance

Normalement, nous traitons les dates d'échéance comme des paramètres donnés. Cette approche reflète le principe selon lequel, dans de nombreuses circonstances réalistes, la date d'échéance est déterminée par le client ou lors de la planification.

Cependant, le producteur peut souvent fixer la date d'échéance ou au moins l'influencer. Nous pourrions considérer la date d'échéance comme une question de négociation entre le

producteur et le client. Néanmoins, un modèle raisonnable de la date d'échéance en tant que paramètre négocié introduirait une complexité beaucoup plus grande. Une étape simple dans cette direction consiste à traiter la date d'échéance comme une variable de décision, éventuellement soumise à une contrainte qui représente un proxy pour le processus de négociation.

Supposons que la date d'échéance puisse être sélectionnée à la date de disponibilité du travail j (r_j). La sélection de la date d'échéance représente un objectif pour la tolérance de flux ou le temps que le travail passera dans le système. Nous pourrions sélectionner les dates d'échéance selon l'un des règles suivantes:

- CON : Constant flow allowance: $d_j = r_j + \gamma$.
- SLK : Equal slack flow allowance: $d_j = r_j + p_j + \beta$.
- TWK : Total work flow allowance: $d_j = r_j + \alpha p_j$.

où chaque règle contient un seul paramètre (γ, β ou α) à spécifier.

Lorsque les dates d'échéance sont complètement discrétionnaires, il n'est pas difficile de minimiser les retards totaux: pour tout programme, nous pourrions sélectionner des dates d'échéance suffisamment lâches pour qu'aucun travail ne soit pas en retard. Cependant, dans un environnement où les échéances peuvent être sélectionnées, il semble raisonnable de rechercher les échéances les plus serrées possibles. Les dates d'échéance serrées correspondent aux indemnités de flux à court terme et représentent donc un engagement aux clients à ce que les commandes soient traitées rapidement. Par conséquent, nous imposons la contrainte qu'aucun travail ne doit être retardé et nous examinons comment définir les dates d'échéance afin qu'elles soient aussi serrées que possible.

Pour mesurer la tension d'un ensemble de dates d'échéance, nous utilisons la somme des dates d'échéance.

$$D = \sum_{j=1}^{n} d_j$$

Le problème devient de minimiser D, sous réserve que $Cj \leq dj$. En principe, nous pouvons facilement trouver une solution optimale à ce problème. Pour tout échéancier, les dates d'échéance les plus serrées possibles sont évidemment données par $d_j = C_j$. Par conséquent, D peut être minimisé en minimisant la somme des temps d'achèvement ou le temps d'écoulement total.

Notre solution peut être trouvée en construisant un calendrier SPT des travaux, en calculant le temps d'achèvement de chaque travail dans ce calendrier et en fixant la date d'échéance de chaque travail à son temps d'achèvement. Cette solution optimale nécessite que la date d'échéance de

chaque travail dépende d'informations spécifiques sur tous les autres travaux de la planification, que nous appelons une base d'informations complète.

Une approche plus pratique consiste à s'appuyer sur des règles telles que CON, SLK et TWK, dans lesquelles la sélection d'une date d'échéance dépend uniquement des informations sur le travail lui-même (sa date de disponibilité et son temps de traitement) et d'un paramètre de précision.

En pratique, une bonne approche consiste à utiliser TWK et à ajuster α par essais et erreurs pour maintenir la performance de la date d'échéance de l'atelier sur la cible.

6. The Number of Tardy Jobs (Nombre de jobs en retard)

Un autre objectif lié à la date d'échéance est $\sum U_j$. Cet objectif peut à première vue sembler quelque peu artificiel et sans intérêt pratique. Cependant, dans le monde réel, il s'agit d'une mesure de la performance souvent contrôlée et en fonction de laquelle les gestionnaires sont mesurés. Cela équivaut au pourcentage d'envois dans les délais.

Un horaire optimal pour $1||\sum U_j$ prend la forme d'un ensemble de travaux qui respectent leurs dates d'échéance et qui sont planifiés en premier, suivis de l'ensemble des travaux restants qui ne respectent pas leurs dates d'échéance et qui sont planifiés en dernier.

Il résulte des résultats de la section $1||L_{max}$ que le premier ensemble de travaux doit être planifié selon EDD afin de s'assurer que L_{max} est négatif; l'ordre dans lequel le deuxième ensemble de travaux est planifié est sans importance.

Le problème $1 \,||\, \sum U_j$ peut être résolu facilement en utilisant un algorithme de type forward. Réorganisez les travaux de telle sorte que $d_1 \leq d_2 \leq \cdot \leq d_n$ (Selon la règle EDD). L'algorithme parcourt n itérations. Dans l'itération k de l'algorithme les travaux 1, 2,. . . , k sont prises en considération.

De ces k jobs, le sous-ensemble J fait référence aux travaux qui, selon un calendrier optimal, peuvent être terminés avant leur date d'échéance et le sous-ensemble J^d fait référence aux travaux qui ont déjà été supprimés et ne respecteront pas leurs dates d'échéance dans le programme optimal. Dans l'itération k, l'ensemble J^c fait référence aux travaux $k + 1$, $k + 2$,. . . , n.

L'algorithme peut être décrit comme suit:

J est l'ensemble des tâches s déjà ordonnancées

J^c est l'ensemble des tâches s restantes

J^d est l'ensemble des tâches considérées pour l'ordonnancement mais post dues

- Etape 1 : poser $J = J^d = \emptyset$; J^c = toutes les tâches
- Etape 2 : soit j^* tel que : $d_{j^*} = \min_{j \in J^c}(d_j)$
  - ajouter j^* à J
  - supprimer j^* de J^c
- Etape 3 : si $\sum_{j \in J} p_j \leq d_{j^*}$
  - procéder à l'étape 4
  - Si non : soit k^* appartenant à J tel que $p_{k^*} = \max(p_j)$
    - Supprimer k^* de J
    - Ajouter k^* à J^d
- Etape 4 : répéter l'étape 2 et 3 jusqu'à ce que $J^c = \emptyset$

Les jobs sont ajoutés à l'ensemble des travaux ponctuels dans l'ordre croissant de leurs dates d'échéance. Si l'inclusion du travail k dans l'ensemble des travaux planifiés implique que le travail k serait terminé en retard, le travail planifié ayant le temps de traitement le plus long (k^*) est marqué comme étant en retard et supprimé.

Exemple: Considérez les 5 travaux suivants

Tâches s	1	2	3	4	5
p_j	7	8	4	6	6
d_j	9	17	18	19	21

Les travaux 1 et 2 peuvent être placés en premier et en second dans la séquence, les deux travaux étant terminés à temps. Mettre le travail 3 en troisième position pose des problèmes. Son délai de réalisation serait 19 et sa date d'échéance est 18.

L'algorithme prescrit la suppression du travail avec le temps de traitement le plus long parmi les trois premiers travaux. Le travail 2 est donc supprimé et les travaux 1 et 3 restent dans les deux premiers postes.

Si le travail 4 suit le travail 3, il est terminé à l'instant 17; cependant, si le travail 5 suit le travail 4, il est terminé en retard. L'algorithme recommande ensuite de supprimer le travail dont le temps de traitement est le plus long parmi ceux déjà planifiés, à savoir le travail 1.

Donc, le calendrier optimal est 3, 4, 5, 1, 2 avec $\sum U_j = 2$.

Exercices complémentaires

Exercice 1

Soit le problème $1||\sum C_j$ avec les données présentées dans le tableau suivant:

Tâches s	1	2	3	4	5	6	7
P_j	4	5	8	6	7	9	6

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 2

Soit le problème $1||\sum w_j C_j$, Avec les données suivantes

Tâches s	1	2	3	4	5	6	7
W_j	18	0	8	8	12	16	17
P_j	6	3	5	4	6	9	8

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 3

Soit le problème $1 \mid chains \mid \sum w_j C_j$, Avec les données suivantes :

Tâches s	1	2	3	4	5	6	7
W_j	18	0	8	8	12	16	17
P_j	6	3	5	4	6	9	8

Les tâches sont groupées dans les chaînes à exécution sans interruption suivantes:

- ✓ 1→2
- ✓ 5→3→4

- ✓ 6→7

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 4

Soit le problème $1|chains|\sum w_j C_j$, Avec les données suivantes :

Tâches	1	2	3	4	5	6	7
W_j	6	18	12	8	8	17	18
P_j	3	6	6	5	4	8	10

Les tâches sont groupées dans les chaînes à exécution avec interruption suivantes:

- ✓ 1→2 →3→4
- ✓ 5→6→7

3. Trouvez une séquence optimale.
4. Trouvez la fonction objectif.

Exercice 5

Soit le problème $1||h_{max}$, Avec les données mentionnées dans le tableau suivant.

Jobs	1	2	3	4	5	6	7
P_j	4	8	12	7	6	9	9
$h_j(C_j)$	$3C_1$	77	C_3^2	$1{,}5C_4$	$70+\sqrt{C_5}$	$1{,}6C_6$	$1{,}4C_7$

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 6

Soit le problème $1|\ prec|h_{max}$, avec les contraintes de précédence suivantes :

5→4

1→7→6

5→7

et les données suivantes :

Jobs	1	2	3	4	5	6	7
P_j	4	8	12	7	6	9	9
$h_j(C_j)$	$3C_1$	77	C_3^2	$1{,}5C_4$	$70+\sqrt{C_5}$	$1{,}6C_6$	$1{,}4C_7$

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 7

Soit le problème $1||L_{max}$, Avec les données suivantes :

Jobs	1	2	3	4	5
P_j	1	2	3	4	5
d_j	9	13	11	15	10

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 8

Soit le problème $1||T_{max}$, Avec les données suivantes :

Jobs	1	2	3	4	5	6	7
P_j	6	18	12	10	10	17	16
d_j	8	42	44	24	80	75	60

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 8

Soit le problème $1||\sum U_j$, Avec les données suivantes :

Jobs	1	2	3	4	5	6	7
P_j	6	18	12	10	10	17	16
d_j	8	42	44	24	80	75	60

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Chapitre 3
Ordonnancement de problèmes à machines en parallèles

1. Introduction :

En général, l'ordonnancement nécessite des décisions de séquencement et d'allocation de ressources. Lorsqu'il n'y a qu'une seule ressource, l'attribution de cette ressource est entièrement déterminée par les décisions de séquencement. En conséquence, dans le modèle mono machine, il n'y a pas de distinction entre le séquençement et l'allocation de ressources. Pour apprécier cette distinction, nous devons examiner les modèles comportant plusieurs machines. La théorie d'ordonnancement couvre plusieurs types de base de modèles multi machines. Parmi ces modèles on peut citer celui avec machines parallèles. Dans les systèmes parallèles, les travaux consistent en une opération, comme dans le modèle mono machine; mais elle peut être traitée sur l'une des différentes machines.

Un système de machines en parallèle est un paramètre important à la fois d'un point de vue théorique et pratique. D'un point de vue théorique, il s'agit d'une généralisation de la machine unique et d'un cas particulier de l'atelier de type flow shop flexible. D'un point de vue pratique, il est important car la présence de ressources en parallèle est courante dans le monde réel. De plus, les techniques pour les machines en parallèle sont souvent utilisées dans les procédures de décomposition pour les systèmes à plusieurs étages.

Un paramètre simple dans lequel nous pouvons étudier les effets du parallélisme est le problème de d'ordonnancement de travaux à une seule opération en présence de plusieurs machines parallèles. Comme dans le modèle de base, *n* tâches sont disponibles simultanément à l'heure zéro. Nous avons également *m* machines parallèles disponibles pour le traitement, et nous supposons qu'un travail peut être traité par, au plus, une machine à la fois. Dans le modèle de machine

parallèle de base, les machines sont identiques et les tâches ne sont pas liées. Lorsque nous abordons les mesures de performance fondamentales dans ce contexte, les solutions reflètent le parallélisme des ressources.

Dans ce chapitre, plusieurs objectifs sont considérés. Les trois objectifs principaux sont la minimisation de la durée de fabrication (Makespan), le temps d'exécution total (total completion time) et le retard maximal (maximum lateness). Avec une seule machine, l'objectif Makespan est généralement intéressant uniquement lorsqu'il existe des temps d'installation dépendants de la séquence; sinon, le Makespan est égal à la somme des temps de traitement et est indépendant de la séquence. Lorsqu'il s'agit de machines en parallèle, le Makespan devient un objectif d'un intérêt considérable. En pratique, il faut souvent régler le problème de l'équilibre de la charge sur des machines; en minimisant le Makespan.

On peut en fait considérer l'ordonnancement des machines parallèles comme un processus en deux étapes. Premièrement, il faut déterminer quels travaux doivent être affectés à quelles machines; deuxièmement, il faut déterminer la séquence des tâches allouées à chaque machine. Avec l'objectif Makespan, seul le processus d'attribution est important.

Avec les machines parallèles, les préemptions jouent un rôle plus important qu'avec une seule machine. Avec une seule machine, les préemptions ne jouent généralement un rôle que lorsque les travaux sont disponibles à des moments différents. En revanche, avec les machines en parallèle, les préemptions sont importantes même lorsque tous les travaux sont libérés au moment zéro.

Les prochaines sections consistent à décrire les différentes approches de résolution des problèmes d'ordonnancement avec machines parallèles. La date de fin (le Makespan), le temps d'achèvement total (The total completion time) et certains objectifs liés à la date d'échéance, sont considérés dans ce chapitre.

2. La minimisation du Makespan (machines identiques, sans préemptions) :

Dans le modèle à une seule machine de base, le Makespan est égal à une constante pour toute séquence de *n* tâches données. Le problème du Makespan ne nécessite donc aucune analyse. Dans le modèle de machine parallèle statique, la séquence des travaux sur une machine particulière est indifférente. Ainsi, le problème de Makespan est uniquement un problème d'attribution de tâches à des machines. Cependant, le problème de Makespan reste très difficile.

Le premier problème considéré est $P_m||\ C_{max}$ où un ensemble de tâches indépendantes doit être planifié sur des processeurs identiques afin de minimiser la date de fin d'ordonnancement. Dans ce problème, nous interdisons la préemption des tâches.

Nous commençons par une analyse de la complexité de ce problème, qui conduit à la conclusion que le problème n'est pas facile à résoudre, car même des cas simples, tels que l'ordonnancement sur deux machines ($P_2||C_{max}$) , peuvent s'avérer être NP-difficiles. Par conséquent, la détermination de calendriers optimaux pour le Makespan nécessite des approches approximatives, heuristiques ... Une de ces heuristiques est décrite ci-dessous.

La règle LPT (Longest Processing Time First) qui consiste à organiser les tâches par ordre décroissant selon leurs temps du traitement. Dans un contexte P_m, cette règle attribue à $t = 0$ les m tâches les plus longues aux m machines. Ensuite, chaque fois qu'une machine est libérée, le travail le plus long parmi ceux qui n'ont pas encore été traités est placé sur la machine. Cette heuristique essaie de placer les travaux les plus courts vers la fin du planning où ils peuvent être utilisés pour équilibrer les charges.

Dans le prochain théorème, une limite supérieure est présentée pour $C_{max}(LPT)/C_{max}(OPT)$ où $C_{max}(LPT)$ indique le Makespan de la séquence LPT et $C_{max}(OPT)$ est le Makespan de la séquence optimale (éventuellement inconnue).

Théorème 3.1:

Dans le problème d'ordonnancement dans un atelier de type machines parallèles identiques avec des tâches non préemptives, la règle LPT donne un Makespan satisfaisant :

$$\frac{C_{\max}(LPT)}{C_{\max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$

Exemple :

Considérez les 7 travaux suivants dans un problème $P_4||\ C_{max}$

Tâches	1	2	3	4	5	6	7
p_j	3	3	3	1	1	1	4

Dans ce problème le Makespan ne peut pas être inferieur à $\sum C_j/4 = 4$.

Si on utilise la règle LPT, on affecte à l'instant 0, la tâche 7 à la machine 1 et les tâches 1, 2, 3 aux machines 2, 3, 4. Puis à l'instant 3, les machines 2, 3 et 4 deviennent libre, donc ces machines traitent les tâches4, 5, 6. Par conséquent, on trouve $C_{max}(LPT)=4$.

Donc, selon le théorème 3.1 on obtient

$$\frac{C_{\max}(LPT)}{C_{\max}(OPT)} = \frac{4}{4} \leq \frac{4}{3} - \frac{1}{3m} = \frac{15}{12}$$

3. Le problème $P_m \mid prec \mid C_{max}$

Passons maintenant au cas des tâches dépendantes et supposées être ordonnancées de manière non préemptive. Lorsque nous ajoutons des relations de précédence au problème Makespan avec des machines parallèles, nous ne facilitons pas le problème. D'un point de vue complexité, ce problème doit être au moins aussi dur que le problème sans contraintes de précédence. Il n'est donc pas surprenant que nous obtenions des résultats uniquement dans des cas particuliers.

Pour obtenir des informations sur les effets des contraintes de priorité, un certain nombre de cas spéciaux doivent être pris en compte. Le cas particulier d'une seule machine est clairement trivial. Il suffit de garder la machine occupée en permanence et la durée de traitement sera égale à la somme des temps de traitement.

Prenons le cas particulier où il y a un nombre illimité de machines en parallèle ou le nombre de machines est au moins aussi grand que le nombre de travaux, c.-à-d. $M \geq n$. Ce problème peut être noté $P_\infty \mid prec \mid C_{max}$. C'est un problème classique dans le domaine de la planification de projet et son étude a conduit à la mise au point de la méthode Critical Path Method (CPM) and Project Evaluation and Review Technique (PERT).

Le calendrier optimal et le Makespan minimum sont déterminés par un algorithme très simple. Cet algorithme consiste à ordonnancer les travaux un par un en commençant à zéro. Lorsqu'un travail est terminé, démarrez tous les travaux pour lesquels tous les prédécesseurs sont terminés c'est-à-dire tous les travaux programmables (ceux qui peuvent être ordonnancés). Il s'avère que dans ce problème, le début du traitement de certains travaux peut généralement être reporté sans augmenter le Makespan. Ces travaux sont appelés les travaux libres (Slak jobs). Les travaux qui ne peuvent pas être reportés sont appelés travaux critiques (Critical jobs). L'ensemble des travaux critiques est appelé chemin (s) critique (s) (critical path(s)).

Afin de déterminer les travaux critiques, commencez par le Makespan, qui est maintenant connu, et avancez vers le temps zéro, tout en respectant les relations de précédences. Ainsi, tous les travaux sont terminés aux derniers instants possibles et, par conséquent, démarrés à ces derniers temps. Les travaux dont les dates de début au plus tôt sont égales à leurs dates de départ au plus tard sont les travaux critiques.

Cet algorithme peut être résumé comme suit:

- ✓ CPM Forward :
  - ordonnancer les tâches (activités) sans prédécesseurs en commençant à t=0.
  - chaque fois qu'une tâche (activité) est terminée, ordonnancer toutes les tâches (activités) dont tous les prédécesseurs sont déjà ordonnancés.
- ✓ CPM backward :
  - Commencer à $t=C_{max}$ et déterminer les dates de fin et de début de toutes les tâches
- ✓ Déduire le chemin critique (critical path) et les temps libres (slack)

Contrairement à $1|prec|C_{max}$ et $P_\infty|prec|\ C_{max}$, le $P_m|prec|C_{max}$ est fortement NP-difficile lorsque $2 \leq m < n$. Même le cas particulier avec tous les temps de traitement égaux à 1, à savoir, $P_m|p_j = 1, prec\ |C_{max}$, n'est pas facile. Cependant, supposer que le graphe de précédences se présente sous la forme d'un arbre (un arbre d'assemblage ou un arbre extérieur (outtree)) crée un problème, c'est-à-dire $P_m|p_j = 1, tree|C_{max}$, qui est facilement soluble.

Dans un arbre d'assemblage (assembly tree ou intree), aucun travail n'a plus d'un successeur direct. De plus, dans un tel arbre, le travail final (le travail sans successeurs) qui est appelé un travail de terminal ou la racine (the root), est situé au niveau 1. Les travaux qui précèdent immédiatement la racine sont au niveau 2; les travaux qui précèdent immédiatement les travaux de niveau 2 sont au niveau 3, et ainsi de suite.

Dans un arbre exterieur (outtree), tous les travaux sans successeurs se situent au niveau 1. Les travaux qui ont uniquement des travaux de niveau 1 comme successeurs immédiats sont au niveau 2; les travaux qui ont uniquement des travaux du niveau 2 comme successeurs immédiats étant au niveau 3,... (Voir figure 3.1).

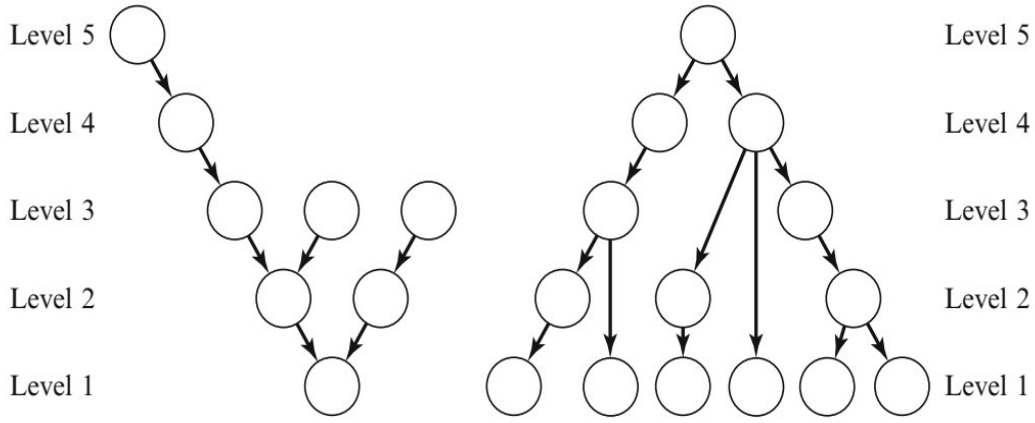


Figure 3.1: Un arbre d'assemblage (intree) et un arbre extérieur (outtree)

De plus, posez p_j = 1 pour tous les travaux, afin que nous ayons des tâches de longueur unitaire. Pour ce cas particulier, nous pouvons résoudre le problème de Makespan avec un algorithme consistant en une phase d'étiquetage suivie d'une phase d'ordonnancement.

Ce problème particulier conduit à une règle d'ordonnancemnt bien connue, la règle de chemin critique ou Critical Path (CP) qui attribue la priorité la plus élevée au travail situé en tête de la plus longue chaîne de travaux du graphe de précédences. La règle CP est équivalente au plus haut niveau en premier (Highest Level first rule).

Théorème 3.2:

La règle Critical Path (CP) est optimale pour le problème $P_m| p_j = 1, intree |C_{max}$ et le problème $P_m|p_j =1, outree|C_{max}$.

Un algorithme qui est utilisé pour optimiser le Makespan dans un problème $P_m|p_j=1,intree|C_{max}$ est appelé **Algorithme de Hu** et peut être décrit comme suit:

Phase d'étiquetage:

Calculez les niveaux des tâches selon les étapes suivantes:

- ✓ étape1: Affecter le zéro à la tâche du terminal.
- ✓ étape 2: Supposons que les étiquettes 1, 2,. . . , j - 1 ont été attribuées, Attribuez le libellé j à tous les travaux dont les successeurs sont étiquetés.
- ✓ étape 3: Répétez l'étape 2 jusqu'à ce que les étiquettes aient été attribuées à tous les travaux.

Phase d'ordonnancement:

La phase d'ordonnancement est essentiellement une procédure de tri des tâches dans un ordre d'étiquette non croissant dans la mesure où les contraintes de précédence le permettent. Cette phase est présentée comme suit:

t=0

Répéter

Construisez la liste L composée de toutes les tâches sans prédécesseurs à l'instant t;

- toutes ces tâches n'ont pas de prédécesseurs
- ou leurs prédécesseurs ont déjà été affectés

Organisez L dans l'ordre décroissant selon les niveaux de tâches;

Attribuez les m premières tâches (le cas échéant) à des processeurs à l'instant t;

Supprimez les tâches assignées du graphique et de la liste.

t=t+1;

Jusqu'à ce que toutes les tâches ont été programmées

Dans cet algortihme, la phase d'étiquetage attribue à chaque travail *j* une étiquette égale au temps requis pour traiter les travaux qui suivent le travail *j* sur le chemin (unique) reliant le travail *j* et le travail de terminal. Ensuite, lorsque la phase d'ordonnancement place les tâches avec les étiquettes les plus grandes dans le planning, elle donne essentiellement la priorité aux tâches qui initient les plus longs chemins dans l'arborescence restante.

En plus, le chemin le plus long est souvent appelé chemin critique. Un exemple d'application de cette procedure peut être présenté par la figure 3.2:

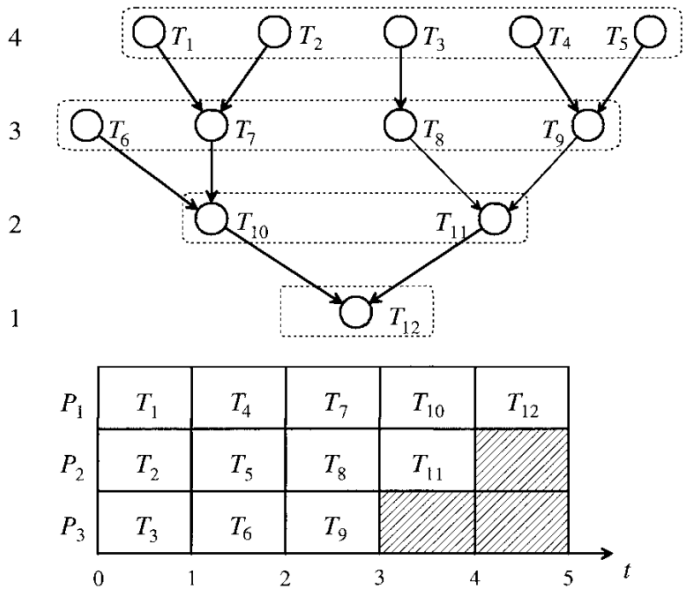


Figure 3.2: Un exemple d'application d'algorithme de Hu

L'algorithme précédent fournit une solution optimale lorsque le problème contient des travaux de longueur unitaire et une structure arborescente. Bien qu'une arborescence ne comporte qu'une tâche de terminal, nous pouvons appliquer l'algorithme à l'ordonnancement de plusieurs arborescences en créant une tâche de terminal fictive qui remplacera les tâches de terminal de chacune des arborescences. Mais, si nous affectons l'étiquette zéro au travail fictif, chaque étiquette représente le travail restant sur le chemin direct depuis le nœud jusqu'à son achèvement (y compris le nœud lui-même).

4. La minimisation du Makespan (machines identiques, avec préemptions) :

Maintenant, on peut essayer une autre façon d'analyser le problème $P_m||C_{max}$ c'est-à-dire que l'on peut relaxer certaines contraintes imposées au problème $P_m||C_{max}$ et permettre des préemptions de tâches. Il semble que le problème $P_m|prmp|C_{max}$ puisse être résolu très efficacement.

Un problème de Makespan plus simple se pose lorsque les travaux ne sont pas liés et que nous autorisons la préemption. Si la préemption est autorisée, le traitement d'une tâche peut être interrompu et le traitement restant peut être effectué ultérieurement, éventuellement sur une autre machine.

La formule pour le Makespan minimum est donnée par le modèle de programmation linéaire suivant:

Minimisez C_{max}

sachant que :

$$\sum_{i=1}^{m} x_{ij} = p_j \qquad j=1,..,n$$

$$\sum_{i=1}^{m} x_{ij} \leq C_{max}, \qquad j=1,..,n$$

$$\sum_{j=1}^{n} x_{ij} \leq C_{max}, \qquad i=1,..,m$$

$$x_{ij} \geq 0, \qquad i=1,..,m \text{ et } j=1,..,n$$

La variable x_{ij} représente le temps total que le travail j passe sur la machine i. Le premier ensemble de contraintes garantit que chaque travail reçoit la quantité de traitement requise. Le second ensemble de contraintes garantit que le nombre total de traitements reçus par chaque travail est inférieur ou égal à la valeur de Makespan. Le troisième ensemble veille à ce que la quantité totale de traitements sur chaque machine soit inférieure à la durée de fabrication.

Ce modèle linéaire peut être résolu en un temps polynomial, mais sa solution ne prescrit pas de programme réel; il spécifie simplement le temps que le travail j doit passer sur la machine i. Cependant, avec cette information, un calendrier peut facilement être construit.

Il existe plusieurs autres algorithmes pour $P_m \mid prmp \mid C_{max}$. L'un de ces algorithmes est basé sur le fait qu'il est facile d'obtenir une expression pour le Makespan dans le programme optimal. Pour ce problème une limite inférieure pour le Makespan est établie, si on pose que le travail 1 est le travail avec le temps de traitement le plus long.

$$C_{max} \geq \max\left(p_1, \sum_{j=1}^{n} {p_j}/{m}\right) = C^*_{max}$$

Dans ces conditions, l'algorithme suivant peut minimiser le Makespan avec préemptions:

- ✓ étape 1: Classer les *n* jobs sur une seule machine (n'importe quel ordre)
- ✓ étape 2: Couper cet ordonnancement en *m* parties de même taille
- ✓ Étape3 : Prendre comme ordonnancement de la machine 1 le premier intervalle; prendre comme calendrier pour la machine 2 la séquence de traitement du second intervalle, et ainsi de suite.

Il est évident que le calendrier qui en résulte est réalisable. Une partie d'un travail peut apparaître à la fin de la planification pour la machine *i*, tandis que la partie restante peut apparaître au début de la séquence pour la machine *i* + 1. Comme les préemptions sont autorisées et que le temps de traitement de chaque travail est inférieur à C^*_{max}, un tel calendrier est réalisable.

Ce problème n'a pas de solution unique et la méthode de construction de l'algorithme ne produit qu'une des nombreuses séquences optimales. En particulier, la méthode ne tente pas de minimiser le nombre de préemptions. Donc, il ne serait pas difficile de construire un calendrier qui réduit le Makespan sans préemptions. En général, toutefois, réduire le nombre de préemptions dans un calendrier optimal est un problème difficile.

Cependant, il reste la question de l'applicabilité pratique de la solution obtenue de cette manière. Il faut se demander si ce modèle d'ordonnancement de tâches préemptives peut être justifié, car on ne peut s'attendre à ce que les préemptions soient gratuites. En règle générale, deux types de coûts de préemption doivent être pris en compte: le temps et les finances.

Les retards dus aux préemptions sont moins cruciaux si le retard causé par une seule préemption est faible par rapport au temps que la tâche passe en permanence au processeur. Les coûts financiers liés aux préemptions, en revanche, réduisent le bénéfice total obtenu par l'exécution préemptive de tâches; mais encore une fois, si le profit réalisé est important par rapport aux pertes causées par les préemptions, le calendrier sera plus utile et plus acceptable.

Ces circonstances suggèrent l'introduction d'un modèle d'ordonnancement dans lequel les préemptions de tâches ne sont autorisées que lorsque les tâches ont été traitées de manière continue pendant un laps de temps donné *k*. La valeur de *k* (granularité de préemption) doit être choisie suffisamment grande pour que le délai et les frais généraux liés aux préemptions soient négligeables.

Si on adapte cette procédure, on note que pour $k = 0$ ce problème se réduit en un problème de planification préventive "classique". D'autre part, si pour un exemple donné, la granularité *k* est supérieure au temps de traitement le plus long parmi les tâches données, aucune préemption n'est autorisée et nous aboutissons à un ordonnancement non préemptive.

Une autre procedure qui peut sembler utile pour le problème $P_m \mid prmp \mid C_{max}$ est la règle **Longest Remaining Processing Time first (LRPT)**. Cette règle est la contrepartie préemptive de la règle LPT (non préemptive). C'est un programme est d'intérêt académique. Mais, d'un point de vue pratique, il présente toujours un sérieux inconvénient. Le nombre de préemptions nécessaires dans le cas déterministe est généralement infini.

Théorème 3.3:

La règle LRPT donne un calendrier optimal pour le problème $P_m \mid prmp \mid C_{max}$ en temps discret.

Exemple : Application de la rège LRPT en temps discret.

Considérons deux machines et trois travaux 1, 2 et 3 avec les temps de traitement 8, 7 et 6. Le planning sous LRPT est illustré à la figure 3.3 et l'espace est égal à 11.

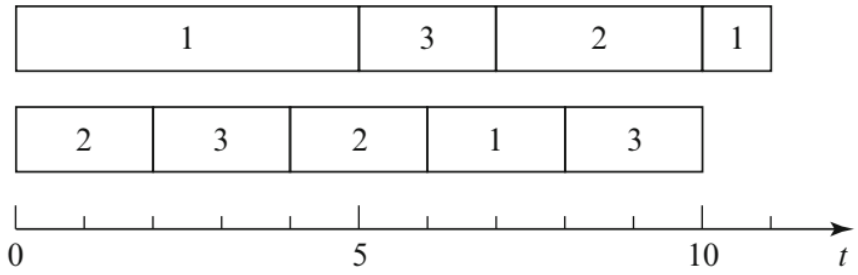


Figure 3.3: Application de la règle LRPT avec préemptions à des moments entiers

LRPT est également optimale en temps continu. L'exemple suivant présente un cas d'application de cette règle pour un problème $P_m \mid prmp \mid C_{max}$ en temps continu.

Exemple : Application de la règle LRPT en temps continu.

Considérez les mêmes travaux que dans l'exemple précédent. Comme les préemptions peuvent être faites à tout moment, le partage du processeur a lieu (Voir figure 3.4). On obtient un Makespan de 10.5.

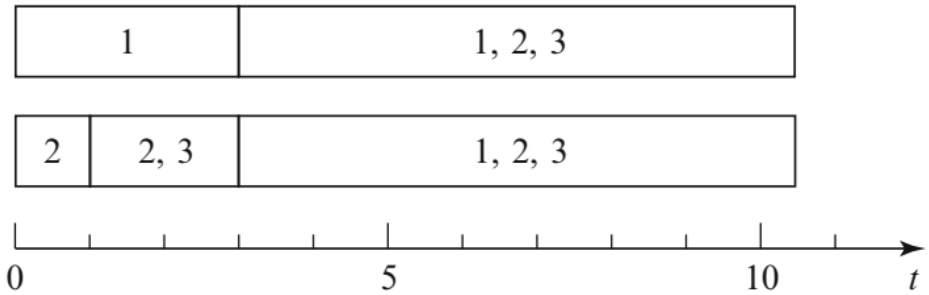


Figure 3.4: Application de la règle LRPT avec préemptions autorisées à tout moment

Nous pouvons appliquer certains des résultats de cette section aux problèmes contenant des travaux préemptifs avec contraintes de précédences. La clé est de considérer chaque travail comme une chaîne de travaux de longueurs unitaires. La figure 3.5a montre un exemple contenant un ensemble de tâches associées ayant des temps de traitement différents, à planifier sur deux machines. La figure 3.5b représente le même ensemble de tâches intégrant la structure de la chaîne. Chaque travail correspond à un noeud de la figure et, à côté de chaque noeud, se trouve le temps de traitement du travail. Plus précisément, les travaux 5, 6 et 7 sont représentés par des chaînes, et dans ce cas, les chaînes ont une longueur de deux.

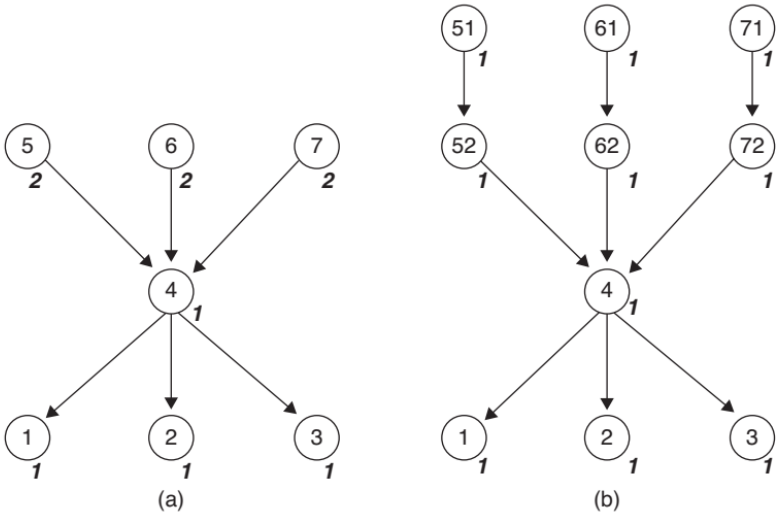


Figure 3.5: Un exemple de sept travaux dans (a) et leur représentation préemptive dans (b)

S'il n'y avait pas de préemptions, alors nous pourrions construire un calendrier à partir de la figure 3.5a. Dans ce cas, il ne serait pas difficile de voir qu'un Makespan optimal sur deux machines a une longueur de $C_{max} = 7$.

On peut également construire un calendrier à partir de la figure 3.5b. L'ordonnancement résultant est illustré dans la figure 3.6, avec une valeur du Makespane de $C_{max} = 6$.

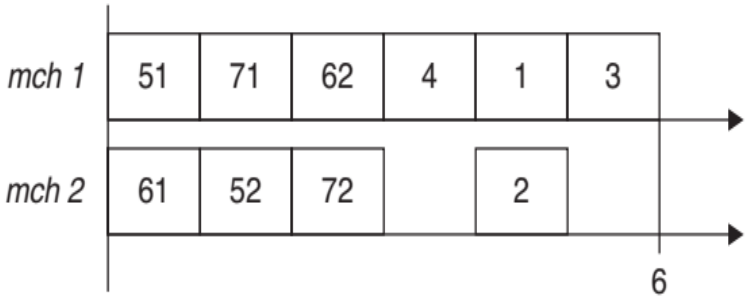


Figure 3.6: Le calendrier des travaux de la figure 3.5b

5. Le problème $Q_m \mid p_j=1 \mid C_{max}$

Commençons par une analyse des tâches indépendantes et de la planification non préemptive dans des machines parallèles à vitesse differentes. Puisque le problème des temps de traitement arbitraires est déjà NP-difficile pour des processeurs identiques, tout ce que nous pouvons trouver est un algorithme d'optimisation de temps polynomial pour les tâches avec des temps de traitement standard d'unité uniquement.

Une telle approche a été donné par Graham et al. où une formulation de réseau de transport a été présentée pour le problème $Q_m \mid p_j=1 \mid C_{max}$. et peut être décrite comme suit:

Soit n sources j, $j = 1,\ldots, n$ et $m*n$ sinks (i, k), $i = 1, 2, \ldots, m$ et $k=1, \ldots, n$. Les sources correspondent aux tâches et les puits (Sinks) aux processeurs et aux positions des tâches sur celles-ci.

Soit C_{ijk} représente le coût de l'arc $(j, (i, k))$; cette valeur correspond au temps d'achèvement de la tâche j traitée sur la machine i dans la position k et depend également de la vitesse de la machine i.

Le flux d'arc X_{ijk} a l'interprétation suivante:

$$X_{ijk} = \begin{cases} 1 & si\ j\ est\ traitée\ dans\ la\ position\ k\ de\ la\ séquence\ de\ la\ machine\ i \\ & autrement \end{cases}$$

Le problème du transport min-max peut maintenant être formulé comme suit:

Minimisez Max $\{C_{ijk}*X_{ijk}\}$

sachant que :

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ijk} = 1 \qquad j=1,\ldots, n$$

$$\sum_{j=1}^{n} x_{ijk} \leq 1, \qquad i=1,\ldots, m \text{ et } k=1,\ldots, n$$

$$x_{ijk} \geq 0, \qquad i=1,\ldots, m, j=1,\ldots, n \text{ et } k=1,\ldots, n$$

6. Le problème $Q_m \mid prmp \mid C_{max}$

Considérons le problème $Q_m \mid prmp \mid C_{max}$, c'est-à-dire m machines en parallèle avec la machine i ayant la vitesse v_i. Supposons que $v_1 \geq v_2 \geq \ldots \geq v_m$ and $P_1 \geq P_2 \geq \ldots \geq P_n$. une limite inférieure peut également être établie pour le Makespan.

$$C_{max} \geq \max\left(\frac{P_1}{v_1}, \frac{P_1+P_2}{v_1+v_2}, \ldots, \frac{\sum_{j=1}^{m-1} P_i}{\sum_{i=1}^{m-1} v_i}, \frac{\sum_{j=1}^{n} P_i}{\sum_{i=1}^{m} v_i}\right)$$

Dans cette inégalité, le Makespan doit être au moins aussi long que le temps nécessaire à la machine la plus rapide pour effectuer le travail le plus long. Il doit également être au moins aussi long que le temps nécessaire pour que la machine la plus rapide et la deuxième plus rapide traite la tâche la plus longue et la deuxième plus longue tout en maintenant les deux machines occupées exactement en même temps. Le reste des premiers m-1 termes sont déterminés de la même manière. Le dernier terme est un peu différent car il s'agit du temps minimum requis pour traiter les n tâches sur les m machines tout en maintenant celles-ci occupées exactement le même temps.

Théorème 3.4:

La règle Longest Remaining Processing Time on the Fastest Machine first (LRPT-FM) est optimale pour le problème $Q_m \mid prmp \mid C_{max}$.

Cette règle est une généralisation de la règle LRPT. Elle affecte, à tout moment, le travail avec le temps de traitement restant le plus long à la machine la plus rapide, le travail avec le deuxième temps de traitement restant le plus long sur la deuxième machine la plus rapide,... Cette règle nécessite généralement un nombre infini de préemptions. Si, au moment t, deux tâches ont le même temps de traitement restant et que ce temps de traitement est le plus long parmi les travaux qui ne sont pas encore terminés au moment t, alors l'un des deux travaux doit passer sur la machine la plus rapide, tandis que l'autre doit continuer sur la deuxième machine plus rapide.

A l'instant $t + \epsilon$, ϵ supposée être très petit, le temps de traitement restant du travail sur la deuxième machine la plus rapide est plus long que le temps de traitement restant du travail sur la machine la plus rapide. Donc, le travail sur la deuxième machine la plus rapide doit passer à la plus rapide et vice versa. Ainsi, la règle LRPT-FM se traduit souvent par un partage du processeur. Un certain nombre de machines, disons m^*, traitent un certain nombre de tâches, disons n^*, $n^* \geq m^*$; les machines répartissent leur capacité totale de traitement de manière uniforme sur les n^* tâches et veillent à ce que les temps de traitement restants des n^* tâches restent égaux à tout moment.

7. The Total Completion Time without Preemptions ($P_m||\sum C_j$)

Considérons m machines en parallèle et n travaux. Rappelons que $p_1 \geq \cdot \geq p_n$. L'objectif à minimiser est le temps total d'achèvement non pondéré ($\sum C_j$). Bien que le problème de Makespan est essentiellement un problème dans l'attribution optimale des travaux aux machines, la minimisation de $\sum C_j$ nécessite la reconnaissance de la séquence ainsi que les décisions d'allocation.

La généralisation à des machines parallèles en utilisant des propriétés de séquencement optimal pour le modèle de base à une seule machine est relativement simple. Lors de l'analyse de la nature de ce critère, on pourrait s'attendre à ce que, comme dans le cas d'un seul processeur, en

affectant des tâches dans un ordre non décroissant de leurs temps de traitement, le temps moyen d'écoulement soit minimisé. En fait, une bonne généralisation de cette règle simple conduit à un algorithme d'optimisation $P_m||\sum C_j$.

Théorème 3.5:

La règle SPT est optimale pour le problème $P_m||\sum C_j$

Le planning SPT n'est pas le seul horaire optimal. De nombreux autres calendriers minimisent également le temps total d'achèvement.

Comme indiqué dans le chapitre précédent, la règle générale WSPT minimise le temps total d'achèvement pondéré dans le cas d'une seule machine. Malheureusement, ce résultat ne peut pas être généralisé aux machines parallèles, comme le montre l'exemple suivant.

Exemple: Application de la règle WSPT

Considérons deux machines et trois travaux.

Tâches s	1	2	3
P_j	1	1	3
W_j	1	1	3

Traiter les tâches 1 et 2 à l'instant zéro et la tâche 3 à l'instant 1 donne une durée totale pondérée de 14, tandis que l'attribution de la tâche 3 à l'instant zéro et les tâches 1 et 2 sur l'autre machine génèrent une durée d'exécution pondérée totale de 12.

Il a été démontré dans la littérature que l'heuristique WSPT est néanmoins une très bonne heuristique pour le temps de réalisation total pondéré sur des machines parallèles. Une analyse des pires cas de cette heuristique donne la limite inférieure:

$$\frac{\sum w_j C_j(WSPT)}{\sum w_j C_j(OPT)} < \frac{1}{2}(1+\sqrt{2})$$

Passons maintenant au problème $P_m \mid prec \mid \sum C_j$ qui est connu pour être fortement NP-difficile dans le cas de contraintes de priorité arbitraires. Cependant, le cas particulier avec tous les temps de traitement égaux à 1 et les contraintes de priorité prenant la forme d'un arbre extérieur peut être résolu en temps polynomial.

Dans ce cas particulier, la règle de chemin critique minimise à nouveau le temps total d'achèvement.

Théorème 3.6:

La règle CP est optimale pour $P_m \mid p_j = 1, outtree \mid \sum C_j$.

8. The Total Completion Time with Preemptions

Dans le théorème 3.5, il est montré que la règle SPT non préemptive minimise$\sum Cj$ dans un environnement de machines parallèles. Il s'avère que la règle SPT non préemptive est également optimale lorsque les préemptions sont autorisées. Ce résultat est un cas particulier du résultat plus général décrit ci-dessous.

Considérons *m* machines en parallèle à vitesses différentes, c'est-à-dire $Q_m \mid prmp \mid \sum C_j$. Ce problème conduit à la règle Shortest Remaining Processing Time on the Fastest Machine (SRPT-FM). Selon cette règle, à tout moment, le travail avec le temps de traitement restant le plus court est affecté à la machine la plus rapide, au deuxième temps de traitement restant le plus court sur la deuxième machine la plus rapide,...

Théorème 3.7:

La règle SRPT-FM est optimale pour $Q_m \mid prmp \mid \sum C_j$

Clairement, cette règle nécessite des préemptions. Chaque fois que la machine la plus rapide termine un travail, la tâche de la deuxième machine la plus rapide passe à la machine la plus rapide, la tâche de la troisième machine la plus rapide passe à la deuxième machine la plus rapide,...

Ainsi, à la première tâche, il y a *m*-1 préemptions, à la deuxième tâche, il y a *m*-1 préemptions, et ainsi de suite jusqu'à ce que le nombre de tâches restantes soit inférieur au nombre de machines. A partir de ce moment, le nombre de préemptions est égal au nombre de travaux restants.

Exemple: Application of the SRPT-FM.

Considérons trois travaux dont les temps du traitement sont 4, 6 et 10 et deux machines dont les vitesses sont V_1=4 et V_2=2.

Selon la règle SRPT-FM on affecte le travail 1 à la machine 1 et le travail 2 à la machine 2. Dans ce cas on obtient C_1=1. A l'instant 1, la machine 2 a déjà traité deux unités du travail 2, donc la durée restante est 4 (6-2*1). Cette tâche passe maintenant à la machine 1 pour compléter son traitement et la tâche 3 passe à la machine 2.

A l'instant 2, la machine 1 dont la vitesse V_1=4 termine le traitement de l'opération 2 (C_2=2) et la durée restante de l'opération 3 est 10-(2*1). Ce travail se déplace vers la machine 1 pour rester 4 unités et se termine à l'instant 6.

9. Certains objectifs liés à la date d'échéance

Les problèmes d'une seule machine avec des objectifs liés à la date d'échéance qui peuvent être résolus en temps polynomial ont généralement pour objectif le retard maximal, par exemple, $1||L_{max}$, $1 \mid prmp \mid L_{max}$ et $1 \mid r_j, prmp \mid L_{max}$. Les problèmes d'une machine avec le retard total ou le retard total pondéré comme objectif ont tendance à être très difficiles.

Passons à l'ordonnancement non préemptif de tâches indépendantes. En prenant en compte de simples transformations entre les problèmes d'ordonnancement et la relation entre les critères C_{max} et L_{max}, on voit que tous les problèmes qui sont NP-difficiles selon le critère C_{max} restent NP-difficiles sous le critère L_{max}.

Donc, il est facile de voir que d'un point de vue complexité $P_m \mid\mid L_{max}$ n'est pas aussi facile que $1 \mid\mid L_{max}$. Considérons le cas particulier où la date d'échéance de tous les travaux est 0. La recherche d'un programme avec un minimum L_{max} est équivalente à $P_m \mid\mid C_{max}$ et est donc NP-difficile.

D'autre part, les temps de traitement unitaires des tâches facilitent le problème et $\boldsymbol{P_m|p_j=1|L_{max}}$ peut être résolu par une application évidente de la règle **EDD**.

En plus, $P_m \mid prmp \mid L_{max}$ est l'un des rares problèmes d'ordonnancement des machines en parallèle avec un objectif lié à la date d'échéance qui peut être résolu en temps polynomial.

Exercices complémentaires

Exercice 1:

Soit le problème $P_4||C_{max}$ avec les données présentées dans le tableau suivant:

Tâches s	1	2	3	4	5	6	7	8	9
P_j	7	7	6	6	5	5	4	4	4

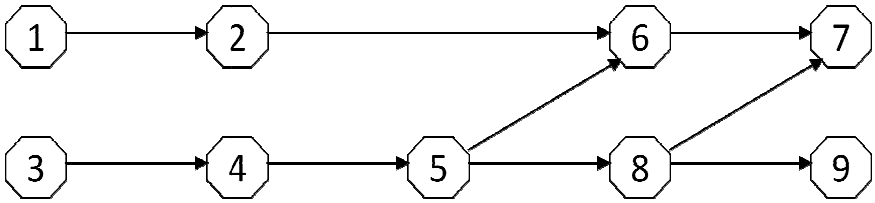
1. Déterminez un ordonnancement satisfaisant en utilisant LPT, déterminez la fonction objectif
2. Déterminez l'ordonnancement optimal (par intuition) comparez sa fonction objectif avec celle de LPT

Exercice 2:

Soit le problème $P_\infty \mid prec \mid C_{max}$ avec les temps d'exécutions et contraintes de précédences suivantes:

Tâches	1	2	3	4	5	6	7	8	9
P_j	4	9	3	3	6	8	8	12	6

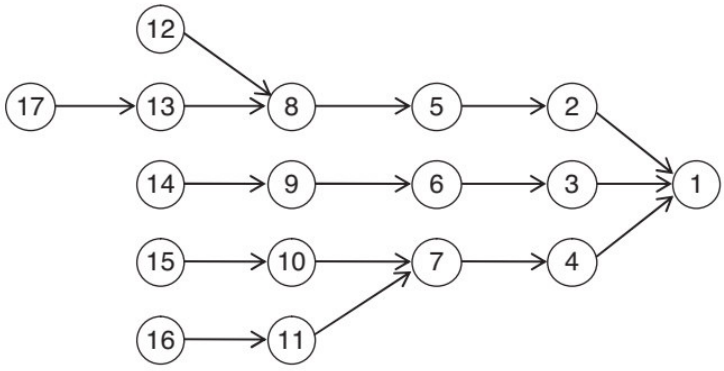
Les contraintes de précédences sont décrites dans la figure suivante:



1. Déterminez le chemin critique et les temps libres (en passant par le CPM forward et backward)

Exercice 3:

Soit le problème $P_3 \mid p_j = 1, intree \mid C_{max}$ avec les contraintes de précédences décrites dans la figure suivante:



1. Déterminez le chemin critique et la fonction objectif.

Exercice 4:

Soit le problème $P_3||C_{max}$ avec les données présentées dans le tableau suivant:

Tâches s	1	2	3	4	5	6	7	8
P_j	1	2	3	4	5	6	7	8

1. Déterminez un ordonnancement satisfaisant en utilisant LPT, déterminez la fonction objectif
2. Considérez le problème $P_3 \mid prmp \mid C_{max}$, déterminez l'ordonnancement optimal avec deux approches.
3. Comparez les résultats de chaque approche avec celle de LPT.

Exercice 5:

Soit le problème $Q_4 \mid prmp \mid C_{max}$, avec les vitesses des différentes machines qui sont V_1= 4, V_2=2, V_3=2, V_4=1. Les temps opératoires sont présentés dans le tableau suivant:

Tâches s	1	2	3	4	5	6	7
P_j	12	18	36	41	45	50	63

1. Déterminez l'ordonnancement optimal ainsi que la valeur de la fonction objectif

Exercice 6:

Soit le problème $P_4 \mid\mid \sum C_j$ avec les données présentées dans le tableau suivant:

Tâches s	1	2	3	4	5	6	7	8	9
P_j	8	7	5	6	4	9	5	3	4

1. Déterminez l'ordonnancement l'optimale ainsi que la fonction objectif

Exercice 7:

Soit le problème $Q_4 \mid prmp \mid \sum C_{max}$, avec les vitesses des différentes machines qui sont V_1= 4, V_2=2, V_3=2, V_4=1.

Les temps opératoires sont présentés dans le tableau suivant:

Tâches	1	2	3	4	5	6	7
P_j	8	16	34	40	45	46	61

1. Déterminez l'ordonnancement optimal ainsi que la valeur de la fonction objectif.

Chapitre 4
Ordonnancement de problèmes à machines en série (Flow shop)

1. Introduction :

Dans de nombreuses installations des systèmes de fabrication et d'assemblage, chaque travail doit être soumis à une série d'opérations. Souvent, ces opérations doivent être effectuées sur tous les travaux dans le même ordre, ce qui implique que les travaux doivent suivre le même itinéraire. Les machines sont ensuite supposées être installées en série et l'environnement est appelé un flow shop.

Dans un flow shop, le travail est divisé en tâches distinctes appelées opérations, et chaque opération est effectuée sur une machine différente. Dans ce contexte, un travail est un ensemble d'opérations avec une structure de priorité spéciale. En particulier, chaque opération après la première a exactement un prédécesseur direct et chaque opération avant la dernière a exactement un successeur direct, comme le montre la figure 4.1. Ainsi, chaque travail nécessite une séquence spécifique d'opérations à effectuer pour qu'il soit complet.

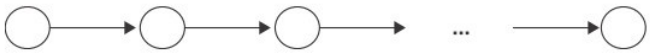


Figure 4.1: La structure de précédences d'un travail dans un flow shop

Les capacités des différents buffers de stockage entre des machines successives peuvent parfois être pratiquement illimitées à toutes fins pratiques. Ceci est souvent le cas lorsque les produits en cours de traitement sont physiquement petits (par exemple, des cartes de circuit imprimé, des circuits intégrés), ce qui rend relativement facile le stockage de grandes quantités entre les machines.

Lorsque les produits sont physiquement volumineux (téléviseurs, copieurs, par exemple), un espace vide entre deux machines successives peut avoir une capacité limitée, ce qui peut provoquer un blocage. Ce dernier se produit lorsque le tampon est plein et que la machine en amont n'est pas autorisée à libérer un travail dans le tampon une fois son traitement terminé. Si tel est le cas, le travail doit rester sur la machine en amont, ce qui empêche un travail de la file d'attente de cette machine de commencer son traitement.

Une grande partie du contenu de ce chapitre concerne l'objectif du Makespan. Cet objectif présente un intérêt pratique et considérable dans la mesure de performances d'un système d'ordonnancement. En plus, sa réduction correspond à d'autres mesures comme la maximisation de l'utilisation des machines. Cependant, ces modèles ont tendance à être d'une telle complexité qu'il est déjà relativement difficile d'obtenir des résultats optimaux. Les autres objectifs liés au temps d'exécution total et à la date d'échéance ont tendance à être encore plus difficiles.

2. Hypothèses inhérentes à un problème à machines en série:

Un atelier à cheminement unique est constitué d'un ensemble de m différentes machines (processeurs) qui effectuent des jobs. Chacun d'entre eux est constitué de m opérations, chacune nécessitant une machine différente. Tous les travaux ont le même ordre de traitement sur les machines, c'est-à-dire qu'un travail est composé d'une liste ordonnée de tâches dans laquelle la tâche i de chaque travail est déterminée par la même machine (la $i^{ème}$ machine de la série) et le temps de traitement correspondant.

Les machines peuvent ainsi être numérotées 1, 2,. . . , m, et les opérations du travail j numérotées $(1, j)$, $(2, j)$,. . . , (m, j), afin qu'ils correspondent à la machine requise. Par exemple, p_{21} indique le temps du traitement sur la machine 2 pour le travail 1.

Avec les machines en série, la majorité des contraintes qui caractérisent le modèle sont similaires à celles du modèle de base à une seule machine. Ces contraintes peuvent être décrites comme suit:

- ✓ Un ensemble de n travaux non liés et comportant plusieurs opérations est disponible pour traitement au moment zéro.
- ✓ Tous les travaux nécessitent une opération sur chaque machine (Chaque travail nécessite m opérations et chaque opération nécessite une machine différente).
- ✓ Chaque machine est une ressource disjonctive.
- ✓ Les temps de configuration ou d'installation des opérations sont indépendants de la séquence et inclus dans les temps de traitement.

- ✓ Les descripteurs de chaque job sont connus à l'avance.
- ✓ Toutes les machines sont disponibles en permanence.
- ✓ Une fois qu'une opération commence, elle se poursuit sans interruption.

Une différence par rapport au cas de base d'une seule machine est que le temps d'inactivité (Idle time) inséré peut être avantageux. Dans le modèle mono machine à arrivées simultanées, on peut assumer que la machine ne reste jamais libre lorsqu'il y'a des opérations en attente de traitement. Par contre, dans un flow shop on peut avoir besoin d'un temps mort inséré pour atteindre l'optimalité.

Exemple :

Considérez les 2 travaux suivants dans un problème $F_4 || C_{max}$

Jobs	1	2
P_{1j}	1	4
P_{2j}	4	1
P_{3j}	4	1
P_{4j}	1	4

Dans ce problème le Makespan est le seul indicateur de performances. Pour ce problème, nous avons deux possibilités d'ordonnancement sans temps libre (soit nous commençons par le job 1 ou le job 2). Les deux plannings sont présentés dans la figure 4.2 a, b, et dans les deux cas, on obtient C_{max}= 24.

Le planning de la figure 4.2 c est optimal, avec C_{max} = 23. Notez que dans ce troisième calendrier, la machine 3 est maintenue libre à l'instant t = 5 où l'opération (3, 1) peut être lancée. Ce temps libre est provoqué afin d'attendre la fin de l'opération (2, 2).

Une autre différence concerne le nombre de permutations. Dans le modèle à une seule machine, il existe une relation un-à-un entre une séquence de tâches et une permutation des nombres 1, 2,. . . , n. Pour trouver une séquence optimale, il est nécessaire d'examiner (au moins implicitement) chacune des séquences correspondant à $n!$ différentes permutations.

En revanche, dans un flow shop, il y en a $n!$ différentes séquences de tâches possibles pour chaque machine, et potentiellement autant que $(n!)^m$ calendriers différents. Alors que nous recherchons un optimum, il serait évidemment utile si nous pouvions ignorer un nombre important de ces possibilités.

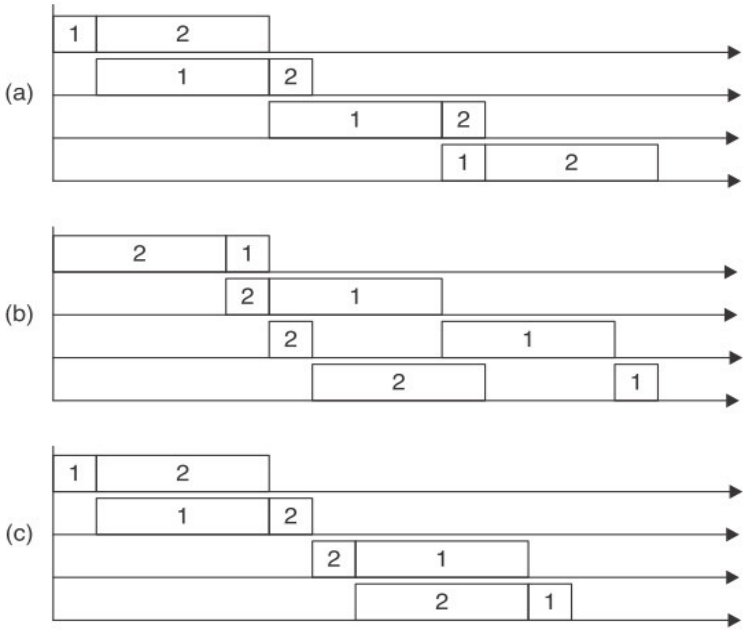


Figure 4.2: Les trois plannings de l'exemple concernant l'importance du temps libre

3. Les problèmes dans flow shop avec stockage intermédiaire illimité:

Lorsqu'on recherche un ordonnancement optimal pour $F_m \mid\mid C_{max}$ la question qui se pose est de savoir s'il suffit de déterminer une permutation dans laquelle les tâches traversent l'ensemble du système.

Physiquement, il peut être possible qu'un travail prend la place d'un autre pendant qu'ils attendent le traitement dans la file d'attente d'une machine occupée. Cela implique que les machines peuvent ne pas fonctionner selon le principe du premier arrivé premier servi (FIFO) et que la séquence dans laquelle les tâches passent par les machines peut changer d'une machine à l'autre.

Le changement de la séquence des tâches en attente dans une file d'attente entre deux machines peut parfois entraîner un Makespan plus petit. Cependant, il peut être démontré qu'il existe toujours un ordonnancement optimal sans changement de séquence des tâches entre les deux premières machines et entre les deux dernières (voir théorème 4.1 et 4.2).

Cela implique qu'il existe des calendriers optimaux pour $F_2||C_{max}$ et $F_3||C_{max}$ qui ne nécessitent pas de changement de séquence entre les machines.

Par conséquent, il n'est pas toujours nécessaire de prendre en compte $(n!)^m$ séquences pour déterminer un optimum. Les deux propriétés de dominance données ci-dessous indiquent des réductions possibles de l'espace de recherche dans les problèmes d'atelier de type flow shop avec buffer de stockage intermédiaire illimité.

Théorème 4.1:

En ce qui concerne le Makespan dans le modèle d'atelier à cheminement unique, il suffit de ne prendre en compte que les ordonnancements dans lesquels la même séquence de travail se produit sur les deux premières machines.

Preuve:

Considérez un calendrier dans lequel les séquences sur les machines 1 et 2 sont différentes. Quelque part dans un tel programme, nous pouvons trouver une paire de tâches, *i* et *j*, avec l'opération (1, *i*) précédant une opération adjacente (1, *j*) mais l'opération (2, *j*) précédant (2, *i*), comme dans la figure 4.3a.

Pour cette paire, on peut imposer à la machine 1 l'ordre des travaux sur la machine 2 (*j* avant *i*), sans nuire à la mesure de performance. Si nous échangeons les opérations (1, *i*) et (1, *j*), nous allons trouver le planning présenté dans la figure 4.3b qui montre que:

- ✓ à l'exception de (1, *i*), aucune opération n'est retardée,
- ✓ l'opération (2, *i*) n'est pas retardée,
- ✓ il peut en résulter un traitement antérieur de (2, *j*), et d'autres opérations.

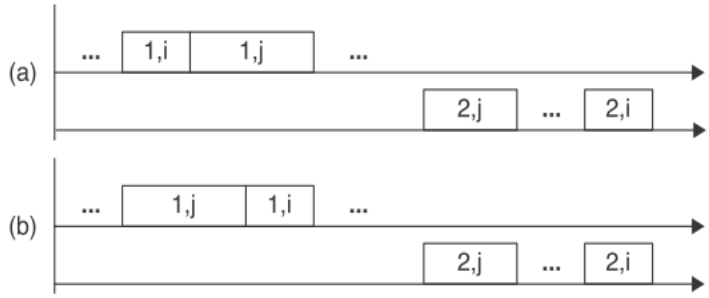


Figure 4.3: Un échange par paire de deux opérations adjacentes sur la machine 1.

Par conséquent, l'échange n'augmenterait pas la date de fin d'une opération sur la machine 2 ou sur une machine ultérieure. Cela signifie qu'aucune augmentation du Makespan ne peut résulter de l'échange. Le même argument s'applique à tout programme dans lequel les séquences de tâches diffèrent sur les machines 1 et 2.

Théorème 4.2:

En ce qui concerne le Makespan dans le modèle d'atelier à cheminement unique, il suffit de ne considérer que les ordonnancements dans lesquels la même séquence de travail se produit sur les deux dernières machines.

Preuve:

Considérons un calendrier dans lequel les séquences sur les machines (m - 1) et m sont différentes. Quelque part dans un tel calendrier, nous pouvons trouver une paire de travaux, i et j, avec l'opération (m, j) précédant une opération adjacente (m, i), mais l'opération (m - 1, i) précédant (m - 1, j). À la suite d'opérations d'échange (m, i) et (m, j), on a:

- ✓ à l'exception de (m, j), aucune opération n'est retardée,
- ✓ l'opération (m, j) se termine avant (m, i) dans le calendrier d'origine,
- ✓ un traitement plus précoce des opérations (m, i) et (m, j) peut en résulter.

Par conséquent, l'échange n'entraînerait pas une augmentation de la date de fin du dernier travail. Encore une fois, ce type d'argument s'applique à tout programme dans lequel les séquences de tâches diffèrent sur les machines (m - 1) et m.

En conséquence des théorèmes 4.1 et 4.2, il est difficile de trouver un programme optimal lorsque les modifications de séquence sont autorisées que de trouver un programme optimal lorsque les modifications de séquence ne sont pas autorisées.

Les flow shop qui ne permettent pas de changement de séquence entre les machines sont appelés flow shop à permutation (permutation flow shop). Dans ces ateliers, la même séquence, ou permutation, de travaux est maintenue tout au long du processus. Les résultats présentés dans ce chapitre se limitent principalement aux ateliers à permutation (permutation flow shop).

Étant donné un calendrier de permutation 1, ..., n avec m machines, le temps d'achèvement du travail k à la machine i peut être calculé facilement à l'aide d'un ensemble d'équations récursives:

$$C_{i,1} = \sum_{l=1}^{i} p_{l,1} \qquad i=1, .., m$$

$$C_{1,k} = \sum_{l=1}^{k} p_{1,l} \qquad k=1, .., n$$

$$C_{i,k} = \max\left(C_{i-1,k}, C_{i,k-1}\right) + p_{i,k} \qquad i=2, .., m,\ \ k=2, .., n$$

La valeur du Makespan sous un programme d'ordonnancement de permutation donné peut également être calculée en déterminant le chemin critique dans un graphe dirigé correspondant à ce programme. Pour une séquence donnée j_1, ..., j_n le graphe orienté peut être construit comme suit:

Pour chaque opération (i, j_k) correspondante au travail j_k sur la machine i, il existe un nœud (i, j_k) avec un poids égal au temps de traitement du travail j_k sur la machine i. Chaque nœud (i, j_k), i=1, ..., m - 1 et k = 1, ..., n - 1, a des arcs allants aux nœuds (i + 1, j_k) et (i, j_k +1)...

Les nœuds correspondants à la machine m n'ont qu'un seul arc sortant, de même que les nœuds correspondants au travail j_n. Le noeud (m, j_n) n'a aucun arc sortant (voir Figure 4.4). Le poids

total du trajet de pondération maximale du nœud $(1, j_1)$ au nœud (m, j_n) correspond à la durée de fabrication sous le schéma de permutation $j_1, ..., j_n$.

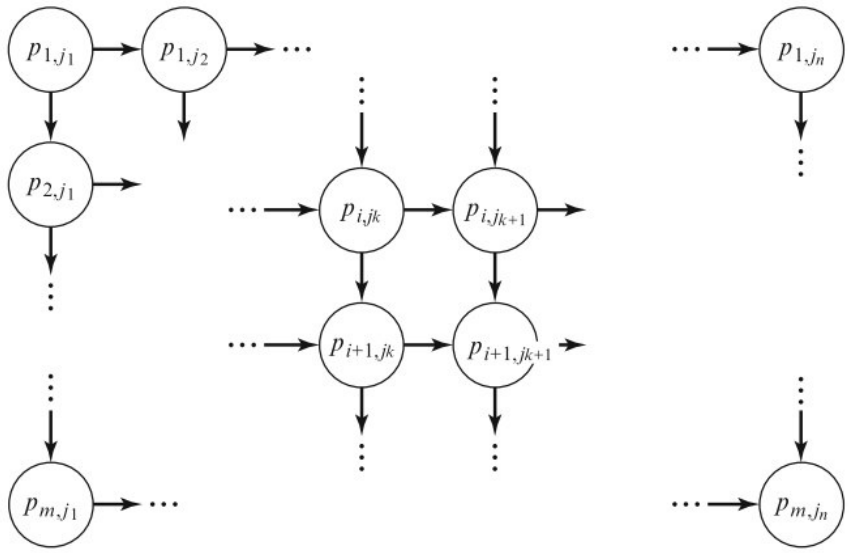


Figure 4.3: Graphe dirigé pour le calcul du Makespan en $F_m|prmu|C_{max}$ sous la séquence $j_1, ..., j_n$.

4. La minimisation du Makespan dans un atelier avec deux machines en série et stockage illimité :

Considérons maintenant le problème $F_2||C_{max}$ avec un stockage illimité entre les deux machines. L'objectif de réduction de la date de fin de fabrication dans un modèle flow shop à permutation à deux machines est également appelé problème de Johnson.

Dans la formulation de ce problème chaque job j est caractérisé par le temps de traitement p_{1j} requis sur la machine 1 et par p_{2j} requis sur la machine 2 après la fin de l'opération sur la machine 1. La règle qui minimise le Makespan est communément appelée règle de Johnson.

Dans cette règle, le job i précède le job j dans une séquence optimale si et seulement si, $\min\{p_{1i}, p_{2j}\} \leq \min\{p_{1j}, p_{2i}\}$.

Une séquence optimale peut être générée comme suit:

- ✓ Partitionnez les tâches en deux groupes tel que les tâches avec $p_{1j}<p_{2j}$ appartiennent au groupe 1 et les tâches avec $p_{1j}>p_{2j}$ appartiennent au groupe 2. Les travaux avec $p_{1j}= p_{2j}$ peuvent être placés dans les deux ensembles (1 ou 2).
- ✓ Les tâches du groupe 1 sont ordonnancées en premier selon la règle SPT.
- ✓ Les tâches du groupe 2 sont ordonnancées en second selon la règle LPT.

- ✓ Planifiez toutes les tâches sur les deux machines dans l'ordre de la séquence de concaténation groupe 1 puis groupe 2.

Dans ce qui suit, une telle séquence est appelée une séquence SPT (1) -LPT (2). Bien entendu, plusieurs programmes peuvent être générés de cette manière.

Théorème 4.3:

Toute séquence SPT (1) -LPT (2) est optimale pour le problème $F_2 \mid\mid C_{max}$.

Preuve:

La preuve est par contradiction. Supposons qu'un autre type de programme est optimal. Dans cet ordonnancement du temps optimal, il doit exister une paire de travaux adjacents, par exemple job *j*, suivi de job *k*, qui remplissent l'une des trois conditions suivantes:

- ✓ *j* appartient à l'ensemble 2 et *k* à l'ensemble 1;
- ✓ les jobs *j* et *k* appartiennent à l'ensemble 1 et $p_{1j} > p_{1k}$;
- ✓ les travaux *j* et *k* appartiennent au groupe 2 et $p_{2j} < p_{2k}$.

Il suffit de montrer que, dans l'une de ces trois conditions, le Makespan est réduit après un échange par paires de travaux *j* et *k*.

Remarques:

La structure d'ordonnancement SPT(1) - LPT (2) ne peut pas être généralisée pour caractériser les séquences optimales pour les ateliers de machines en série comptant plus de deux machines. Cependant, la minimisation de la date de fin du traitement dans un flow shop de permutation avec un nombre arbitraire de machines c'est-à-dire, $F_m \mid prmu \mid C_{max}$, peut être formulé en tant que programme à nombres entiers mixtes (Mixed Integer Program (MIP)).

Une autre remarque concerne l'algorithme de Johnson qui peut être décrit comme suit (deuxième alternative):

- ✓ **étape 1:** Trouvez le temps de traitement minimum parmi les travaux non ordonnancés.
- ✓ **étape 2 a:** Si le minimum à l'étape 1 est atteint sur la machine 1, placez le travail associé dans la première position disponible dans la séquence. Allez à l'étape 3.
- ✓ **étape 2 b:** Si le minimum à l'étape 1 est atteint sur la machine 2, placez le travail associé dans la dernière position disponible dans l'ordre.
- ✓ **étape 3:** Supprimez la tâche attribuée et revenez à l'étape 1 jusqu'à ce que toutes les positions de la séquence soient remplies.

Exemple:

Considérons un problème contenant 5 jobs dans un flow shop avec deux machines, les temps du traitement de ces jobs sont mentionnés dans le tableau suivant:

Tâches	1	2	3	4	5
$P_{1,j}$	3	5	1	6	7
$P_{2,j}$	6	2	2	6	5

Les différentes étapes de résolution de ce problème sont présentées dans le tableau suivant:

Itérations	**Jobs non ordonnancés**	**$\text{Min}_j\{p_{1j}, p_{2j}\}$**	**La position dans la séquence**	**La séquence obtenue**
1	1, 2, 3, 4, 5	p_{13}	1	3 x x x x
2	1, 2, 4, 5	p_{22}	5	3 x x x 2
3	1, 4, 5	p_{11}	2	3, 1, x x 2
4	4, 5	p_{25}	4	3, 1, x 5, 2
5	4	p_{14} ou p_{24}	3	3, 1, 4, 5, 2

5. Le Makespan dans un atelier avec deux machines en série et stockage limité :

Considérons m machines en série avec zéro stockage intermédiaire entre machines successives. Lorsqu'une machine termine le traitement d'un travail, ce dernier ne peut pas passer à la machine suivante si cette dernière est occupée. Le travail doit rester sur la première machine, qui ne peut donc pas démarrer le traitement des travaux suivants. Comme indiqué précédemment, ce phénomène est appelé blocage.

Dans ce qui suit, seuls les ateliers avec zéro stockage intermédiaire sont considérés. Le problème de la minimisation de la durée de fabrication dans un atelier à machines en série avec zéro stockage intermédiaire est appelé $F_m \mid block \mid C_{max}$.

Soit D_{ij} le temps auquel le travail j quitte réellement la machine i. Clairement, $D_{ij} >= C_{ij}$. L'égalité est valable quand le job j n'est pas bloqué. L'instant auquel job j commence à être traité sur la première machine est désigné par D_{0j}.

Les relations récursives suivantes sont valables sous la séquence 1, .., n:

$$D_{i,1} = \sum_{l=1}^{i} p_{l,1}$$

$$D_{i,k} = \max\left(D_{i-1,k} + p_{i,k}, D_{i+1,k-1}\right)$$

$$D_{m,k} = D_{i-1,k} + p_{i,k}$$

Pour ce modèle, on peut également calculer le Makespan sous un programme de permutation donné en déterminant le chemin critique dans un graphe dirigé.

Dans ce graphe orienté, le nœud (i, j_k) indique l'instant de départ du travail j_k de la machine i. Contrairement au graphe de la section 3 pour les ateliers avec stockages intermédiaires illimités, dans ce graphe, les arcs, plutôt que les nœuds, ont des poids.

Le nœud (i, j_k), $i = 1, ..., m - 1$; $k = 1, ..., n - 1$, a deux arcs sortants; un arc va au nœud $(i + 1, j_k)$ et a un poids ou une distance $p_{i+1, jk}$, l'autre arc va au nœud $(i - 1, j_{k+1})$ et a un poids nul. Le nœud (m, j_k) n'a qu'un arc sortant vers un nœud $(m - 1, j_{k+1})$ de poids zéro.

Le nœud (i, j_n) a un seul arc sortant vers le nœud $(i + 1, j_n)$ de poids $p_{i+1,jn}$. Le nœud (m, j_n) n'a pas d'arc sortant (voir Figure 4.4). Le C_{max} de la séquence $j_1, ..., j_n$ est égal à la longueur du chemin de poids maximum du nœud $(0, j_1)$ au nœud (m, j_n).

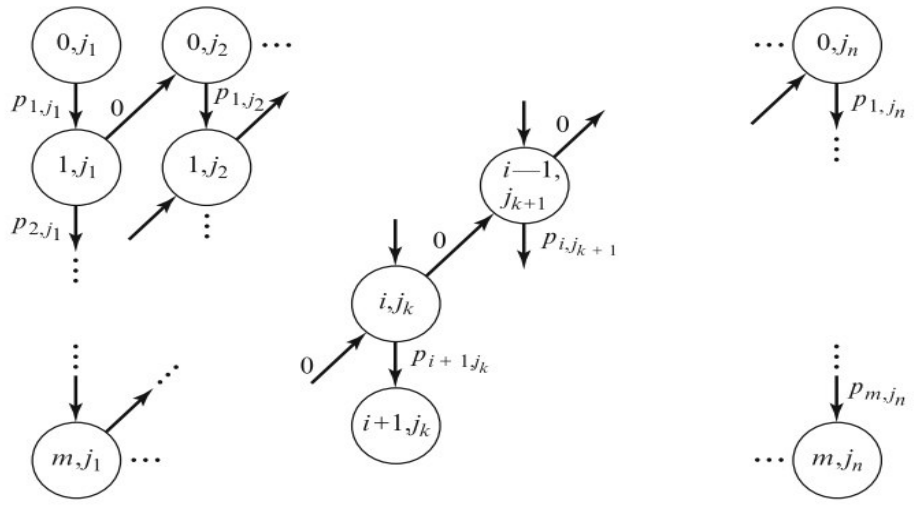


Figure 4.4: Graphe dirigé pour le calcul du Makespan.

6. Flow shops proportionnels avec stockage intermédiaire illimité

Puisque $F_m \mid prmu \mid C_{max}$ étant fortement NP-difficile, il est intéressant d'étudier des cas particuliers qui présentent certaines propriétés structurelles qui peuvent faciliter le problème.

Un cas particulier et important de l'atelier à cheminement unique de permutation avec stockage intermédiaire illimité est appelé flow shop proportionnel de permutation (proportionate permutation flow shop) avec stockage intermédiaire illimité.

Dans cet atelier, les temps de traitement du travail *j* sur chacune des *m* machines sont égaux à p_j, c'est-à-dire que $p_{1j} = p_{2j} = \cdot\cdot = p_{mj} = p_j$. La réduction de la date de fin dans ce type d'atelier est indiquée par $F_m \mid prmu, p_{ij} = p_j \mid C_{max}$.

Théorème 4.4:

Dans un $F_m \mid prmu, p_{ij} = p_j \mid C_{max}$, le Makespan est égal à :

$$C_{max} = \sum_{j=1}^{n} p_j + (m-1)\max(p_1, .., p_n)$$

et il est indépendant du calendrier.

Le résultat énoncé dans le théorème ci-dessus implique que, dans un aspect, le flow shop proportionnel de permutation est similaire à la machine unique: le Makespan ne dépend pas de la séquence. En réalité, il existe beaucoup plus de similitudes entre ce type d'ateliers et la machine unique. Les résultats suivants, qui sont tous des extensions simples de leurs homologues d'une seule machine, illustrent une partie de ce fait:

- ✓ La règle SPT est optimale pour $1 \mid\mid \sum C_j$ ainsi que pour $F_m \mid prmu, p_{ij} = p_j \mid \sum C_j$.
- ✓ L'algorithme qui génère un calendrier optimal pour $1 \mid\mid \sum U_j$ génère également un planning optimal pour $F_m \mid prmu, p_{ij} = p_j \mid \sum U_j$
- ✓ L'algorithme qui génère un calendrier optimal pour $1 \mid\mid h_{max}$ génère également un planning optimal pour $F_m \mid prmu, p_{ij} = p_j \mid h_{max}$

Le modèle flow shop proportionnel de permutation peut être généralisé pour inclure des machines à différentes vitesses. Si la vitesse de la machine *i* est v_i, alors le temps du traitement du travail *j* dans la machine *i* est $p_{ij} = p_j / v_i$.

La machine avec le plus petit v_i s'appelle la machine à goulot d'étranglement. Dans ce problème, le Makespan n'est plus indépendant de la séquence.

Théorème 4.5:

Si dans un $F_m \mid prmu, p_{ij} = p_j \mid C_{max}$ avec des machines à vitesses différents et la première (dernière) machine est le goulot d'étranglement. Alors, la règle LPT (SPT) minimise le Makespan.

Exercices complémentaires

Exercice 1:

Considérer 5 travaux dans un $F_4|\ prmp\ |C_{max}$ avec les temps de traitement présentés dans le tableau ci-dessous:

Tâches	1	2	3	4	5
$P_{1,j}$	5	5	3	6	3
$P_{2,j}$	4	4	2	4	4
$P_{3,j}$	4	4	3	4	1
$P_{4,j}$	3	6	3	2	5

1. Etablir le graphe dirigé de la séquence 1, 2, 3, 4, 5
2. Etablir le diagramme de Gantt correspondant.
3. Déterminer le Makespan

Exercice 2:

Soit le problème $F_2||C_{max}$, avec les données suivantes :

Tâches	1	2	3	4	5	6	7	8	9	10
$P_{1,j}$	5	4	3	5	7	7	3	6	8	2
$P_{2,j}$	3	6	4	3	6	8	6	7	3	4

1. Déterminez un ordonnancement optimal en utilisant les deux alternatives de l'algorithme de Johnson.
2. Calculez la fonction objectif pour chaque ordonnancement.

Exercice 3:

Considérer les données présentées dans l'exercice 1 dans un $F_4|\ block\ |C_{max}$.

1. Etablir le graphe dirigé de la séquence 1, 2, 3, 4, 5.

2. Etablir le diagramme de Gantt correspondant.
3. Déterminer le Makespan

Exercice 4:

Soit le problème $F_4 \mid prmu,\ p_{ij} = p_j \mid \sum C_j$ avec les données présentées dans le tableau suivant:

Tâches	1	2	3	4	5	6	7
P_j	4	8	12	10	5	7	9

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 5:

Soit le problème $F_4 \mid prmu,\ p_{ij} = p_j \mid \sum U_j$, avec les données suivantes :

Jobs	1	2	3	4	5	6
p_j	5	3	4	4	9	3
d_j	17	19	21	22	24	24

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 6

Soit le problème $F_4 \mid prmu,\ p_{ij} = p_j \mid h_{max}$, Avec les données mentionnées dans le tableau suivant:

Jobs	1	2	3	4	5	6	7
p_j	4	8	12	7	6	9	9
$h_j(C_j)$	$3C_1$	77	C_3^2	$1{,}5C_4$	$70+\sqrt{C_5}$	$1{,}6C_6$	$1{,}4C_7$

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 7

Soit le problème $F_4 \mid prmu, p_{ij} = p_j \mid C_{max}$ avec les vitesses des différentes machines qui sont V_1= 1, V_2=2, V_3=2, V_4=2.

Les temps opératoires sont présentés dans le tableau suivant:

Tâches	1	2	3	4	5	6	7
p_j	4	8	12	10	6	14	2

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Chapitre 5
Ordonnancement de problèmes de type Job shop et Open shop

1. Introduction :

Dans ce chapitre, nous continuons l'ordonnancement des tâches sur des processeurs ou des machines dédiées. Ce chapitre traite certains modèles multi opérations qui diffèrent des modèles d'atelier de type flow shop décrits dans le chapitre précédent. Nous supposons que les tâches appartiennent à un ensemble de jobs, chacune étant caractérisée par sa propre séquence de machines.

Le premier type d'implantation de l'usine présenté dans ce chapitre est le job shop ou l'atelier à cheminements multiples. Il fournit une forme de fabrication plus flexible que celle de l'atelier à cheminement unique. Le problème classique de l'ordonnancement des ateliers de type job shop diffère du problème des ateliers de type flow shop sur un point important. Dans un flow shop, tous les travaux suivent le même itinéraire. Cependant, le flux de travail n'est pas unidirectionnel dans un job shop où les routes sont fixés mais pas nécessairement les mêmes pour chaque travail.

Les éléments du problème de type job shop sont un ensemble de *m* machines et un ensemble de *n* jobs à ordonnancer. Chaque travail consiste en plusieurs opérations avec la même structure de priorité linéaire (sa propre gamme de fabrication), comme dans le modèle de type flow shop. Bien qu'un travail puisse avoir un nombre quelconque d'opérations. Comme le flux de travail dans un atelier de travail n'est pas unidirectionnel, nous pouvons considérer chaque machine de l'atelier comme disposante des flux de travail en entrée et en sortie. Contrairement au modèle à machines en série, il n'existe pas de machine initiale effectuant uniquement la première opération d'un travail, ni de machine terminale effectuant uniquement la dernière opération d'un travail.

Ce chapitre traite également d'autres modèles multi opérations qui diffèrent des modèles d'atelier de type flow shop et job shop. Dans un flow shop, les jobs ont un seul routage et dans un job shop chaque travail a un itinéraire fixe qui est prédéterminé. Dans la pratique, il arrive souvent que le routage du travail soit indifférent et décidé par l'ordonnanceur de la production. Lorsque les itinéraires des travaux sont libres, le modèle est appelé atelier de type open shop ou atelier à cheminements libres.

La première partie de ce chapitre porte sur les représentations et les formulations du problème classique du job shop avec l'objectif Makespan. La deuxième section est réservée aux modèles de type open shop non préemptif avec l'objectif Makespan

2. Programmation Disjonctive pour la minimisation du Makespan dans un job shop

Comme nous avons mentionné dans la section précédente, un atelier de type job shop consiste en un ensemble de différentes machines (telles que des tours, des fraiseuses, des perceuses,...) qui effectuent des tâches. Chaque travail a un ordre de traitement spécifié via les machines, c.-à-d. un travail est composé d'une liste ordonnée des opérations dont chacune est déterminée par la machine requise et par la durée de traitement correspondante. Il existe plusieurs contraintes sur les travaux et les machines prises en compte dans cette section. Ces conditions peuvent être résumées dans les points suivants:

- ✓ Il n'y a pas de contraintes de précédence (contrainte conjonctive) entre les tâches des travaux différents (les contraintes conjonctives décrivent l'ordre des opérations du même travail).
- ✓ Les tâches ne peuvent pas être interrompues (la préemption n'est pas autorisée).
- ✓ Chaque machine ne peut gérer qu'une seule opération à la fois.
- ✓ Chaque tâche ne peut être effectuée que sur une machine à la fois.

Considérons le problème le plus simple $J_2 \mid\mid C_{max}$, il y a deux machines et n travaux. Certaines tâches doivent être traitées d'abord sur la machine 1, puis sur la machine 2. Les tâches restantes doivent être traitées d'abord sur la machine 2, puis sur la machine 1. Le temps de traitement du travail j sur la machine 1 (2) est de p_{1j} (p_{2j}). L'objectif est de minimiser le Makespan.

Ce problème peut être réduit à $F_2 \mid\mid C_{max}$ comme suit. Soit $J_{1,2}$, l'ensemble des tâches à traiter en premier sur la machine 1 et $J_{2,1}$, l'ensemble des tâches à traiter en premier sur la machine 2. Notez que lorsqu'un travail de $J_{1,2}$ a terminé son traitement sur la machine 1, le report de son traitement sur la machine 2 n'affecte pas le Makespan tant que la machine 2 est occupée. La même

chose pour chaque travail de $J_{2,1}$; si un tel travail a terminé son traitement sur la machine 2, le report de son traitement sur la machine 1 (tant que la machine 1 est occupée) n'affecte pas le Makespan.

Par conséquent, un travail de $J_{1,2}$ a sur la machine 1 une priorité plus élevée que tout travail de $J_{2,1}$, alors qu'un travail de $J_{2,1}$ a sur la machine 2 une priorité supérieure à tout travail de $J_{1,2}$. Il reste à déterminer dans quelle séquence les travaux de $J_{1,2}$ passent par la machine 1 et les travaux de $J_{2,1}$ passent par la machine 2.

La première de ces deux séquences peut être déterminée en considérant $J_{1,2}$ comme un problème $F_2 \mid\mid C_{max}$ avec la machine 1 configurée en premier puis la deuxième machine. Par contre, la deuxième séquence peut être déterminée en considérant $J_{2,1}$ comme un autre problème $F_2 \mid\mid C_{max}$ avec la machine 2 configurée d'abord et la machine 1 seconde. Cela conduit à des séquences SPT (1) -LPT (2) pour chacun des deux ensembles, avec des priorités entre les ensembles, comme indiqué ci-dessus.

Ce problème de deux machines est l'un des rares problèmes d'ordonnancement d'ateliers de type job shop pour lesquels un algorithme à temps polynomial peut être trouvé. Certains d'autres problèmes pour lesquels des algorithmes de temps polynomiaux peuvent être obtenus mais nécessitent généralement que tous les temps de traitement soient 0 ou 1.

Le reste de cette section est dédié au problème $J_m \mid\mid C_{max}$ avec des temps de traitement arbitraires et aucune recirculation (Aucune opération ne peut visiter la même machine plus d'une fois).

Réduire le Makespan d'un atelier de type job shop sans recirculation $J_m \mid\mid C_{max}$ peut être représenté par un graphe disjonctif. Considérons un graphe orienté G avec un ensemble de nœuds N et deux ensembles d'arcs A et B. Les nœuds N correspondent à toutes les opérations (i, j) des n travaux. Les arcs appelés conjonctifs (solides) A représentent les itinéraires des travaux.

Si l'arc $(i, j) \rightarrow (k, j)$ fait partie de A, le travail j doit être traité sur la machine i avant d'être traité sur la machine k, c'est-à-dire que l'opération (i, j) précède l'opération (k, j).

Deux opérations appartenant à deux travaux différents et devant être traitées sur la même machine sont connectées l'une à l'autre par deux arcs dits disjonctifs (cassés) qui vont dans des directions opposées. Les arcs disjonctifs B forment m cliques d'arcs doubles, une clique pour chaque machine. Toutes les opérations (nœuds) dans la même clique doivent être effectuées sur la même machine.

Tous les arcs émanant d'un nœud, qu'ils soient conjonctifs ou disjonctifs, ont pour longueur le temps de traitement de l'opération représentée par ce nœud. De plus, il existe une source U et un

puits *V*, qui sont des nœuds factices. Le nœud source U a *n* arcs conjonctifs émanant des premières opérations des *n* travaux et le nœud de puits *V* a *n* arcs conjonctifs provenant de toutes les dernières opérations. Les arcs émanant de la source ont une longueur zéro. Ce graphe est noté $G = (N, A, B)$. (Voir Figure 5.1).

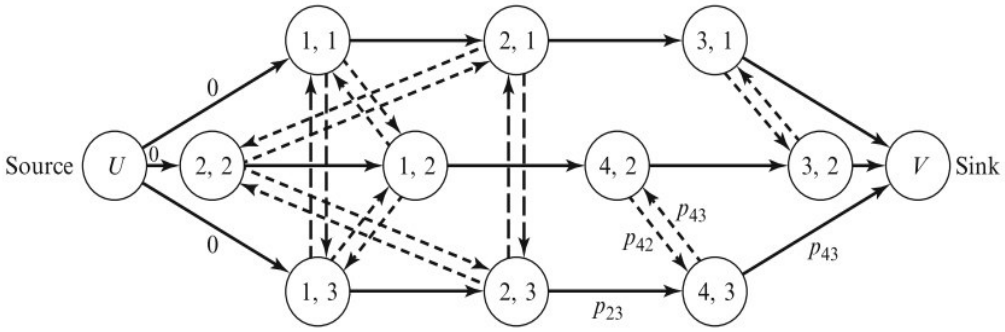


Figure 5.1: Un graphe dirigé d'un job shop pour minimiser le Makespan

Un ordonnancement réalisable correspond à une sélection d'un arc disjonctif de chaque paire, de sorte que le graphe dirigé résultant soit acyclique. Cela implique qu'une sélection d'arcs disjonctifs d'une clique doit être acyclique.

La date de fin du planning réalisable est déterminée par le plus long chemin de la source *U* au puits *V* dans le graphe acyclique correspondant. Ce chemin consiste en un ensemble d'opérations dont la première commence à l'instant 0 et la dernière se termine à l'instant du Makespan. Chaque opération sur ce chemin est immédiatement suivie de l'opération suivante sur la même machine ou de l'opération suivante du même travail sur une autre machine. Le problème de la minimisation de Makespan est réduit à la recherche d'une sélection d'arcs disjonctifs minimisant la longueur du plus long chemin (c'est-à-dire le chemin critique).

Il existe également plusieurs formules de programmation mathématique pour les job shops sans recirculation, notamment un certain nombre de formules de programmation en nombres entiers. Cependant, la formulation la plus utilisée est la programmation disjonctive. Cette formulation de programmation disjonctive est étroitement liée à la représentation graphique disjonctive de l'atelier à cheminements multiples.

Pour présenter la formulation de programmation disjonctive, la variable y_{ij} indique la date de début de l'opération (i, j). L'ensemble N désigne l'ensemble de toutes les opérations (i, j) et A l'ensemble des contraintes de routage $(i, j) \rightarrow (k, j)$ qui nécessitent que le travail j soit traité sur la machine i avant son traitement sur la machine k. Le programme mathématique suivant minimise le Makespan:

Minimiser C_{max}

Sachant que:

$y_{kj} - y_{ij} \geq p_{ij}$ Pour chaque $(i,j) \rightarrow (k,j) \in A$

$C_{max} - y_{ij} \geq p_{ij}$ Pour chaque $(i,j) \in N$

$y_{ij} - y_{il} \geq p_{il}$ ou $y_{il} - y_{ij} \geq p_{ij}$ Pour chaque (i, l) et (i,j), $i = 1,...,m$

$y_{ij} \geq 0$ Pour chaque $(i,j) \in N$

Dans cette formulation, la première contrainte garantit que l'opération (k, j) ne peut pas démarrer avant que l'opération (i, j) soit terminée. La troisième contrainte s'appelle contrainte disjonctive; elle garantit qu'il existe un certain ordre parmi les opérations de différentes tâches qui vont être traitées sur la même machine.

3. Le Makespan dans un open shop sans préemptions

Considérons le problème $O_2 \mid\mid C_{max}$; c'est-à-dire qu'il y a deux machines et n jobs. Chaque travail j peut être traité d'abord sur la machine 1, puis sur la machine 2 ou inversement; le décideur peut déterminer les itinéraires. Afin de minimiser le Makespan dont une borne inferieure a été définie comme suit:

$$C_{max} \geq \max\left(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$$

Selon l'inégalité, il est clair que le Makespan ne peut pas être inférieur à la charge de travail de chaque machine.

Cette section considère uniquement les ordonnancements sans délai. Autrement dit, si un travail est en attente de traitement lorsqu'une machine est libre, cette machine n'est pas autorisée à rester inactive. Une période d'inactivité peut se produire sur une machine si et seulement si une tâche est en attente de traitement sur cette machine et, lorsque cette machine est disponible, cette dernière tâche est en cours de traitement sur l'autre machine.

Dans le but de minimiser le Makespan, on considère la règle suivante: Chaque fois qu'une machine est libre, choisir parmi les tâches restantes, la tâche avec le temps d'exécution le plus long sur l'autre machine. Cette règle est décrite ci-après sous le nom de Longest Alternate Processing Time first (LAPT) rule.

À l'instant zéro, lorsque les deux machines sont disponibles, il peut arriver que le même travail devienne prioritaire sur les deux machines. Si tel est le cas, il est traité sur une des deux machines. En plus, selon cette règle, chaque fois qu'une machine est libérée, les tâches dont le

traitement est déjà terminé sur l'autre machine ont la priorité la plus basse, c'est-à-dire zéro, sur la machine qui vient d'être libérée. Il n'y a donc pas de distinction entre les priorités de deux tâches déjà traitées sur l'autre machine.

Théorème 5.1:

La règle LAPT est optimale pour le problème $O_2 \mid\mid C_{max}$ avec un Makespan :

$$C_{max} = \max\left(max_j(p_{1j}, p_{2j}), \sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$$

Cependant, cette règle n'aboutit pas toujours à un ordonnancement optimal car $O_m \mid\mid C_{max}$ est NP-difficile lorsque $m \geq 3$. Par contre le problème $O_m \mid p_{ij}=\{0,1\} \mid C_{max}$ peut être résolu en un temps polynomial. Pour trouver une solution optimale à ce problème, on applique les deux règles suivantes:

- ✓ JLMO-MLNO (job with largest number of operations first on the machine with the largest number of operations)
- ✓ MLNO-JLMO (machine with the largest number of operations first for job with the largest number of operations)

Exercices complémentaires

Exercice 1:

Soit le problème $J_2 \mid\mid C_{max}$, Avec les données suivantes :

Tâches	1	2	3	4	5	6	7	8	9	10
$P_{1,j}$	5	4	3	5	7	7	3	6	8	2
$P_{2,j}$	3	6	4	3	6	8	6	7	3	4

1. Déterminez un ordonnancement optimal.
2. Calculez la fonction objectif pour chaque ordonnancement.

Exercice 2:

Soit le problème $O_2||\ C_{max}$ avec les données présentées dans le tableau suivant:

Tâches	1	2	3	4	5	6	7	8	9
P_{1j}	5	4	3	5	7	7	3	6	8
P_{2j}	3	6	4	3	6	8	6	7	3

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Exercice 3:

Soit le problème $O_5|p_{ij}=\{0,1\}|C_{max}$, Avec les données suivantes :

Jobs	1	2	3	4	5
P_{1j}	1	0	0	1	1
P_{2j}	1	1	1	0	0
P_{3j}	0	1	0	1	1
P_{4j}	1	1	1	0	1
P_{5j}	1	1	1	1	0

1. Trouvez une séquence optimale.
2. Trouvez la fonction objectif.

Références

Livres :

Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Gunter Schmidt and Jan Weglarz (2007). Handbook on scheduling: from theory to applications. Springer 2007.

Kenneth R. Baker and Dan Trietsch (2009). Principles of sequencing and scheduling. John Wiley & Sons 2009.

Michael L. Pinedo (2005). Planning and Scheduling in Manufacturing and Services. Springer 2005.

Michael L. Pinedo (2016). Scheduling Theory, Algorithms, and Systems (Fifth edition). Springer 2016.

Pierre Lopez and François Roubellat (2008) Production Scheduling. John Wiley & Sons (2008).

Thèses de doctorat :

Abir Ben Hmida Sakly (2009). Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible. Thèse de doctorat, Université de Toulouse. France 2009.

Fatma Tangour Toumi. Ordonnancement dynamique dans les industries agroalimentaires. Thèse de doctorat, Université des Sciences et Technologies de Lille, France. (2007).

Guillaume Pinot, Coopération homme-machine pour l'ordonnancement sous incertitudes, Thèse de Doctorat, Université de Nantes, France 2008

Letouzey Agnès (2001). Ordonnancement interactif basé sur des indicateurs: Applications à la gestion de commandes incertaines et à l'affectation des opérateurs. Thèse de doctorat, L'Institut National Polytechnique de Toulouse, France (2001).

Mehdi Souier, Investigations sur la sélection de routages alternatifs en temps réel basées sur les métaheuristiques -les essaims particulaires-, Thèse de doctorat, Université de Tlemcen, Algérie (2012).

Sadia Azem (2010). Ordonnancement des systèmes flexibles de production sous contraintes de disponibilité des ressources, Thèse de doctorat, École Nationale Supérieure des Mines de Saint-Étienne, France (2010).

Samia Ourari, De l'ordonnancement déterministe à l'ordonnancement distribué sous incertitudes. Thèse de doctorat, Université Paul Sabatier de Toulouse, France 2011.

Autres polycopiés et cours:

Mohamed Ali Aloulou, Introduction aux problèmes d'ordonnancement, Université Paris Dauphine, France

Zaki Sari, Ordonnancement de la production, Université de Tlemcen, Algérie.