

MINISTRE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

Université Mohamed Boudiaf - M'Sila

FACULTE DES SCIENCES ET SCIENCES DE L'INGENIEUR

Département De Mathématiques

Mémoire

Présenté par

LATRECHE FAIZ

En vue de l'obtention du diplôme de

MAGISTER

En : Mathématiques

Option : Mathématiques Discrètes

Thème:

LES PROBLEMES D'ORDONNANCEMENT A CHEMINEMENT UNIQUE « FLOW SHOP » ET MULTIPLE « JOB SHOP »

Soutenu le : 07/07./2004

Devant la commission d'examen composée de :

Président :	M.MOUSSAI	Prof Université de M'sila
Rapporteur :	H.BELOUADAH	M.C Université de M'sila
Examineurs:	B.BOUDERAH	M.C Université de M'sila
	D.MIHOUBI	M.C Université de M'sila
	A.BENHOCINE	M.C Université de Sétif

ملخص :

تتضمن الأطروحة دراسة نمطين من مسائل ترتيب الأشغال :

المسائل ذات السير الموحد (flow shop) و المتعدد (job shop)

حاولنا في القسم النظري تقديم هذان النمطان من خلال بعض التعاريف و الخوارزميات المشهورة لحل حالات خاصة.

القسم التطبيقي تم فيه اقتراح خوارزميتين تقريبيتين A و B بنفس مبدأ طريقة CDS لحل المسألة العامة $n/m/F/C_{max}$. بعد ذلك قمنا ببرمجة الطرق الثلاث واختبارها على مجموعة من الامثلة .

الكلمات المفاتيح : ترتيب الأشغال ، السير الموحد والمتعدد خوارزمية

Abstract :

In this memory , we study two types of scheduling problems "flow shop" and "job shop". We tried in the theoretical part to give some definitions and famous algorithms intended for the resolution of the particular cases. In the practical part we proposed two heuristics methods A , B for the problem $n/m/F/C_{max}$ and we tried to test and compare the three heuristics A , B and CDS for several examples .

Key-words : scheduling , flow shop , job shop , heuristics.

Résumé :

Ce mémoire s'intéresse aux problèmes d'ordonnancement a cheminement unique (flow shop) et multiple (job shop) . la partie théorique est consacrée à la représentation de ces deux type des problèmes avec quelque cas particuliers. Dans la partie pratique , on a proposé deux heuristiques A , B basés sur le même principe que CDS pour résoudre le problème général $n/m/F/C_{max}$ puis on a programmé ces trois méthodes (A , B et CDS) pour les tester et comparer sur différents exemples.

Mots – clefs : ordonnancement , cheminement unique et multiple , heuristiques

Remerciement

Je tiens à remercier monsieur BELOUADH HOCINE pour sa contribution à ma formation et son encadrement de mon travail, ainsi pour ces conseils qui m'ont été utiles.

J'exprime mes remerciements au jury composé de: Mr MOUSSAI, Mr BOUDERAH, Mr MIHOUBI et Mr BENHOCINE qui m'ont fait l'honneur de juger mon travail.

Je tins aussi à remercier mes collègues: LAKHEL MEFTAH, BN ORAIRA ALI pour ces aides apportés pour la réalisation de ce document.

TABLE DES MATIÈRES

INTRODUCTION	1
CHAPITRE 1 : Les problèmes d’ordonnancement	3
1.1 Définition générale	4
1.2 Les tâches	4
1.3 Les ressources	4
1.4 Les contraintes	5
1.5 Les critères.....	6
1.6 Représentation des solutions.....	6
1.7 Classification des problèmes	7
1.8 Notions de complexité et algorithme efficace	8
 CHAPITRE 2 : Les problèmes de type Flow-shop	10
2.1 Les problèmes de type Flow-shop	11
2.1.1 Algorithme de Johnson pour le problème $n/2/F/C_{\max}$	13
2.1.2 Un cas particulier pour le problème $n/3/F/C_{\max}$	15
2.1.3 Méthodes heuristiques	16
2.1.4 Heuristiques pour le problème général $n/m/F/C_{\max}$	18
2.1.4.1 Heuristique de Palmer	20
2.1.4.2 Heuristique CDS	21
2.1.4.3 Heuristique de Dannebring	22
 CHAPITRE 3 : Cas spécifiques des problèmes « flow shop » et le problème « job shop »	24
3.1 Cas spécifiques des problèmes flow shop	25
3.1.1 Flow Shop avec lags entre tâches	25
3.1.2 Flow shop avec relations de précédence	26
3.1.3 Le flow shop hybride HFS	26
3.2 Les problèmes de type job shop	28
3.2.1 Algorithme de Johnson pour le problème $n/2/G/C_{\max}$	29
 CHAPITRE 4 : Heuristiques, proposées et programmées, pour le problème général $n/m/F/C_{\max}$	32
4.1 Heuristique A	33
4.2 Heuristique B	35

4.3 Programmes	38
4.4 Tests et résultats	50
4.5 Discussion	58
Conclusion générale	60
Références bibliographiques	62

INTRODUCTION

Les problèmes d'ordonnancement apparaissent dans plusieurs circonstances pratiques et sous une large variété d'aspects. L'ordonnancement intervient massivement, notamment, dans le domaine de production en industrie, et dans d'autres domaines économiques et militaires ... etc. Pour cette raison, l'ordonnancement est devenu un outil stratégique et moyen nécessaire dans la vie humaine.

L'ordonnancement est l'activité de parvenir à décider, dans le détail, des tâches à accomplir, des ressources (en particulier les machines) à utiliser pour accomplir ces tâches.

L'objet de ce mémoire, est de traiter deux cas particuliers des problèmes d'ordonnancement :

Les problèmes de type flow shop et job shop dont l'objectif est minimiser le makespan (date de fin de projet : $C_{\max}$).

Dans le problème de type flow shop, chaque tâche passe sur chaque machine une fois seulement, et l'ordre de passage sur les machines est le même pour toutes les tâches, alors que pour le job-shop, cet ordre peut être différent.

Le chapitre 1 est consacré aux notions générales des problèmes d'ordonnancement, on y donne leur définition générale, avec les termes y liés (tâche, ressource, contraintes ...). Ce chapitre se termine par des notions de complexité, où, on met en évidence les classes des problèmes P et NP ainsi les problèmes dits NP-difficile.

Dans le chapitre 2, on s'intéresse aux problèmes de type « flow shop » (où à cheminement unique). On y aborde le problème de type flow shop à deux machines

$n/2/F/C_{\max}$ résolu par la règle de Johnson. Le cas de flow shop à trois machines $n/3/F/C_{\max}$, avec certaine condition, est résolu par l'algorithme de Johnson y adapté. La méthode graphique de Aker pour le problème $2/m/F/C_{\max}$ a été aussi présentée.

Après une discussion sur le concept des heuristiques, on s'intéresse au cas général de flow shop à m machine $n/m/F/C_{\max}$ pour lequel des méthodes de résolution dites heuristiques, trouvées dans la littérature ont été mise (heuristique de Palmer, CDS et Dannenbring).

Une heuristique est, en fait, une procédure de calcul qui conduit à une solution dont on peut calculer l'écart à l'optimum, ou pour laquelle on peut donner un écart maximal à l'optimum.

Le chapitre 3 traite des cas spécifiques des problèmes de type « flow shop » à savoir le « flow shop » avec lags entre tâches, flow shop avec relations de précédence et le flow shop hybride (HFS). On y aborde aussi le problème de type « Job shop » (où à cheminement multiple).

On propose dans le chapitre suivant, deux heuristiques appelées A et B pour résoudre le problème général $n/m/F/C_{\max}$. Après la présentation de ces deux heuristiques avec des exemples, on les programme, ainsi que l'heuristique CDS, par le langage Delphi, on procède ensuite au test de ces trois heuristiques sur plusieurs exemples de 3,4,5,6 et 7 machines.

Ce chapitre est clôturé par une discussion des résultats trouvés à travers ces différents exemples.

Une conclusion générale sur tout le travail a été mise en dernier.

CHAPITRE 1

LES PROBLEMES

D'ORDONNANCEMENT

1.1 Définition générale

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation des tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement,) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps et vise à satisfaire un ou plusieurs objectifs.

De manière plus précise, on parle d'ordonnancement lorsqu'on fixe les dates de début ou de fin de chacune des tâches, alors qu'on réserve le temps de séquençement au cas où seul l'ordre relatif des tâches (séquence) serait fixé, indépendamment des dates d'exécution.

1.2 Les tâches

Une tâche ; est une entité élémentaire de travail localisée dans le temps par une date de début t_i ou de fin C_i , dont la réalisation nécessite une durée $p_i = C_i - t_i$, et qui consomme des moyens k avec une intensité a_{ik} .

Selon les problèmes, les tâches peuvent être exécutées par morceaux, ou doivent être exécutées sans interruption ; on parle alors respectivement de problèmes préemptifs et non préemptifs.

Lorsque les tâches ne sont pas liées entre elles par des contraintes de cohérence technologique (par exemple contraintes d'enchaînement liées aux procédés de réalisation), elles sont dites indépendantes.

1.3 Les ressources

Une ressource k est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche, et disponible en quantité limitée, sa capacité A_k .

Plusieurs types de ressource sont à distinguer. Une ressource est renouvelable si, après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les hommes, les machine, l'espace, l'équipement en général,...) ; la quantité de ressource utilisable à chaque instant est limitée. Dans le cas contraire, elle est

consommable (matière première, budget, ...); la consommation globale (ou cumul) au cours du temps est limitée.

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut à priori varier au cours du temps. On distingue par ailleurs - principalement dans le cas de ressources renouvelables- les ressources disjonctives (on non partageables) qui ne peuvent exécuter qu'une tâche à la fois (machine-outil, robot manipulateur) et les ressources cumulatives (ou partageables) qui peuvent être utilisées par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail).

Dans notre travail, on s'intéressera au cas des ressources renouvelables disjonctives représenté par les machines.

1.4 Les contraintes

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision que ce soit sur le temps (variable d'ordonnancement) ou sur les ressources (variable d'affectation). On distingue :

1.4.1 Les contraintes temporelles

- Contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limitées des tâches(délais de livraisons, disponibilité des approvisionnements).
- Contraintes d'antériorité et plus généralement contrainte de cohérence technologique (exemple : contrainte de gammes dans le cas des problèmes d'atelier), qui décrivent des positionnements relatifs devant être respectés entre tâches.

1.4.2 Les contraintes de ressources

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

1.5 Les critères

Les facteurs importants dans l'évaluation d'un problème d'ordonnancement sont l'utilisation efficace des ressources, le délai global et le respect du plus grand nombre possible de contraintes.

Les critères, encore appelés fonctions économiques, les plus utilisées font intervenir la durée totale, le délai d'exécution pondérée, les retards et le coût des stocks d'encours :

1.5.1 *Durée totale*

La durée totale de l'ordonnancement, notée $C_{\max}$ est égale à la date d'achèvement de la tâche la plus tardive: $C_{\max} = \max_{i \in I} \{C_i\}$, avec C_i est la date de fin d'exécution de la tâche i .

1.5.2 *Respect des dates au plus tard*

On peut chercher à minimiser le plus grand retard $T_{\max} = \max_{i \in I} \{T_i\}$, ou la somme des retards $\sum_{i \in I} w_i T_i$ avec $T_i = \max(0, C_i - d_i)$ pour une tâche i de délai d_i .

1.5.3 *Minimisation d'un coût:*

Ce type de critère peut s'exprimer sous des formes très variées. Par exemple, le critère $\sum_{i \in I} w_i C_i$ permet d'estimer le coût des stocks d'encours.

Un critère à minimiser $f(C_1, C_2, \dots, C_n)$ est dit régulier si :

$$(C'_1 \leq C_1, \dots, C'_i \leq C_i, \dots, C'_n \leq C_n) \Rightarrow f(C'_1, \dots, C'_i, \dots, C'_n) \leq f(C_1, \dots, C_i, \dots, C_n).$$

Il en résulte que si un ordonnancement S' est meilleur qu'un ordonnancement S , alors au moins pour une tâche i , on a : $C'_i < C_i$.

1.6 Représentation des solutions

Le diagramme de Gantt est le type de représentation le plus simple pour visualiser graphiquement l'exécution des tâches et l'occupation des ressources au cours du temps.

Dans le cas du problème d'atelier, le diagramme de Gantt est composé d'une ligne horizontale pour chaque ressource. Il permet de visualiser les zones d'occupation et d'oisiveté, la séquence des opérations sur les ressources et, bien entendu, la durée de l'ordonnancement.

Notation

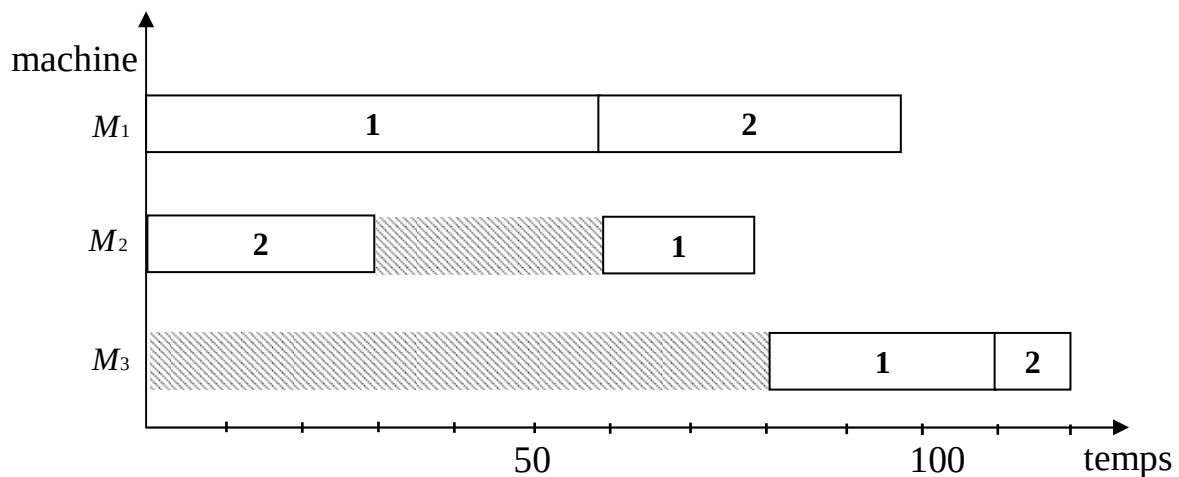
On note par p_{ij} la durée d'exécution de la tâche i sur la machine j , que l'on appelle la durée opératoire.

Exemple

Soit le tableau suivant qui donne les durées opératoires p_{ij} des tâches 1,2 sur les machines M_1 , M_2 et M_3 .

		Machine		
		M_1	M_2	M_3
Tâche	1	$p_{11}=60$	$p_{12}=20$	$p_{13}=30$
	2	$p_{21}=40$	$p_{22}=30$	$p_{23}=10$

Le diagramme de Gantt ci-dessous, représente un ordonnancement de durée totale égale à cent vingt unités de temps.



1.7 Classification des problèmes

Il est commode d'avoir une simple notation pour représenter les types des problèmes d'ateliers.

On classe un problème selon quatre paramètres : $n/m/A/B$.

n est le nombre de tâches

m est le nombre de machines

A décrit la discipline dans les machines.

Si $m=1$, A est laissé vide. A pourrait être :

F pour le cas de flow-shop.

P pour le cas de permutation flow-shop.

G pour le cas Job-shop général

B représente le critère par lequel l'ordonnancement est à évaluer.

Comme exemple : $n/2/F/C_{\max}$ est le problème flow-shop de n tâches, 2 machines et pour but de minimiser le makspan.

Remarque

Les notions de flow-shop et job-shop seront illustrées dans le prochain chapitre.

1.8 Notions de complexité et algorithme efficace

Un « algorithme » de résolution d'un problème $\mathcal{P}$ donné est une procédure, décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de n'importe quel exemple du problème $\mathcal{P}$ en une chaîne de caractères représentant les résultats de $\mathcal{P}$.

L'efficacité d'un algorithme pour un problème est évaluée par le nombre maximum d'étapes de calcul – en fonction de n – nécessaires pour obtenir une solution optimale; elle est caractérisée par la complexité notée O .

Si le nombre d'itérations nécessaires pour obtenir une solution optimale est décrit par $3n^2+8n+10$, l'algorithme est dit en $O(n^2)$, ce qui caractérise une complexité polynomiale en la taille du problème. En revanche pour un nombre d'itérations en $O(2^n)$, on parle de complexité exponentielle.

Un algorithme est considéré comme « efficace » si, et seulement si, il est polynomial.

Dans la théorie de la complexité, on fait une distinction entre problèmes d'optimisation et problèmes de décision. Dans le dernier cas, la réponse attendue à une question posée est **oui** ou **non**. A chaque problème d'optimisation, on peut associer un problème de décision ; le problème de décision associé à la recherche d'un ordonnancement de durée minimale consiste à statuer sur l'existence d'un ordonnancement de durée inférieur à une valeur donnée. Si l'on dispose d'un algorithme en temps polynomial pour le problème d'optimisation, alors on dispose d'un algorithme en temps polynomial pour le problème de décision associé, et vice versa.

1.8.1 Les classes P et NP

La classe P se compose de tous les problèmes pour lesquels des algorithmes en temps polynomial ont été trouvés. La classe NP est l'ensemble des problèmes pour les-

quels des algorithmes de complexité exponentielle ont été trouvés. On dit que les problèmes de la classe P sont « faciles ».

Il est clair que NP contient P, et dans ce contexte, il est utile de mentionner la fameuse conjecture $P \neq NP$, autrement dit, il existe des problèmes qui ne possèdent pas des algorithmes de complexité polynomiale pour les résoudre, ou la classe NP contient des problèmes qui sont « plus difficiles » que ceux de la classe P.

1.8.2 Réduction polynomiale et problème NP-complet

On dit qu'un problème Π_1 se réduit polynomialement à un problème Π_2 et on écrit $\Pi_1 \leq \Pi_2$ s'il existe un algorithme polynomial **A** construisant, à partir d'une donnée D_1 de Π_1 , une donnée D_2 de Π_2 telle que la réponse pour D_1 soit **oui** si et seulement si la réponse pour D_2 est **oui**.

On dit qu'un problème Π dans NP est NP-complet si tout autre problème Π' dans NP se réduit polynomialement à Π : $\Pi' \leq \Pi$ pour tout Π' dans NP.

La propriété de « NP-complétude » est très forte puisqu'elle signifie, que si on avait un algorithme polynomial pour un problème NP-complet, alors on aurait un algorithme polynomial pour tous les problèmes de la classe NP. C'est-à-dire qu'on saurait répondre (positivement) à la conjecture $P = NP$.

La classe des problèmes NP peut donc se partitionner en 3 sous-classes, les problèmes de la classe P, les problèmes NP-complet et les autres. Evidemment, si la conjecture $P \neq NP$ est fausse, ces deux dernières sous-classes sont vides. On peut donc dire que les problèmes NP-complet sont « les plus difficiles » de la classe NP (si toutefois $P \neq NP$).

1.8.3 La réduction de Turing et les problèmes NP- difficiles

On dit qu'un problème de recherche Π_1 se réduit polynomialement à un problème de recherche Π_2 par la réduction de Turing s'il existe pour résoudre Π_1 un algorithme A_1 utilisant comme sous programme un algorithme A_2 résolvant Π_2 , de telle sorte que la complexité de A_1 est polynomiale, quand on évalue arbitrairement chaque appel de A_2 par une constante. Un problème est NP-difficile s'il est plus difficile qu'un problème NP-complet, c'est-à-dire s'il existe un problème NP-complet se réduisant à ce problème par une réduction de Turing.

CHAPITRE 2

LES PROBLEMES DE TYPE

«FLOW SHOP »

2.1 Les problèmes de type flow-shop

Dans un problème de "flow-shop", tout travail ou tâche visite chaque machine de l'atelier et l'ordre de passage d'un travail sur les différentes machines est le même pour toutes les tâches (flot unidirectionnel). Cet ordre ou gamme unique est une donnée du problème, le tableau suivant présente un flow-shop de n tâches et m machines

Tâche	Ordre d'exécution sur les machines				
J_1	M_1	M_2	M_3	$\dots$	M_m
J_1	M_1	M_2	M_3	$\dots$	M_m
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
J_n	M_1	M_2	M_3	$\dots$	M_m

Note

Dans tous ce qui suit, on suppose que les tâche sont disponibles à l'instant 0.

Théorème 2.1

Pour un problème $n/m/F/B$ avec B un critère régulier, il suffit de considérer les ordonnancements dans lesquels la même séquence est donnée pour les deux premières machines.

Preuve

Si un ordonnancement S_1 n'a pas le même ordre sur les deux premières machines M_1 et M_2 , il y' aurait une tâche I qui précède directement une tâche K sur M_1 , mais elle suit K , peut-être, après d'autres tâches intermédiaires sur M_2 .

M_1		I	K	
M_2			K	I

Sur M_1 , on pourrait renverser l'ordre de I et K sans augmentation des dates de début des tâches sur M_2 . Donc cet inter-changement ne peut pas augmenter la date de fin d'aucune tâche, et par suit la mesure régulière B ne peut pas augmenter.

Cette opération d'échange des tâches sur M_1 pourrait être répétée jusqu'à l'obtention d'un ordonnancement S'_1 avec le même ordre sur M_1 et M_2 . Clairement, S'_1 n'est pas mauvais que S_1 sur une mesure régulière B .

Théorème 2.2

Pour les problèmes $n/m/F/C_{\max}$, on a pas besoin de considérer les ordonnancements avec un ordre différent sur les machines M_{m-1} et M_m .

Preuve

Si un ordonnancement S_2 n'a pas le même ordonnancement sur les machines M_{m-1} et M_m , il y'aurait une tâche I qui précède directement une tâche K sur M_{m-1} , mais suit K , peut être, après d'autres tâches intermédiaires sur M_m .

M_{m-1}		I	K	
M_m			K	
				I

L'inter-changement des tâches I et K sur M_m pourrait modifier les dates de séjour de certaines tâches. Cependant, le $C_{\max}$ ne peut pas augmenter.

On répète cette opération d'inter-changement des tâches jusqu'à l'obtention d'un ordonnancement S'_2 avec le même ordre sur M_{m-1} et M_m . Cet ordonnancement S'_2 , ainsi trouvé, n'est pas mauvais que S_2 en terme de $C_{\max}$.

A cause du premier théorème, on se contente de considérer seulement, les ordonnancements de permutations pour le problème $n/2/F/B$, avec

B régulier. Par la combinaison avec le résultat du deuxième théorème, encore on peut considérer, seulement les ordonnancements de permutations pour le problème $n/3/F/C_{\max}$.

2.1.1 Algorithme de Johnson pour le problème $n/2/F/C_{\max}$

Supposons qu'on est en face au problème $n/2/F/C_{\max}$; i. e. on voudrait ordonnancer n tâches sur deux machines, chaque tâche dans l'ordre M_1, M_2 , et pour but de minimiser le critère $C_{\max}$.

Pour résoudre ce problème, l'idée fondamentale de l'algorithme de Johnson, est de créer une permutation de $\{J_1, J_2, \dots, J_n\}$ (avec $J_i, i=1 \dots n$, représente les tâches), de telle sorte que les premières tâches dans cette séquence, ont des durées d'exécution, ou opératoires, courte sur la machine M_1 , alors que les dernières tâches dans la séquence ont des durées courtes sur la machine M_2 . La permutation ainsi créée, serait l'ordonnancement optimale.

Nous donnons maintenant, l'algorithme de Johnson en mettant les notations suivantes:

a_i : la durée opératoire de J_i sur M_1

b_i : la durée opératoire de J_i sur M_2

Algorithme (Johnson):

Etape 1: prendre $k = 1, l = n$.

Etape 2 : prendre la liste courante des tâches non ordonnancées $\{J_1, J_2, \dots, J_n\}$.

Etape 3 : trouver le plus petit de tous les a_i et b_i pour les tâches couramment non-ordonnancées.

Etape 4 : si le plus petit des a_i et b_i est un a_i d'une J_i , alors:

- i) Ordonnancer J_i dans la k^{eme} position de la séquence.
- ii) Supprimer J_i de la liste courante des tâches non ordonnancées.
- iii) Incrémenter k à $k+1$.
- iv) Aller à l'étape 6.

Etape 5 : si le plus petit des a_i et b_i est un b_i , alors:

- i) Ordonnancer J_i dans la l^{eme} position de la séquence.

- ii) Supprimer J_i de la liste courante des tâches non ordonnancées.
- iii) Réduire l à $l - 1$.
- iv) Aller à l'étape 6.

Etape 6 : S'il existe encore une tâche non ordonnancée, aller à l'étape 3. Sinon, arrêter.

Remarque

Si on a plus d'une tâche, possédant le même plus petit des éléments, à l'étape 3, sélectionner J_i arbitrairement.

Exemple

Appliquons l'algorithme de Johnson au problème $7/2/F/C_{\max}$ avec les données suivante:

Tâche	Durée opératoire sur la machine	
	M_1	M_2
1	6	3
2	2	9
3	4	3
4	1	8
5	7	1
6	4	5
7	7	6

En suivant les étapes de l'algorithme, on trouve ceci:

Tâche 4 ordonnancée : 4 — — — — —

Tâche 5 ordonnancée : 4 — — — — — 5

Tâche 2 ordonnancée : 4 2 — — — — — 5

Tâche 3 ordonnancée : 4 2 — — — 3 5

Tâche 1 ordonnancée : 4 2 — — 1 3 5

Tâche 6 ordonnancée : 4 2 6 — 1 3 5

Tâche 7 ordonnancée : 4 2 6 7 1 3 5

Donc on devrait ordonnancer les tâches selon la séquence (4,2,6,7,1,3,5). Notons que dans cet exemple, on a eu deux choix arbitraires. On aurait pu mettre la tâche 5 dans la dernière position de la séquence avant

l'ordonnancement de la tâche 4. Ici la séquence résultant serait la même. Aussi, On aurait pu ordonnancer la tâche 1 dans la sixième position au lieu de la tâche 3, ceci donnerait une séquence différente, mais équivalente (4,2,6,7,3,1,5).

Théorème 2.3

L'algorithme de Johnson trouve un ordonnancement optimal pour le problème $n/2/F/C_{\max}$

Preuve

La preuve se base sur la proposition suivante :

Si $a_k = \min\{a_1, a_3, \dots, a_n, b_1, b_2, \dots, b_n\}$, il existe un ordonnancement optimal avec J_k première dans la séquence.

Si $b_k = \min\{a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n\}$, il existe un ordonnancement optimal avec J_k dernière dans la séquence.

Cette proposition nous montre que la première cycle de l'algorithme positionne une tâche de façon optimale.

La preuve du théorème 2.3 se fait par induction. Supposons que les premières ν cycles ont positionné ν tâches de façon optimale. A cause de ce fait le $(\nu+1)$ ieme cycles ne prend pas en compte ces tâches prises dans les cycles précédentes, mais il choisirait une de $(n-\nu)$ tâches restantes et le positionne dans une des $(n-\nu)$ places restantes dans la séquence. Le positionnement de la $(\nu+1)$ ieme tâche minimise le $C_{\max}$ pour les $(n-\nu)$ tâches considérées seulement, et l'algorithme minimise le $C_{\max}$ pour toutes les n tâches. Alors, le $(\nu+1)$ ieme cycle positionne une tâche de façon optimale.

Donc, tous les cycles positionnent les tâches de façon optimal, et par suite, l'algorithme construit un ordonnancement optimal.

2.1.2 Un cas particulier pour le problème $n/3/F/C_{\max}$

L'algorithme de Johnson pour le problème $n/2/F/C_{\max}$ peut être étendu à un cas particulier du problème $n/3/F/C_{\max}$, ce cas particulier impose l'une des deux conditions suivantes:

$$\min_{i=1}^n \{p_{i1}\} \geq \max_{i=1}^n \{p_{i2}\} \quad (1)$$

ou
$$\min_{i=1}^n \{p_{i3}\} \geq \max_{i=1}^n \{p_{i2}\}$$

avec p_{ij} ($j=1,2,3$) est la durée opératoire de la tâche i sur la machine j .

si (1) ou (2) est vérifié, un ordonnancement optimal peut être trouvé en mettant:

$$a_i = p_{i1} + p_{i2},$$

$$b_i = p_{i2} + p_{i3},$$

et en ordonnant les tâches par la considération de deux machines seulement, mais cette fois-ci, avec pour toute tâche des durées opératoires a_i et b_i sur la première et deuxième machines respectivement. Cet ordonnancement se fait par l'algorithme de Johnson.

Exemple

Considérer le problème $6/3/F/C_{\max}$ avec les données suivantes:

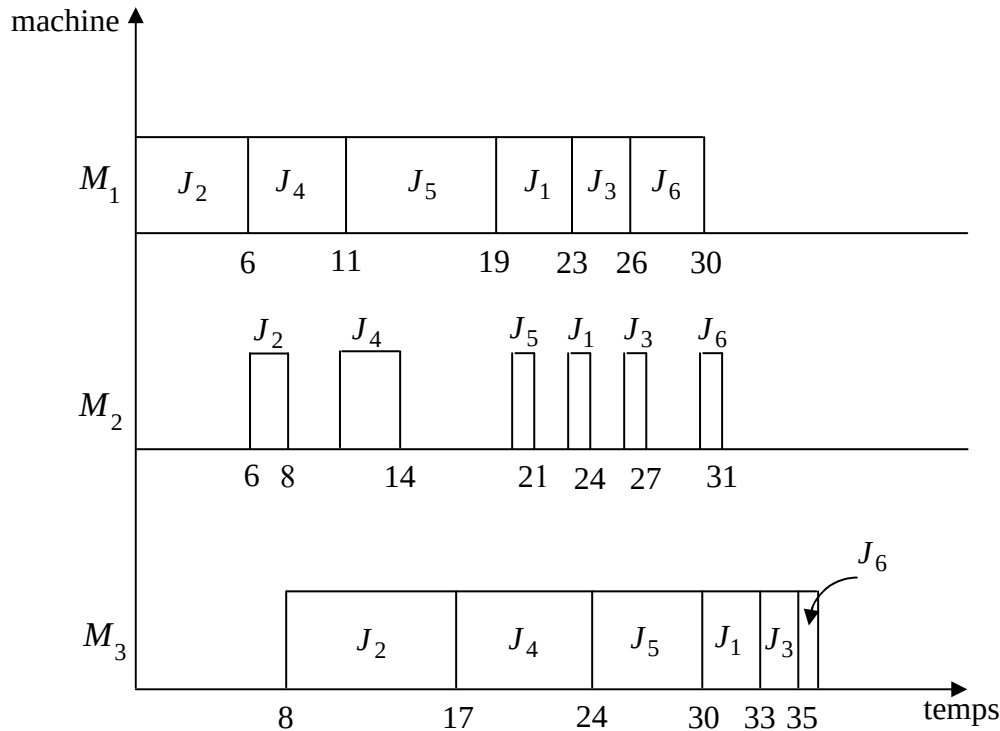
Tâche	Durées opératoires actuelles			Durées opératoires construites	
	M ₁	M ₂	M ₃	1 ^{ère} machine	2 ^{ère} machine
1	4	1	3	5	4
2	6	2	9	8	11
3	3	1	2	4	3
4	5	3	7	8	10
5	8	2	6	10	8
6	4	1	1	5	2

On a

$$\min_{i=1}^6 \{p_{i1}\} = 3 ; \max_{i=1}^6 \{p_{i2}\} = 3 ; \text{ et } \min_{i=1}^6 \{p_{i3}\} = 1$$

$$\min_{i=1}^6 \{p_{i1}\} = 3 \geq \max_{i=1}^6 \{p_{i2}\} \text{ donc (1) est vérifié.}$$

Les durées construites a_i et b_i sont données sur le tableau. En appliquant l'algorithme de Johnson, on obtient la séquence (2,4,5,1,3,6), qui a le diagramme de Gantt suivant :



2.1.3 Solution graphique de Aker pour le problème $2/m/F/C_{\max}$

Cette méthode due à Aker [AKE 56] tente de trouver graphiquement un ordonnancement optimal du problème $2/m/F/C_{\max}$ en représentant le problème sur un graphe cartésien dont un axe porte les durées opératoires de la première tâche, et l'autre pour la deuxième.

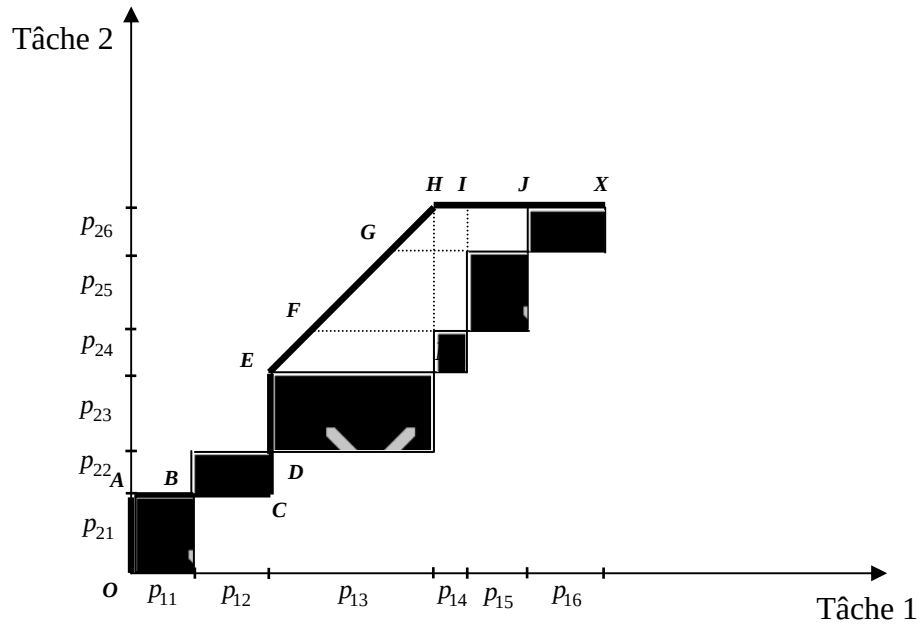
Un ordonnancement y représenté par une ligne parcourant de $O=(0,0)$ à

$X=(\sum_{j=1}^m p_{1j}, \sum_{j=1}^m p_{2j})$ composé des segments pouvant parcourir horizontalement

indiquant le travail sur la tâche 1 seulement, où verticalement indiquant le travail sur la tâche 2 seulement, où à 45 degré indiquant le travail sur les tâches 1 et 2 toutes les deux. Ces segments ne doivent pas couper les surfaces

hachurées, expliqué par le fait qu'une machine ne peut pas exécuter deux tâches simultanément.

Illustrons ça sur l'exemple $2/6/F/C_{\max}$.



Trouver un ordonnancement optimal revient à construire un ordonnancement ayant le minimum longueur des segments verticaux, ou de façon équivalente, le minimum longueur des segments horizontaux.

Les algorithmes discutés précédemment, sont des méthodes exactes, c'est à dire, elles conduisent à des solutions optimales, mais aux cas particuliers des problèmes et avec certaines conditions. Pour le cas général $n/m/F/C_{\max}$ ($m \geq 3$), le problème devient NP-complet. Des méthodes, trouvées dans la littérature, appelées Heuristiques, donnent des solutions approchées à ce problème général.

Avant d'introduire ces heuristiques, abordons tout d'abord le concept des méthodes heuristiques.

2.1.3 Méthodes heuristiques

On rencontre souvent, dans le domaine de l'ordonnancement, des problèmes qui sont NP-difficile dont on ne pouvait pas trouver la solution optimale en temps polynomial. En pratique, il arrive qu'on ne puisse pas laisser un problème NP-difficile sans résolution car il est très lié à la réalité. Si on ne peut pas trouver un bon ordonnancement pour un tel problème, parmi plusieurs qu'on a, sûrement, on ne pouvait pas abandonner la recherche et choisir au hasard. On devrait, plutôt, utiliser ses connaissances et ses expériences pour trouver un ordonnancement qui, s'il n'est pas optimal, pourrait au moins, satisfaire mieux qu'en moyenne. L'algorithme donnant un tel ordonnancement s'appelle une heuristique.

Mais la question qui se pose : quel genre de performance peut-on garantir en utilisant une telle heuristique?.

Exemple illustratif

Considérons, par exemple le problème d'ordonnancement de n tâches indépendantes $\{J_1, J_2, \dots, J_n\}$ sur m machines identiques $\{M_1, M_2, \dots, M_m\}$. Chaque tâche demande une seule opération exécutable sur n'importe quelle machine. La durée p_i de la tâche i est la même quelque soit la machine utilisée, les tâches sont exécutables dans n'importe quel ordre et sont disponibles à l'instant 0. L'objectif de ce problème est de minimiser le $C_{\max}$.

Ce problème est NP-complet pour $m > 1$. On se ramène à y considérer des méthodes heuristiques. Avant cela, donnons des définitions y nécessaires :

Instance

Appelons une instance I du problème, la spécification des données $n, m, p_1, p_2, \dots, p_n$. On définit $Opt(I)$ le $C_{\max}$ donné par un ordonnancement optimal pour l'instance I , et $A(I)$ le $C_{\max}$ donné par une heuristique A pour l'instance I .

Rapport de la performance relative

Appelons le rapport $R_A(I) = \frac{A(I)}{opt(I)}$ la performance relative de A . Notons que

$R_A(I) \geq 1$ pour tout A et I .

Prenons comme heuristique à ce problème l'algorithme de liste. Cette méthode consiste à lister les tâches dans un ordre quelconque, et les affecter selon cet ordre aux machines dès qu'elles soient libres.

On a le théorème suivant :

Théorème 2.4 (Graham) :

Si A est l'algorithme de liste, alors :

$$R_A(I) \leq 2 - \frac{1}{m} \text{ pour toute instance } I.$$

Appelons la borne fournie par ce théorème la garantie de performance de A.

Après avoir présenter l'heuristique de liste A, on se demande s'il trouve d'autres heuristiques donnant une garantie meilleure que A. pour cela, supposons l'algorithme LPT (longest processing time). Cette méthode liste les tâches selon l'ordre décroissant des durées p_i .

On a le théorème suivant :

Théorème 2.5 (Graham)

Si A est l'algorithme LPT, alors

$$R_A(I) \leq \frac{4}{3} - \frac{1}{3m} \text{ pour toute instance.}$$

Et le processus d'amélioration de la garantie de performance pourrait ainsi, se poursuivre.

2.1.4 Heuristique pour le problème général $n/m/F/C_{max}$

Dans cette section, on aborde quelques heuristiques, qui ont été conçues pour construire un solution aussi proche de l'optimale concernant le problème général $n/m/F/C_{max}$.

Notons que toute ces méthodes mènent a des ordonnancements de permutation.

2.1.4.1 Heuristique de Palmer

L'idée de cette heuristique est basée sur le concept du « slope index » qui donne sa grande valeur aux tâches qui ont une forte tendance de progresser de courtes ou longues durées opératoires en passant d'une machine à l'autre.

Donc, quand les tâches sont ordonnancées en ordre décroissant selon le « slope index », on pourrait espérer trouver un ordonnancement proche de l'optimal.

Pour le problème général $n/m/F/C_{\max}$, son « slope index » pour J_i est :

$$S_i = -(m-1) p_{i1} - (m-3) p_{i2} - \dots + (m-3) p_{i(m-1)} + (m-1) p_{im} \\ = -\sum_{j=1}^m [m - (2j-1)] p_{ij}$$

En prenant $m=7$, on obtient :

$$S_i = -6 p_{i1} - 4 p_{i2} - 2 p_{i3} + 0 p_{i4} + 2 p_{i5} + 4 p_{i6} + 6 p_{i7}$$

on illustre cette méthode sur l'exemple $4/3/F/C_{\max}$ avec les données suivantes :

Tâche	Durées opératoires		
	p_{i1}	p_{i2}	p_{i3}
1	1	8	4
2	2	4	5
3	6	2	8
4	3	9	2

Ici on trouve les « slope indices » :

$$S_1 = -2 \times 1 + 0 \times 8 + 2 \times 4 = 6.$$

$$S_2 = -2 \times 2 + 0 \times 4 + 2 \times 5 = 6.$$

$$S_3 = -2 \times 6 + 0 \times 2 + 2 \times 8 = 4.$$

$$S_4 = -2 \times 3 + 0 \times 9 + 2 \times 2 = -2.$$

Donc cette heuristique propose la séquence (1,2,3,4) ou (2,1,3,4). Pour décider laquelle à prendre, on évalue le $C_{\max}$ pour les deux cas, et on choisit le mieux.

2.1.4.2 Heuristique CDS

L'heuristique de Campbell, Dudek et Smith (CDS) utilise une application multiple de l'algorithme de Johnson à deux machines pour tenter d'obtenir un bon ordonnancement, de la manière suivante :

On crée $(m-1)$ problèmes fictifs à deux machines, puis on applique l'algorithme de Johnson à chaque problème, et on sélectionne la meilleure solution. Notons que chaque problème fictif $k(k=1,2,...,(m-1))$ est défini comme suivant :

$$a_i^{(k)} = \sum_{j=1}^k p_{ij} \text{ durée opératoire de la tâche } i \text{ sur la première machine.}$$

$$b_i^{(k)} = \sum_{j=m-k+1}^m p_{ij} \text{ durée opératoire de la tâche } i \text{ sur la deuxième machine.}$$

Appliquons cet algorithme sur notre exemple précédent, on construit deux problèmes fictifs à deux machines correspondants à $k=1$ et $k=2$.

Tâche J_i	durées opératoires $p_{i1} \quad p_{i2} \quad p_{i3}$			problème fictif $k=1$ $a_i^{(1)} = p_{i1} \quad b_i^{(1)} = p_{i3}$		problème fictif $k=2$ $a_i^{(2)} = p_{i1} + p_{i2} \quad b_i^{(2)} = p_{i2} + p_{i3}$	
1	1	8	4	1	4	9	12
2	2	4	5	2	5	6	9
3	6	2	8	6	8	8	10
4	3	9	2	3	2	12	11

L'application de l'algorithme de Johnson donne la séquence (1,2,3,4) pour le problème $k=1$ et (2,3,1,4) pour le problème $k=2$.
pour la première $C_{\max}=28$, tandis que $C_{\max} \geq 29$ pour la deuxième. Donc la méthode devrait choisir la séquence (1,2,3,4), qui est le même résultat obtenu par l'heuristique de Palmer.

2.1.4.3 Heuristique de Dannebring

Cette heuristique tente de combiner les avantages des deux heuristiques précédentes. Son idée est de construire seulement un problème à 2machines, sur lequel on applique l'algorithme de Johnson, mais cette fois-ci avec des durées opératoires reflétant le même comportement que « slope index » de Palmer.

La durée opératoire construite pour la première machine :

$$a_i = mp_{i1} + (m-1)p_{i2} + \dots + 1p_{im}$$

$$= \sum_{j=1}^m (m-j+1)p_{ij};$$

et pour la deuxième machine :

$$b_i = p_{i1} + 2p_{i2} + \dots + mp_{im}$$

$$= \sum_{j=1}^m jp_{ij};$$

Appliquons cette méthode, toujours sur le même exemple

Tâche J_i	durées opératoires			durées opératoires construites	
	p_{i1}	p_{i2}	p_{i3}	$a_i = 3p_{i1} + 2p_{i2} + 3p_{i3}$	$b_i = p_{i1} + 2p_{i2} + 3p_{i3}$
1	1	8	4	23	29
2	2	4	5	19	25
3	6	2	8	30	34
4	3	9	2	29	27

L'application de l'algorithme de Johnson nous donne la séquence (2,1,3,4)
avec $C_{\max} = 28$.

CHAPITRE 3

CAS SPECIFIQUES DES

PROBLEMES «FLOW SHOP » ET LE

PROBLEME «JOB SHOP »

3.1 Cas spécifiques des problèmes flow shop

Les problèmes de type flow shop peuvent varier selon leurs caractéristiques. On mentionne ici trois cas spécifiques intéressants des problèmes de type flow shop qui sont le flow shop avec lags entre tâches, flow shop avec relations de précédence et le flow shop hybride. Commençons par :

3.1.1 Flow Shop avec lags entre tâches

Jackson [JAC 56] et Mitten [MIT 59] ont étendu la règle de Jonson au cas du problème de flow-shop de permutation à deux machines avec lags entre tâches : F_2 / l_j , permut / $C_{\max}$. Dans ce problème, un lags l_j représente la différence minimale (éventuellement négative) séparant le début de l'exécution de la tâche J_j sur la seconde machine de la fin de son exécution sur la première machine.

Pour résoudre F_2 / l_j , permut / $C_{\max}$, nous définissons une relation transitive entre tâche $\leq$ par :

$$J_i \leq J_j \Leftrightarrow \min(p_{i,1} + l_i ; l_j + p_{j,2}) \leq \min(l_i + p_{i,2} ; p_{j,1} + l_j)$$

Théorème 2.6

Soit P un problème du flow shop de permutation à deux machines avec lags entre tâches. Si $J_i \leq J_j$, alors il existe un ordonnancement optimal pour P dont la tâche J_i précède la tâche J_j .

Une caractéristique fort intéressante du problème F_2 / l_j , permut / $C_{\max}$ est la possibilité de substituer une tâche dite composite à toute séquence ininterrompue de tâches, et ce, sans modifier la durée de l'ordonnancement résultant, avec la tâche composite $\langle J_i, J_j \rangle$ constitué par les tâches J_i et J_j vaut :

- $p_{\langle i,j \rangle,1} = p_{i,1} + p_{j,1}$
- $l_{\langle i,j \rangle} = \max(l_i + p_{i,2} ; p_{j,1} + l_j) - p_{i,2} - p_{j,1}$
- $p_{\langle i,j \rangle,2} = p_{i,2} + p_{j,2}$

de façon plus précise on a :

Théorème 2.7

Soit P un problème F_2 / l_j , permut / $C_{\max}$. Supposons que $\Pi = (\Pi_1, \dots, \Pi_h, \dots, \Pi_k, \dots, \Pi_n)$ est une permutation de tâches, et soit $\sigma = \langle \Pi_h, \dots, \Pi_k \rangle$ une tâche composite. Soit $\Pi' = (\Pi_1, \dots, \Pi_{h-1}, \sigma, \Pi_{k+1}, \dots, \Pi_n)$. Alors les dates de fin de Π_k et de σ sur les machines M_1 et M_2 sont identiques.

Notons que l'utilisation des tâches composites demeure vraie dans le contexte du flow shop classique à deux machines, car on sait que le problème $n/2/F/C_{\max}$ (ou simplement $F_2 // C_{\max}$ admet une solution qui un ordonnancement de permutation.

Malheureusement, ce résultat ne peut pas être étendu au flow shop classique avec lags, $F_2 / l_j / C_{\max}$ est NP-complet.

3.1.2 Flow shop avec relations de précedence

On définit les relations de précedence entre deux opérations, et deux tâches de la manière suivante:

$$(O_{i,k} \rightarrow O_{j,k}) \Rightarrow t_{i,k} + p_{i,k} \leq t_{j,k} ; (J_i \rightarrow J_j) \Rightarrow \forall k, (O_{i,k} \rightarrow O_{j,k})$$

avec $p_{i,k}$ est la durée opératoire de $O_{i,k}$, et $t_{i,k}$ désigne le début d'exécution de la tâche J_i sur la machine M_k .

On a le résultat important suivant:

Lemme 2.8

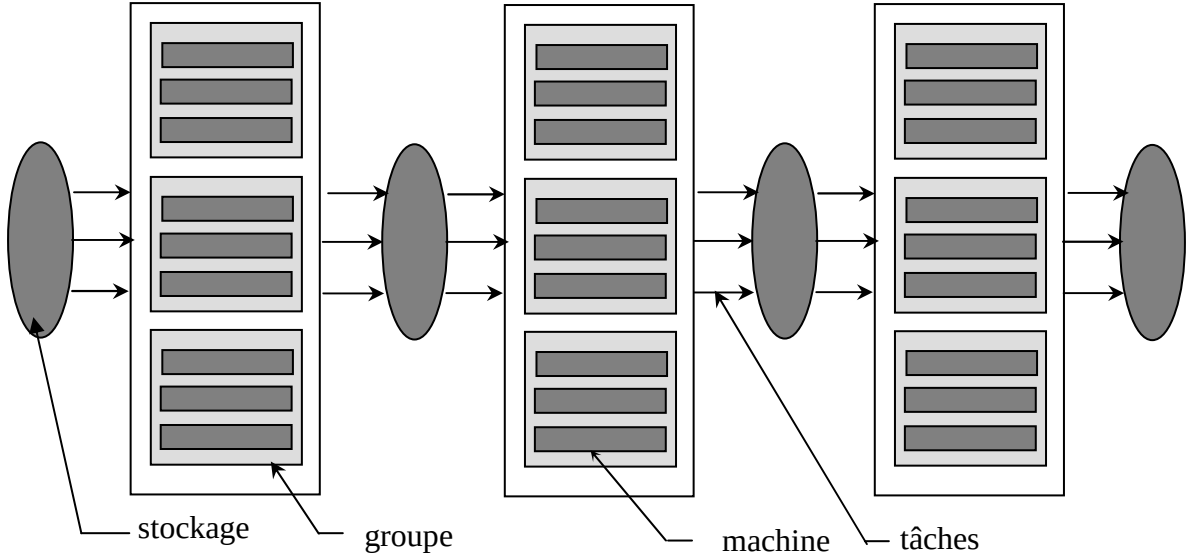
Soit P un problème de flow-shop à deux machines avec relations de précedence entre opérations sur la première machine. Il existe une solution optimale qui un ordonnancement de permutation.

Notons dans ce contexte, que le problème flow-shop à deux machines avec relations générales de précedence $F_2 / prec / C_{\max}$ a été prouvé NP-complet.

3.1.3 Le flow shop hybride HFS

Le multiprocesseur flow-shop ou flow shop hybride (HFS) consiste du flow shop à plusieurs phases (multi-phases), où chaque phase $s \in \{1, \dots, K\}$, $K \geq 2$, est composé de M_s machines parallèles et identiques. Chaque machine ne peut exécuter qu'une seule tâche à la fois. Un flow shop est considéré comme flow

shop hybride si chaque phase est constituée d'une ou plusieurs machines. La figure suivante représente un flow shop hybride à trois phases.



Notre objectif consiste à ordonnancer un ensemble de tâches $I=\{1,...,n\}$, où une tâche est composé d'une seul opération pour chaque phase, et les opérations doivent être réalisées séquentiellement.

Chaque tâche i a une durée opératoire p_{is} sur chaque phase s , $i \in I$, et $s \in \{1,...,K\}$ Chaque tache pourrait avoir une date de disponibilité initiale r_{i1} à la première phase et une queue finale q_{iK} à la dernière phase K

De façon plus générale, pour chaque phase $s \in \{1,...,K\}$ et chaque tâche $i \in I$,

- La date de disponibilité r_{is} représente la période (heure, jour,...) dont à la fin, une tâche i est disponible à être exécutée à la phase s . La tâche i peut donc s'exécuter à la phase s au début de période $r_{is} + 1$.
- La queue q_{is} désigne la quantité minimale du temps qu'une tâche, après avoir exécutée à la phase s , puisse prendre en restant dans l'atelier avant le makespan (la date de fin du projet).

On devrait construire un ordonnancement avec un makespan minimum, avec le makespan μ définit par $\mu = \max[t_{iK} + p_{iK} + q_{iK} - 1]$, avec $t_{iK} > r_{iK}$ le

temps de début de la dernière (K^{ieme}) opération de la tâche i qui se termine donc au $t_{iK} + p_{iK} - 1$.

Un ordonnancement consiste à affecter une machine spécifique à chaque opération d'une tâche ainsi que la détermination de la séquence de toutes les opérations affectées à chaque machine, telles que les opérations d'une tâche ne se chevauchent pas et chaque machine exécute au plus une tâche à la fois.

Du fait que le problème d'une seule phase avec des tâches ayant des dates de disponibilité et queues, est NP-difficile [CAR 87], et que même le problème à une machine avec dates de disponibilités et queue [LEN 76], on déduit que notre problème général HFS est NP-difficile.

3.2 Les problèmes de type Job-shop

Dans un Job-shop, chaque travail ou tâche passe sur les mêmes machines dans un ordre fixé, mais à la différence du flow-shop, cet ordre peut être différent pour chaque travail (flot multi-directionnel)

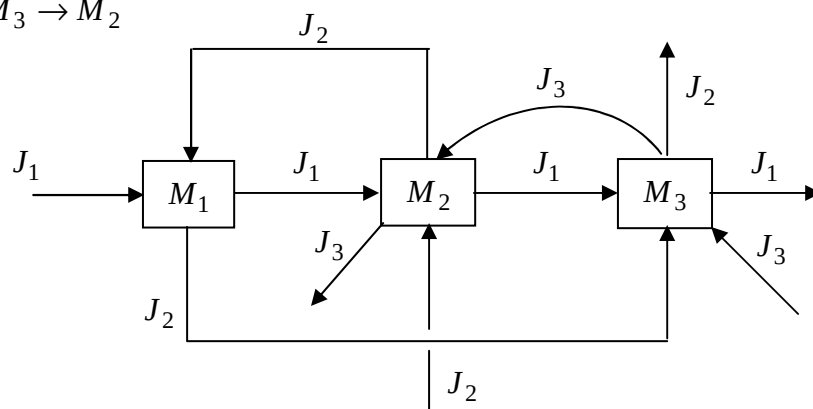
Exemple

La figure suivante présente un job-shop à trois machines et trois travaux (ou tâches), les gammes de chacun d'eux étant:

$$J_1 : M_1 \rightarrow M_2 \rightarrow M_3$$

$$J_2 : M_2 \rightarrow M_1 \rightarrow M_3$$

$$J_3 : M_3 \rightarrow M_2$$



Le cas du Job-shop se rencontre dès qu'un atelier fabrique des pièces présentant des caractéristiques différentes, mais nécessitant néanmoins des traitements dont les séquences sont connues à l'avance.

3.2.1 Algorithme de Johnson pour le problème $n/2/G/C_{\max}$

Supposons que l'ensemble de n tâches $\{J_1, J_2, \dots, J_n\}$ soit partitionnées en quatre types de tâches comme suivant:

Type A: pour celles qui passent sur la machine M_1 seulement.

Type B: pour celles qui passent sur la machine M_2 seulement.

Type C: pour celles qui passent sur les deux machines dans l'ordre M_1 puis M_2 .

Type D: pour celles qui passent sur les deux machines dans l'ordre M_2 puis M_1 .

Les étapes de cet algorithme sont :

- 1) Ordonnancer les tâches de type A dans un ordre quelconque pour donner la séquence S_A .
- 2) Ordonnancer les tâches de type B dans un ordre quelconque pour donner la séquence S_B .
- 3) Ordonnancer les tâches de type C selon l'algorithme de Johnson pour $n/2/F/C_{\max}$ pour donner la séquence S_C .
- 4) Ordonnancer les tâches de type D selon l'algorithme de Johnson pour $n/2/F/C_{\max}$ pour donner la séquence S_D .

(ici on considère M_2 comme première machine et M_1 la deuxième).

Un ordonnancement optimal est donc

Machine	ordre
M_1	(S_C, S_A, S_D)
M_2	(S_D, S_B, S_C)

Exemple

Considérons le problème $9/2/G/C_{\max}$ avec les données suivantes:

Tâche	Ordre et durées			
	1 ^{ère} machine		2 ^{ème} machine	
1	M_1	8	M_2	2
2	M_1	7	M_2	5
3	M_1	9	M_2	8
4	M_1	4	M_2	7
5	M_2	6	M_1	4
6	M_2	5	M_1	3
7	M_1	9	/	
8	M_2	1	/	
9	M_2	5	/	

Pour trouver un ordonnancement optimal:

Type A : uniquement la tâche 7 qui passe sur M_1 seulement.

Type B: tâche 8 et 9 demandent M_2 seulement. Sélectionner arbitrairement l'ordre (8,9)

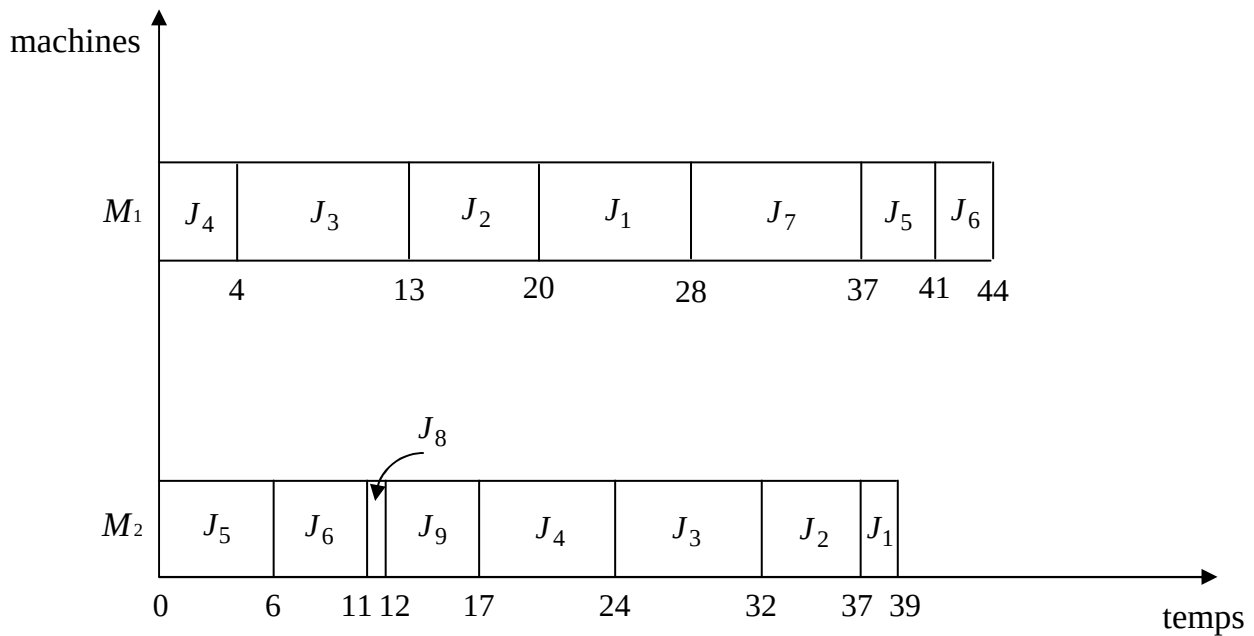
Type C: tâches 1,2,3 et 4 demandent M_1 d'abord puis M_2 . L'algorithme de Johnson pour ce problème $4/2/F/C_{\max}$ donne la séquence (4,3,2,1).

Type D: tâches 5,6 demandent M_2 d'abord puis M_1 . L'algorithme de Johnson pour ce problème $2/2/F/C_{\max}$ donne la séquence (5,6).

Donc la séquence optimale pour notre problème $9/2/G/C_{\max}$ est:

	séquence
Machine M_1	(4,3,2,1,7,5,6)
Machine M_2	(5,6,8,9,4,3,2,1)

Avec le diagramme de Gantt suivant :



Donc on obtient $C_{\max} = 44$ pour un ordonnancement optimal.

Il est très utile de terminer ce chapitre par des résultats concernant la complexité d'autre problème de flow-shop et job-shop :

Pour le flow-shop:

les problèmes $n/2/F/\sum w_i c_i$ (w_i représente le poids de tâche J_i), $n/2/F/\sum T_i$ et $n/2/F/L_{\max} = \max(c_i - d_i)$ sont NP-complet [RIN 76].

Pour le Job-shop:

les problèmes $n/2/G$, $n_i \leq 3/C_{\max}$, $n/3/G$, $n_i \leq 2/C_{\max}$ (avec $n_i = |J_i|$: nombre d'opération de J_i), sont NP-complet [RIN 76].

CHAPITRE 4

HEURISTIQUES, PROPOSEES ET

PROGRAMMEES, POUR LE

PROBLEME GENERAL

$n/m/F/C_{\max}$

Nous présentons dans ce chapitre deux heuristiques, qui sont basées sur la règle de Johnson donnée auparavant, pour la résolution de problème général $n/m/F/C_{\max}$, puis nous les programmons par le langage Delphi, et enfin nous les testerons sur des exemples divers de 3,4,5,6 et 7 machines, en faisant une comparaison avec l'heuristique CDS présentée dans le chapitre précédant, qui est également programmée.

On commence donc par la présentation des heuristiques proposées notées A et B.

4.1 Heuristique A

A l'instar de la méthode CDS, cet heuristique génère $(m-1)$ problèmes fictifs de deux machines à partir d'un problème général $n/m/F/C_{\max}$, mais à la différence de la manière de choisir les deux machines de chaque problème fictif. L'heuristique A procède comme ceci :

Etape 1 : Créer les $(m-1)$ problèmes fictifs :

Pour le problème fictif $\mathcal{P}_1$: on considère la première machine et le reste des machines qui constituent la deuxième machine avec $a_i^{(1)}, b_i^{(1)}$ les durées opératoires de la tâche i sur la première et la deuxième machine respectivement, dans ce problème $\mathcal{P}_1$ sont définies comme suivant :

$$\begin{aligned} a_i^{(1)} &= p_{i1} \\ b_i^{(1)} &= \sum_{j=2}^m p_{ij} \end{aligned}$$

avec p_{ij} est la durée opératoire de la tâche i sur la machine j .

Pour le problème fictif $\mathcal{P}_2$: on considère les deux premières machines comme une seule machine et le reste des machines qui constituent la deuxième machine avec les durées opératoires

$$\begin{aligned} a_i^{(2)} &= p_{i1} + p_{i2} \\ b_i^{(2)} &= \sum_{j=3}^m p_{ij} \end{aligned}$$

de façon générale, pour un problème fictif $\mathcal{P}_l$ avec $1 \leq l \leq m-1$, on a :

$$a_i^{(l)} = \sum_{j=1}^l p_{ij}$$

$$b_i^{(l)} = \sum_{j=l+1}^m p_{ij}$$

Etape 2: Appliquer la règle de Johnson à chaque problème fictif $\mathcal{P}_l$ et calculer le $C_{\max}^l$ du problème initial en utilisant la permutation l qui est l'optimale de $\mathcal{P}_l$.

Etape 3: Trouver $C_{\max}^* = \min_{l=1}^{m-1} C_{\max}^{(l)}$

Etape 4: Prendre $C_{\max}^*$ comme solution donnée par l'heuristique A.

Exemple

Considérons le problème $4/4/F/C_{\max}$ suivant :

	J_1	J_2	J_3	J_4
M_1	2	4	1	3
M_2	3	1	5	2
M_3	1	4	3	1
M_4	5	1	2	4

Appliquons l'heuristique A à ce problème :

On crée 3 problèmes fictifs :

Le problème fictif $\mathcal{P}_1$ est :

	J_1	J_2	J_3	J_4
M'_1	2	4	1	3
M'_2	9	6	10	7

L'application de la règle de Johnson à ce problème fictif nous donne la permutation 3-1-4-2 avec $C_{\max} = 21$

Le problème fictif $\mathcal{P}_2$ est :

	J_1	J_2	J_3	J_4
M_1''	5	5	6	5
M_2''	6	5	5	5

La permutation obtenue est 1-2-4-3 avec $C_{\max} = 21$

Le problème fictif $\mathcal{P}_3$ est :

	J_1	J_2	J_3	J_4
M_1'''	6	9	9	6
M_2'''	5	1	2	4

La permutation obtenue est 1-4-3-2 avec $C_{\max} = 20$

Donc finalement, l'heuristique A propose la permutation 1-4-3-2 avec $C_{\max} = 20$ comme solution.

4.2 Heuristique B

Cette heuristique, basée aussi sur la règle de Johnson, utilise le même principe de générer les $(m-1)$ problèmes fictifs à partir du problème initial donné $n/m/F/C_{\max}$, mais avec sa propre manière de définir les problèmes fictifs et ses machines associées. Nous donnons les étapes de cette heuristique:

Etape 1: Créer les $(m-1)$ problèmes fictifs de la façon suivante :

pour le problème fictif $\mathcal{P}_1$: on considère la première machine et la dernière machine, et donc les durées opératoires de chaque tâche sont :

$$a_i^{(1)} = p_{i1}$$

$$b_i^{(1)} = p_{im}$$

Pour le problème fictif $\mathcal{P}_2$: on considère la deuxième machine comme première machine, et la dernière avec la première comme deuxième machine. Les durées opératoires de $\mathcal{P}_2$ sont :

$$a_i^{(2)} = p_{i2}$$

$$b_i^{(2)} = p_{im} + p_{i1}$$

de façon générale, pour un problème fictif $\mathcal{P}_l$ avec $1 \leq l \leq m-1$ on a :

$$a_i^{(l)} = p_{il}$$

$$b_i^{(l)} = p_{im} + \sum_{j=1}^{l-1} p_{ij}$$

Etape 2: Appliquer la règle de Johnson à chaque problème fictif $\mathcal{P}_l$, et calculer le $C_{\max}^l$ du problème initial en utilisant la permutation l qui est l'optimale de $\mathcal{P}_l$.

Etape 3: Trouver $C_{\max}^* = \min_{l=1}^{m-1} C_{\max}^{(l)}$

Etape 4: Prendre $C_{\max}^*$ comme solution donnée par l'heuristique B.

Exemple

Reconsidérons le même problème $4/4/F/C_{\max}$:

	J_1	J_2	J_3	J_4
M_1	2	4	1	3
M_2	3	1	5	2
M_3	1	4	3	1
M_4	5	1	2	4

Appliquons l'heuristique B à ce problème :

On crée 3 problèmes fictifs :

Le problème fictif $\mathcal{P}_1$ est :

	J_1	J_2	J_3	J_4
M'_1	2	4	1	3
M'_2	5	1	2	4

La règle de Johnson appliquées à ce problème donne la permutation 3-1-4-2 avec $C_{\max}=21$

Le problème fictif $\mathcal{P}_2$ est :

	J_1	J_2	J_3	J_4
M''_1	3	1	5	2
M''_2	7	5	3	7

La permutation obtenue est 2-4-1-3 avec $C_{\max}=22$

Le problème fictif $\mathcal{P}_3$ est :

	J_1	J_2	J_3	J_4
M'''_1	1	4	3	1
M'''_2	10	6	8	9

La permutation obtenue est 1-4-3-2 avec $C_{\max}=20$

Finalement, l'heuristique B prend la permutation 1-4-3-2 avec $C_{\max}=20$ comme solution du problème qui est le même résultat donné par l'heuristique A.

Après avoir présenté les heuristiques proposées A et B, on les programme sous Delphi, qui est un langage et environnement de programmation visuel, permettant de créer des applications Microsoft Windows très efficaces, avec un minimum de codages manuel, et fournissant tous les outils qui sont nécessaires pour développer, tester des applications

4.3 Programmes

La partie suivante est consacrée aux programmes complets des méthodes A et B, avec la règle de Johnson comme sous procédure ainsi l'heuristique CDS.

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Grids;

type

TForm1 = class(TForm)

Tab: TStringGrid;

Label1: TLabel;

Edit1: TEdit;

Label2: TLabel;

Edit2: TEdit;

Button1: TButton;

Button2: TButton;

Preposition1: TButton;

G: TGroupBox;

VectU: TStringGrid;

Edit3: TEdit;

Label4: TLabel;

Label3: TLabel;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Preposition2: TButton;

Preposition3: TButton;

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Preposition1Click(Sender: TObject);

procedure Preposition2Click(Sender: TObject);

procedure Preposition3Click(Sender: TObject);

```
private
  { Private declarations }
```

```
public
  { Public declarations }
```

```
end;
```

```
var
  Form1: TForm1;
```

```
implementation
```

```
{ ----- Declaration globale ----- }
```

```
var TabTach:array[1..100,1..100]of integer;
    TabJNS,OrdOpt:array[1..2,1..100]of integer;
    Tmaxx,Tmaxy,Cmax:integer;
    Tmax:array[1..100]of integer;
```

```
{$R *.DFM}
```

```
{ ----- Méthode de JOHNSON ----- }
```

```
procedure M_JOHNSON;
var i1,mxy1,mxy2,c,k,l,i2,i,j,j1,j2,m1,m2:integer;
    tabInter:array[1..3,1..100]of integer;
```

```
begin
  k:=1;
  l:=Tmaxy;
```

```

c:=0;
for i:=1 to tmaxy do
begin
  tabInter[1][i] := i;
  tabInter[2][i] := TabJNS[1][i];
  tabInter[3][i] := TabJNS[2][i];
end;
while c<Tmaxy do
begin
  i1 := tabInter[1][1];
  j1 :=1;
  j2 :=1;
  i2 := tabInter[1][1];
  mxy1 := tabInter[2][1];
  mxy2 := tabInter[3][1];
  for i:=2 to Tmaxy-c do
  begin
    if TabInter[2][i]<mxy1 then
      begin
        mxy1 := TabInter[2][i];
        i1 := TabInter[1][i];
        j1 := i;
      end;
    if TabInter[3][i]<mxy2 then
      begin
        mxy2 := TabInter[3][i];
        i2 := TabInter[1][i];
        j2 := i;
      end;

  end;
  if mxy1<mxy2 then
    begin
      OrdOpt[1][k]:=i1;

```

```

        tabInter[1][j1]:=tabInter[1][Tmaxy-c];
        tabInter[2][j1]:=tabInter[2][Tmaxy-c];
        tabInter[3][j1]:=tabInter[3][Tmaxy-c];
        k:=k+1;
        c:=c+1;
    end
    else
    begin
        OrdOpt[1][l]:=i2;
        tabInter[1][j2]:=tabInter[1][Tmaxy-c];
        tabInter[2][j2]:=tabInter[2][Tmaxy-c];
        tabInter[3][j2]:=tabInter[3][Tmaxy-c];
        l:=l-1;
        c:=c+1;
    end;
end;
for i:= 1 to Tmaxx do
    Tmax[i] := 0;
    for i:=1 to Tmaxy do
    begin
        j := OrdOpt[1][i];
        Tmax[1] := Tmax[1] + TabTach[1][j];
        for k:=2 to Tmaxx do
            if Tmax[k] < Tmax[k-1] then
                Tmax[k] := Tmax[k-1] + TabTach[k][j]
            else
                Tmax[k] := Tmax[k] + TabTach[k][j]
            end;
        end;

        cmax := Tmax[Tmaxx];
    end;
end;

```

{ ----- Réserve de l'interface ----- }

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
  Tmxy := StrToInt(Edit1.Text);
  Tmaxx := StrToInt(Edit2.Text);
  tab.RowCount := Tmaxx + 1;
  tab.ColCount := Tmxy + 1;

  for i:=1 to tab.RowCount do
    tab.Cells[0,i] := 'm'+ IntToStr(i);

  for i:=1 to tab.ColCount do
    tab.Cells[i,0] := IntToStr(i);

end;
```

{ ----- Appel de la Méthode de JOHNSON ----- }

```
procedure TForm1.Button2Click(Sender: TObject);
var i,j:integer;
begin
  g.Height := 130;
  for i:=1 to Tmaxx do
    for j:=1 to Tmxy do
      TabTach[i][j] := StrToInt (tab.Cells[j,i]);
  for i:=1 to 2 do
    for j:=1 to Tmxy do
      TabJNS[i][j] := TabTach[i][j];
  M_JOHNSON;
  VectU.ColCount := Tmxy;
  for i:=1 to Tmxy do
    VectU.Cells[i-1,0] := IntToStr(OrdOpt[1][i]);
```

```
edit3.Text := inttostr(Cmax);
```

```
end;
```

```
{ ----- Heuristique A ----- }
```

```
procedure TForm1.Preposition1Click(Sender: TObject);
```

```
var i,j,h,k:integer;
```

```
    Cmin,ind:integer;
```

```
    Opt:array[1..100]of integer;
```

```
begin
```

```
    g.Height := 185;
```

```
    for i:=1 to Tmaxx do
```

```
        for j:=1 to Tmaxy do
```

```
            TabTach[i][j] := StrToInt (tab.Cells[j,i]);
```

```
    Cmin := 0;
```

```
    ind := 1;
```

```
    for i:=1 to Tmaxx-1 do
```

```
        begin
```

```
            for k:=1 to Tmaxy do
```

```
                begin
```

```
                    TabJNS[1][k] := 0;
```

```
                    TabJNS[2][k] := 0;
```

```
                    for h:=1 to i do
```

```
                        TabJNS[1][k]:=TabJNS[1][k] + TabTach[h][k];
```

```
                    for h:=i+1 to Tmaxx do
```

```
                        TabJNS[2][k]:=TabJNS[2][k] + TabTach[h][k];
```

```
                end;
```

```
            M_JOHNSON;
```

```

if i=1 then
    begin
        Cmin := Cmax;
        for h:=1 to Tmaxy do
            Opt[h] := OrdOpt[1][h];
        end
    else
        if Cmax<Cmin then
            begin
                Cmin := Cmax;
                ind := i;
                for h:=1 to Tmaxy do
                    Opt[h] := OrdOpt[1][h];
                end;
            end;

end;

VectU.ColCount := Tmaxy;
for i:=1 to Tmaxy do
    VectU.Cells[i-1,0] := IntToStr(Opt[i]);

edit3.Text := inttostr(Cmin);
label7.Caption := "";
label8.Caption := "";
for i:=1 to Tmaxx do
    begin
        if i<=ind then
            if i=ind then
                label7.Caption := label7.Caption + 'm' + inttostr(i)+'.'
            else
                label7.Caption := label7.Caption + 'm' + inttostr(i)+' , ';
        if i>ind then
            if i=Tmaxx then
                label8.Caption := label8.Caption + 'm' + inttostr(i)+'.'

```

```

        else
            label8.Caption := label8.Caption + 'm' + inttostr(i)+' , ';
        end;
    end;
end;

```

{ ----- Heuristique CDS ----- }

```

procedure TForm1.Preposition2Click(Sender: TObject);
var i,j,h,k:integer;
    Cmin,ind:integer;
    Opt:array[1..100]of integer;
begin
    g.Height := 185;
    for i:=1 to Tmaxx do
        for j:=1 to Tmaxy do
            TabTach[i][j] := StrToInt (tab.Cells[j,i]);
        Cmin := 0;
        ind := 1;
        for i:=1 to Tmaxx-1 do
            begin
                for k:=1 to Tmaxy do
                    begin
                        TabJNS[1][k] := 0;
                        TabJNS[2][k] := 0;
                        for h:=1 to i do
                            TabJNS[1][k]:=TabJNS[1][k] + TabTach[h][k];
                        for h:=Tmaxx-i+1 to Tmaxx do
                            TabJNS[2][k]:=TabJNS[2][k] + TabTach[h][k];
                        end;
                    M_JOHNSON;
                if i=1 then

```

```

begin
  Cmin := Cmax;
  for h:=1 to Tmaxy do
    Opt[h] := OrdOpt[1][h];
  end
  else
    if Cmax<Cmin then
      begin
        Cmin := Cmax;
        ind := i;
        for h:=1 to Tmaxy do
          Opt[h] := OrdOpt[1][h];
        end;
      end;
    end;

VectU.ColCount := Tmaxy;
for i:=1 to Tmaxy do
  VectU.Cells[i-1,0] := IntToStr(Opt[i]);

edit3.Text := inttostr(Cmin);
label7.Caption := "";
label8.Caption := "";
for i:=1 to ind do
  if i=ind then
    label7.Caption := label7.Caption + 'm' + inttostr(i)+'.'
  else
    label7.Caption := label7.Caption + 'm' + inttostr(i)+'.';
  end;
for i:=Tmaxx-ind+1 to Tmaxx do
  if i=Tmaxx then
    label8.Caption := label8.Caption + 'm' + inttostr(i)+'.'
  else
    label8.Caption := label8.Caption + 'm' + inttostr(i)+'.';
  end;

```

end;

{ ----- Heuristique B ----- }

procedure TForm1.Preposition3Click(Sender: TObject);

var i,j,h,k:integer;

 Cmin,ind:integer;

 Opt:array[1..100]of integer;

begin

 g.Height := 185;

 for i:=1 to Tmaxx do

 for j:=1 to Tmaxy do

 TabTach[i][j] := StrToInt (tab.Cells[j,i]);

 Cmin := 0;

 ind := 1;

 for i:=1 to Tmaxx-1 do

 begin

 for k:=1 to Tmaxy do

 begin

 TabJNS[1][k]:= TabTach[i][k];

 TabJNS[2][k] := 0;

 for h:= 1 to i-1 do

 TabJNS[2][k]:=TabJNS[2][k] + TabTach[h][k];

 TabJNS[2][k]:=TabJNS[2][k] + TabTach[Tmaxx][k];

 end;

 M_JOHNSON;

 if i=1 then

 begin

 Cmin := Cmax;

 for h:=1 to Tmaxy do

 Opt[h] := OrdOpt[1][h];

 end

```

else
    if Cmax<Cmin then
        begin
            Cmin := Cmax;
            ind := i;
            for h:=1 to Tmaxy do
                Opt[h] := OrdOpt[1][h];
            end;

end;

VectU.ColCount := Tmaxy;
for i:=1 to Tmaxy do
    VectU.Cells[i-1,0] := IntToStr(Opt[i]);

edit3.Text := inttostr(Cmin);
label7.Caption := "";
label8.Caption := "";
label7.Caption := label7.Caption + 'm' + inttostr(ind)+' .';
for i:=1 to ind-1 do
    label8.Caption := label8.Caption + 'm' + inttostr(i)+' ,';
    label8.Caption := label8.Caption + 'm' + inttostr(Tmaxx)+' . ';

end;
end.

```

4.4 Tests et résultats

Maintenant, on voudrait tester les trois méthodes: A, B et CDS sur différents exemples de 3,4,5,6 et 7 machines. Les durées opératoires des tâches suivent une loi uniforme sur $[1,99]$.

Exemple 1

	J_1	J_2	J_3
M_1	24	27	50
M_2	92	99	29
M_3	7	22	67

$$C_{\max}(A) = 277$$

$$C_{\max}(B) = 277$$

$$C_{\max}(CDS) = 277$$

Exemple 2

	J_1	J_2	J_3
M_1	95	14	29
M_2	76	10	81
M_3	66	2	98

$$C_{\max}(A) = 276$$

$$C_{\max}(B) = 276$$

$$C_{\max}(CDS) = 276$$

Exemple 3

	J_1	J_2	J_3
M_1	9	47	85
M_2	84	91	65
M_3	34	23	88

$$C_{\max}(A) = 337$$

$$C_{\max}(B) = 273$$

$$C_{\max}(CDS) = 273$$

Exemple 4

	J_1	J_2	J_3
M_1	49	53	42
M_2	98	19	66
M_3	38	50	67

$$C_{\max}(A) = 280$$

$$C_{\max}(B) = 280$$

$$C_{\max}(CDS) = 280$$

Exemple 5

	J_1	J_2	J_3	J_4
M_1	95	56	86	20
M_2	20	96	4	71
M_3	12	3	52	26

$$C_{\max}(A) = 289$$

$$C_{\max}(B) = 356$$

$$C_{\max}(CDS) = 289$$

Exemple 6

	J_1	J_2	J_3	J_4
M_1	20	3	90	33
M_2	78	88	75	50
M_3	62	76	38	56

$$C_{\max}(A) = 332$$

$$C_{\max}(B) = 332$$

$$C_{\max}(CDS) = 332$$

Exemple 7

	J_1	J_2	J_3	J_4	J_5
M_1	14	17	61	71	33
M_2	8	90	27	55	70
M_3	13	14	23	93	93

$$C_{\max}(A) = 362$$

$$C_{\max}(B) = 339$$

$$C_{\max}(CDS) = 339$$

Exemple 8

	J_1	J_2	J_3	J_4	J_5
M_1	58	38	95	61	29
M_2	87	72	20	17	96
M_3	74	17	77	4	94

$$C_{\max}(A) = 391$$

$$C_{\max}(B) = 391$$

$$C_{\max}(CDS) = 391$$

Exemple 9

	J_1	J_2	J_3	J_4
M_1	16	86	74	47
M_2	42	39	65	63
M_3	10	26	93	7
M_4	45	36	83	54

$$C_{\max}(A) = 414$$

$$C_{\max}(B) = 414$$

$$C_{\max}(CDS) = 414$$

Exemple 10

	J_1	J_2	J_3	J_4
M_1	46	78	49	89
M_2	86	65	5	32
M_3	85	1	23	26
M_4	47	14	33	43

$$C_{\max}(A) = 354$$

$$C_{\max}(B) = 354$$

$$C_{\max}(CDS) = 354$$

Exemple 11

	J_1	J_2	J_3	J_4	J_5	J_6
M_1	46	60	87	98	22	72
M_2	57	6	2	78	64	41
M_3	79	42	76	44	32	74
M_4	7	31	96	50	95	27

$$C_{\max}(A) = 540$$

$$C_{\max}(B) = 540$$

$$C_{\max}(CDS) = 540$$

Exemple 12

	J_1	J_2	J_3	J_4	J_5
M_1	30	7	52	58	21
M_2	66	98	34	76	38
M_3	29	58	43	53	78
M_4	47	43	23	64	68

$$C_{\max}(A) = 424$$

$$C_{\max}(B) = 443$$

$$C_{\max}(CDS) = 424$$

Exemple 13

	J_1	J_2	J_3	J_4
M_1	82	35	72	37
M_2	64	29	31	70
M_3	81	34	83	55
M_4	66	53	57	45

$$C_{\max}(A) = 437$$

$$C_{\max}(B) = 439$$

$$C_{\max}(CDS) = 439$$

Exemple 14

	J_1	J_2	J_3	J_4	J_5
M_1	44	63	37	32	4
M_2	92	14	57	2	61
M_3	68	21	45	39	61
M_4	22	61	5	68	3
M_5	83	63	4	10	3

$$C_{\max}(A) = 389$$

$$C_{\max}(B) = 389$$

$$C_{\max}(CDS) = 389$$

Exemple 15

	J_1	J_2	J_3	J_4	J_5
M_1	18	87	14	48	54
M_2	69	70	72	10	8
M_3	27	75	89	65	49
M_4	46	97	18	94	42
M_5	84	40	5	46	30

$$C_{\max}(A) = 472$$

$$C_{\max}(B) = 472$$

$$C_{\max}(CDS) = 472$$

Exemple 16

	J_1	J_2	J_3	J_4	J_5
M_1	29	76	37	72	68
M_2	75	65	70	98	2
M_3	10	80	17	37	65
M_4	9	38	81	73	94
M_5	72	31	47	65	61

$$C_{\max}(A) = 535$$

$$C_{\max}(B) = 568$$

$$C_{\max}(CDS) = 561$$

Exemple 17

	J_1	J_2	J_3	J_4	J_5	J_6
M_1	30	66	25	95	96	30
M_2	64	34	71	87	30	69
M_3	46	48	86	69	99	49
M_4	91	47	41	31	44	9
M_5	25	70	43	82	2	20

$$C_{\max}(A) = 564$$

$$C_{\max}(B) = 600$$

$$C_{\max}(CDS) = 565$$

Exemple 18

	J_1	J_2	J_3	J_4	J_5	J_6
M_1	72	81	10	18	12	58
M_2	73	7	51	82	47	56
M_3	62	54	88	61	68	26
M_4	98	20	61	84	14	24
M_5	16	60	6	12	46	3
M_6	21	30	94	65	8	39

$$C_{\max}(A) = 542$$

$$C_{\max}(B) = 542$$

$$C_{\max}(CDS) = 542$$

Example 19

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
M_1	12	46	3	51	49	29	88
M_2	65	8	39	71	90	67	76
M_3	12	58	75	86	3	58	65
M_4	47	56	57	74	63	7	96
M_5	68	26	41	98	94	87	18
M_6	14	24	20	86	5	49	14

$$C_{\max}(A) = 657$$

$$C_{\max}(B) = 719$$

$$C_{\max}(CDS) = 661$$

Example 20

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
M_1	7	14	41	17	79	37	52
M_2	54	88	40	74	79	76	10
M_3	83	51	42	38	39	38	73
M_4	15	95	38	46	26	89	1
M_5	18	13	90	5	35	19	60
M_6	8	6	67	56	96	37	95
M_7	82	38	95	37	48	91	40

$$C_{\max}(A) = 749$$

$$C_{\max}(B) = 755$$

$$C_{\max}(CDS) = 763$$

3.5 Discussion.

Tout d'abord, notons que dans le cas de deux machines, les trois heuristiques A, B et CDS sont identiques à la règle de Johnson, et par conséquent, elles donnent toutes la même solution optimale.

A travers les résultats du test de ces méthodes sur différents problèmes de 3 machines, avec le nombre de tâches varie entre 3 et 5, on constate bien qu'elles donnent la même solution, dans plusieurs cas (à savoir dans les exemples 1,2,4,6 et 8). Dans le reste des cas :

De l'exemple 3 : $C_{\max}(CDS) = C_{\max}(B) = 273 < 337 = C_{\max}(A)$

De l'exemple 5 : $C_{\max}(CDS) = C_{\max}(A) = 289 < 356 = C_{\max}(B)$

De l'exemple 7 : $C_{\max}(CDS) = C_{\max}(B) = 339 < 362 = C_{\max}(A)$

Ces résultats nous indiquent le bon comportement de l'heuristique CDS, ce qui pourrait être expliqué par le fait que l'heuristique CDS est une généralisation de la méthode exacte de Johnson à trois machines.

Concernant les cas de 4 machines, nous avons :

De l'exemple 9 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 414$

De l'exemple 10 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 354$

De l'exemple 11 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 540$

De l'exemple 12 : $C_{\max}(CDS) = C_{\max}(A) = 424 < 443 = C_{\max}(B)$

Alors que pour l'exemple 13 $C_{\max}(A) = 437 < 439 = C_{\max}(CDS) = C_{\max}(B)$

Ce dernier exemple montre, évidemment, que A donne un résultat mieux que CDS et B

Le teste sur 5 machines nous a donné :

De l'exemple 14 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 389$

De l'exemple 15 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 472$

Alors que de l'exemple 16 : $C_{\max}(A) = 535 < 561 = C_{\max}(CDS) < C_{\max}(B) = 568$

Et de l'exemple 17 : $C_{\max}(A) = 564 < 565 = C_{\max}(CDS) < C_{\max}(B) = 600$

Ces deux derniers exemples (16 et 17) montrent aussi que A donne mieux que B et CDS

Pour le cas de 6 machines, on a :

De l'exemple 18 : $C_{\max}(CDS) = C_{\max}(A) = C_{\max}(B) = 542$

Alors que de l'exemple 19 : $C_{\max}(A) = 657 < 661 = C_{\max}(CDS) < C_{\max}(B) = 719$

Cet exemple montre le bon résultat de A par rapport à CDS et B.

La même remarque pour l'exemple 20 de 7 machines qui donne :
 $C_{\max}(A) = 749 < 755 = C_{\max}(B) < C_{\max}(CDS) = 763$.

D'après ce qui précède, on déduit que les heuristiques A et B donnent, en global, des résultats aussi satisfaisants que CDS (à travers ces exemples), et en plus, l'heuristique A peut donner mieux que CDS dans des cas de 4,5,6 et 7 machines (exemples 13,16,17,19 et 20), ce qui indique la possibilité d'avoir un bon comportement de l'heuristique A par rapport à B et CDS pour d'autres problèmes de ces tailles et même pour des tailles assez grandes.

CONCLUSION GENERALE

L'ordonnancement d'atelier a été étudié de façon extensive en plusieurs variétés. Ce sujet constitue une classe importante dans le domaine d'optimisation et recherche opérationnelle en général. Le modèle de base consiste en m machines et n tâches (ou travaux). Chaque tâche est constituée d'un ensemble d'opérations associées aux m machines avec des durées d'exécution (ou opératoires) données.

Deux types célèbres et intéressants s'y présentent : l'ordonnancement à cheminement unique « flow shop » et l'ordonnancement à cheminement multiple « job shop ».

Dans le « flow shop », chaque tâche passe sur chaque machine une fois seulement, et l'ordre de passage sur les machines est le même pour toutes les tâches, alors que pour le « job shop » cet ordre peut être différent. Le critère d'évaluation est de rendre minimum la date de fin d'exécution de toutes les tâches dans l'atelier ou le makspan, noté $C_{\max}$.

Dans ce travail, on a abordé ces deux types importants des problèmes d'ordonnancement rencontrés souvent dans la vie pratique. On s'est intéressé plus des problèmes de type Flow shop avec cet objectif de trouver un ordre de lancement des tâches qui minimise le makespan $C_{\max}$ (date fin de projet).

Ce genre de problèmes, qui intervient fréquemment, dans les pratiques industrielles, a suscité depuis tant d'années des recherches et des études exhaustives.

En 1954, Johnson a proposé un algorithme [JOH 54] qui résout le problème du flow shop à deux machines $n/2/F/C_{\max}$ en atteignant la solution optimale.

L'idée de l'algorithme de Johnson a été étendue pour désigner des méthodes heuristiques pour le problème du flow shop à m machines telle que la méthode CDS [CAM 70].

Le problème est reconnu NP-complet lorsque l'on considère plus de deux machines. Cela implique que, dans les problèmes industriels qui comportent généralement plus de deux machines, on est obligé de mettre en œuvre des algorithmes heuristiques.

A l'instar de l'heuristique CDS, et d'autres trouvées dans la littérature (Palmer [PAL 65] et Dannenbring [DAN 77]) concernant ce type de problèmes d'ordonnancement, on a proposé deux heuristiques appelées A et B pour le cas général du Flow shop à m machines. Après avoir programmé les trois heuristiques A, B et CDS, on a procédé aux tests sur différents exemples de 3, 4, 5, 6 et 7 machines. A travers ces tests, on a constaté que les heuristiques A et B donnent, en général, des résultats aussi satisfaisants que CDS pour des problèmes à 3 machines (au moins $C_{\max}(A)$ ou $C_{\max}(B)$ est égal au $C_{\max}(CDS)$), et que l'heuristique A peut aboutir à des résultats mieux que la méthode CDS dans des cas pris de 4, 5, 6 et 7 machines. Cela donne la possibilité d'avoir un bon comportement de l'heuristique A par rapport à CDS pour une classe des problèmes de taille importante (au terme du nombre de tâches et machines).

Le problème général du flow shop reste un sujet fertile de recherche et mérite des efforts pour mettre en œuvre des méthodes et algorithmes, et en améliorer tous les résultats trouvés jusqu'à présent.

REFERENCES BIBLIOGRAPHIQUES

- [AKE 56] S. B. Aker, *A graphical approach to production scheduling problems*. Ops. Res., 4, 244-245, 1956
- [CAM 70] H. G. Campbell, R. A. Dudek and M. L. Smith, *A heuristic algorithm for the n-job, m-machine sequencing problem*. Mgmt. Sci., 16, B630-B637, 1970.
- [CAR 88] J. Carlier et P. Chrétienne, *problèmes d'ordonnancement*. Modélisation, Complexité, Algorithmes. Masson, 1988.
- [CAR 87] J. Carlier, *Scheduling jobs with release dates and tails on identical machines*, 1987.
- [CHU 96] C. Chu et J. Proth, *L'ordonnancement et ses applications*. Masson, 1996.
- [DAN 77] D. G. Dannenbring, *An evaluation of flow-shop sequencing heuristics*. Mgmt. Sci., 23, 1174-1182, 1977.
- [ESQ 99] P. Esquirol et P. Lopez, *L'ordonnancement*. Economica, 1999.
- [FRE 82] S. French, *Sequencing and Scheduling*. An introduction to the mathematics of the Job-Shop. John Wiley & sons New york, 1982.
- [JAC 56] J. R. Jackson, *An extension of Johnson's results on Job-lot scheduling*. Naval research logistics Quarterly, 3, 1956.
- [JOH 54] S. M. Johnson, *Optimal two- and three – stage production schedules with set-up times*. Nav. Res. Logist. Q., 1, 61- 68, 1954.

- [LEN 76] J. K. Lenstra, *Sequencing by Enumerative Methods*, Mathematisch Centrum, Amsterdam, 1976.
- [MIT 59] L. G. Mitten, *Sequencing n jobs on two machines with arbitrary time lags*. Management science, 5(3), 1959.
- [MOR 99] Morsli Omar, *The hybrid flow shop scheduling problem*. Chapter 2 of Ph. D. Thesis n° 329. Université catholique de Louvain, ESPO, Belgium, 1999.
- [PAL 65] D. S. Palmer, *Sequencing jobs through a multi-stage process in the minimum total time- a quick method of obtaining a near optimum*. Opl. Res. Q., 16,101-107, 1965.
- [PER 99] L. Péridy, E. Pinson et D. Rivreau, *Elimination rules for the Flow-shop and permutation Flow-shop problems*. Université d'Angers, 1999.
- [RIN 76] A. H. G. Rinnooy KAN, *Machine scheduling problems Classification, Complexity, and Computations*. Marinus Nijhoff, the Hague, 1976.
- [SAK 84] M. Sakarovitch, *Optimisation combinatoire. Méthodes mathématiques et Algorithmiques*. Hermann, 1984..