Ordonnancement des systèmes de production flexibles soumis à différents types de contraintes de blocage

Wajdi Trabelsi

THÈSE

Pour l'obtention du grade de

Docteur de l'Université de Lorraine

en

Automatique, Traitement du signal et des images, Génie informatique

École doctorale IAEM Lorraine
UFR Mathématiques, Informatique, Mécanique - Metz

Présentée par

Wajdi TRABELSI

Ordonnancement des systèmes de production flexibles soumis à différents types de contraintes de blocage

Soutenue le 14 Novembre 2012, devant la commission d'examen:

Rapporteurs	
M. DAUZERE-PERES Stéphane	Professeur, École des Mines de Saint-Etienne
M. YALAOUI Farouk	Professeur, Université de Technologie de Troyes
Examinateurs	
M. CRAMA Yves	Professeur, Université de Liège
M. MOUKRIM Aziz	Professeur, Université de Technologie de Compiègne
Directrice de thèse	
Mme SAUER Nathalie	Professeur, Université de Lorraine
Co-directeur de thèse	
M. SAUVEY Christophe	Maître de conférences, Université de Lorraine

Résumé

Ce sujet de thèse concerne de manière générale l'évaluation des performances et l'ordonnancement dans des systèmes de production flexibles et principalement les problèmes d'ordonnancement d'atelier de type Flow-Shop et Flow-Shop hybride. Le problème d'ordonnancement d'un Flow-Shop peut être défini ainsi : un ensemble de N jobs composés chacun de M opérations, doivent passer sur M machines dans le même ordre. Une machine peut exécuter une seule opération à la fois, chaque job ne peut avoir qu'une seule opération en cours de réalisation simultanément et la préemption n'est pas autorisée. Dans le cas des Flow-Shops hybrides, M_k machines identiques sont disponibles à chaque étage k en un ou plusieurs exemplaires. Pour cette étude, notre objectif est toujours de minimiser le temps total d'exécution aussi appelé makespan.

Les problèmes d'ordonnancement les plus répandus sont de type Flow-Shop classique où les espaces de stockage entre les machines sont considérées comme infinies. D'autres problèmes sont caractérisés par des capacités de stockage limitées ou nulles qui engendre une seule contrainte de blocage. Cette contrainte peut être un blocage classique (de type *RSb*) ou particulier (de type *RCb* ou *RCb**). Dans nos travaux de recherche, nous présentons un cas général qui peut être tiré de l'industrie et modélisé sous forme de systèmes de type Flow-Shop et Flow-Shop hybride soumis simultanément à plusieurs types de blocage.

Pour résoudre ce genre de problèmes, nous avons étudié dans cette thèse la complexité de ces systèmes et nous avons proposé des méthodes exactes, des méthodes approchées ainsi que des bornes inférieures.

Mots clés

Systèmes de production flexibles, Flow-Shop, blocage, ordonnancement, optimisation.

Abstract

This thesis deals mainly with makespan minimization in Flow-Shop and hybrid Flow-Shop scheduling problems where mixed blocking constraints are considered. In Flow-Shop scheduling problem, a set of *N* jobs must be executed on a set of *M* machines. All jobs require the same operation order that must be executed according to the same manufacturing process. Each machine can only execute one job at any time. Pre-emptive operation is not authorized in presented work. In case of hybrid Flow-Shop, at any processing stage *k*, there exist one or more identical machines M_k. Objective function consists in determining best schedule in order to reduce makespan, i.e. time where all operations are completed.

The most common scheduling problem is classical flowshop where buffer space capacity between machines is considered as unlimited. Other problems are characterized by the fact that the storage capacity is limited or null and which generates one blocking constraint. This constraint can be a classical blocking (*RSb*) or particular blocking (*RCb* or *RCb**). In our works, we present a general case which can be derived from industry and modeled as Flow-Shop and hybrid Flow-Shop systems subject simultaneously to different blocking.

To solve these problems, we studied in this thesis complexity of these systems and we proposed exact methods, approached methods and lower bounds.

Keywords

Flexible production systems, flowshop, blocking, scheduling, optimization

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Pour obtenir des profits élevés, les usines de production modernes cherchent généralement à maximiser leur productivité tout en réduisant les coûts. Pour atteindre cet objectif, elles doivent ordonnancer d'une façon optimale ou quasi optimale leurs opérations dans les processus de production tout en réduisant le plus possible les capacités de stockage. En effet, une telle réduction permet de gagner de l'espace donc de l'argent, de réduire les coûts de stockage, de s'adapter à l'augmentation de la complexité de la gestion de la production, d'éviter l'obsolescence des produits, de répondre au besoin de réactivité à des changements dans la production, etc. Cependant, la limitation des espaces de stockage entre les machines peut conduire à des situations connues dans la littérature comme situations de blocage entre les machines. En effet, si le stock entre deux machines est inexistant ou plein, lorsque la première machine a terminé de traiter un produit, elle ne peut pas le libérer immédiatement si la seconde machine n'est pas disponible.

Les modèles d'ordonnancement diffèrent selon les technologies utilisées et les contraintes auxquelles est soumis le système. Les problèmes d'ordonnancement les plus répandus dans la littérature sont de type Flow-Shop classique où les espaces de stockage entre les machines sont considérés comme infinis. D'autres problèmes sont caractérisés par des capacités de stockage limitées ou nulles qui engendre une seule contrainte de blocage. Cette contrainte peut être un blocage classique (de type *RSb*) ou particulier (de type *RCb* ou *RCb**). Dans le cas d'un blocage classique, un job reste bloqué sur une machine tant que la machine suivante n'est pas disponible ou qu'il n'y a pas de place en stock. Dans le cas *RCb**, une machine est disponible pour traiter la prochaine opération après que son opération soit finie sur la machine suivante. Le cas *RCb* est similaire au cas *RCb**, mais le job doit également avoir quitté la machine suivante.

Dans le cadre de cette thèse, nous étudions un cas général qui peut se présenter dans l'industrie. Nous considérons des systèmes de type Flow-Shop ou Flow-Shop hybride soumis simultanément à plusieurs types de blocage et en considérant le temps total d'exécution « *makespan* » comme critère d'optimisation. Nous nous plaçons dans un contexte déterministe où toutes les données sont connues avec certitude. Le mémoire de cette thèse est organisé en sept chapitres répartis en trois parties.

La première partie de la thèse est dédiée à la présentation globale du problème ainsi qu'à la présentation des différentes contraintes de blocage. Elle est composée de deux chapitres. Dans le premier chapitre, nous présentons les caractéristiques d'un système de type Flow-Shop et Flow-Shop hybride. Ensuite, une description détaillée des différentes contraintes de blocage ainsi que quelques exemples d'applications sont présentés. Dans le deuxième chapitre, nous présentons un état de l'art des principaux travaux qui ont été développés pour les systèmes de type Flow-Shop et Flow-Shop hybride, avec et sans blocage.

La deuxième partie est consacrée à l'étude des systèmes de production de type Flow-Shop soumis simultanément à plusieurs types de blocage. Dans le troisième chapitre, nous présentons des résultats de complexité pour les problèmes comportant la contrainte de blocage *RCb**, ainsi que pour des problèmes où différentes contraintes de blocage sont mélangées. Ensuite, pour résoudre le problème, nous présentons, dans le quatrième chapitre, une méthode de résolution exacte basée sur la modélisation mathématique en nombres entiers. Dans le dernier chapitre de cette deuxième partie, nous proposons des bornes inférieures, des heuristiques et un algorithme génétique.

Dans la troisième et dernière partie, nous étudions les systèmes de production de type Flow-Shop hybride avec blocage mixte. Nous proposons d'abord, dans le sixième chapitre, une méthode de résolution exacte basée sur la modélisation mathématique en nombres entiers pour des instances de petite taille. Dans le dernier chapitre du mémoire, nous proposons des bornes inférieures qui vont nous servir à évaluer nos heuristiques et méta-heuristiques pour les instances de plus grande taille.

Finalement, nous présentons la conclusion en synthétisant les apports de nos travaux et en formulant quelques propositions pour les évolutions futures des méthodes que nous avons développée.

Première partie
Présentation du problème et état de l'art

Introduction

La première partie de cette thèse est consacrée à la présentation des problèmes classiques d’ordonnancement des systèmes de production de type Flow-Shop et Flow-Shop hybride, ainsi que les problèmes particuliers présentant la caractéristique de flexibilité sur lesquels porte notre étude. Cette partie est composée de deux chapitres.

Dans le premier chapitre, nous présentons les caractéristiques des systèmes de production de type Flow-Shop et Flow-Shop hybride. Nous rappelons différentes contraintes de blocage déjà étudiées dans la littérature et nous introduisons une nouvelle contrainte de blocage. Nous présentons enfin quelques exemples d'applications.

Dans le deuxième chapitre, nous présentons un état de l'art des principaux travaux portant sur la minimisation du makespan dans les systèmes de type Flow-Shop et Flow-Shop hybride avec et sans blocage. Nous nous sommes aussi intéressés aux travaux qui considèrent des contraintes de blocage mixtes dans ces mêmes systèmes de production.

Chapitre 1
Présentation du problème

1. 1. Introduction

La résolution des problèmes d'ordonnancement consiste à optimiser une fonction objectif en déterminant la séquence suivant laquelle un ensemble de *N* tâches doit être exécuté sur un ensemble de *M* ressources et soumis à des contraintes, ainsi qu'à déterminer les instants de début et de fin d'exécution des différentes tâches sur chacune des ressources.

Les problèmes d'ordonnancement existent dans de nombreux secteurs d'activités comme la gestion de production, dans l'industrie manufacturière on encore les systèmes informatiques, où les tâches représentent les programmes et les ressources sont les processeurs ou la mémoire, la conception des emplois des temps, etc.

Parmi ces nombreux types de problèmes d'ordonnancement, nous nous sommes intéressés dans le cadre de cette thèse aux problèmes d'atelier. Dans ces problèmes, l'ensemble des opérations (tâches) qui doivent être exécutées sur les différentes machines (ressources) constitue un travail appelé job. En fonction de l'ordre de passage des opérations sur les machines, on distingue trois classes de problèmes, à savoir :

- Flow-Shop : Tous les jobs suivent le même ordre de passage sur chacune des machines.
- Job-Shop : Chaque job suit une gamme qui lui est propre et chacun d'entre eux peut exécuter plusieurs opérations sur une même machine.
- Open-Shop : L'ordre d'exécution des opérations qui composent un job (la gamme opératoire) n'est pas fixé à l'avance et peut donc être différent d'une solution proposée du problème à une autre.

Les modèles d'ordonnancement diffèrent aussi selon les technologies utilisées et les contraintes auxquelles le système est soumis. Les problèmes d'ordonnancement les plus répandus sont de type Flow-Shop classique où les espaces de stockage entre les machines sont considérés comme infinis. D'autres problèmes sont caractérisés par une contrainte dite de blocage où ces capacités de stockage sont limitées ou nulles. Dans nos travaux de thèse, nous présentons un cas général qui peut s'appliquer à l'industrie et est modélisé sous forme de systèmes de type Flow-Shop ou

Flow-Shop hybride soumis simultanément à plusieurs types de blocage en considérant le temps total d'exécution, aussi appelé makespan, comme critère d'optimisation.

Dans les deux paragraphes suivants, nous présentons de manière plus détaillée les systèmes de production de type Flow-Shop et Flow-Shop hybride qui font l'objet de notre étude. Ensuite, nous proposons une description détaillée des différentes contraintes de blocage ainsi que quelques exemples d'applications industrielles.

1. 2. Systèmes de production de type Flow-Shop

1. 2. 1. Flow-Shop classique

Dans le problème d'ordonnancement de systèmes de type Flow-Shop (F), un ensemble de N jobs, $J = \{J_1, J_2, \ldots, J_N\}$, doit être traité sur un ensemble de M machines, $M = \{M_1, M_2, \ldots, M_M\}$. Tous les jobs passent sur toutes les machines dans le même ordre. Ils sont donc composés de M opérations, $O_i = \{O_{i1}, O_{i2}, \ldots, O_{iM}\}$. L'opération O_{ik} a besoin d'un temps d'exécution P_{ik} sur la machine $M_k \in M$. Chaque machine ne peut effectuer qu'une seule opération à la fois et chaque job ne peut avoir qu'une seule opération en cours de réalisation simultanément. La capacité de stockage inter-machines est définie et la préemption d'opérations n'est pas autorisée.

La Figure 1.1 illustre un exemple de système de production de type Flow-Shop classique à N jobs et M machines.

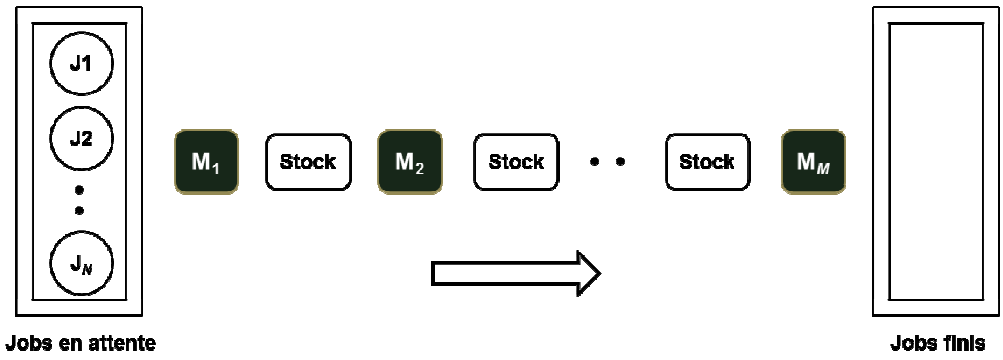


Figure 1.1. Flow-Shop classique

La résolution du problème consiste à déterminer l'ordre de passage des jobs sur l'ensemble des machines ainsi que les instants de début et de fin des opérations des jobs afin de réduire au minimum le temps total d'exécution de tous les jobs, aussi appelé makespan et noté *Cmax*.

1. 2. 2. Flow-Shop hybride

Afin d'être toujours plus réactives et productives, les entreprises ont cherché à augmenter la flexibilité de leurs systèmes de production. Pour atteindre ce but, il est possible de multiplier le nombre des machines qui peuvent réaliser une même opération. Ces machines, considérées comme identiques dans le cadre de cette thèse, sont regroupées en étage. Le modèle résultant est connu dans la littérature sous le nom de Flow-Shop hybride (FH).

Le Flow-Shop hybride est donc une généralisation du Flow-Shop classique au cas où plusieurs machines sont disponibles sur un ou plusieurs étages pour exécuter les différentes tâches du Flow-Shop. Ces problèmes présentent alors une difficulté supplémentaire par rapport aux problèmes sans flexibilité des ressources. En effet, la machine qui sera utilisée pour exécuter une opération n'est pas connue d'avance, mais doit être sélectionnée parmi un ensemble donné pour construire une solution au problème.

Dans un problème d'ordonnancement de type Flow-Shop hybride, un ensemble de N jobs, $J = \{J_1, J_2, \ldots, J_N\}$, doit être traité dans un atelier de production composé de K étages, $E = \{E_1, E_2, \ldots, E_K\}$. Chaque étage E_k contient M_k machines parallèles identiques, avec (k= 1,2,...,K). Tous les jobs exigent le même ordre des opérations, $O_i = \{O_{i1}, O_{i2}, \ldots, O_{iK}\}$, qui doivent être exécutées selon le même processus de fabrication. L'opération O_{ik} a besoin d'un temps d'exécution P_{ik} sur l'étage $E_k \in E$. Une machine ne peut appartenir qu'à un seul étage et ne peut effectuer qu'une seule opération à la fois. Chaque job ne peut avoir qu'une seule opération en cours de réalisation simultanément et doit être traité par une seule machine de l'étage E_k, sans interruption.

Un exemple de système de production de type Flow-Shop hybride à N jobs, K étages et M_k machines par étage, est présenté sur la Figure 1.2.

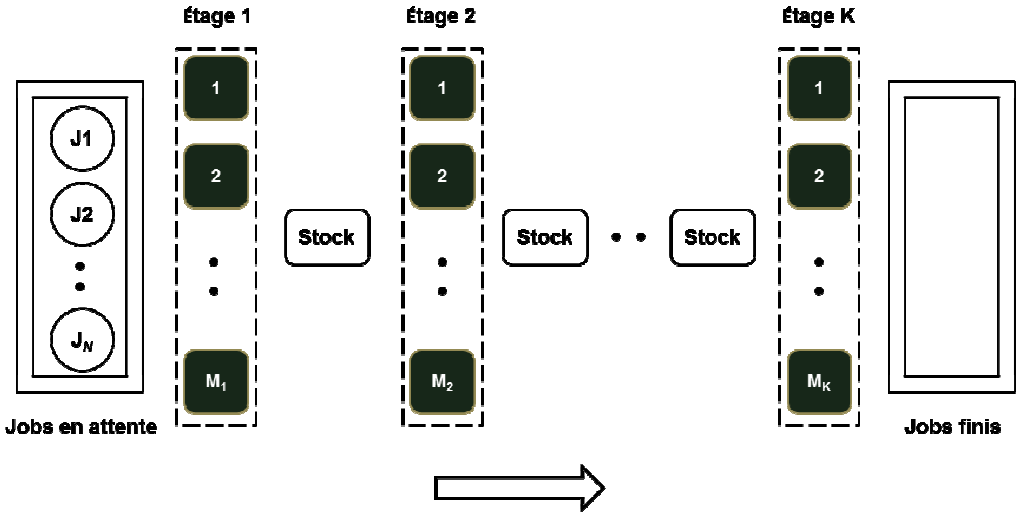


Figure 1.2. Flow-Shop hybride

La résolution du problème consiste à trouver un ordonnancement faisable minimisant le makespan en déterminant l'ordre de passage des jobs sur l'ensemble des machines ainsi que les dates de début et de fin des opérations sur chaque machine.

1. 3. Contraintes de blocage

De nombreuses études et recherches ont été réalisées au sujet du problème de type Flow-Shop classique et le problème du blocage compte parmi les plus importants sujets de recherche abordés récemment. En effet, des situations dites de blocage, engendrées par des contraintes industrielles, organisationnelles ou économiques sont présentes dans de nombreux problèmes réels.

Dans les systèmes industriels, la capacité des zones de stockage intermachines est considérée comme l'un des problèmes les plus couramment rencontrés. Ceci peut entraîner des situations de blocage dans les lignes de productions et agit sur les dates de passage des opérations sur les machines. En effet, si on considère que cette capacité de stockage est illimitée (c'est le cas du Flow-Shop classique), une machine est immédiatement disponible pour traiter la prochaine opération après que l'opération en cours est accomplie (Figure 1.3). Alors que pour une capacité limitée voire nulle (c'est le cas du Flow-Shop avec contrainte de blocage *RSb*), un job reste bloqué sur une machine tant que la machine suivante n'est pas disponible ou qu'il n'y a pas de place en stock (Figure 1.4). Cette hypothèse est souvent considérée à cause des problèmes d'espace entre les lignes de production et des coûts élevés que cela engendre.

Dans la suite, on associera la notation *Wb* (*Without blocking constraint*) aux problèmes de type Flow-Shop classique, pour indiquer qu'il n'y a pas de contrainte de blocage entre les machines.

Pour illustrer l'impact des différentes contraintes de blocage sur le makespan du système, nous considérons un exemple d'un Flow-Shop à 4 jobs et 5 machines. Les temps opératoires de chaque job sont donnés dans la matrice suivante P_{ik}.

$$P_{i,k} = \begin{pmatrix} 1 & 1 & 2 & 1 & 2 \\ 1 & 3 & 2 & 2 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 3 & 2 & 1 & 1 & 1 \end{pmatrix} \quad i \in \{1,..,4\},\ k \in \{1,..,5\}$$

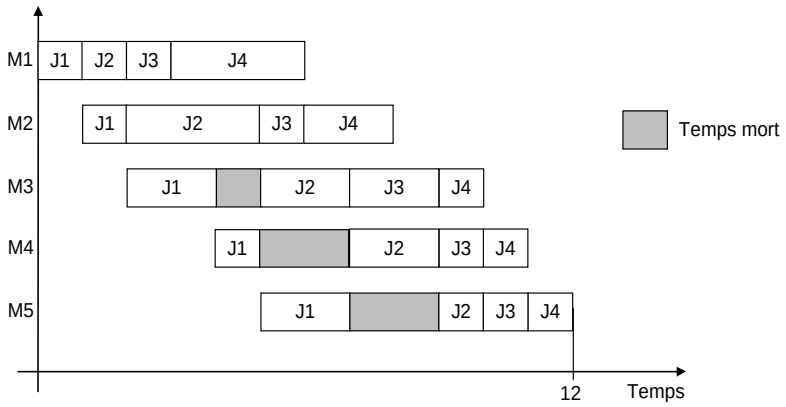


Figure 1.3. Flow-Shop classique (Wb)

Dans l'exemple de la figure 1.3 qui représente le diagramme de Gantt d'un Flow-Shop classique « sans blocage », la machine M_1 est disponible pour traiter l'opération du job J_4 dès que l'opération du job J_3 sur la machine M_1 est terminée. Le makespan correspondant est égal à 12 ut.

On peut noter que l'absence de blocage n'implique pas l'absence de temps mort pour les machines, ni pour les jobs.

1. 3. 1. Contrainte de blocage classique (blocage *RSb*)

Dans les situations de blocage classique, une machine est immédiatement disponible pour traiter sa prochaine opération après que la machine suivante est disponible ou qu'il y ait de la place dans

le stock (Figure 1.4). Cette contrainte de blocage, appelée contrainte de blocage de type *RSb* (*Release when Starting blocking*), a été rencontrée dans plusieurs applications industrielles, comme par exemple les chaînes robotisées des usines de production d'aciers (Hall *et al*., 1998).

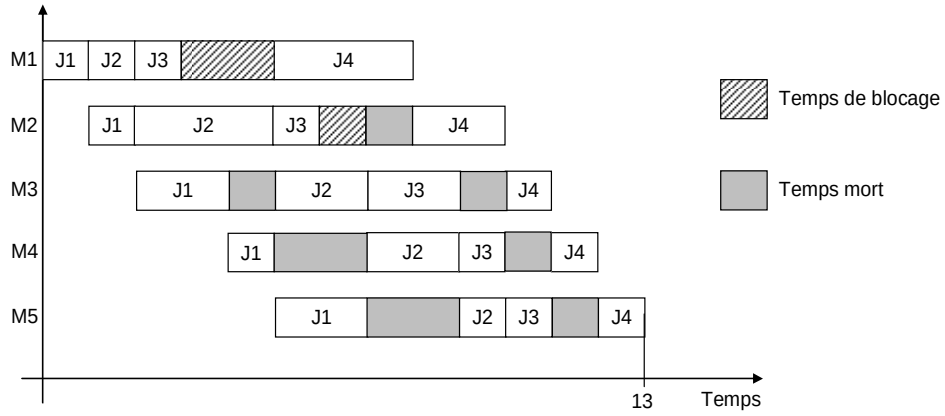


Figure 1.4. Flow-Shop avec la contrainte de blocage classique (RSb)

Nous reprenons l'exemple précédent du Flow-Shop à 4 jobs et 5 machines et nous appliquons la contrainte de blocage de type *RSb* entre toutes les machines. Le temps total d'exécution pour cet exemple augmente d'une unité temps. Cela est dû au retardement du début de la première opération du job J_4 sur la machine M_1. En effet, le job J_3 reste bloqué sur la machine M_1 tant que la machine M_2 n'est pas disponible. Ainsi, le job J_4 ne peut pas commencer son opération sur la machine M_1 avant que le job J_3 passe sur la machine M_2.

1. 3. 2. Contrainte de blocage particulière (blocage *RCb*)

Dans les situations de blocage soumises à cette contrainte particulière notée *RCb* (Release when Completing blocking), une machine est disponible pour traiter la prochaine opération après que son opération soit finie sur la machine suivante et que le job ait quitté cette machine (Figure 1.5). Cette contrainte a été étudiée pour la première fois par (Dauzère-Pérès *et al*., 2000).

Ce genre de problème a été rencontré à diverses reprises dans l'industrie. Elle apparaît par exemple dans les applications industrielles suivantes : le traitement de déchets industriels avant enfouissement et la fabrication de pièces métalliques avec traitement thermique. Ces deux exemples sont détaillés dans (Martinez, 2005).

Dans le premier cas, une compagnie reçoit différents types de déchets industriels et agricoles pour les traiter avant de les enfouir. Les déchets sont apportés par camions et déchargés dans des silos. Chaque cargaison est déchargée dans un seul silo. Chaque produit (ou cargaison) est ensuite traité par une seule machine (le malaxeur). Comme les produits s'écoulent lentement du silo vers le malaxeur, un silo est libéré seulement à la fin du traitement sur le malaxeur de la totalité des déchets qu'il contenait.

La deuxième application concerne une société qui réalise des pièces métalliques pour l'industrie aéronautique. Ces pièces doivent tout d'abord être chauffées à bonne température dans un ou plusieurs fours. Elles sont ensuite embouties sur une presse. Un four qui traite un lot est bloqué jusqu'à ce que toutes les pièces qu'il contenait aient été embouties par la presse.

Nous pouvons citer une troisième application industrielle sur laquelle s'applique ce genre de contrainte de blocage qui concerne le brassage du cidre. Dans cette application, on ne peut pas brasser les pommes des différents clients simultanément. Dans une première étape, les pommes sont versées dans un bain, puis pressées pour faire du jus de pomme. Les pommes d'un nouveau client ne peuvent pas être versées dans le bain avant que la totalité des pommes du premier client soit pressée.

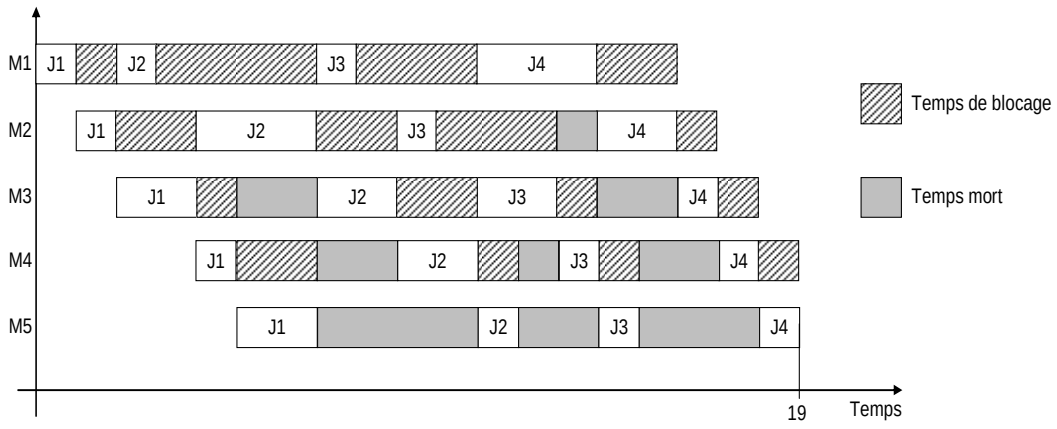


Figure 1.5. Flow-Shop avec la contrainte de blocage particulière (RCb)

Nous reprenons notre exemple du Flow-Shop à 4 jobs et 5 machines et nous appliquons la contrainte de blocage de type *RCb* entre toutes les machines. La valeur du makespan du problème augmente significativement pour cet exemple. Cela est dû à l'accroissement des temps d'inactivité pour les machines composés essentiellement par les temps de blocage (Figure 1.5).

Par exemple, la machine M_1 reste bloquée par le job J_3 jusqu'à ce que son opération sur la machine M_2 soit finie, mais aussi que le job J_3 quitte cette machine. Cette date correspond finalement à la date de début de l'opération du job J_3 sur la machine M_3.

1. 3. 3. Nouvelle contrainte de blocage (blocage *RCb**)

Dans ce travail, nous présentons une variante de la contrainte de blocage *RCb*, notée *RCb**, pour laquelle une machine sera disponible pour traiter la prochaine opération d'un job après que l'opération qu'elle traitait soit finie sur la machine suivante, sans tenir compte du fait que ce job quitte ou non la machine (Figure 1.6).

Cette contrainte de blocage a été proposée pour la première fois dans (Trabelsi *et al*., 2010). Dans ces travaux, nous avons proposé des heuristiques pour résoudre les problèmes de type Job-Shop avec les contraintes *RCb* et *RCb**.

On peut rencontrer ce genre de contrainte dans plusieurs cas industriels comme par exemple dans une ligne de production où deux machines successives sont dépendantes d'une même ressource (opérateur, outil,…), de sorte qu'elles ne peuvent pas fonctionner simultanément.

Un autre exemple de cette contrainte peut être relevé de l'application précédente du brassage du cidre. En effet, avant que les pommes ne soient versées dans le bain pour presse, elles doivent d'abord être triées dans un conteneur. Une fois l'étape de tri est terminée, le contenu du conteneur est versé dans le bain. Le conteneur ne peut trier le prochain produit avant que la totalité des pommes ne soient versées.

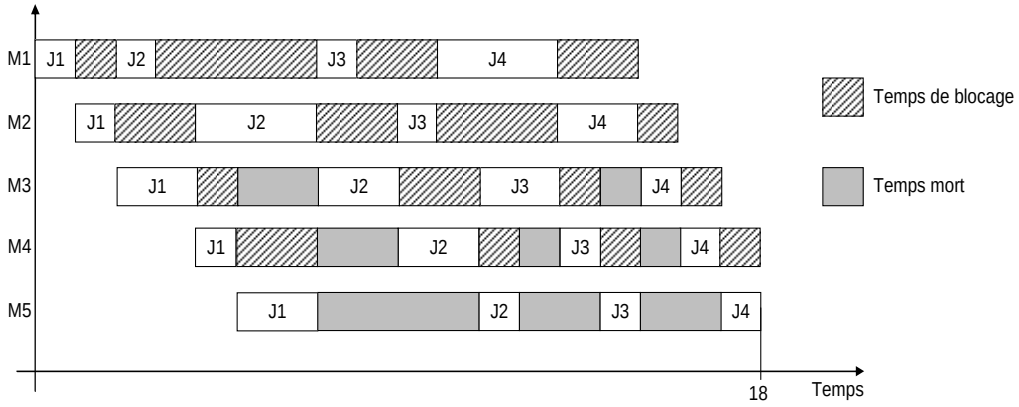


*Figure 1.6. Flow-Shop avec la contrainte de blocage RCb**

Nous reprenons notre exemple du Flow-Shop à 4 jobs et 5 machines et nous appliquons cette fois la contrainte de blocage de type *RCb** entre toutes les machines. Nous remarquons que la valeur du makespan pour cet exemple avec la contrainte *RCb** a diminué d'une unité de temps par rapport au même exemple avec la contrainte *RCb*. Cela est dû au décalage de la date de début de la première opération du job J_4 sur la machine M_1 (Figure 1.6). En effet, la machine M_1 est disponible pour traiter l'opération du job J_4 immédiatement après que le job J_3 termine son opération sur la machine M_2. Dans le cas du blocage *RCb**, le job J_4 n'attend pas que le job J_3 quitte la machine M_2.

1. 3. 4. Blocage mixte

Dans ce paragraphe, nous présentons un cas général qui peut directement être utilisé dans l'industrie et modélisé sous forme d'un système de type Flow-Shop dans lequel on peut choisir quelle contrainte de blocage appliquer entre deux machines successives.

Pour décrire ce genre de problèmes, nous considérons une séquence de contraintes de blocage V qui sera adaptée selon le nombre de machines du système et qui représente les différentes contraintes de blocage entre deux machines successives, *i.e.* V_k est la contrainte de blocage entre les machines M_k et M_{k+1}. Ce vecteur contient M-1 éléments, c'est-à-dire autant que de transitions entre les machines.

La Figure 1.7 illustre un exemple de système de production de type Flow-Shop avec blocage mixte à 4 jobs et 5 machines, où V = (*RCb*, *RSb*, *RCb**, *Wb*), *i.e. RCb* est la contrainte de blocage entre les machines M_1 et M_2, *RSb* est la contrainte de blocage entre les machines M_2 et M_3, etc.

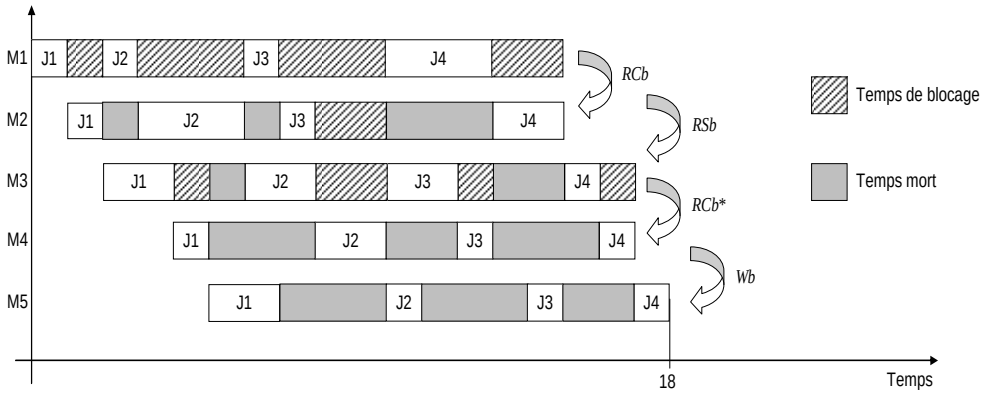


Figure 1.7. Flow-Shop avec blocage mixte

1. 4. Notations

Pour faciliter la description des différents problèmes d'ordonnancement qui sont abordés dans la littérature ainsi que dans le cadre de cette thèse, nous allons adopter la notation de Lawler (Graham *et al.*, 1979) qui consiste à décrire un problème d'ordonnancement donné par trois champs séparés par des « *slashs* » $\alpha|\beta|\gamma$, dont la signification de chaque champ est la suivante :

- Le champ α : Environnement machine. Ce premier champ spécifie le type et le nombre de machines considérées.
- Le champ β : Contraintes sur tache. Ce deuxième champ donne les caractéristiques des taches (jobs).
- Le champ γ : Critères d'optimisation. Ce troisième champ indique le critère à optimiser.

Par exemple, *F*2 | *RSb* | *Cmax* désigne la minimisation du makespan pour un problème de type Flow-Shop à deux machines avec contrainte de blocage de type *RSb*.

1. 5. Conclusion

Dans ce chapitre, nous avons présenté les caractéristiques des systèmes que nous étudions dans le cadre de cette thèse. Nous avons rappelé la définition des systèmes de production de type Flow-Shop et Flow-Shop hybride en premier lieu. Puis, nous avons présenté différentes contraintes de blocage qui peuvent être rencontrées dans diverses applications industrielles.

Dans le chapitre suivant, nous nous intéressons à l'état de l'art des systèmes de type Flow-Shop et Flow-Shop hybride, avec et sans contraintes de blocage, ainsi qu'aux problèmes qui considèrent des contraintes de blocage mixtes.

Chapitre 2
État de l'art

2. 1. Introduction

Au cours des dernières années, les chercheurs dans le domaine de la recherche opérationnelle se sont intéressés aux systèmes de production présentant une flexibilité sur les ressources. L'intérêt porté à ces systèmes est largement motivé par ce caractère de flexibilité donnant plus de degrés de liberté aux systèmes de production et s'approchant de plus en plus des problèmes d'ateliers réels.

Dans ce chapitre, nous établissons un état de l'art des problèmes d'ordonnancement de type Flow-Shop et Flow-Shop hybride, avec et sans contrainte de blocage et ayant comme critère d'optimisation la minimisation du makespan.

Ce chapitre est organisé comme suit : Nous présentons tout d'abord les différentes méthodes de résolution utilisées pour résoudre les problèmes d'ordonnancement. Puis, nous présentons un état de l'art des approches développées pour le Flow-Shop classique, avec et sans blocage. Enfin, nous présentons un état de l'art des approches développées pour le Flow-Shop hybride, avec et sans blocage.

2. 2. Les méthodes de résolution

La résolution des problèmes d'ordonnancement d'atelier consiste à programmer un ensemble de tâches et les affecter à un ensemble de ressources de façon à optimiser un ou plusieurs critères de performance, en respectant un ensemble de contraintes. Pour cela, plusieurs méthodes d'optimisation combinatoire sont proposées dans la littérature. Nous allons passer en revue les méthodes les plus connues en les classant en deux catégories : les méthodes exactes qui garantissent la résolution optimale du problème et les méthodes approchées qui perdent cette propriété pour gagner en temps de calcul.

2. 2. 1. Les méthodes exactes

Les méthodes de résolution exacte sont des applications simples des techniques d'optimisation. Elles examinent d'une manière implicite la totalité de l'espace de recherche pour produire la meilleure solution par rapport au critère choisi. Elles peuvent être très intéressantes pour des problèmes de petite taille, mais peuvent demander des temps de traitement importants lorsque la taille du problème augmente.

Parmi les principales méthodes de résolution exacte, on peut citer :

- **La programmation dynamique** : C'est une méthode d'optimisation introduite par (Bellman, 1957). Cette méthode est basée essentiellement sur la décomposition du problème en une série de sous-problèmes reliés entre eux par une relation de récurrence permettant de décrire la valeur optimale du critère à une étape donnée en fonction de sa valeur à l'étape précédente. La solution optimale du problème est obtenue en calculant les solutions des sous-problèmes les plus petits, pour ensuite en déduire petit à petit les solutions du problème complet. Elle est par exemple utilisée pour développer des algorithmes polynomiaux et pseudo polynomiaux pour résoudre les problèmes à une machine et à machines parallèles (Lawler, 1973), (Carlier et Chrétienne, 1988).
- **La programmation linéaire** : C'est un outil d'optimisation très puissant qui permet de résoudre un grand nombre de modèles linéaires. Son utilisation demande que le problème posé puisse se ramener à l'optimisation d'une fonction de forme linéaire, en respectant un ensemble de contraintes elles aussi linéaires, fonctions des mêmes variables positives ou nulles. Des contraintes très variées peuvent être ajoutées de la même façon. L'étape de modélisation du problème peut s'avérer difficile, et la résolution d'un programme linéaire complexe peut demander un temps relativement long. Parmi les logiciels de résolution des problèmes de programmation linéaire, on peut citer LP-Solve, Cplex, Mosel-Xpress (qui est aussi un langage de modélisation et de résolution des problèmes d'optimisation).
- **La méthode de séparation et évaluation (Branch and Bound)** : Cette méthode a été introduite pour la première fois par (Dantzig *et al.*, 1954) pour résoudre le problème de voyageur de commerce. Il s'agit de représenter l'ordonnancement sous forme d'un arbre dont on essayera d'énumérer toutes les solutions possibles du problème avec le moins de branches possibles pour éviter la numération des mauvaises solutions. Elle se base sur quelques principes simples :
  - Chaque nœud de l'arbre représente un choix binaire, c'est-à-dire que l'on réalise telle opération ou non (procédure de séparation).

- La capacité d'associer à chaque décision partielle une borne de l'objectif poursuivi (procédure d'évaluation progressive).
- Chaque séparation sera choisie de manière à maximiser la différence d'évaluation de l'objectif entre les deux branches.

Cette méthode est optimale, mais son temps de réponse peut être important. En effet, la performance d'une méthode de branch and bound dépend, entre autres, de la qualité de sa capacité d'exclure des solutions partielles tôt.

Ces méthodes exactes sont souvent utilisées pour pouvoir évaluer l'erreur que commettent les méthodes approchées qui donnent beaucoup plus rapidement une solution aux problèmes considérés.

2. 2. 2. Les méthodes approchées

Comme nous l'avons mentionné dans le paragraphe précédent, les méthodes exactes nécessitent des temps de traitement prohibitifs pour résoudre les problèmes de grande taille. Pour obtenir malgré tout des solutions, des méthodes approchées ont été développées. Ces méthodes donnent des solutions certes sous-optimales, mais en un temps de calcul raisonnable. La performance de telles méthodes est estimée par le pourcentage d'erreur entre la solution fournie et la valeur de la solution optimale si elle est calculable. Dans les cas où la solution optimale est non calculable, on peut évaluer ces méthodes en comparant les solutions fournies à des bornes inférieures.

Les méthodes approchées peuvent être classées en deux types : les méthodes constructives et les méthodes d'amélioration.

- **Les méthodes constructives** : Ces méthodes, dites aussi méthodes heuristiques, sont des méthodes de résolution de problèmes non fondées sur un modèle formel et qui fournissent rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-Difficile. Elles se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. La recherche commence à partir de zéro et à chaque itération, une solution partielle du problème est complétée en utilisant des règles de priorité.
- **Les méthodes d'amélioration** : Ces méthodes, dites aussi métaheuristiques, sont basées elles-mêmes sur l'utilisation de méthodes constructives. La notion de voisinage de la solution est généralement utilisée par ces méthodes pour conduire à une exploration plus exhaustive de l'espace des solutions. Il s'agit de générer une solution voisine en appliquant

plusieurs fois et de façon différente, une petite transformation. Cette solution est ensuite évaluée et comparée à la solution courante.

On distingue parmi ces méthodes, des **recherches locales** qui, à chaque itération, recherchent une solution meilleure que la solution courante dans son voisinage. Les **recherches globales** acceptent de ne pas forcément améliorer la valeur de la fonction objectif à chaque étape, ce qui permet de sortir des minima locaux.

Les métaheuristiques ont la possibilité de s'appliquer à une grande variété de problèmes, montrent une bonne efficacité et permettent d'obtenir une bonne performance en tenant compte de la qualité des solutions et du temps de calcul.

Les métaheuristiques ont un certain nombre de caractéristiques communes (Dréo et al., 2003) :

- Elles abordent souvent la combinatoire des problèmes par des méthodes stochastiques, c'est-à-dire des tirages probabilistes qui permettront d'effectuer des choix.
- Elles sont souvent inspirées par des analogies avec la physique (recuit simulé), avec la biologie (algorithmes génétiques) ou avec l'éthologie (colonies de fourmis, essaims particulaires…).
- Leurs performances ont pour inconvénient de dépendre du réglage de leurs paramètres, qui peut être délicat.
- Leurs temps de calcul peuvent être élevés.

2. 3. Flow-Shop

Dans cette partie, nous établissons un état de l'art des travaux concernant les problèmes d'ordonnancement de type Flow-Shop classique, avec et sans contrainte de blocage et ayant comme critère d'optimisation la minimisation du makespan.

Nous présentons tout d'abord les résultats de complexité pour chaque type de système. Nous donnons ensuite les résultats des travaux utilisant des méthodes de résolution exacte. Enfin, nous présentons quelques travaux présentés dans la littérature qui concernent la résolution de ces problèmes par des méthodes approchées.

2. 3. 1. Flow-Shop classique (*Wb*)

- **Complexité**

  Le premier article traitant le problème d'ordonnancement de type Flow-Shop classique a été publié il y a une cinquantaine d'années (Johnson, 1954), dans lequel l'auteur montre que la minimisation du makespan du problème d'un Flow-Shop classique à deux machines F2||Cmax est polynomiale. Dans le même article, Johnson démontre que pour les problèmes de Flow-Shop classique à trois machines F3||Cmax, si les temps opératoires sur la deuxième machine sont uniformément inférieurs à ceux de la première ou la troisième machine, le problème peut être résolu en temps polynomial. Par contre, si cette condition n'est pas vérifiée, les auteurs dans (Garey *et al*., 1976) montrent que le problème F3||Cmax est NP-Difficile au sens fort.

  De nombreux auteurs se sont intéressés à d'autres aspects de ce problème. Par exemple, pour le cas où la préemption est autorisée, les auteurs dans (Gonzalez et Sahni, 1978) ont montré que le problème de type Flow-Shop préemptif à deux machines F2|*pmtn*|Cmax peut être résolu par l'algorithme de Johnson et est donc polynomial, tandis qu'il devient NP-Difficile au sens fort pour le problème à trois machines F3|*pmtn*|Cmax. Nous pouvons citer aussi le cas du problème avec les dates de disponibilité pour les jobs F2|*ri*|Cmax, qui est un problème NP-Difficile au sens fort (Lenstra *et al*., 1977). Le cas préemptif de ce dernier problème F2|*ri,pmtn*|Cmax est aussi NP-Diffcile au sens fort (Cho et Sahni, 1981). Parmi les travaux les plus récents, nous pouvons citer ceux de (Baptiste et Timkovsky, 2004) et (Averbakh *et al*., 2005) qui ont montré que les problèmes F2|*prec,rj,pij*=1|∑Cj et FM|*pij*=1,*intree*|∑Cj, respectivement, peuvent être résolus polynomialement.

- **Méthodes exactes**

  Parmi les méthodes exactes utilisées pour la résolution des problèmes d'ordonnancement de type Flow-Shop classique, on trouve la procédure par Séparation et Evaluation (Branch and Bound). Elle a été appliquée sur le problème de type Flow-Shop à *M* machines FM||Cmax par plusieurs auteurs dont on peut citer : (Lomnicki, 1965), (Gupta, 1969, 1970). Dans (Potts, 1980), l'auteur a également proposé une procédure Branch and Bound avec laquelle il est possible de résoudre des instances jusqu'à 15 jobs et 4 machines. Parmi les procédures Branch and Bound les plus récentes, on peut citer celle proposée dans (Carlier et Rebaï, 1996) qui peut résoudre des instances jusqu'à 50 jobs et 10 machines.

  En comparant le nombre de variables entières binaires requises dans trois modèles linéaires en nombre entiers pour différents problèmes d'ordonnancement, Pan (1997) conclue que pour les problèmes de Flow-Shop de permutation, le modèle de Manne (1960) est la

meilleure formulation programmée, le modèle de Wanger (1959) est classé deuxième et le modèle de Wilson (1989) arrive derrière le modèle de Wanger.

Une autre étude a été menée dans (Tseng *et al*., 2004) pour comparer quatre modèles mathématiques : (Wagner, 1959), (Wilson, 1989), (Manne, 1960) et (Liao et You, 1992). Dans cette étude, les auteurs utilisent un critère différent de celui utilisé dans Pan (1997). En effet, ils comparent le temps de calcul que nécessite chaque modèle pour trouver la solution optimale, et obtiennent un classement différent :

1. Le modèle de Wagner est la meilleure formulation PLNE.
2. Le modèle de Wilson est la deuxième meilleure formulation PLNE.
3. Le modèle Manne traîne derrière la formulation de Liao-You.

- **Méthodes approchées**

Parmi les méthodes constructives qui ont été proposées pour résoudre les problèmes d'ordonnancement de type Flow-Shop classique à *M* machines FM||Cmax, nous pouvons citer celles de (Campbell *et al*., 1970), (Dannenbring, 1977) et (Rock et Schmidt, 1983) qui proposent des algorithmes basés sur une adaptation de l'algorithme de Johnson (Johnson, 1954). L'heuristique NEH (Nawaz *et al*., 1983) est une des heuristiques les plus connues pour résoudre ce genre de problèmes. Elle est non seulement la plus efficace (Ruiz et Marotto, 2005), mais aussi très simple à programmer. Nous revenons plus en détail sur cette heuristique dans le chapitre 5.

Pour ce qui concerne la résolution des problèmes FM||Cmax par les méthodes amélioratives, les auteurs dans (Taillard *et al*., 1990) proposent une recherche tabou et la comparent aux meilleures heuristiques existantes. Dans (Nowicki et Smutnicki, 1996), les auteurs proposent aussi une recherche tabou considérée comme l'une des meilleures méthodes de recherche globale pour résoudre ce problème. Les auteurs de (Osman et Potts, 1989) et (Ishibuchi et Misaki, 1995) se sont intéressés à la méthode de recuit simulé pour résoudre ces problèmes. Une autre méthode d'amélioration basée sur un Branch and Bound tronqué a été proposée par (Haouari et Ladhari, 2003). Elle a permis de résoudre efficacement des problèmes de grande taille (jusqu'à 200 jobs et 10 machines). Enfin, pour les travaux qui ont proposé des algorithmes génétiques pour résoudre les problèmes FM||Cmax, nous citons (Iyer et Saxena, 2004), (Siarry et Michalewicz, 2007) et (Zobolas *et al*., 2009). Dans ce dernier article, des résultats ont été obtenus pour des problèmes de très grande taille (jusqu'à 200 jobs et 20 machines).

Le Tableau 2.1 présente un résumé des principaux travaux que nous venons de citer.

Compléxité		
F2||Cmax	(Johnson, 1954)	Problème polynomial
F3||Cmax	(Johnson, 1954)	Polynomial si pour tout $i \leq N$, $P_{i2} \leq P_{i1}$ ou $P_{i2} \leq P_{i3}$
F3||Cmax	(Garey et al., 1976)	NP-Difficile
F2|*pmtn*|Cmax	(Gonzalez et Sahni, 1978)	Polynomial
F3|*pmtn*|Cmax	(Gonzalez et Sahni, 1978)	NP-Difficile
Méthodes exactes		
FM||Cmax	(Lomnicki, 1965)	Parmi les premières B&B
FM||Cmax	(Potts, 1980)	B&B (jusqu'à 15 jobs et 4 machines)
FM||Cmax	(Carlier et Rebaï, 1996)	B&B (jusqu'à 50 jobs et 10 machines)
FM||Cmax	(Wagner, 1959), (Manne, 1960), (Wilson, 1989) et (Liao et You, 1992)	PLNE
Méthodes approchées		
FM||Cmax	(Campbell et al., 1970), (Dannenbring, 1977) et (Rock et Schmidt, 1983)	Algorithmes basés sur une adaptation de l'algorithme de Johnson
FM||Cmax	(Nawaz et al., 1983)	Heuristique très efficace
FM||Cmax	(Nowicki et Smutnicki, 1996)	Recherche tabou très efficace
FM||Cmax	(Osman et Potts, 1989) et (Ishibuchi et Misaki, 1995)	Recuit simulé
FM||Cmax	(Haouari et Ladhari, 2003)	Métaheuristique basée sur un Branch and Bound tronqué (jusqu'à 200 jobs et 10 machines)
FM||Cmax	(Zobolas *et al*., 2009)	Algorithme génétique (jusqu'à 200 jobs et 20 machines)

Tab 2.1 : Etat de l'art pour le Flow-Shop classique

2. 3. 2. Flow-Shop avec blocage (*RSb*, *RCb*, *RCb**, mixte)

Cette partie concerne les blocages de type *RSb*, *RCb*, *RCb** et les problèmes qui considèrent le blocage mixte.

Parmi les travaux qui ont présenté un état de l'art des problèmes d'ordonnancement de type Flow-Shop avec différentes contraintes de blocage (no-wait, *RSb*, cellules robotisées), nous pouvons citer ceux de (Bagchi *et al*., 2006). Dans cet article, les problèmes sont souvent transformés en un problème de voyageur de commerce.

Comme dans le paragraphe précédent, nous donnons les résultats des travaux réalisés sur la complexité, puis nous présentons les travaux utilisant des méthodes de résolution exacte. Enfin, nous citons les principaux travaux présentés dans la littérature qui concernent la résolution de ces problèmes par des méthodes approchées.

- **Complexité**

  La contrainte sans attente entre les machines (no-wait) est un autre type de contrainte rencontrée dans les systèmes industriels et étudiée par de nombreux scientifiques. Les résultats de complexité pour cette contrainte ont montré qu'un problème F2|*no-wait*|Cmax est équivalent au problème F2|*RSb*|Cmax où la capacité de stockage est considérée comme nulle (b=0). Ce problème peut être résolu polynomialement par l'algorithme proposé dans (Gilmore et Gomory, 1964), en transformant le problème en un problème de voyageur de commerce avec des distances particulières (Reddi et Ramamoorthy, 1972). Pour le problème à 3 machines F3|*no-wait*|Cmax, Röck (1984) montre que ce problème est NP-Difficile au sens fort.

  Pour les problèmes de type Flow-Shop où on considère une capacité de stockage limitée F2|*b finie*|Cmax, Papadimitriou et Kanellakis (1980) prouvent que le problème est NP-Difficile au sens fort. Dans (Hall et Sriskandarajah, 1996), les auteurs montrent que le problème F3|b=0| Cmax est NP-Difficile au sens fort. Par conséquent, le problème FM|$0 \leq b \leq 1$|Cmax où $M \geq 3$, est NP-Difficile.

  Pour ce qui concerne la contrainte de blocage *RCb*, la résolution des problèmes d'ordonnancement de type Flow-Shop avec cette contrainte a été prouvée dans (Martinez *et al.*, 2006), polynomiale pour $M \leq 3$ machines et NP-Difficile pour $M \geq 5$. Bien que le problème reste ouvert pour les cas avec $M = 4$, certains cas particuliers importants ont été prouvés polynomiaux.

  Enfin, à notre connaissance, très peu de travaux considèrent des systèmes de production soumis simultanément à plusieurs types de blocage. Des premiers résultats de complexité pour les problèmes de Flow-Shop avec blocages *RSb* et *RCb* ont été obtenus (Martinez *et al.*, 2006) : certains problèmes à quatre machines restent polynomiaux. Par contre, à partir de cinq machines, tous les problèmes sont NP-Difficiles.

- **Méthodes exactes**

  Le fait de transformer le problème FM|*no-wait*|Cmax en un problème de voyageur de commerce (Bagchi *et al.*, 2006), permet de gagner considérablement en temps de calcul en utilisant les méthodes exactes proposées dans (Carpaneto et Toth, 1980) et (Carpaneto *et al.*, 1995).

  Pour résoudre le problème FM|*RSb*|Cmax, des Branch and Bound ont été proposés dans (Levner, 1969) et (Ronconi, 2005). Dans (Dutta et Cunningham, 1975), les auteurs proposent une méthode de programmation dynamique pour résoudre de façon optimale des

instances de petite taille pour le problème F2|*b finie*|Cmax. Alors que pour résoudre le problème FM|*b finie*|Cmax où on considère une faible différence entre le nombre de jobs à ordonnancer et la capacité de stockage, Reddi (1976) propose une méthode de programmation dynamique dont la complexité est fonction de cette différence.

Pour ce qui concerne la contrainte de blocage *RCb*, deux modèles mathématiques ont été présentés dans (Martinez, 2005), dont le plus efficace permet de résoudre en temps raisonnable des problèmes jusqu'à 12 jobs / 20 machines, 11 jobs / 50 machines et 10 jobs / 100 machines.

- **Méthodes approchées**

Le problème FM|*no-wait*|Cmax, transformé en un problème de voyageur de commerce, peut être résolu d'une manière efficace avec les heuristiques proposées dans (Lawler *et al*., 1985), (Cirasella *et al*., 2001) et (Johnson *et al*., 2002).

Pour la résolution des problèmes de Flow-Shop avec la contrainte *RSb* par les méthodes approchées, des heuristiques ont été proposées dans (Abadi *et al*. 2000) et (Ribas *et al*, 2011) pour résoudre respectivement les problèmes FM|*RSb*|C^t (minimisation du temps de cycle dans un système de type Flow-Shop sans capacité de stockage) et FM|*RSb*|Cmax. Dans (Carraffa *et al*., 2001), les auteurs proposent un algorithme génétique pour résoudre le problème FM|$b = 0$|C^t. Cet algorithme utilise la diminution de la vitesse du traitement des jobs proposée dans (Abadi *et al*. 2000). Les résultats de cet algorithme fournissent une meilleure performance que les résultats donnés dans (Abadi *et al*. 2000). Dans (Leisten, 1990), une comparaison entre différentes heuristiques montre que l'algorithme NEH proposé par Nawaz, Enscore et Ham (Nawaz *et al*., 1983) pour le Fm||Cmax et adapté au cas sans capacité de stockage, est le plus performant. Dans (Nowicki, 1996), l'auteur a proposé une méthode de recherche tabou pour le problème FM|*b finie, prmu*|Cmax. Parmi les publications scientifiques les plus récentes et qui ont traité le problème de minimisation du makespan d'un système de type Flow-Shop sans capacité de stockage, nous pouvons citer (Wang *et al*., 2006) dont les auteurs ont eu recours à un algorithme génétique.

Pour ce qui concerne la contrainte de blocage *RCb*, plusieurs heuristiques ainsi qu'un recuit simulé sont présentées dans (Martinez, 2005) pour le cas du Flow-Shop jusqu'à 12 jobs et 100 machines. Ce problème a été résolu dans (Yuan et Sauer, 2007) par une métaheuristique « Electromagnetism-like mechanism ».

Le Tableau 2.2 présente un résumé des principaux travaux que nous venons de citer.

Compléxité		
F2| no-wait |Cmax	(Reddi et Ramamoorthy, 1972)	Problème équivalent au F2|*RSb* |Cmax (F2|*b* =0|Cmax) et polynomial selon (Gilmore et Gomory, 1964)
F3|*b* =0| Cmax	(Hall et Sriskandarajah, 1996)	NP-Difficile au sens fort
FM|*RCb* |Cmax	(Martinez *et al* ., 2006)	Polynomial pour $M \leq 3$ machines et NP-Difficile pour $M \geq 5$
F3|*RCb, RSb* |Cmax	(Martinez *et al* ., 2006)	Polynomial
F3|*RSb, RCb* |Cmax	(Martinez *et al* ., 2006)	Polynomial
F4|*RCb, RSb, RCb* |Cmax	(Martinez *et al* ., 2006)	Polynomial
F4|*RCb, RCb, RSb* |Cmax	(Martinez *et al* ., 2006)	NP-Difficile
F4|*RSb, RCb, RCb* |Cmax	(Martinez *et al* ., 2006)	NP-Difficile
F4|*RSb, RCb, RSb* |Cmax	(Martinez *et al* ., 2006)	NP-Difficile
F(M >4)|*RSb, RCb* |Cmax	(Martinez *et al* ., 2006)	NP-Difficile
Méthodes exactes		
FM|*RSb* |Cmax	(Levner, 1969) et (Ronconi, 2005)	Branch and Bound
F2|*b finie* |Cmax	(Dutta et Cunningham, 1975)	Programmation dynamique
FM|*b finie* |Cmax	(Reddi, 1976)	Programmation dynamique
F2|*RCb* |Cmax	(Martinez, 2005)	Deux modèles mathématiques (12 J / 20 M)
Méthodes approchées		
FM|*RSb* |Cmax	(Leisten, 1990)	Comparaison entre différentes heuristiques montre que l'algorithme NEH adapté à la contrainte *RSb* est le plus performant
FM|*RSb* |Cmax	(Ribas *et al.* , 2011)	Heuristiques
FM|*b finie, prmu*| Cmax	(Nowicki, 1996)	Recherche tabou
FM|*RSb* |Cmax	(Wang *et al* ., 2006)	Algorithme génétique
F2|*RCb* |Cmax	(Martinez, 2005)	Plusieurs heuristiques et un recuit simulé (12 J / 100 M)
F2|*RCb* |Cmax	(Yuan et Sauer, 2007)	Métaheuristique « Electromagnetism-like mechanism »

Tab 2.2 : Etat de l'art pour le Flow-Shop avec blocage

2. 1. Flow-Shop hybride

Dans cette partie, nous établissons un état de l'art des travaux concernant les problèmes d'ordonnancement de type Flow-Shop hybride (FH), avec et sans contrainte de blocage et portant sur la minimisation du makespan dans ces systèmes de production.

De la même façon que celle dont nous avons présenté l'état de l'art des problèmes d'ordonnancement de type Flow-Shop classique, nous optons pour la même démarche pour le cas du Flow-Shop hybride. Nous présentons d'abord les résultats de complexité, puis les travaux utilisant des méthodes de résolution exacte et enfin les travaux qui concernent la résolution par des méthodes approchées.

Parmi les travaux qui nous ont aidés à établir un état de l'art des problèmes d'ordonnancement de type Flow-Shop hybride, nous pouvons citer ceux de (Linn et Zhang, 1999), (Vignier *et al*., 1999) et (Ruiz et Vázquez-Rodríguez, 2010).

2. 1. 1. Flow-Shop hybride classique (*Wb*)

- **Complexité**

  Dans (Gupta, 1988), l'auteur s'est intéressé aux problèmes de type Flow-Shop hybride à deux étages (K=2) avec le makespan comme critère d'optimisation. Il a montré dans ce travail que le problème avec plus d'une machine sur au moins un étage FH2(max(M_1,M_2)>1)||Cmax est NP-Difficile. Dans (Hoogeveen *et al*., 1996), les auteurs montrent que le cas préemptif du même problème est aussi NP-Difficile. Par conséquent, les problèmes de type Flow-Shop hybride au cas général (plus de deux étages) sont NP-Difficiles.

- **Méthodes exactes**

  Pour résoudre le problème FH2(M_1=2, M_2=1)||Cmax, un Branch and Bound a été testé sur des exemples de moins de 10 jobs à cause des temps de calcul élevés dans (Arthanary et Ramaswamy, 1971). Dans (Allaoui et Artiba, 2006), les auteurs ont proposé un Branch and Bound pour le problème FH2(M_1=1, M_2=m)||Cmax. Toujours avec les problèmes FH à deux étages, un Branch and Bound basé sur une méthode exacte pour résoudre un problème d'ordonnancement pour machines identiques parallèles a été proposé dans (Haouari *et al*., 2006) pour résoudre le problème FH2(M_1=m1, M_2=m2)||Cmax.

  Pour résoudre les problèmes à K étages FHK||Cmax, une méthode exacte basée sur la modélisation du problème par un programme linéaire mixte (Hunsucker et Brah, 1987), et un Branch and Bound pour les problèmes de petite taille (Brah et Hunsucker, 1991) ont été proposés.

  Dans (Perregaard, 1995), l'auteur propose un nouveau schéma de branchement qui se rapproche de celui proposé dans (Brah et Hunsucker, 1991), du fait que les séquences de tâches sur les machines ne sont pas construites chronologiquement.

  Dans (Carlier et Néron, 2000), les auteurs proposent une procédure de Branch and Bound non classique, dont le schéma de séparation non chronologique est original et permet de traiter des problèmes de taille importante (jusqu'à 150 tâches).

- **Méthodes approchées**

  Pour le problème de Flow-Shop hybride à deux étages avec m machines identiques sur le premier étage et une seule machine sur le deuxième étage FH2(M_1=m, M_2=1)||Cmax, une heuristique basée sur la règle de Johnson a été proposée (Gupta, 1988). Pour le problème inverse FH2(M_1=1, M_2=m)||Cmax, des bornes supérieure et inférieure utilisées dans une

heuristique basée sur une méthode de Branch and Bound a été proposée dans (Gupta et Tunc, 1991). Cette heuristique donne de bonnes solutions quand le nombre de jobs est inférieur à 9.

Pour le problème à deux étages, où le nombre de machines parallèles identiques sur chaque étage est le même FH2(M_1=M_2=m)||Cmax, des méthodes heuristiques ont été proposées pour le cas non-préemptif (Langston, 1987) et (Shen et Chen, 1972), ainsi que dans le cas préemptif (Buten et Shen, 1973).

Pour les problèmes avec plusieurs machines sur les deux étages FH2(M_1=m1, M_2=m2)||Cmax, une heuristique et des bornes inférieures ont été proposées dans (Lee et Vairaktarakis, 1994). Deux heuristiques plus performantes que celle-ci, basées sur le recuit simulé et la recherche tabou ont été proposées dans (Haouari et M'Hallah, 1997) pour ce même problème.

Dans (Riane *et al.*, 1998), les auteurs proposent deux heuristiques pour résoudre un problème réel à trois étages FH3(M_1=1, M_2=2, M_3=1)‖Cmax. L'une repose sur la programmation linéaire et l'autre est basée sur le Branch and Bound. Une expérimentation étendue a prouvé les excellentes performances des heuristiques développées.

Pour le problème de Flow-Shop hybride à K étages FHK||Cmax, des heuristiques ont été proposées dans (Nowicki et Smutnicki, 1998) qui sont basées sur la recherche tabou avec un voisinage basée sur la notion de blocs proposée par (Grabowski *et al.*, 1983). Une étude comparative de six heuristiques pour la résolution de ce problème a été proposée dans (Hunsucker et Shah, 1994). Parmi les travaux qui ont proposé le recuit simulé comme méthodes approchées pour la résolution des problèmes FHK||Cmax, nous pouvons citer (Gourgand *et al*., 1999) et (Jin *et al*., 2006). Quant à l'algorithme génétique, il a été utilisé dans les travaux de (Ruiz *et al*., 2005) et (Kahraman *et al*, 2008).

Dans (Portmann *et al.*, 1998), les auteurs ont proposé une amélioration du Branch and Bound développé dans (Brah et Hunsucker, 1991). Différentes heuristiques sont proposées pour calculer des bornes inférieures initiales. Ces dernières sont améliorées par un algorithme génétique au cours de la résolution. Une étude expérimentale montre que leur approche permet de résoudre optimalement des problèmes de petite taille (N = 10, 15 et K = 2, 3, 5).

Le Tableau 2.3 présente un résumé des principaux travaux que nous venons de citer.

Complexité		
FH2(M_1, M_2)||Cmax	(Gupta, 1988)	Problème NP-Difficile si max(M_1, M_2)>1
FH2(M_1, M_2)|*pmtn*|Cmax	(Hoogeveen *et al*., 1996)	NP-Difficile au sens fort si max(M_1, M_2)>1
Méthodes exactes		
FH2(M_1=1, M_2=*m*)||Cmax	(Allaoui et Artiba, 2006)	B&B
FH2(M_1=*m* 1, M_2=*m* 2)||Cmax	(Haouari *et al*., 2006)	B&B
FHK||Cmax	(Hunsucker et Brah, 1987)	Modélisation par un programme linéaire mixte
FHK||Cmax	(Brah et Hunsucker, 1991)	B&B
FHK||Cmax	(Carlier et Néron, 2000)	B&B original (jusqu'à 150 tâches)
Méthodes approchées		
FH2(M_1=*m* , M_2=1)||Cmax	(Gupta, 1988)	Heuristique basée sur la règle de Johnson
FH2(M_1=1, M_2=*m*)||Cmax	(Gupta et Tunc, 1991)	Bornes supérieure et inférieure utilisées dans une heuristique basée sur une méthode de B&B
FH2(M_1=M_2=*m*)|| Cmax	(Langston, 1987) et (Shen et Chen, 1972)	Heuristiques
FH2(M_1=M_2=*m*)| *pmtn* | Cmax	(Buten et Shen, 1973)	Heuristiques
FH2(M_1=*m* 1, M_2=*m* 2)|| Cmax	(Lee et Vairaktarakis, 1994)	Heuristique et des bornes inférieures
FH2(M_1=*m* 1, M_2=*m* 2)|| Cmax	(Haouari et M'Hallah, 1997)	Deux heuristiques performantes basées sur le recuit simulé et la recherche tabou
FH3(M_1=1, M_2=2, M_3=1)||Cmax	(Riane *et al.* , 1998)	Deux heuristiques très performantes
FHK||Cmax	(Nowicki et Smutnicki, 1998)	Heuristiques basées sur la recherche tabou
FHK||Cmax	(Hunsucker et Shah, 1994)	Une étude comparative de six heuristiques
FHK||Cmax	(Gourgand *et al*., 1999) et (Jin *et al*., 2006)	Recuit simulé
FHK||Cmax	(Ruiz *et al*., 2005) et (Kahraman *et al* , 2007)	Algorithme génétique

Tab 2.3 : Etat de l'art pour le Flow-Shop hybride sans blocage

2. 1. 2. Flow-Shop hybride avec blocage (*RSb*, *RCb*, *RCb**, mixte)

- **Complexité**

Dans (Martinez, 2005), l'auteur a montré que les problèmes de type Flow-Shop hybride avec blocage sont NP-Difficiles, puisque si l'on considère le cas particulier où les temps opératoires de tous les jobs sur les *k* étages ($k \geq 2$) sont égaux à zéro, le problème devient équivalent au problème d'ordonnancement d'un système à un seul étage avec des machines parallèles et identiques. Et comme ce dernier problème a été prouvé NP-Difficile dans (Karp, 1972), alors l'ordonnancement d'un FH avec blocage est aussi NP-Difficile.

- **Méthodes exactes**

Pour résoudre les problèmes de Flow-Shop hybrides avec blocage, plusieurs auteurs ont proposé des méthodes exactes, comme par exemple la modélisation par un programme linéaire en nombres entiers utilisée dans (Sawik, 2000) pour résoudre les problèmes FHK|*b*=0|C_{max} et FHK|*b finie*|*Cmax.* Il a présenté un exemple numérique pour des

instances jusqu'à 10 jobs, 3 étages avec 2, 3 et 2 machines par étage respectivement et des capacités de stockage entre les étages de b=0 et b=3 jobs.

Parmi les autres travaux qui ont proposé comme technique de résolution la programmation linéaire en nombres entiers pour la minimisation du makespan des problèmes de FH avec la contrainte de blocage *RSb*, nous pouvons citer les travaux de (Liu et Karimi, 2008) et (Gicquel *et al*., 2012).

Pour ce qui concerne la contrainte de blocage *RCb*, un modèle mathématique basé sur la discrétisation du temps a été proposé dans (Martinez, 2005) pour le problème FH2(M_1=3, M_2=1)||Cmax. Les résultats obtenus sur des problèmes de différentes tailles générés aléatoirement ont montré que le facteur le plus influent est la plage de variation des durées opératoires. Pour des petites plages horaires, des problèmes jusqu'à 15 jobs et 3 machines parallèles au premier étage sont résolus en un temps raisonnable.

Pour résoudre les problèmes à K étages FHK|*RCb*|Cmax, les auteurs dans (Yuan *et al*., 2009) proposent un modèle linéaire en nombres entiers pour minimiser le makespan et qui peut résoudre des problèmes jusqu'à 10 jobs et 5 étages.

- **Méthodes approchées**

Pour le cas du blocage classique *RSb*, Sawik (1993, 1995) a proposé une heuristique pour le Flow-Shop à plusieurs étages, avec et sans capacité de stockage respectivement. Dans (Thornton et Hunsucker, 2004), les auteurs ont développé une heuristique pour le Flow-Shop hybride à plusieurs étages sans capacité de stockage.

Dans (Wardono et Fathi, 2004), les auteurs ont présenté une méthode de recherche tabou pour le Flow-Shop avec capacité de stockage limitée. Pour le même problème FHK|*b finie*|Cmax, un algorithme génétique a été développé dans (Wang *et al.*, 2006).

Pour ce qui concerne la contrainte de blocage *RCb*, quatre algorithmes constructifs qui sont des adaptations des algorithmes proposés dans (Gupta, 1988) pour le Flow-Shop hybride à deux étages et (Nawaz *et al*., 1983) pour le Flow-Shop classique, et un recuit simulé utilisant plusieurs voisinages ont été proposés dans (Martinez, 2005). La meilleure solution obtenue par ces heuristiques a été utilisée comme solution initiale pour le recuit simulé. Les résultats ont montré que deux heuristiques MNEH et GUPMEH sont de bonnes méthodes de résolution et ont un pourcentage d'erreur relative faible pour des instances jusqu'à 100 jobs et 3 machines. En appliquant le recuit simulé, les résultats ont été encore améliorés, fournissant ainsi des résultats presque optimaux pour des instances avec un grand nombre de jobs et un grand nombre de machines sur le premier étage.

Enfin, pour les articles qui ont traité différents types de contraintes dans un même système de production, nous pouvons citer les travaux de (Grabowski et Pempera, 1999) qui traitent un problème réel d'une industrie de construction modélisé en un problème de type Flow-Shop hybride soumis à une contrainte de blocage mixte (sans-attente/*RSb*). Pour ce problème, une méthode tabou est proposée. A notre connaissance, aucun autre auteur ne s'intéresse à un problème de type Flow-Shop hybride avec blocage mixte.

Le Tableau 2.4 présente un résumé des principaux travaux que nous venons de citer.

Compléxité		
FH|*bloc* |Cmax	(Martinez, 2005)	Problème NP-Difficile, puisqu'il est plus complexe qu'un système à un seul étage avec des machines parallèles et identiques qui a été prouvé NP-Difficile dans (Karp, 1972)
Méthodes exactes		
FHK|b =0| Cmax et FHK|*b finie*| Cmax	(Sawik, 2000)	Modélisation par PLNE jusqu'à 10 jobs, 3 étages avec M_1=2, M_2=3 et M_3=2
FHK|*RS b* |Cmax	(Liu et Karimi, 2008) et (Gicquel *et al* ., 2012)	PLNE
FH2(M_1=3, M_2=1)||Cmax	(Martinez, 2005)	Modèle mathématique basé sur la discrétisation du temps, problèmes résolus jusqu'à 15 jobs
FHK|*RCb* |Cmax	(Yuan *et al* ., 2009)	PLNE (jusqu'à 10 jobs et 5 étages)
Méthodes approchées		
FHK|*b finie* |Cmax	(Sawik, 1993)	Heuristique
FHK|b =0|Cmax	(Thornton et Hunsucker, 2004) et (Sawik, 1993)	Heuristique
FHK|*b finie* |Cmax	(Wardono et Fathi, 2004)	Recherche tabou
FHK|*b finie* |Cmax	(Wang *et al.* , 2006)	Algorithme génétique
F2|*RCb* |Cmax	(Martinez, 2005)	Adaptations des algorithmes proposés dans (Gupta, 1988) et (Nawaz *et al* ., 1983), et un recuit simulé utilisant plusieurs voisinages : performant jusqu'à 100 jobs et 3 machines au premier étage.
FHK|*no-wait /RSb* |Cmax	(Grabowski et Pempera, 1999)	Recherche tabou

Tab 2.4 : Etat de l'art pour le Flow-Shop hybride avec blocage

2. 1. Conclusion

Dans ce chapitre, nous avons présenté les différentes méthodes utilisées pour la résolution des problèmes d'ordonnancement de type Flow-Shop et Flow-Shop hybride, avec et sans blocage.

Nous avons également présenté les différentes études qui ont été réalisées pour résoudre ces types de problèmes.

Cette étude de l’état de l’art nous a permis de faire un tour d’horizon assez large des approches de résolution développées et permet de dégager les méthodes actuellement les plus performantes par type de problème. Nous avons aussi constaté que différents cas classiques de ces systèmes ont été largement étudiés, mais pour les cas avec blocage mixte, les travaux existants sont beaucoup moins nombreux.

Dans les chapitres suivants, nous apportons donc notre contribution à la résolution de ce type de problème en proposant des résultats de complexité, des bornes inférieures, ainsi que des méthodes exactes et approchées.

Deuxième partie
Le Flow-Shop avec blocage mixte

Introduction

Dans cette partie nous nous intéressons à la minimisation du makespan dans des systèmes de production de type Flow-Shop avec différents types de blocage entre les machines.

Cette partie est organisée de la façon suivante. Dans le chapitre 3, nous présentons une étude de la complexité des problèmes de type Flow-Shop avec les différentes contraintes de blocage. Dans ce chapitre, nous démontrons quelques nouveaux théorèmes portant sur cette complexité.

Nous proposons ensuite, dans le chapitre 4, une méthode de résolution exacte basée sur la modélisation linéaire en nombres entiers du problème.

Dans le chapitre 5, nous proposons une borne inférieure, des méthodes heuristiques et un algorithme génétique comme métaheuristique.

Chapitre 3
Complexité

3. 1. Introduction

Afin d'avoir une idée de la difficulté à résoudre un problème, il est important d'étudier sa complexité. Dans ce chapitre, nous étudions la complexité des problèmes de type Flow-Shop avec la nouvelle contrainte de blocage *RCb** ou lorsque différents types de contraintes de blocage sont présentes. Nous présentons d'abord les graphes disjonctifs classique et modifié pour prendre en compte les différentes contraintes de blocage. Ensuite, nous proposons des résultats portant sur la complexité des systèmes de type Flow-Shop avec uniquement la contrainte *RCb** ou comportant des contraintes de blocage différentes d'une machine à l'autre.

3. 2. Modélisation

La représentation par le graphe disjonctif (Roy et Sussman, 1964) est souvent utilisée pour modéliser les problèmes de planification d'ateliers. Un de ses avantages principaux est qu'il est facile à construire et représente bien les différentes contraintes entre les temps de début et de fin des opérations.

3. 2. 1. Graphe disjonctif

Dans un graphe disjonctif (Figure 3.1), on trouve :

- les nœuds qui représentent :
  - les débuts des opérations du problème d'ordonnancement notées O_{ij} ;
  - les opérations fictives qui correspondent à la fin de chaque job, noté Φ_i ;
  - les dates du début et de la fin de l'ordonnancement, notées respectivement O et F.

- les arcs conjonctifs qui représentent les contraintes de précédence entre deux opérations consécutives (O_{ik} , O_{ik+1}), valuées par la durée P_{ik} de l'opération O_{ik} (les arcs reliant le nœud O (*resp. F*) à la première (*resp.* dernière) opération de chaque job ont un poids nul).
- les arcs disjonctifs qui représentent des conflits potentiels d'utilisation de ressources. Deux opérations reliées par un tel arc ne doivent pas s'exécuter en même temps (Mati, 2002). Ainsi, toutes les opérations qui doivent être exécutées par la même machine sont connectées par un arc disjonctif.

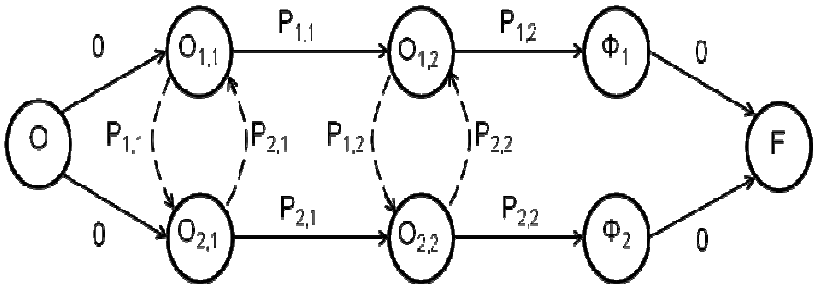


Figure 3.1. Graphe disjonctif classique

Pour résoudre un problème d'ordonnancement, il s'agit d'orienter les arcs disjonctifs. Pour que l'ordonnancement soit parfaitement défini, il faut que l'arbitrage soit complet (toutes les disjonctions sont arbitrées) et compatible (le graphe est sans circuit). Le makespan est donné par la longueur du plus long chemin du sommet source *O* au sommet puits *F*, appelé chemin critique (Figure 3.2).

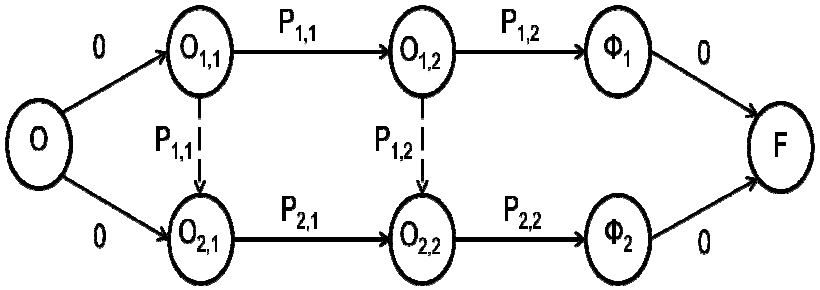


Figure 3.2. Graphe disjonctif classique orienté

Soit $S_{i,k}$ la date de début de l'opération du job J_i sur la machine M_k, de durée $P_{i,k}$. On peut représenter les deux types de contraintes de précédence par les équations suivantes :

- La contrainte de précédence entre deux opérations successives d'un même job :

$$S_{i,k} \geq S_{i,k-1} + P_{i,k-1}\ ,\ \forall i \in \{1,..,N\}, \forall k \in \{2,..,M\} \qquad 3.1$$

- La contrainte de précédence entre deux opérations qui doivent être exécutées par la même machine :

$$S_{i,k} \geq S_{i-1,k} + P_{i-1,k}\ ,\ \forall i \in \{2,..,N\}, \forall k \in \{1,..,M\} \qquad 3.2$$

3. 2. 2. Graphe disjonctif avec blocage *RSb*

Pour traiter la contrainte de blocage *RSb*, Mati (2002) a étendu la notion de graphe disjonctif. Dans ce cas, chaque arc disjonctif de la représentation classique est remplacé par un couple disjonctif. Plus précisément, pour chaque paire d'opération O_{ik} et $O_{i'k'}$ partageant la même machine, un couple disjonctif de deux arcs est introduit : l'un du sommet $O_{i(k+1)}$ à $O_{i'k'}$ et un autre de $O_{i'(k'+1)}$ à O_{ik}. Le poids des couples disjonctifs est zéro. Ceci signifie que l'opération peut commencer aussitôt que la machine est relâchée (Figure 3.3).

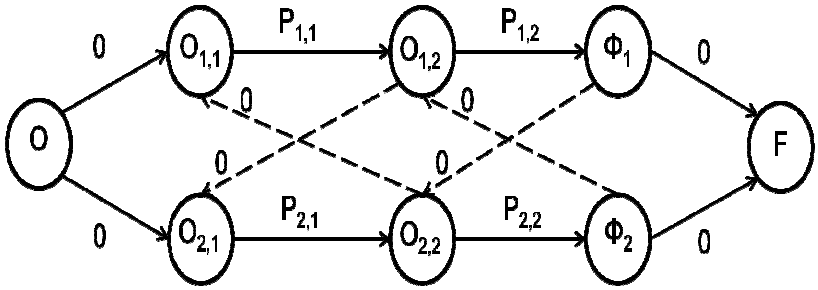


Figure 3.3. Graphe disjonctif avec blocage RSb

Dans ce cas de blocage (*RSb*), la contrainte de précédence entre deux opérations qui doivent être exécutées par la même machine devient :

$$S_{i,k} \geq S_{i-1,k+1}\ ,\ \forall i \in \{2,..,N\}, \forall k \in \{1,..,M-1\} \qquad 3.3$$

3. 2. 3. Graphe disjonctif avec blocage *RCb*

Dans le cas d'un blocage *RCb*, Martinez *et al*. (2006) ont étendu la notion précédente. Dans cette représentation, un arc disjonctif représente le fait que la date de début de l'opération du job J_2 sur la machine M_1 doit être supérieure ou égale à la date à laquelle le job précédent J_1 quitte la machine M_2 (Figure 3.4).

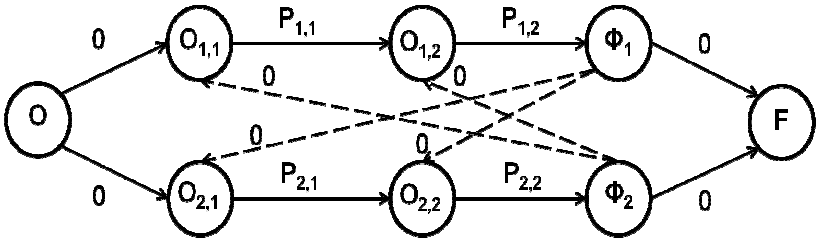


Figure 3.4. Graphe disjonctif avec blocage RCb

Dans le cas d'un blocage *RCb*, la contrainte de précédence entre deux opérations qui doivent être exécutées par la même machine devient :

$$S_{i,k} \geq S_{i-1,k+2}\ ,\ \forall\, i \in \{2,..,N\}, \forall\, k \in \{1,..,M-2\} \qquad 3.4$$

3. 2. 4. Graphe disjonctif avec blocage *RCb**

Nous avons également transformé le graphe disjonctif pour prendre en compte le blocage *RCb**. Dans cette représentation, un arc disjonctif représente le fait que la date de début de l'opération du job J_2 sur la machine M_1 doit être supérieure ou égale à la date à laquelle l'opération du job précédent J_1 sur la machine M_2 soit finie.

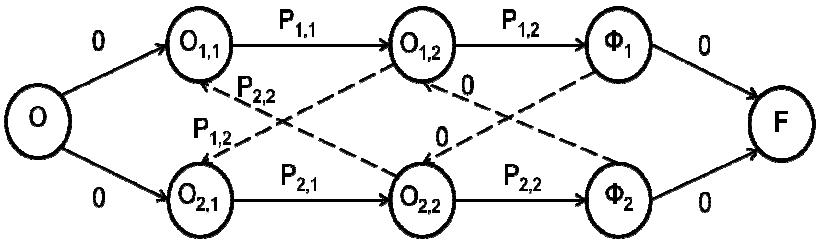


*Figure 3.5. Graphe disjonctif avec blocage RCb**

Dans le cas d'un blocage *RCb**, la contrainte de précédence entre deux opérations qui doivent être exécutées par la même machine devient :

$$S_{i,k} \geq S_{i-1,k+1} + P_{i-1,k+1}\ ,\ \forall i \in \{2,..,N\}, \forall k \in \{1,..,M-1\} \qquad 3.5$$

On peut constater que la représentation du blocage entre les deux dernières machines est la même dans le cas d'un blocage *RCb* et d'un blocage *RCb**. On a donc le résultat suivant.

Théorème 1. *Les problèmes* ***FM|… , … , RCb(M-1,M)|Cmax*** *sont équivalents aux problèmes* ***FM|… , … , RCb*(M-1,M)|Cmax***.

Preuve : Quel que soit le type de contrainte de blocage (*RCb* ou *RCb**) entre les deux dernières machines, la date de début de l'opération du job J_i sur la machine M_{M-1} doit être supérieure ou égale à la date à laquelle l'opération du job précédent J_{i-1} sur la dernière machine M_M soit finie (Figure 3.6).

En effet, la contrainte de blocage *RCb* entre les machines *M*-1 et *M* ne s'applique pas vraiment puisque il n'y a pas de contraintes de démarrage sur les opérations de la dernière machine *M*.

D'où l'équation de précédence dans les deux cas est la suivante :

$$S_{i,M-1} \geq S_{i-1,M} + P_{i-1,M}\ ,\ \forall i \in \{2,..,N\} \qquad 3.6$$

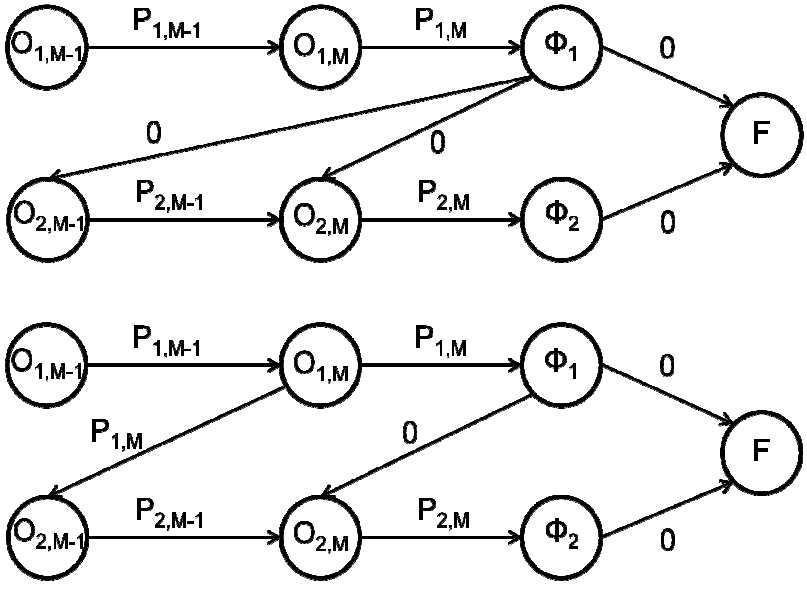


Figure 3.6. Contraintes RCb/RCb entre les deux dernières machines*

3. 3. Complexité

Nous avons présenté dans le chapitre 2 un état de l'art des travaux qui ont étudié la complexité des problèmes de type Flow-Shop classique avec blocage *RSb, RCb* ou sans blocage, ainsi que des exemples avec blocage mixte. Nous avons remarqué que les problèmes contenant à la fois les contraintes de blocage *RSb* et *RCb* sont plus faciles que ceux dont toutes les machines sont soumises à la même contrainte de blocage. Dans cette partie, nous étudions donc la complexité de problèmes mixant différentes contraintes de blocage.

La première partie de cette section est consacrée à la présentation de la complexité des problèmes de type Flow-Shop avec la nouvelle contrainte de blocage *RCb**. Ensuite, nous considérons différents cas de blocage mixte.

3. 3. 1. Flow-Shop avec la contrainte de blocage *RCb**

Flow-Shop à 2 machines avec blocage *RCb**

La complexité d'un système de production de type Flow-Shop à deux machines avec la contrainte de blocage *RCb** est la même que celui avec la contrainte de blocage *RCb*.

Corollaire 1. *Le problème* ***F2\|RCb*(1,2)\|Cmax*** *est polynomial.*

Preuve : En effet, d'après le théorème 1, pour un nombre de machines $M = 2$, Le problème F2|*RCb*(1,2)|Cmax est équivalent au problème F2|*RCb**(1,2)|Cmax. Or, le problème F2|*RCb*(1,2)|Cmax a été prouvé, dans (Martinez *et al*., 2006), équivalent au problème 1||Cmax qui est polynomial. Par conséquent, le problème F2|*RCb**(1,2)|Cmax est polynomial et l'ordre dans lequel les jobs sont ordonnancés n'influe pas sur la valeur finale du makespan qui est donnée par l'équation 3.7 :

$$C_{\max} = \sum_{i=1}^{N} (P_{i,1} + P_{i,2}) \qquad 3.7$$

Flow-Shop à 3 machines avec blocage *RCb**

L'ordonnancement d'un système de production de type Flow-Shop à trois machines avec la contrainte de blocage *RCb** est de même complexité que l'ordonnancement du problème F2|*RSb*|Cmax.

Théorème 2. *Le problème* ***F3|RCb*(1,2) , RCb*(2,3)|Cmax*** *est polynomial.*

Preuve : Supposons que la séquence d'une solution donnée pour le problème F3|*RCb**|Cmax est composée de N jobs, $J = \{J_1, J_2, \ldots, J_N\}$, de trois opérations chacun ($O_{i,1}$, $O_{i,2}$ et $O_{i,3}$) et que $S_{i,k}$ est la date de début de l'opération du job J_i, sur la machine M_k et de durée $P_{i,k}$.

La valeur d'un plus long chemin de O jusqu'à l'opération $O_{i,3}$ pour un job donné J_i est déterminé par l'équation 3.8 :

$$S_{i,3} = \max\left\{\Phi_{i-1} + 0\,,\, S_{i,2} + P_{i,2}\right\} \qquad 3.8$$

Comme $S_{i,2} \geq \Phi_{i-1}$ (car $S_{i,2} \geq S_{i-1,3} + P_{i-1,3} = \Phi_{i-1}$) et $P_{i,2} \geq 0$ (les temps d'exécution sont supposés non-négatifs), $S_{i,3} = S_{i,2} + P_{i,2}$. Par conséquent, tous les sommets $O_{i,2}$ appartiennent à un chemin critique (Figure 3.7).

Un plus long chemin L partant de $O_{i,2}$ et arrivant à $O_{i+1,2}$ d'un job donné J_i est déterminé par l'équation 3.9 :

$$L = P_{i,2} + \max\left\{P_{i+1,1}\,,\, P_{i,3}\right\} \qquad 3.9$$

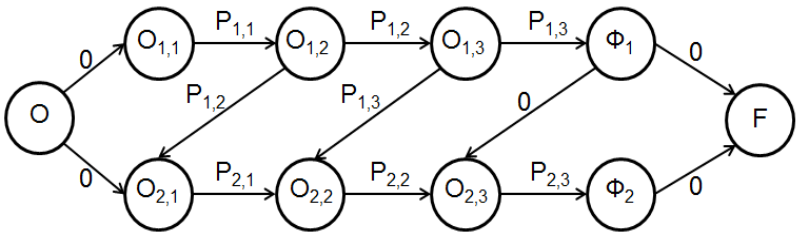


Figure 3.7. Graphe conjonctif pour le problème F3|RCb|Cmax avec 2 jobs*

Alors, la longueur du chemin critique est donnée par :

$$C_{\max} = P_{1,1} + \sum_{i=1}^{N-1} \max\left\{P_{i+1,1}\,,\, P_{i,3}\right\} + P_{N,3} + \sum_{i=1}^{N} P_{i,2} \qquad 3.10$$

Le dernier terme peut être ignoré car il est constant et ne change pas avec la séquence. Par conséquent, ce problème est équivalent au problème F2|*RSb*|Cmax avec les machines M_1 *et* M_3, qui peut être transformé en un problème de voyageur de commerce avec des distances

particulières (Reddi et Ramamoorthy, 1972), lequel peut être résolu polynomialement par l'algorithme de (Gilmore et Gomory, 1964).

CQFD

Flow-Shop à 4 machines avec blocage *RCb**

Théorème 3. *Le problème* ***F4\| RCb*(1,2) , RCb*(2,3) , RCb*(3,4)\|Cmax*** *est NP-Difficile.*

Preuve : On remarque que le problème F4|*RCb**|Cmax contient au moins une partie (encadrée en traits discontinus dans la Figure 3.8) qui représente un problème F3||Cmax.

Dans ce nouveau problème F3||Cmax, les machines α, β *et* γ représentent respectivement les sommets ($O_{1,2}$, $O_{2,1}$), ($O_{1,3}$, $O_{2,2}$) et ($O_{1,4}$, $O_{2,3}$).

Ce type de problèmes a été prouvé NP-Difficile dans (Garey et al., 1976).

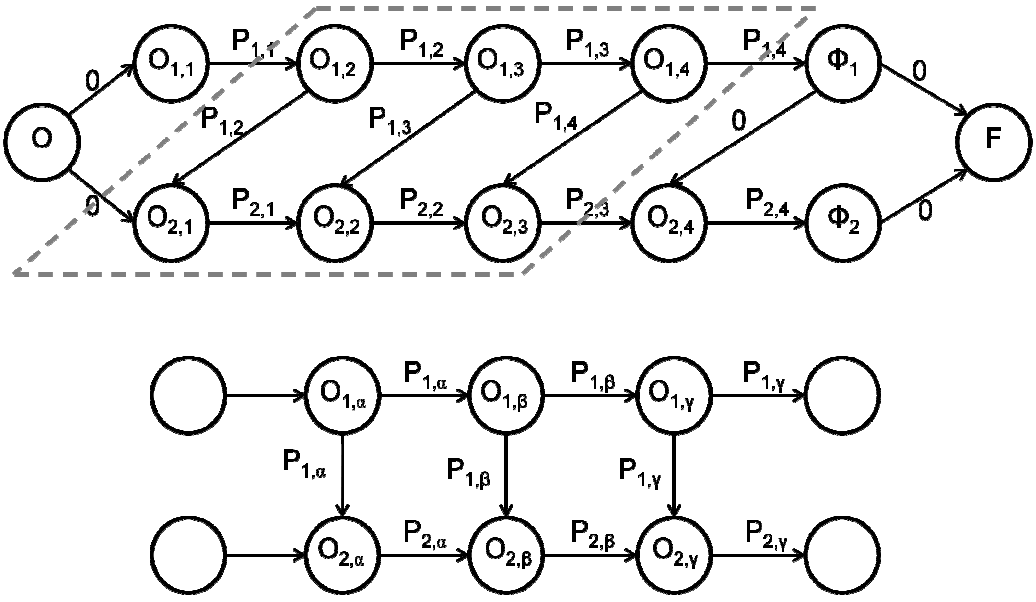


Figure 3.8. Graphe conjonctif pour le problème F4|RCb|Cmax avec 2 jobs*

De ce fait, quel que soit le nombre de jobs, des problèmes ayant au moins quatre machines soumis à la contrainte de blocage *RCb** sont de complexité NP-Difficile.

3. 3. 2. Flow-Shop avec blocage mixte (2 contraintes)

La complexité des problèmes où des contraintes de blocage *RSb* et *RCb* se trouvent mélangées a été prouvée inférieure à la complexité des problèmes avec un seul type de contraintes entre toutes les machines (Martinez *et al*., 2006). Ceci nous a poussé à définir la complexité d'autres problèmes avec différentes combinaisons de contraintes de blocage.

On notera par la suite le cas classique où il n'y a aucune contrainte de blocage par *Wb* (*Without blocking*).

Flow-Shop à 4 machines avec blocage *RCb, Wb, RCb*

Théorème 4. *Le problème* ***F4|RCb(1-2), Wb(2-3), RCb(3-4)|Cmax*** *est polynomial*.

Preuve : Supposons que la séquence d'une solution donnée pour le problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax est composée de N jobs, $J = \{J_1, J_2, \ldots, J_N\}$, de quatre opérations chacun ($O_{i,1}$, $O_{i,2}$, $O_{i,3}$ et $O_{i,4}$) et que $S_{i,k}$ est la date de début de l'opération du job J_i, sur la machine M_k et de durée $P_{i,k}$.

La valeur d'un plus long chemin de O jusqu'à l'opération $O_{i,2}$ pour un job donné J_i est déterminé par l'équation 3.11 :

$$S_{i,2} = \max\left\{S_{i-1,2} + P_{i-1,2}, S_{i,1} + P_{i,1}\right\} \qquad 3.11$$

Comme $S_{i,1} \geq S_{i\text{-}1,2} + P_{i\text{-}1,2}$ et $P_{i,1} \geq 0$ (les temps d'exécution sont supposés non-négatifs), $S_{i,2} = S_{i,1} + P_{i,1}$. Par conséquent, les machines M_1 et M_2 peuvent être fusionnées en une machine composée α sans perdre aucune information sur la date de début de l'opération $O_{i,1}$ de chaque job J_i. Le temps d'exécution de la machine composée α est égal à la somme des temps d'exécution sur les machines M_1 et M_2.

La valeur d'un plus long chemin de O jusqu'à l'opération $O_{i,4}$ pour un job donné J_i est déterminé par l'équation 3.12 :

$$S_{i,4} = \max\left\{\Phi_{i-1} + 0, S_{i,3} + P_{i,3}\right\} \qquad 3.12$$

Comme $S_{i,3} \geq \Phi_{i\text{-}1}$ et $P_{i,3} \geq 0$, $S_{i,4} = S_{i,3} + P_{i,3}$. Par conséquent, les machines M_3 et M_4 peuvent être fusionnées en une machine composée β sans perdre aucune information sur la date de début de l'opération $O_{i,3}$ de chaque job J_i. Le temps d'exécution de la machine composée β est égal à la somme des temps d'exécution sur les machines M_3 et M_4.

Ainsi, le problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax peut-être transformé en un problème F2|*RSb*| C_{max} qui peut être résolu en temps polynomial (Reddi et Ramamoorthy, 1972). Par conséquent, le problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax est polynomial.

CQFD

La solution optimale pour le problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax (*i.e.* les dates de début $S_{i,k}$ de chaque job J_i sur chaque machine M_k) peut être calculée de la façon suivante :

- $S_{i,1} = S'_{i,\alpha}$ $\quad \forall\ i$
- $S_{i,2} = S'_{i,\alpha} + P_{i,1}$ $\quad \forall\ i$
- $S_{i,3} = S'_{i,\beta}$ $\quad \forall\ i$
- $S_{i,4} = S'_{i,\beta} + P_{i,3}$ $\quad \forall\ i$

où $S'_{i,j}\ \ \forall\ j \in \{\alpha, \beta\}$ est la date de début du job J_i sur la machine M_j dans la solution optimale du problème F2|*RSb*|Cmax équivalent.

La Figure 3.9 illustre la preuve du théorème 4 sur un exemple à deux jobs.

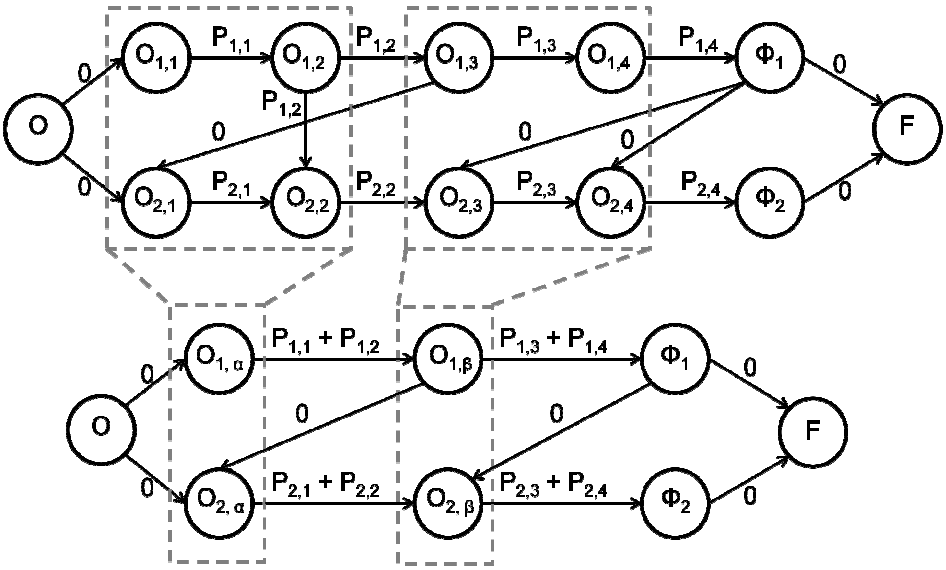


Figure 3.9. Graphe conjonctif pour le problème F4|RCb(1-2), Wb(2-3), RCb(3-4)|Cmax avec 2 jobs

Corollaire 2 : *Les problèmes* ***F3|RCb(1-2), Wb(2-3)|C_{max}*** *et* ***F3|Wb(1-2), RCb(2-3)|C_{max}*** *sont polynomiaux*.

Preuve : En effet, d'après le théorème 4, le problème F3|*RCb*(1-2), *Wb*(2-3)|Cmax (resp. F3|*Wb*(1-2), *RCb*(2-3)|Cmax) correspond au cas particulier du problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax, dans lequel les temps d'exécution de tous les jobs sur la machine M_4 (resp. M_1) sont nuls.

Flow-Shop à 4 machines avec blocage *RCb*, Wb, RCb**

Théorème 5. *Le problème* ***F4|RCb*(1-2), Wb(2-3), RCb*(3-4)|Cmax*** *est polynomial*.

Preuve : Tout comme les cas du problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax, les machines M_1 et M_2, ainsi que les machines M_3 et M_4, peuvent être fusionnées en deux machines composées α et β respectivement, sans perdre aucune information sur les dates de début des opérations $O_{i,1}$ et $O_{i,3}$ de chaque job J_i . Le temps d'exécution de la machine composée α (resp. β) est égal à la somme des temps d'exécution sur les machines M_1 et M_2 (resp. M_3 et M_4). Ainsi, le problème F4|*RCb**(1-2), *Wb*(2-3), *RCb**(3-4)|Cmax peut-être transformé en un problème F2||Cmax qui peut être résolu en temps polynomial par l'algorithme de Johnson (Johnson, 1954). Par conséquent, le problème F4|*RCb**(1-2), *Wb*(2-3), *RCb**(3-4)|Cmax est polynomial.

CQFD

La solution optimale pour le problème F4|*RCb**(1-2), *Wb*(2-3), *RCb**(3-4)|Cmax (*i.e.* les dates de début $S_{i,k}$ de chaque job J_i sur chaque machine M_k) peut être calculée de la façon suivante :

- $S_{i,1} = S'_{i,\alpha}$ $\quad \forall\ i$
- $S_{i,2} = S'_{i,\alpha} + P_{i,1}$ $\quad \forall\ i$
- $S_{i,3} = S'_{i,\beta}$ $\quad \forall\ i$
- $S_{i,4} = S'_{i,\beta} + P_{i,3}$ $\quad \forall\ i$

où $S'_{i,j}\ \forall\ j \in \{\alpha, \beta\}$ est la date de début du job J_i sur la machine M_j dans la solution optimale du problème F2||Cmax équivalent.

La Figure 3.10 illustre la preuve du théorème 5 sur un exemple à deux jobs.

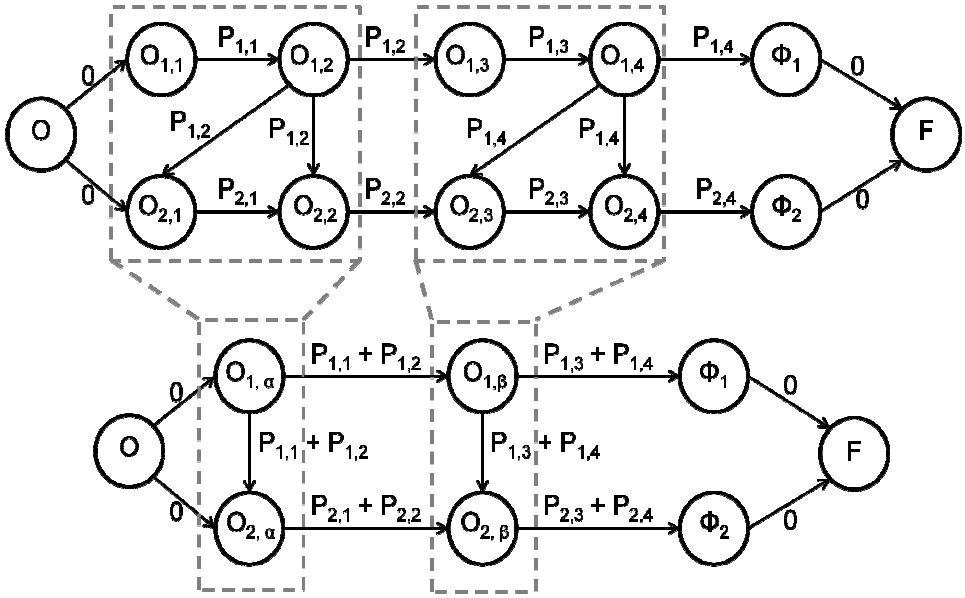


Figure 3.10. Graphe conjonctif pour le problème F4|RCb(1-2), Wb(2-3), RCb*(3-4)|Cmax avec 2 jobs*

Corollaire 3 : *Les problèmes* ***F*3|*RCb**(1-2), *Wb*(2-3)|*Cmax*** *et* ***F*3|*Wb*(1-2), *RCb**(2-3)|*Cmax*** *sont polynomiaux.*

Preuve : En effet, d'après le théorème 5, le problème F3|*RCb**(1-2), *Wb*(2-3)|Cmax (resp. F3|*Wb*(1-2), *RCb**(2-3)|Cmax) correspond au cas particulier du problème F4|*RCb**(1-2), *Wb*(2-3), *RCb**(3-4)|Cmax, dans lequel les temps d'exécution de tous les jobs sur la machine M_4 (resp. M_1) sont nuls.

Corollaire 4. *Les problèmes* ***F*(M>3)|…, *Wb*(M_m-M_{m+1}), *Wb*(M_{m+1}-M_{m+2}), …|*Cmax*** *et* ***F*(M>3)|…, *RSb*(M_m-M_{m+1}), *RSb*(M_{m+1}-M_{m+2}), …|*Cmax*** *sont NP-Difficiles*.

Preuve : En effet, les problèmes *F*3|*Wb*(M_1-M_2), *Wb*(M_2-M_3)|Cmax et *F*3|*RSb*(M_1-M_2), *RSb*(M_2-M_3)|Cmax ont été prouvés NP-Difficiles dans (Garey *et al*., 1976) et (Hall et Sriskandarajah, 1996) respectivement. Donc, quel que soit la nouvelle contrainte ajoutée avant ou après deux contraintes identiques et successives (*Wb* ou *RSb*), la complexité de ces problèmes reste NP-Difficile

Corollaire 5. *Les problèmes* ***F*(M>4)|…, *RCb**(M_m-M_{m+1}), *RCb**(M_{m+1}-M_{m+2}) , *RCb**(M_{m+2}-M_{m+3}), …|*Cmax*** *sont NP-Difficiles*.

Preuve : En effet, d'après le théorème 3, le problème F4|*RCb**(M_1-M_2), *RCb**(M_2-M_3), *RCb**(M_3-M_4)|Cmax est NP-Difficile. Donc, quelle que soit la nouvelle contrainte ajoutée avant ou après ces trois contraintes successives (*RCb**), la complexité de ce problème reste NP-Difficile.

Flow-Shop à 4 machines avec blocage *Wb, RCb*, Wb*

Théorème 6. *Le problème* ***F4\|Wb(1-2), RCb*(2-3), Wb(3-4)\|Cmax*** *est NP-Difficile.*

Preuve : La valeur d'un plus long chemin de *O* jusqu'à l'opération $O_{i,3}$ pour un job donné J_i est déterminé par l'équation 3.13 :

$$S_{i,3} = \max\left\{S_{i-1,3} + P_{i-1,3}, S_{i,2} + P_{i,2}\right\} \qquad 3.13$$

Comme $S_{i,2} \geq S_{i\text{-}1,3} + P_{i\text{-}1,3}$ et $P_{i,2} \geq 0$, $S_{i,3} = S_{i,2} + P_{i,2}$. Par conséquent, les machines M_2 et M_3 peuvent être fusionnées en une machine composée α sans perdre aucune information sur la date de début de l'opération $O_{i,2}$ de chaque job J_i. Le temps d'exécution de la machine composée α est égal à la somme des temps d'exécution sur les machines M_2 et M_3. Ainsi, le problème F4|*Wb*(1-2), *RCb**(2-3), *Wb*(3-4)|Cmax peut-être transformé en un problème $F3||\ C_{max}$ lequel est NP-Difficile (Garey et al., 1976). Par conséquent, le problème F4|*Wb*(1-2), *RCb**(2-3), *Wb*(3-4)|Cmax est NP-Difficile.

La Figure 3.11 illustre la preuve du théorème 6 sur un exemple à deux jobs.

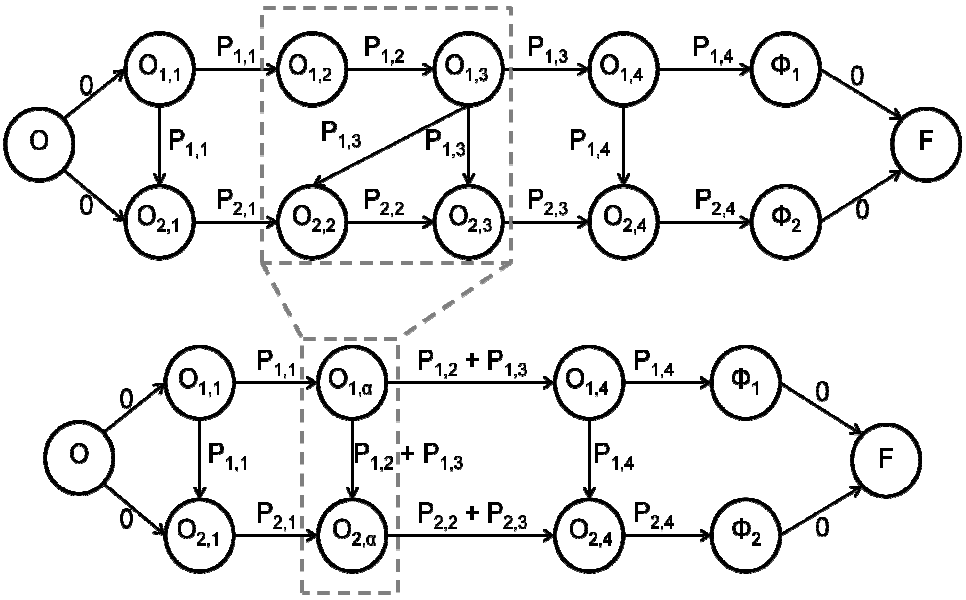


Figure 3.11. Graphe conjonctif pour le problème F4|Wb(1-2), RCb(2-3), Wb(3-4)|Cmax avec 2 jobs*

3. 3. 3. Flow-Shop avec blocage mixte (3 contraintes)

Flow-Shop à 4 machines avec blocage *RCb*, Wb, RCb*

Corollaire 6. *Le problème* ***F4|RCb*(1-2), Wb(2-3), RCb(3-4)|Cmax*** *est polynomial*.

Preuve : En effet, d'après le théorème 1, on peut remplacer la dernière contrainte *RCb* par la contrainte *RCb**, par conséquent ce problème est équivalent au problème F4|*RCb**(1-2), *Wb*(2-3), *RCb**(3-4)|Cmax, lequel est prouvé polynomial d'après le théorème 5.

Flow-Shop à 4 machines avec blocage *RCb, Wb, RCb**

Corollaire 7. *Le problème* ***F4|RCb(1-2), Wb(2-3), RCb*(3-4)|Cmax*** *est polynomial*.

Preuve : En effet, d'après le théorème 1, on peut remplacer la dernière contrainte *RCb** par la contrainte *RCb*, par conséquent ce problème est équivalent au problème F4|*RCb*(1-2), *Wb*(2-3), *RCb*(3-4)|Cmax, lequel est prouvé polynomial d'après le théorème 4.

Flow-Shop à 4 machines avec blocage *RCb, RSb, RCb**

Corollaire 8. *Le problème* ***F4|RCb(1-2), RSb(2-3), RCb*(3-4)|Cmax*** *est polynomial*.

Preuve : En effet, d'après le théorème 1, on peut remplacer la dernière contrainte *RCb** par la contrainte *RCb*, par conséquent ce problème est équivalent au problème F4|*RCb*(1-2), *RSb*(2-3), *RCb*(3-4)|Cmax, lequel est prouvé polynomial dans (Martinez *et al*., 2006).

3. 4. Conclusion

Quelques résultats de complexité que nous avons démontré dans ce chapitre ont été présentés à la conférence ***International Federation of Automatic Control 'IFAC11'*** *(Trabelsi et al., 2011a).*

Dans ce chapitre, nous avons montré que le problème d'ordonnancement de type Flow-Shop avec la contrainte de blocage *RCb** est polynomial pour $M \leq 3$ machines et devient NP-Difficile à partir de 4 machines. Pour les problèmes qui contiennent des contraintes de blocage différentes entre les machines, certains cas à 3 et 4 machines ont été prouvés polynomiaux (Tableau 3.1).

*RCb**	F2|*RCb* *|Cmax	Polynomial
	F3|*RCb* *|Cmax	Polynomial
	F4|*RCb* *|Cmax	NP-Difficile
Mixte (2 contraintes)	F3|*RCb, Wb* |Cmax	Polynomial
	F3|*Wb, RCb* |Cmax	Polynomial
	F3|*RCb*, Wb* |Cmax	Polynomial
	F3|*Wb, RCb* *|Cmax	Polynomial
	F4|*RCb, Wb, RCb* |Cmax	Polynomial
	F4|*RCb*, Wb, RCb* *|Cmax	Polynomial
	F4|*Wb, RCb* *, *Wb* |Cmax	NP-Difficile
	F(*M* >3)|---, *Wb, Wb,* --- |Cmax	NP-Difficile
	F(*M* >3)|---, *RSb, RSb,* ---|Cmax	NP-Difficile
	F(M>4)|---, *RCb*, RCb*, RCb*,* ---|Cmax	NP-Difficile
Mixte (3 contraintes)	F4|*RCb*, Wb, RCb* |Cmax	Polynomial
	F4|*RCb, Wb, RCb** |Cmax	Polynomial
	F4|*RCb, RSb, RCb** |Cmax	Polynomial

Tab. 3.1 : Complexité de différents types de problème

Les problèmes qui contiennent deux contraintes de blocage différentes entre les machines, dont l'une est de type *RSb*, restent ouverts.

Pour ce qui concerne les problèmes qui contiennent trois contraintes de blocage différentes entre les machines, nous avons prouvé trois cas polynomiaux. Tous les autres problèmes non traités restent ouverts.

Dans le chapitre suivant, nous allons nous intéresser au développement de méthodes de résolution exactes et approchées pour résoudre des problèmes de type Flow-Shop avec blocage mixte.

Chapitre 4
Méthodes de résolution exacte

4. 1. Introduction

Après avoir déterminé la complexité de différents problèmes, nous allons, dans ce chapitre, calculer une solution optimale pour résoudre le problème de minimisation du makespan d'un Flow-Shop avec différentes contraintes de blocage. Nous avons proposé un modèle mathématique linéaire en nombres entiers. Cette modélisation est inspirée du modèle présenté par (Martinez, 2005) pour minimiser le makespan dans un Flow-Shop avec uniquement le blocage *RCb* entre toutes les machines du problème. Nous avons adapté ce modèle afin de considérer différentes contraintes de blocage en même temps.

Nous allons commencer par présenter les paramètres, variables et contraintes du modèle. Nous présentons ensuite des résultats numériques, afin d'en montrer les performances.

4. 2. Modèle mathématique

Comme nous sommes dans le cas d'un Flow-Shop de permutation, la séquence des jobs est la même sur toutes les machines et le problème revient donc à déterminer la permutation *G* des jobs qui minimise la date de fin de l'ordonnancement (appelée makespan). Ceci est équivalent à minimiser la date de fin du dernier job de la séquence sur la dernière machine.

4. 2. 1. Paramètres

Les paramètres utilisés pour la formulation mathématique du problème d'ordonnancement de type Flow-Shop avec blocage mixte sont les suivants :

- N : Nombre de jobs.
- M : Nombre de machines.
- $P_{i,k}$: Temps opératoire du job J_i sur la machine M_k.

- $B_{h,k}$ = 1 si il y a un blocage de type *h* entre la machine M_k et la machine M_{k+1} et 0 sinon. Avec,
  - *h* = 1 si il n'y a pas de blocage entre la machine M_k et la machine M_{k+1}. (*Wb*)
  - *h* = 2 si il y a un blocage de type *RSb* entre la machine M_k et la machine M_{k+1}.
  - *h* = 3 si il y a un blocage de type *RCb** entre la machine M_k et la machine M_{k+1}.
  - *h* = 4 si il y a un blocage de type *RCb* entre la machine M_k et la machine M_{k+1}.

4. 2. 2. Variables

Les variables de décision du modèle permettant d'obtenir l'ordre de passage des jobs sur les machines sont les suivantes :

- $S_{j,k}$ = Date de début du job en position *j* dans la séquence *G* sur la machine M_k.
- $C_{j,k}$ = Date de fin du job en position *j* dans la séquence *G* sur la machine M_k.
- $G_{i,j}$ = 1 si le job J_i se trouve à la position *j* dans la séquence *G* et 0 sinon.

4. 2. 3. Modèle

En utilisant les paramètres et les variables décrits précédemment, le problème peut être modélisé de la façon suivante :

$$Min\ C_{\max} \tag{4.1}$$

Sous les contraintes :

$$C_{\max} \geq C_{j,M} \quad \forall j \in \{1,..,N\} \tag{4.2}$$

$$S_{j,k} \geq S_{j,k-1} + \sum_{i=1}^{N} P_{i,k-1}.G_{i,j}\ , \quad \forall j \in \{1,..,N\}, \forall k \in \{2,..,M\} \tag{4.3}$$

$$S_{j,k} \geq C_{j-1,k}.B_{1,k} + S_{j-1,k+1}.B_{2,k} + C_{j-1,k+1}.B_{3,k} + S_{j-1,k+2}.B_{4,k} \quad \forall j \in \{2,..,N\}, \forall k \in \{1,..,M\text{-}2\} \tag{4.4}$$

$$S_{j,M-1} \geq C_{j-1,M-1}.B_{1,M-1} + S_{j-1,M}.B_{2,M-1} + C_{j-1,M}.B_{3,M-1}\ , \quad \forall j \in \{2,..,N\} \tag{4.5}$$

$$S_{j,M} \geq C_{j-1,M} \quad \forall j \in \{2,..,N\} \tag{4.6}$$

$$C_{j,k} = S_{j,k} + \sum_{i=1}^{N} P_{i,k} . G_{i,j} \ , \ \forall j \in \{1, .., N\}, \forall k \in \{1, .., M\} \quad 4.7$$

$$\sum_{i=1}^{N} G_{i,j} = 1 \ , \forall j \in \{1, .., N\} \quad 4.8$$

$$\sum_{j=1}^{N} G_{i,j} = 1 \ , \forall i \in \{1, .., N\} \quad 4.9$$

$$G_{i,j} \in \{0, 1\} \ , \forall i \in \{1, .., N\}, \forall j \in \{1, .., N\} \quad 4.10$$

$$S_{j,k} \geq 0 \ , \forall j \in \{1, .., N\}, \forall k \in \{1, .., M\} \quad 4.11$$

$$C_{j,k} \geq 0, \forall j \in \{1, .., N\}, \forall k \in \{1, .., M\} \quad 4.12$$

4. 2. 4. Signification des équations

Nous donnons dans ce paragraphe, la signification de chacune des contraintes de notre modèle :

- Équation 4.1 : La fonction objectif de notre problème est de minimiser la date de fin de l'ordonnancement, appelée C_{max} ou encore makespan.
- Équation 4.2 : Cette équation impose que la valeur du makespan soit supérieure ou égale à la date de fin de tous les jobs sur la dernière machine.
- Équation 4.3 : Cette équation représente la contrainte de précédence entre deux opérations successives d'un même job. Un job doit finir son opération sur la machine M_k, avant de pouvoir démarrer son opération sur la machine suivante M_{k+1}.
- Équation 4.4 : Cette équation modélise la contrainte de précédence en prenant en considération les différentes contraintes de blocage représentées par le paramètre $B_{h,k}$. Par exemple, s'il y a un blocage de type *RSb* entre la machine M_k et la machine M_{k+1}, l'équation 4.4 devient :

$$S_{j,k} \geq S_{j-1,k+1}$$

- Équation 4.5 : Cette équation est l'application de l'équation précédente au cas particulier de l'avant dernière machine.
- Équation 4.6 : Cette équation est l'application de l'équation 4.4 au cas particulier de la dernière machine, qui ne peut être que sans blocage.
- Équation 4.7 : Cette équation calcule la date de fin des opérations de chaque job.
- Équation 4.8 : À chaque position j de la séquence G n'est affecté qu'un seul job J_i.
- Équation 4.9 : Tout job J_i n'occupe qu'une seule position j dans la séquence G.

- Équation 4.10 : $G_{i,j}$ est une variable booléenne. Elle est égale à 1 si le job J_i est à la position j dans la séquence G et 0 sinon.
- Équation 4.11 : La date de début des opérations des jobs ne peut pas être de valeur négative.
- Équation 4.12 : La date de fin des opérations des jobs ne peut pas être de valeur négative.

Lorsque $B_{h,k} = 4$, on retrouve le modèle de Martinez duquel nous sommes partis :

$$S_{j,k} \geq S_{j-1,k+2}, \forall k \in \{1, .., M\text{-}2\}$$

Les problèmes de type *Wb* sont aussi un cas particulier des problèmes avec blocage mixte où on considère qu'il n'existe aucune contrainte de blocage entre toutes les machines ($B_{h,k} = 1$). D'où, l'équation 4.4 du modèle devient :

$$S_{j,k} \geq C_{j-1,k}, \forall k \in \{1, .., M\}$$

4. 2. 5. Résultats expérimentaux

Dans cette partie, nous présentons les résultats numériques obtenus pour des problèmes de différentes dimensions.

Pour cela, nous avons utilisé les benchmarks de (Martinez, 2005) : 100 instances différentes pour chaque dimension fixée de N jobs et M machines (20 instances pour les problèmes marqués par un astérisque) ont été générées. Les durées des opérations ont été générées uniformément dans l'intervalle [0, 99]. Les Tableaux 4.1, 4.2 et 4.3 indiquent respectivement les temps moyens de calcul obtenus en utilisant le logiciel Xpress-MP pour trouver la solution optimale des problèmes d'ordonnancement de type *Wb*, *RCb* et avec blocage mixte. Ces calculs ont été réalisés sur un PC Core 2 Duo' 3.16 GHz.

Jobs	Machines							
	5	6	7	10	15	20	50	100
5	0,11	0,14	0,15	0,19	0,27	0,28	0,58	0,60
6	0,17	0,18	0,24	0,27	0,37	0,45	0,95	1,35
7	0,20	0,25	0,29	0,46	0,65	0,83	1,98	3,63
8	0,24	0,34	0,38	0,71	1,19	1,56	5,90	11,76
9	0,31	0,46	0,60	1,17	2,32	3,53	16,45	50,46
10	0,39	0,69	1,02	2,28	5,34	10,07	65,45	229,81
11	0,63	1,16	1,69	5,17	19,45*	35,99*	369,97*	1053,77*
12	1,10	2,04	3,46	9,59*	55,17*	127,52*	1665,65*	5833,65*

Tab. 4.1 : temps d'exécution moyen par problème de type Wb (en secondes)

Jobs	Machines							
	5	6	7	10	15	20	50	100
5	0,07	0,08	0,10	0,15	0,19	0,25	0,48	0,58
6	0,12	0,13	0,15	0,20	0,27	0,34	0,76	1,29
7	0,15	0,19	0,22	0,31	0,49	0,58	1,68	4,07
8	0,22	0,28	0,35	0,53	0,93	1,34	5,34	14,99
9	0,36	0,58	0,63	1,02	2,09	3,20	17,49	73,49
10	1,00	1,46	1,61	3,24	5,95	9,93	88,37	330,29
11	2,83	3,93	5,52	10,41	22,19*	51,03*	386,25*	1513,56*
12	9,93	12,18	25,32	51,61*	109,16*	154,99*	1603,76*	7756,95*

Tab. 4.2 : temps d'exécution moyen par problème de type RCb (en secondes)

Pour les problèmes avec blocage mixte, nous avons choisi arbitrairement la séquence de contraintes de blocage suivante : V = (*RCb*, *RSb*, *RCb**, *Wb*, *RCb*, *RSb* …) qui sera adaptée selon le nombre de machines du système. Ce vecteur contient M-1 éléments, c'est-à-dire autant que de transitions entre les machines.

Jobs	Machines							
	5	6	7	10	15	20	50	100
5	0,05	0,06	0,09	0,12	0,18	0,22	0,46	0,74
6	0,08	0,09	0,11	0,19	0,25	0,32	0,63	1,09
7	0,13	0,13	0,16	0,28	0,45	0,51	1,31	2,59
8	0,17	0,17	0,20	0,37	0,62	0,86	2,88	7,51
9	0,33	0,26	0,30	0,51	1,05	1,56	7,82	30,66
10	0,48	0,40	0,53	1,04	2,08	3,75	29,66	120,38
11	1,62	1,22	1,88	1,97	5,91*	14,32*	119,89*	556,46*
12	6,32	5,65	3,09	4,91*	21,49*	33,22*	395,01*	2348,07*

Tab. 4.3 : temps d'exécution moyen par problème avec blocage mixte (en secondes)

Les temps de calcul moyens des tableaux 4.1, 4.2 et 4.3 sont obtenus en utilisant le même modèle mathématique proposé dans ce chapitre pour résoudre les problèmes d'ordonnancement avec blocage mixte.

Il est très intéressant de comparer les temps de calcul moyens de ces tableaux. Nous pouvons constater que le temps moyen d'exécution pour les problèmes avec une seule contrainte de blocage (*Wb* ou *RCb*) est nettement supérieur à celui des problèmes avec blocage mixte. Pour le problème à 12 jobs et 100 machines, le temps moyen de calcul pour le cas sans blocage est estimé à une heure et demie, plus de deux heures pour le cas du blocage *RCb*, alors que ça ne prend qu'une quarantaine de minutes pour le cas du blocage mixte. Cela peut s'expliquer par le fait que certains cas avec blocage mixte sont plus faciles qu'avec une seule contrainte de blocage, pour le même nombre de machines comme le montre la complexité de ces problèmes étudiée dans le chapitre 3.

4. 3. Conclusion

Ce modèle mathématique a été présenté lord de la conférence ***International Federation of Automatic Control 'IFAC11'*** *(Trabelsi et al., 2011a).*

Dans ce chapitre, nous avons présenté un modèle linéaire en nombres entiers pour résoudre le problème d'ordonnancement d'un Flow-Shop soumis à différentes contraintes de blocage. Cette modélisation est une adaptation du modèle proposé par Martinez afin de considérer le blocage mixte.

Ce modèle a été validé avec le logiciel d'optimisation XPress-MP. Les résultats obtenus sur des problèmes de différentes tailles montrent que des problèmes de taille inférieure à 12 jobs et 100 machines sont résolus dans un temps raisonnable. Nous constatons également que les temps moyens d'exécution pour les problèmes avec blocage mixte sont nettement inférieurs à ceux des problèmes avec une seule contrainte de blocage (*RCb* ou *Wb*).

Dans le chapitre suivant, nous allons proposer des méthodes approchées pour résoudre les problèmes de taille importante dans un temps raisonnable. Le modèle mathématique développé dans ce chapitre va nous permettre d'évaluer les performances des heuristiques et métaheuristiques pour des problèmes de petites et moyennes tailles.

Chapitre 5
Méthodes approchées

5. 1. Introduction

Dans le chapitre précédent, nous avons constaté que les temps d'exécution augmentent avec le nombre de jobs et de machines et que, pour les problèmes de taille importante, il n'est pas possible d'obtenir une solution exacte à ces problèmes en un temps raisonnable. Il est donc nécessaire de développer des méthodes approchées et d'estimer le pourcentage d'erreur avec la solution optimale (si elle est calculable) ou des bornes inférieures de qualité.

Dans ce chapitre, nous présentons différentes méthodes de résolution approchées pour résoudre les problèmes de type Flow-Shop avec blocage mixte. Nous présentons tout d'abord une borne inférieure basée sur le temps maximal d'occupation des machines. Puis, nous proposons une nouvelle heuristique TSS, ainsi qu'une adaptation de l'heuristique NEH développée initialement pour le Flow-Shop de permutation. Ensuite, nous apportons des améliorations à ces heuristiques. Enfin, nous proposons comme méta-heuristique un algorithme génétique.

5. 2. Borne inférieure

Dans le but d'estimer le pourcentage d'erreur des heuristiques que nous développons, nous proposons une borne inférieure basée sur le temps maximal d'occupation des machines. Cette proposition est inspirée de la borne inférieure présentée par (Gorine et Sauer, 2008) pour les problèmes de type Job-Shop avec blocage *RCb*. Nous avons adapté ce modèle afin de considérer différentes contraintes de blocage entre différentes machines du même problème.

5. 2. 1. Théorème

Théorème 7. Soit un Flow-Shop composé de N jobs et M machines soumis à différentes contraintes de blocage. Une borne inférieure du makespan du problème d'ordonnancement de ce système est donnée par :

$$B_{\inf} = \max_{k=1,\ldots,M}\left[\sum_{i=1,\ldots,N}\left[P_{ik} + P_{i(k+1)}.\psi_k\right] + \min_{i=1,\ldots,N}\left[\sum_{s=1}^{k-1} P_{is}\right] + \min_{i=1,\ldots,N}\left[\left(\sum_{s=k+1}^{M} P_{is}\right) - P_{i(k+1)}.\psi_k\right]\right]$$

avec :

$$\psi_k = \begin{cases} 1 & s'il\ y\ a\ un\ blocage\ de\ type\ RCb\ ou\ RCb*\ entre\ la\ machine\ M_k\ et\ la\ machine\ M_{k+1}\ \forall\ k \prec M \\ 0 & sinon \end{cases}$$

Preuve :

Soit une machine M_k. Cette machine est soit libre, soit en cours d'exécution d'une opération, soit bloquée par un job (si l'opération effectuée par ce job sur la machine M_k n'est pas la dernière opération dans la gamme du job et si le job n'a pas encore quitté la machine suivante dans la gamme). Le temps total d'exécution d'une opération plus le temps de blocage (on ne considère que le blocage de type *RCb* ou *RCb**) est supérieur ou égal à :

$$T1 = \sum_{i=1,\ldots,N}\left[P_{ik} + P_{i(k+1)}.\psi_k\right] \qquad 5.1$$

Avant de pouvoir débuter l'exécution d'une opération O_{ik} sur la machine M_k, il faut que toutes les opérations qui précèdent O_{ik} dans la gamme de J_i soient terminées. Par conséquent, M_k ne peut pas débuter une opération avant le plus petit temps nécessaire, noté $T2$ (Figure 5.1):

$$T2 = \min_{i=1,\ldots,N}\left[\sum_{s=1}^{k-1} P_{is}\right] \qquad 5.2$$

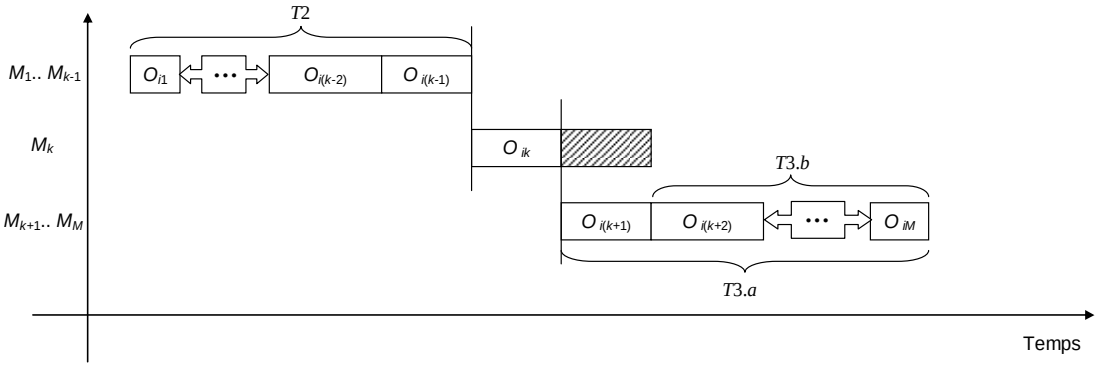


Figure 5.1. Temps d'attente avant et après l'exécution d'une opération O_{ik} sur M_k pour un job J_k

De la même façon, après l'exécution d'une opération O_{ik} sur la machine M_k, il faut que toutes les opérations qui suivent O_{ik} dans la gamme de J_i se terminent (Figure 5.1).

Le temps $T3.a$ est défini comme suit dans le cas sans blocage ou avec blocage *RSb* :

$$T3.a = \min_{i=1,\dots,N}\left[\sum_{s=k+1}^{M} P_{is}\right] \qquad 5.3$$

Dans le cas d'un blocage de type *RCb* ou *RCb** entre la machine M_k et la machine M_{k+1}, l'opération $O_{i(k+1)}$ est comptée dans le terme $T1$. Ainsi, pour ces deux cas de blocage il faut attendre au moins le temps $T3.b$ suivant avant la fin de l'ordonnancement (Figure 5.1):

$$T3.b = \min_{i=1,\dots,N}\left[\sum_{s=k+2}^{M} P_{is}\right] \qquad 5.4$$

Par conséquent, après toutes les opérations O_{ik} qui sont réalisées sur M_k, il faut attendre au moins le temps $T3$ suivant avant la fin de l'ordonnancement :

$$T3 = \min_{i=1,\dots,N}\left[\left(\sum_{s=k+1}^{M} P_{is}\right) - P_{i(k+1)}.\psi_k\right] \qquad 5.5$$

En conclusion, le C_{max} est supérieur ou égal à $T1+T2+T3$.

Si on considère toutes les machines du problème, on obtient :

$$C_{max} \geq B_{\inf} = \max_{k=1,\dots,M}\left[\sum_{i=1,\dots,N}\left[P_{ik} + P_{i(k+1)}.\psi_k\right] + \min_{i=1,\dots,N}\left[\sum_{s=1}^{k-1} P_{is}\right] + \min_{i=1,\dots,N}\left[\left(\sum_{s=k+1}^{M} P_{is}\right) - P_{i(k+1)}.\psi_k\right]\right]$$

CQFD

Exemple :

Considérons un problème d'ordonnancement de type Flow-Shop constitué de 3 jobs et 5 machines (Figure 5.2) avec les temps opératoires suivants :

$$P_{i,k} = \begin{pmatrix} 1 & 1 & 2 & 1 & 2 \\ 1 & 3 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 1 \end{pmatrix} \quad \forall i \in \{1,..,3\},\ \forall k \in \{1,..,5\}$$

Nous sommes dans le cas d'un Flow-Shop, donc chaque job passe par toutes les machines dans le même ordre, de la machine M_1 jusqu'à la machine M_5. Nous considérons le vecteur de contraintes

de blocage V = (*RCb*, *RSb*, *RCb**, *Wb*). Ce vecteur indique le type de contrainte entre les machines successives du problème (*i.e*. *RCb* est la contrainte de blocage entre les machines M_1 et M_2, *RSb* est celle entre les machines M_2 et M_3, etc.).

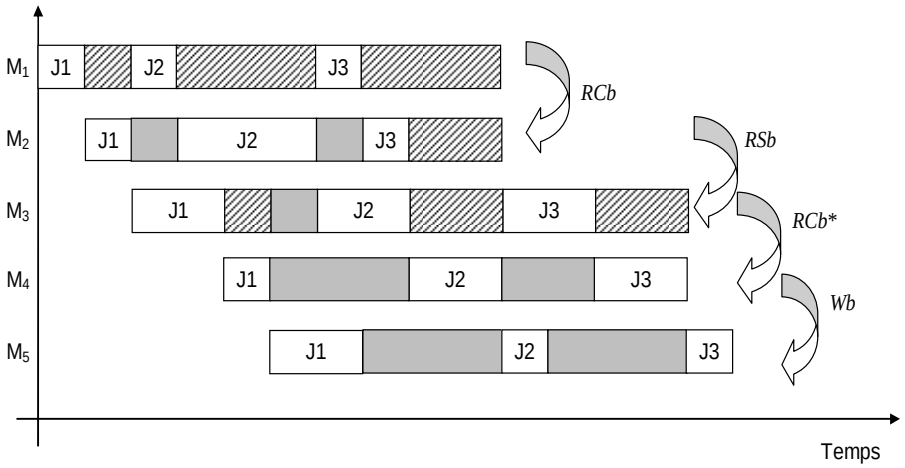


Figure 5.2. Flow-Shop (3 jobs, 5 machines) avec contraintes de blocage mixtes

Pour k = 1,.., M, on note :

$$B_k = \sum_{i=1,\ldots,N} \left[P_{ik} + P_{i(k+1)} \cdot \psi_k\right] + \min_{i=1,\ldots,N}\left[\sum_{s=1}^{k-1} P_{is}\right] + \min_{i=1,\ldots,N}\left[\left(\sum_{s=k+1}^{M} P_{is}\right) - P_{i(k+1)} \cdot \psi_k\right]$$

Nous allons calculer B_k pour chacune des 5 machines :

- Machine M_1

  Nous avons un blocage de type *RCb*, donc $\psi_k = 1$:

$$B_1 = \left[P_{11} + P_{12} + P_{21} + P_{22} + P_{31} + P_{32}\right] + 0 + \min\left(P_{13} + P_{14} + P_{15}, P_{23} + P_{24} + P_{25}, P_{33} + P_{34} + P_{35}\right)$$
$$= [1+1+1+3+1+1] + 0 + \min\left(2+1+2, 2+2+1, 2+2+1\right) = 13\,ut$$

- Machine M_2 (ici, $\psi_k = 0$) :

$$B_2 = \left[P_{12} + P_{22} + P_{32}\right] + \min\left(P_{11}, P_{21}, P_{31}\right) + \min\left(P_{13} + P_{14} + P_{15}, P_{23} + P_{24} + P_{25}, P_{33} + P_{34} + P_{35}\right)$$
$$= [1+3+1] + \min(1,1,1) + \min\left(2+1+2, 2+2+1, 2+2+1\right) = 11\,ut$$

- Machine M_3 (ici, $\psi_k = 1$) :

$$B_3 = [P_{13} + P_{14} + P_{23} + P_{24} + P_{33} + P_{34}] + \min(P_{11} + P_{12}, P_{21} + P_{22}, P_{31} + P_{32}) + \min(P_{15}, P_{25}, P_{35})$$
$$= [2 + 1 + 2 + 2 + 2 + 2] + \min(1 + 1, 1 + 3, 1 + 1) + \min(2, 1, 1) = 14\ ut$$

- Machine M_4 (ici, $\psi_k = 0$) :

$$B_4 = [P_{14} + P_{24} + P_{34}] + \min(P_{11} + P_{12} + P_{13}, P_{21} + P_{22} + P_{23}, P_{31} + P_{32} + P_{33}) + \min(P_{15}, P_{25}, P_{35})$$
$$= [1 + 2 + 2] + \min(1 + 1 + 2, 1 + 3 + 2, 1 + 1 + 2) + \min(2, 1, 1) = 10\ ut$$

- Machine M_5 (ici, $\psi_k = 0$) :

$$B_5 = [P_{15} + P_{25} + P_{35}] + \min(P_{11} + P_{12} + P_{13} + P_{14}, P_{21} + P_{22} + P_{23} + P_{24}, P_{31} + P_{32} + P_{33} + P_{34}) + 0$$
$$= [2 + 1 + 1] + \min(1 + 1 + 2 + 1, 1 + 3 + 2 + 2, 1 + 1 + 2 + 2) + 0 = 9\ ut$$

Par conséquent, $B_{\inf} = \max(B_1, B_2, B_3, B_4, B_5) = 14$ ut.

Sur cet exemple, la solution optimale est de 14 ut et la borne inférieure B_{inf} donne cette solution.

5. 2. 2. Résultats expérimentaux

Pour mieux évaluer la borne inférieure proposée, nous avons utilisé les mêmes 100 instances pour chaque dimension fixée de *N* jobs et *M* machines (20 instances pour les problèmes marqués par un astérisque) que celles utilisées dans le chapitre 4. Nous avions précisé, dans les Tableau 4.1 à 4.3, les temps d'exécution moyens pour obtenir la solution optimale.

Le Tableau 5.1 donne le pourcentage d'erreurs de la borne inférieure proposée par rapport au makespan optimal ainsi que le nombre de fois où la borne inférieure est égale à la solution optimale (nombre entre parenthèses). Le pourcentage d'erreur est défini par la formule suivante :

$$\%err = \frac{S_{opt} - B_{\inf}}{S_{opt}} \times 100 \qquad 5.6$$

D'après le tableau 5.1, on remarque que pour les instances de petite taille (de 5 à 7 machines), notre borne inférieure est proche de la solution optimale voire égale à S_{opt} pour quelques instances (on obtient la solution optimale une à trois fois sur dix).

Jobs	Machines							
	5	6	7	10	15	20	50	100
5	3,82 (23)	5,02 (19)	5,93 (11)	8,5 (4)	12,15 (0)	12 (0)	12,9 (0)	11,48 (0)
6	3,29 (28)	4,06 (20)	4,52 (13)	8,1 (3)	11,6 (0)	11,7 (0)	13,46 (0)	12,03 (0)
7	3,06 (17)	3,63 (23)	4,09 (9)	7,14 (3)	10,88 (0)	11,78 (0)	14,22 (0)	12,86 (0)
8	2,74 (22)	3,05 (26)	3,53 (14)	6,24 (4)	10,57 (0)	11,82 (0)	14,54 (0)	13,5 (0)
9	2,67 (21)	2,64 (29)	3,25 (16)	5,95 (4)	9,56 (0)	10,9 (0)	14,32 (0)	13,9 (0)
10	2,81 (20)	2,35 (27)	2,95 (10)	5,33 (6)	9,12 (1)	10,75 (0)	14,63 (0)	14,24 (0)
11	3 (15)	2,07 (25)	3,13 (10)	4,55 (6)	9,49 (0*)	9,95 (0*)	14,29 (0*)	14,62 (0*)
12	2,62 (13)	2 (24)	2,6 (9)	5,02 (1*)	9,18 (0*)	10,05 (0*)	13,85 (0*)	

Tab. 5.1 : Pourcentage d'erreur de la borne inférieure et le nombre de fois où $B_{inf} = S_{opt}$

Cette borne inférieure est moins efficace pour les problèmes où le nombre de machines est égal à 10 et le nombre de fois où $B_{inf} = S_{opt}$ est faible.

Enfin, dès que le nombre de machines devient plus important, l'erreur augmente et on n'obtient quasiment plus la solution optimale (à partir de 15 machines, on n'a obtenu qu'une seule fois la solution optimale pour le problème à 10 jobs et 15 machines).

Néanmoins, nous pouvons observer qu'un nombre élevé de machines ne dégrade pas significativement l'efficacité de cette borne, pour les nombres de jobs considérés. En effet, entre 15 et 100 machines, elle présente un pourcentage d'erreur compris entre 9 et 15%.

Cette borne inférieure a été présentée à la conférence ***Industrial Engineering and Systems Management 'IESM11'*** *(Trabelsi et al., 2011b).*

5. 3. Heuristiques

Comme nous l'avons dit dans le chapitre précédent, les temps de traitement nécessaires avec les méthodes exactes peuvent être importants, surtout pour les problèmes de grandes dimensions. Par conséquent, pour être capable de traiter ces problèmes, il est nécessaire de développer des

méthodes approchées et d'estimer leur erreur moyenne par rapport à une solution optimale quand elle est calculable, ou par rapport à la borne inférieure présentée dans le paragraphe précédent.

Dans ce paragraphe, nous proposons donc des heuristiques avec quelques améliorations locales afin d'obtenir rapidement une solution réalisable dans un temps raisonnable, mais pas nécessairement une solution optimale. Nous présentons dans un premier temps la méthode NEH que nous avons adaptée pour satisfaire les contraintes de notre problème, puis nous proposons une nouvelle heuristique TSS. Nous utilisons également des méthodes d'amélioration pour les deux algorithmes.

5. 3. 1. Heuristique NEH

L'heuristique NEH est l'une des heuristiques les plus connues pour résoudre les problèmes d'ordonnancement de type Flow-Shop (Nawaz, Enscore et Ham, 1983). Cette heuristique est non seulement efficace, mais aussi très simple à programmer. Il s'agit d'un algorithme constructif qui sélectionne le job le plus long parmi les jobs non encore placés et essaie de l'insérer dans toutes les positions possibles de la séquence partielle en cours de construction. Elle choisit enfin la séquence qui augmente le moins la durée de l'ordonnancement partiel formé par les jobs déjà placés. Un grand nombre de travaux ont considéré cette heuristique comme une référence pour comparer leurs résultats (Zobolas *et al*., 2009), (Ruiz et Stützle, 2007). Une étude comparative de certaines heuristiques a été réalisée dans (Ruiz et Maroto, 2005) dans laquelle, là encore, cette heuristique montre sa supériorité par rapport à toutes ses concurrentes.

L'algorithme suivant rappelle le fonctionnement de cette heuristique.

Algorithme 1

Soit $J = (J_1, J_2, .., J_N)$ la liste des jobs triés dans l'ordre décroissant des temps opératoires.
Pour $i = 1$ à N
    Sélectionner le job J_i de J.
    $C_{pmax} = \infty$
    Pour $k = 1$ à i
        Placer le job J_i à la position k dans la séquence partielle sans changer la position relative des autres jobs déjà placés.
        Calculer le C_{max} de l'ordonnancement partiel formé.
        Si $C_{max} < C_{pmax}$ alors
            $Best_k = k$
            $C_{pmax} = C_{max}$
        Fin si
    Fin pour

Placer le job J_i à la position $Best_k$ dans la séquence partielle.
Fin pour

L'adaptation de cette heuristique à notre problème consiste simplement à respecter les différentes contraintes de blocage entre les machines lorsque l'on calcule le makespan partiel à chaque itération.

5. 3. 2. Heuristique TSS

L'heuristique proposée est une heuristique par construction. Cette proposition est inspirée de l'algorithme développé dans (Trabelsi *et al*., 2010) pour résoudre les problèmes de type Job-Shop avec des contraintes de blocage *RCb* et *RCb**. A partir d'une solution partielle, il faut choisir quel job on doit placer lors de l'étape suivante. Pour cela, nous avons proposé un critère qui combine trois paramètres :

- *Cpmax*: makespan partiel. Il s'agit de calculer la date de fin du nouvel ordonnancement formé en plaçant le job choisi, tout en tenant compte du blocage engendré.
- *SomTpsIn*: la somme des temps d'inactivité. Il s'agit de calculer le temps pendant lequel les machines sont inactives en incluant les temps morts et les temps de blocage.
- *SomTpsEx*: la somme des temps opératoires de tous les jobs placés dans l'ordonnancement partiel.

Pour avoir un ordonnancement le plus optimal possible, on a tendance d'une part à minimiser les temps d'inactivité des machines et la date de fin des opérations, et d'autre part à maximiser le taux d'utilisation des machines. D'où le critère :

$Cr = \min (Cpmax + SomTpsIn - SomTpsEx)$.

De plus, *N* possibilités d'ordonnancement peuvent être construites en changeant le job à placer en première position. Parmi les solutions générées, on choisit à la fin celui qui donne le makespan final *Cmax* le plus petit.

La structure générale de l'algorithme TSS utilisé pour résoudre les problèmes de type Flow-Shop avec les différentes contraintes de blocage est la suivante :

Algorithme 2

Introduire les données du problème (nombre de jobs, nombre de machines, temps d'exécution des opérations de chaque job et le type de blocage entre les machines).

Pour i = 1 à N (*on calcule toutes les possibilités en changeant le job à placer en premier*)
    On place en premier le job J_i.
    Tant qu'il reste encore des jobs à placer, faire
        Pour t = 1 à [N - (nombre de jobs déjà placés)]
            Calculer *Cpmax, SomTpsIn, SomTpsEx*.
            Calculer C_{rt} (le critère de choix après avoir placé le job J_t).
        Fin pour
        Placer le job dont le C_{rt} est le plus petit.
    Fin tant que
    Calculer $C_{max,i}$ (*makespan final lorsqu'on place le job J_i en premier*).
Fin pour
Choisir l'ordonnancement qui donne le makespan final $C_{max,i}$ le plus petit.

Exemple:

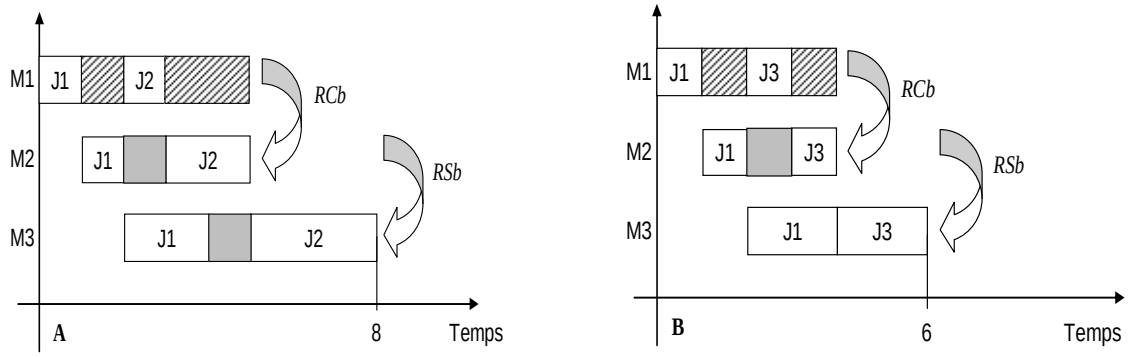
On considère un problème d'ordonnancement Flow-Shop à quatre jobs et trois machines.

Les temps opératoires sont les suivants :

$$P_{i,k} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix} \quad \forall i \in \{1,..,4\},\ \forall k \in \{1,..,3\}$$

On considère le vecteur de contraintes de blocage V = (*RCb*, *RSb*) , *i.e. RCb* est la contrainte de blocage entre les machines M_1 et M_2, et *RSb* est la contrainte de blocage entre les machines M_2 et M_3.

Soit J_1 le job à placer en premier. Nous avons donc trois possibilités (puisqu'il y a quatre jobs à ordonnancer) pour placer le job suivant : J_2, J_3 ou J_4 (Figure 5.3).



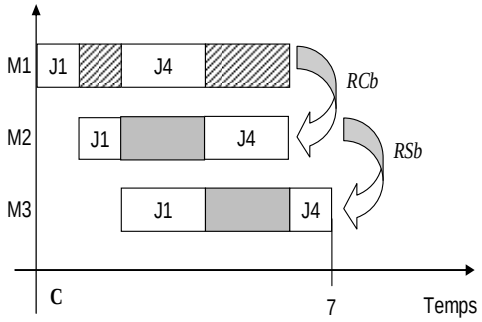


Figure 5.3. Trois cas possibles d'ordonnancement

Nous expliquons ici comment se fait le calcul des différents paramètres avec le premier cas, *i.e.* quand le job J_2 est placé (Figure 5.3 - A) :

- *SomTpsEx* = $\sum_{i=1}^{2} \sum_{j=1}^{3} P_{i,j}$ = 10 ut. Après avoir placé les deux premiers jobs, on calcule la somme des temps d'exécution de toutes les opérations de ces deux jobs.
- *SomTpsIn* = 1+2+1+1 = 5 ut. C'est la somme des temps inactifs qui est composée d'un temps de blocage et d'un temps mort. Le temps de blocage est dû à la contrainte *RCb* entre les deux premières machines M_1 et M_2 (intervalles [1-2] et [3-5] sur la machine M_1). Le temps mort est celui qui est considéré lorsque les machines M_2 et M_3 sont disponibles mais ne traitent aucun job (intervalles [2-3] sur la machine M_2 et [4-5] sur la machine M_3).
- *Cpmax* = 8 ut. Le makespan partiel représente la date minimale à partir de laquelle toutes les machines sont disponibles.

Ainsi, pour l'exemple considéré et expliqué précédemment, le calcul de *Cr*(*A*) donne :

Cr(*A*) = 8+5-10 = 3 ut.

Les résultats de calcul pour chaque cas (J_2, J_3 ou J_4) sont donnés dans le Tableau 5.2.

	Somme des temps d'exécution « *SomTpsEx* » (ut)	Temps de blocage + temps mort « *SomTpsIn* » (ut)	Makespan partiel « *Cpmax* » (ut)	Critère « *Cr* »
A	10	3 + 2	8	3
B	**8**	**2 + 1**	**6**	**1**
C	9	4 + 3	7	5

Tab. 5.2 : Critère de choix du job à placer en premier

Dans notre exemple, c'est donc le job J_3 qui sera placé en deuxième position dans la séquence partielle, puisqu'il donne le critère le plus petit (Cr = 1 ut).

La solution finale, donnée par l'algorithme TSS pour cet exemple de problème $F3|\text{Mixte}|C_{max}$, est présentée sur le diagramme de Gantt suivant (Figure 5.4) :

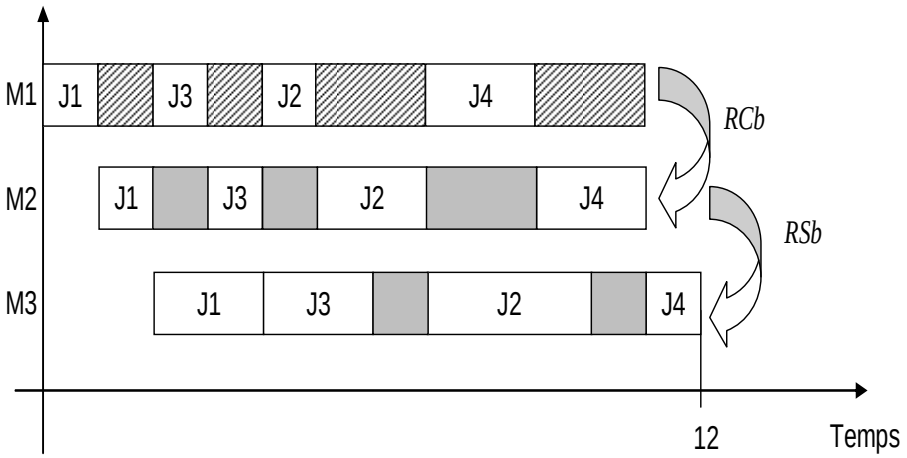


Figure 5.4. Ordonnancement complet pour le problème F3|Mixte|Cmax

5. 3. 3. Amélioration locale NEH

Cette amélioration locale est basée sur la méthode itérative NEH. En effet, nous partons d'une séquence d'ordonnancement obtenue à partir d'une heuristique initiale, puis nous utilisons la partie itérative de l'algorithme NEH tant qu'on obtient une amélioration de la solution courante.

La structure de l'algorithme d'amélioration locale NEH est la suivante :

Algorithme 3

Soit J la séquence des jobs dans l'ordre obtenu à partir d'une heuristique initiale.
C_{max} = makespan obtenu avec la séquence J
$C = 0$
Tant que $C \leq C_{max}$ faire
    Appliquer l'heuristique NEH à partir de la séquence en cours J
    J = séquence obtenue
    C = makespan de la solution obtenue
    Si $C \leq C_{max}$ alors $C_{max} = C$
Fin tant que

5. 3. 4. Amélioration locale TSS

Dans les problèmes de type Flow-Shop classiques, les contraintes de blocage *RCb* et *RCb** provoquent un temps de blocage supplémentaire, défini comme temps de blocage différentiel (Sauvey et Sauer, 2009).

Considérons l'exemple de la Figure 5.5 : Après avoir placé le premier job, les auteurs donnent deux définitions pour le temps de blocage différentiel causé par les temps opératoires du deuxième job.

- Le temps de blocage différentiel « d'attente » sur la machine M_1 est dû à l'insuffisance du temps opératoire du job J_2 sur la même machine par rapport au temps opératoire du job précédent J_1 sur la machine M_3.
- Les temps de blocage différentiels « d'inactivité » sur les machines M_3 et M_4 sont dus à l'excédence du temps opératoire du job J_2 sur la machine M_2 par rapport au temps opératoire du job précédent J_1 sur la machine M_4.

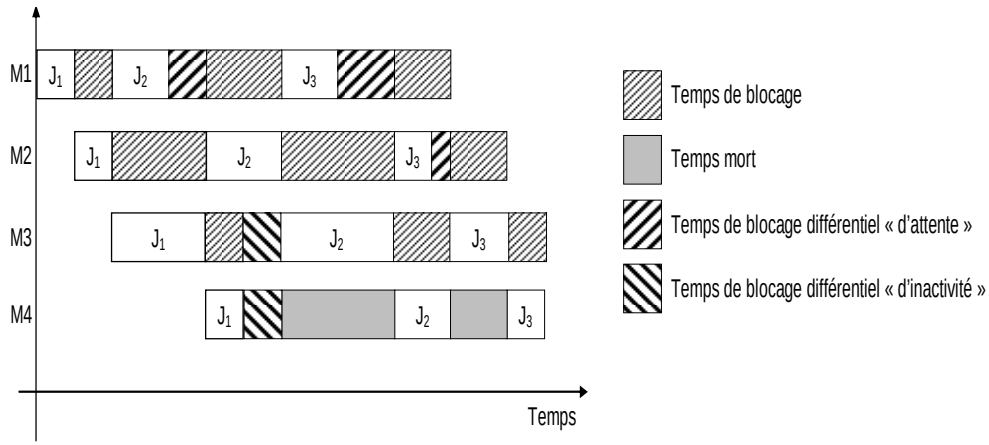


Figure 5.5. Temps de blocages différentiels sur un Flow-Shop avec contrainte RCb

L'amélioration locale TSS est basée sur l'hypothèse qu'un bon ordonnancement limite les temps de blocages différentiels à une valeur la plus faible possible. Ainsi, on prend, dans une séquence donnée, le job présentant le temps de blocage différentiel le plus grand et on le permute successivement avec chacun des autres jobs de la séquence. On garde la meilleure permutation, tant qu'elle améliore le makespan.

La structure de l'algorithme d'amélioration locale TSS est la suivante :

Algorithme 4

Soit J la séquence des jobs dans l'ordre obtenu à partir d'une heuristique initiale.
C_{max} = makespan obtenu avec la séquence J
$C = 0$
Tant que $C \leq C_{max}$ faire
    Chercher le job J_x présentant le temps de blocage différentiel le plus grand
    Pour $t = 1$ à $N \setminus (J_x)$
        Permuter les jobs J_t et J_x
        Calculer le nouveau makespan obtenu C
    Fin pour
    Garder la meilleure permutation possible (C le plus petit)
    Si $C \leq C_{max}$ alors $C_{max} = C$
Fin tant que

5. 3. 5. Résultats expérimentaux

Dans cette partie, nous présentons les résultats numériques que nous avons obtenus en utilisant les heuristiques puis les améliorations locales afin de montrer leur efficacité.

Pour valider la qualité de notre heuristique TSS, nous faisons, dans une première étape (a), la comparaison avec l'heuristique NEH. Ensuite, nous appliquons l'amélioration locale NEH sur chacune des deux heuristiques (b), puis l'amélioration locale TSS (c). Enfin, nous donnons les résultats lorsque la meilleure solution, celle qui a le plus petit makespan, est retenue (d) (Tableau 5.3).

Pour chaque dimension fixée de N jobs et M machines, nous avons utilisé les mêmes 100 instances différentes (20 instances pour les problèmes de taille supérieure à 11 jobs / 10 machines) que celles utilisées dans le chapitre 4.

Le tableau 5.3 présente le pourcentage d'erreur entre la solution optimale S_{opt} et le makespan C_{max} trouvé en utilisant les différentes heuristiques. Le pourcentage d'erreur est calculé de la façon suivante :

$$\%err = \frac{C_{\max} - S_{opt}}{S_{opt}} \times 100 \qquad 5.7$$

	Machines	5		6		7		10		15		20		50		100	
Jobs	Heuristiques	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH	TSS	NEH
5	a	4,46	3,75	3,39	4,61	3,94	3,72	2,86	4,25	2,55	3,42	1,76	3,70	1,20	2,55	0,86	1,73
	b	0,48	0,38	0,21	0,17	0,29	0,29	0,40	0,32	0,44	0,34	0,39	0,22	0,36	0,27	0,20	0,12
	c	0,45	0,23	0,19	0,15	0,23	0,19	0,38	0,13	0,38	0,25	0,35	0,15	0,32	0,17	0,19	0,08
	d	0,07		0,10		0,08		0,08		0,11		0,08		0,08		0,06	
6	a	4,15	4,02	4,25	3,97	5,37	4,58	4,14	3,98	4,35	3,82	2,91	4,09	1,78	2,56	1,12	1,84
	b	0,41	0,74	0,38	0,33	0,61	0,43	0,39	0,54	0,86	0,81	0,73	0,50	0,58	0,24	0,39	0,21
	c	0,35	0,59	0,37	0,32	0,53	0,30	0,36	0,45	0,76	0,71	0,67	0,43	0,55	0,19	0,35	0,18
	d	0,11		0,17		0,17		0,22		0,45		0,28		0,12		0,11	
7	a	5,21	4,07	4,87	3,72	5,68	4,24	4,92	4,02	4,58	3,84	3,95	4,01	2,46	2,65	1,72	2,07
	b	0,50	0,87	0,46	0,46	0,78	0,89	0,82	0,68	0,93	0,73	0,84	0,58	0,88	0,37	0,53	0,26
	c	0,48	0,69	0,46	0,44	0,66	0,75	0,69	0,57	0,84	0,59	0,75	0,49	0,85	0,34	0,48	0,23
	d	0,24		0,25		0,35		0,31		0,31		0,32		0,22		0,17	
8	a	5,35	3,87	5,21	3,26	6,42	3,77	5,90	3,95	5,80	4,10	4,36	3,95	3,09	2,82	2,16	2,18
	b	0,82	0,95	0,54	0,52	0,94	0,79	1,13	1,03	1,23	1,00	1,09	0,76	0,96	0,52	0,64	0,42
	c	0,75	0,79	0,50	0,48	0,84	0,63	1,02	0,84	1,14	0,75	1,01	0,68	0,93	0,48	0,61	0,40
	d	0,47		0,28		0,41		0,57		0,53		0,44		0,36		0,28	
9	a	5,33	3,67	6,10	3,45	6,35	4,01	6,27	4,12	5,74	4,17	5,19	4,25	3,33	3,37	2,43	2,42
	b	0,57	0,85	0,80	0,73	1,25	1,23	1,53	1,05	1,49	1,26	1,18	0,87	1,26	0,73	0,92	0,62
	c	0,53	0,76	0,75	0,59	1,14	1,02	1,49	0,85	1,41	1,14	1,11	0,80	1,13	0,72	0,82	0,61
	d	0,39		0,38		0,78		0,65		0,80		0,59		0,52		0,46	
10	a	5,51	3,72	5,57	3,17	6,33	4,02	6,86	4,54	6,77	4,56	5,67	4,86	3,95	3,20	2,94	2,57
	b	0,81	0,94	0,74	0,81	1,46	1,27	1,31	1,25	2,16	1,63	1,51	1,32	1,60	0,92	1,08	0,63
	c	0,72	0,87	0,68	0,71	1,31	1,16	1,22	1,14	2,10	1,49	1,43	1,18	1,51	0,87	1,01	0,58
	d	0,41		0,39		0,82		0,74		1,17		0,89		0,75		0,47	
11	a	5,52	3,33	6,21	3,19	7,20	4,23	7,03	4,39	7,16	5,74	6,03	3,89	4,32	3,87	3,43	2,56
	b	0,75	0,96	1,04	0,95	1,57	1,52	1,49	1,30	1,67	1,90	1,97	1,44	1,44	1,11	1,07	0,72
	c	0,65	0,87	0,97	0,82	1,44	1,43	1,32	1,18	1,57	1,83	1,95	1,44	1,43	1,05	1,01	0,67
	d	0,45		0,53		0,97		0,80		1,09		1,13		0,94		0,52	
12	a	5,47	3,31	5,27	3,21	7,15	4,07	7,58	5,12	7,85	5,57	6,41	4,84	4,80	3,64	3,49	2,50
	b	0,77	0,81	0,96	0,88	1,34	1,46	2,00	1,70	2,33	2,02	1,67	1,49	1,56	1,11	1,12	0,86
	c	0,75	0,79	0,94	0,81	1,24	1,30	1,85	1,69	2,22	1,87	1,46	1,43	1,56	1,01	1,06	0,82
	d	0,51		0,52		0,90		1,15		1,61		0,98		0,84		0,58	

a : Écart relatif entre l'heuristique (TSS ou NEH) et la solution optimale

b : Amélioration de l'heuristique de départ par NEH

c : Amélioration de la dernière solution par TSS

d : Meilleure solution des deux heuristiques de départ : min (TSS+AL ; NEH+AL)

Tab. 5.3 : Pourcentage d'erreur entre la solution optimale et le makespan trouvé en utilisant différentes heuristiques

D'après le Tableau 5.3, nous pouvons constater que pour les problèmes de petite taille (de 5 à 8 jobs), les heuristiques NEH et TSS donnent des solutions avec des pourcentages d'erreur raisonnables et comparables, même si TSS est souvent meilleure pour les problèmes avec un

grand nombre de machines. Par contre, pour les problèmes de grande taille, NEH est souvent meilleure que TSS. (Tableau 5.3, ligne -a-)

En ajoutant l'amélioration locale NEH sur les résultats obtenus, nous remarquons que les pourcentages d'erreur sont considérablement réduits et qu'ils deviennent de plus en plus comparables pour les deux heuristiques proposées. Nous constatons aussi que, pour le cas où la solution de départ est obtenue par l'heuristique TSS, nous obtenons de meilleur résultats pour les problèmes à 5 machines et quelques autres problèmes (6/10, 7/7, 10/6...). (Tableau 5.3, ligne -b-)

Comme l'amélioration locale NEH apporte une amélioration significative aux résultats, et que les temps de calcul sont négligeables, nous avons eu l'idée d'ajouter d'autres améliorations. Nous avons ajouté l'amélioration locale TSS, ce qui nous a permis d'améliorer encore nos résultats. (Tableau 5.3, ligne -c-)

Finalement, après avoir ajouté les différentes améliorations locales aux heuristiques TSS et NEH, nous gardons la meilleure solution des deux cas (Tableau 5.3, ligne -d-) et nous remarquons, encore une fois, que le pourcentage d'erreur est considérablement réduit, ce qui s'explique par le fait que les meilleures solutions trouvées à partir des heuristiques de départ TSS ou NEH sont souvent différentes.

En conclusion, nous constatons l'intérêt de varier les deux heuristiques et les deux améliorations locales pour les raisons suivantes :

- Cela nous permet d'attaquer les problèmes sous différents angles. Les solutions obtenues sont plutôt différentes, ce qui nous permet de choisir la meilleure.
- Comme les temps de calcul des problèmes résolus par ces heuristiques et leurs améliorations locales sont négligeables, nous pouvons alors les combiner pour améliorer d'avantage nos résultats.

Nous remarquons enfin que les pourcentages d'erreur les plus élevés en utilisant notre méthode, sont au niveau des problèmes à 15 machines. L'hypothèse que nous avançons pour expliquer cette constatation est que ce nombre de 15 machines est de l'ordre de grandeur des nombres de jobs pour lesquels nous avons mené nos expériences. Les heuristiques sont habituellement basées sur les machines ou bien basées sur les jobs. Lorsque ces deux nombres sont voisins, l'efficacité des heuristiques a tendance à baisser.

5. 4. Métaheuristique

Afin d'améliorer davantage les solutions obtenues par les méthodes heuristiques, nous proposons d'utiliser les algorithmes génétiques qui ont prouvé leur efficacité sur les problèmes d'optimisation discrète depuis 1962. Les travaux de (Holland, 1962) sur l'adaptation à différents domaines d'application ont été largement étendus (Holger et Stützle, 2005), (Siarry et Michalewicz, 2007).

5. 4. 1. Algorithme génétique

Les résultats obtenus par l'algorithme génétique pour des problèmes d'ordonnancement de type Flow-Shop avec la contrainte de blocage *RCb* exposés dans les travaux de (Sauvey et Sauer, 2012) nous ont encouragé à utiliser cette méthode sur les problèmes de type Flow-Shop avec contraintes de blocages mixtes.

L'algorithme génétique est composé de deux phases principales: la sélection et l'évaluation (Figure 5.6).

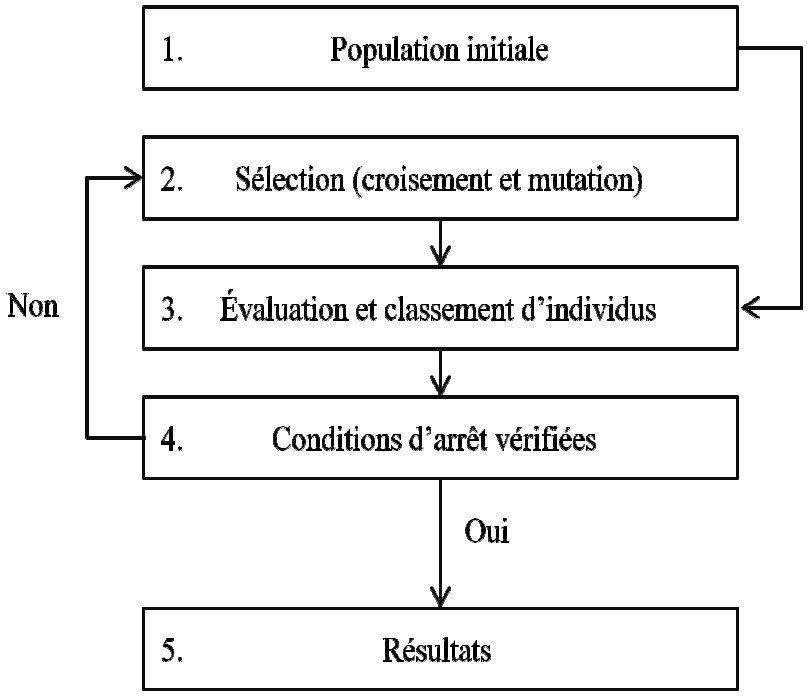


Figure 5.6. Diagramme de l'algorithme génétique

Une population initiale est générée aléatoirement et directement évaluée. La population est un ensemble d'individus comportant chacun *N* gênes (jobs) qui forment le chromosome (séquence).

L'évaluation dans notre problème consiste à calculer le makespan de chaque individu afin de trier la population. Ensuite, une sélection est opérée sur les individus afin de déterminer ceux qui sont susceptibles de donner de meilleurs résultats. Notre sélection comprend le croisement, la mutation et l'ajout de nouveaux individus à chaque nouvelle étape :

- Le croisement permet d'enrichir la population en sélectionnant deux individus, les parents, et en générant deux autres individus, les enfants, en manipulant leurs chromosomes de la façon suivante : On coupe les chromosomes de chacun des deux parents à un même endroit choisi aléatoirement. Chaque enfant est construit provisoirement avec le début du chromosome d'un de ses parents et la fin du chromosome de l'autre (Figure 5.7). Comme ce procédé engendre des répétitions de gênes, nous procédons à une réparation qui consiste à échanger les gênes répétés d'un enfant à l'autre (Figure 5.7). Après cette phase de réparation, nous obtenons deux enfants « viables », c'est-à-dire correspondant effectivement à une solution possible du problème d'ordonnancement.
- La mutation est une modification aléatoire des gènes d'un individu. On choisit aléatoirement deux gênes repérés par deux positions sur le chromosome de l'individu de départ (celui qui va subir la mutation). On opère ensuite la permutation de ces deux gênes afin d'obtenir l'individu solution, c'est-à-dire celui qui résulte de la mutation.
- Une solution au problème d'ordonnancement est donnée par un ordre particulier de jobs (par exemple $J_6 - J_2 - J_3$…). Chaque solution est représentée dans l'algorithme génétique par un individu dont le code génétique est la suite de nombres correspondants aux numéros des jobs (par exemple 6 – 2 – 3…).

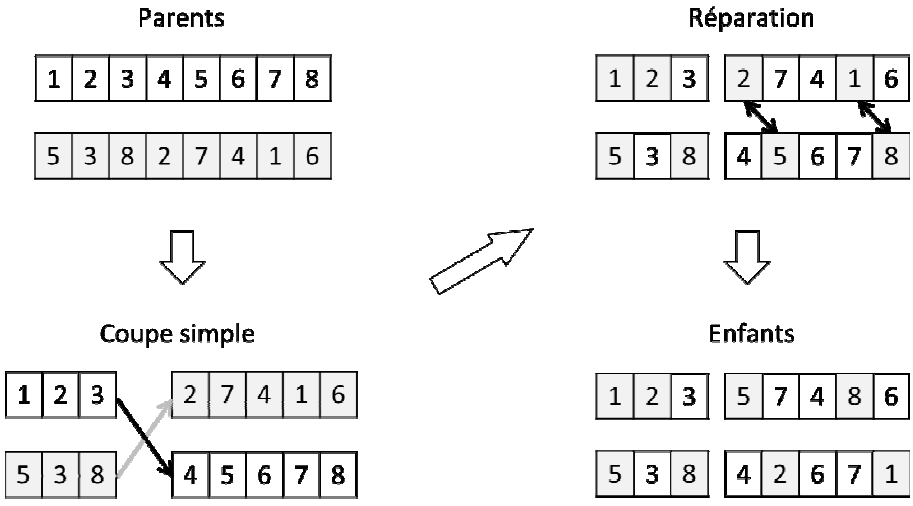


Figure 5.7. Croisement de deux individus

Dans notre algorithme génétique, nous pouvons imposer le pourcentage d'individus (*pc_best*) gardés d'une génération à la suivante, le pourcentage d'insertion d'une nouvelle population générée de façon aléatoire (*pc_new*) et le pourcentage de croisement (*pc_cross*). Les individus sélectionnés pour le croisement sont pris parmi les meilleurs individus de la génération précédente et de la nouvelle population générée aléatoirement. Le pourcentage restant de la génération créée est construit par mutation. Dans notre algorithme, les individus sélectionnés pour la mutation le sont parmi les meilleurs individus de la génération précédente. En effet, en procédant ainsi, on souhaite voir si des individus au code génétique très proche sont encore meilleurs. Cette mutation d'individus se rapproche de la notion de voisinage que l'on peut trouver dans les algorithmes de recuit simulé.

Comme représenté sur la Figure 5.6, le contenu de la boucle composée des étapes 2 et 3 est répété aussi longtemps que la condition de l'étape 4 est vraie. Nous avons choisi d'arrêter l'algorithme quand la population n'évolue plus assez rapidement. Nous espérons alors qu'elle se trouve à proximité de la solution optimale. Le critère que nous avons retenu pour arrêter nos simulations est le nombre de fois où la meilleure solution est identique (*ctbvi*). Il est simple de mettre un compteur de boucle pour compter le nombre de fois où la boucle 2-3-4 est effectuée. À chaque itération de cette boucle, nous espérons que l'évaluation de la nouvelle population donnera de meilleurs individus que le meilleur individu déjà trouvé jusqu'à présent. Ensuite, si le meilleur individu de la nouvelle population améliore la solution de l'algorithme, il est alors mémorisé comme la solution temporaire de l'algorithme et le compteur de cette boucle est réinitialisé à 0. Si la solution temporaire n'a pas été améliorée par la population de la boucle en cours, ce compteur est incrémenté. La solution retenue par l'algorithme est une solution temporaire qui n'a pas évolué depuis *ctbvi* boucles.

5. 4. 2. Résultats expérimentaux

Les pourcentages de croisement, de mutation et de nouveaux individus sont réglables. Pour effectuer les travaux que nous présentons, nous avons défini pour nos expériences, les pourcentages suivants : *pc_best* = 0,1 ; *pc_new* = 0,1 ; *pc_cross* = 0,6. La valeur de *ctbvi* que nous avons choisie pour arrêter nos simulations est *ctbvi* = 500. Ces valeurs ont été choisies à partir d'un plan d'expériences pour trouver les meilleurs paramètres définis pour lancer l'algorithme génétique dans les meilleures conditions. L'algorithme génétique a été programmé avec le langage "C++" et exécutés sur un PC Core 2 Duo' 3.16 GHz.

Dans le Tableau 5.4, nous pouvons voir, dans un premier temps (lignes -a-), le pourcentage d'erreur moyen entre la solution optimale et le makespan obtenu en utilisant un algorithme

génétique avec 50 individus. Nous apportons les améliorations locales TSS et NEH sur les solutions retenues (lignes -b-). Puis, (lignes -c-) nous prenons la meilleure solution qui donne le meilleur makespan entre le dernier algorithme génétique avec les améliorations locales (-b-) et un autre algorithme génétique avec 100 individus. Le temps de calcul de (-c-) est ensuite présenté.

Métaheuristiques	Jobs/Mach	5	6	7	10	15	20	50	100
a		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
b	5	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Temps d'exécution (s)		0	0	0	0	0	0	1	2,06
a		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
b	6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Temps d'exécution (s)		0	0	0	0	0	0,02	1,03	3,01
a		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
b	7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
c		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Temps d'exécution (s)		0	0	0	0,02	0,1	0,11	1,42	3,45
a		0,01	0,00	0,03	0,03	0,03	0,05	0,00	0,02
b	8	0,01	0,00	0,01	0,03	0,02	0,05	0,00	0,02
c		0,00	0,00	0,00	0,00	0,00	0,01	0,00	0,00
Temps d'exécution (s)		0	0,01	0,05	0,25	0,4	0,59	1,97	5,06
a		0,04	0,05	0,08	0,09	0,15	0,07	0,09	0,04
b	9	0,03	0,05	0,08	0,08	0,14	0,07	0,09	0,04
c		0,00	0,00	0,00	0,02	0,04	0,01	0,02	0,01
Temps d'exécution (s)		0,05	0,08	0,14	0,51	0,85	1,15	3,61	7,15
a		0,11	0,08	0,10	0,19	0,21	0,17	0,17	0,10
b	10	0,08	0,07	0,10	0,18	0,21	0,17	0,17	0,09
c		0,04	0,02	0,06	0,05	0,10	0,04	0,05	0,05
Temps d'exécution (s)		0,13	0,05	0,13	0,44	0,76	1,1	3,23	6,1
a		0,15	0,13	0,22	0,23	0,36	0,39	0,28	0,07
b	11	0,12	0,11	0,22	0,21	0,36	0,39	0,28	0,07
c		0,02	0,04	0,09	0,10	0,20	0,29	0,12	0,04
Temps d'exécution (s)		0,48	0,23	0,69	0,82	2,15	2,95	6,5	9,95
a		0,19	0,14	0,31	0,36	0,64	0,64	0,34	0,23
b	12	0,17	0,13	0,29	0,35	0,57	0,60	0,34	0,23
c		0,06	0,06	0,17	0,25	0,34	0,27	0,23	0,13
Temps d'exécution (s)		0,56	0,39	0,61	1	2,35	2,65	5,3	11,15

Tab. 5.4 : Pourcentage d'erreur entre la solution optimale et le makespan trouvé en utilisant différentes métaheuristiques

Nous constatons que les algorithmes génétiques sont très efficaces : 0% d'erreur pour des problèmes jusqu'à 8 jobs avec un temps de calcul souvent négligeable et le pourcentage d'erreur reste inférieur à 0,34% pour tous les autres problèmes jusqu'à 12 jobs et 100 machines avec un temps de calcul raisonnable.

Si nous explorons un peu plus les résultats, nous trouvons qu'il n'y a pas une énorme amélioration des résultats suite à l'ajout des améliorations locales NEH et TSS (ligne -b-). Il faut tempérer cette constatation par le fait que, même s'il est faible, les améliorations locales ont quand même un effet. Cet effet nous parait d'autant plus faible que la solution donnée par l'algorithme génétique est déjà de très bonne qualité (moins de 1% d'erreur). Il serait intéressant d'observer cet effet des améliorations locales lorsque les solutions fournies par l'algorithme génétique seront de moins bonne qualité.

Toutefois, au niveau de la ligne (-c-), nous voyons une nette amélioration sur les solutions obtenues, ce qui nous permet de constater que si on aborde un problème sous différents angles (à savoir l'utilisation de deux algorithmes génétiques ou plus), on aura plus de chance d'améliorer la solution obtenue que d'ajouter d'autres améliorations locales sur la solution en cours. Il est donc intéressant de lancer plusieurs fois cette métaheuristique sur un même problème.

5. 5. Conclusion

Ces heuristiques et l'algorithme génétique ont fait le sujet d'une publication dans la revue internationale ***Computers & Operations Research*** *(Trabelsi et al., 2012a).*

Dans ce chapitre, nous avons présenté différentes méthodes de résolution approchées pour résoudre les problèmes de type Flow-Shop avec blocage mixte. Nous avons, tout d'abord, présenté une borne inférieure basée sur le temps maximal d'occupation des machines. Puis, nous avons proposé une nouvelle heuristique TSS, ainsi qu'une adaptation de l'heuristique NEH qui compte parmi les méthodes heuristiques les plus performantes pour résoudre les problèmes de type Flow-Shop de permutation. Ensuite, nous avons apporté successivement deux améliorations locales à ces heuristiques. La première est basée sur la méthode itérative NEH et la seconde est basée sur la permutation du job qui a le plus grand temps de blocage différentiel. Enfin, nous avons aussi proposé un algorithme génétique pour améliorer davantage nos solutions.

Pour les instances de petite taille, notre borne inférieure est assez performante (on obtient la solution optimale une à trois fois sur dix, et l'erreur est faible pour les autres cas). Elle est moins efficace pour les problèmes où le nombre de machines est supérieur à 10.

Nos heuristiques, quand elles sont combinées, améliorent remarquablement et rapidement nos solutions : moins de 1,61% d'erreurs entre la solution optimale et le makespan obtenu pour un temps de calcul inférieur à 1 seconde pour tous les problèmes testés.

En utilisant l'algorithme génétique, nous avons encore amélioré les solutions (l'erreur maximale n'est plus que de 0,34%), mais en augmentant le temps de calcul : jusqu'à 11,15 secondes pour les tailles les plus grandes des problèmes testés, ce qui reste encore acceptable.

Dans le chapitre suivant, nous adaptons toutes ces méthodes au traitement du cas du Flow-Shop hybride avec blocage mixte afin d'obtenir des résultats comparables.

Troisième partie
Le Flow-Shop hybride avec blocage mixte

Introduction

Dans cette partie, nous nous intéressons à la minimisation du makespan dans des systèmes de production de type Flow-Shop hybride soumis à des contraintes de blocage différentes entre les étages. Ce genre de problème peut être plus difficile à résoudre que les problèmes d'ordonnancement de type Flow-Shop classique. En effet, le Flow-Shop classique est un cas particulier du Flow-Shop hybride où tous les étages ne contiennent qu'une seule machine. Donc, pour le cas du Flow-Shop hybride, en plus de déterminer les dates de début et de fin des tâches, il faut aussi les affecter sur les différentes machines de chaque étage.

Cette partie est organisée de la façon suivante : dans le chapitre 6, nous présentons une méthode de résolution exacte basée sur la modélisation linéaire en nombres entiers du problème. Nous proposons ensuite, dans le chapitre 7, une borne inférieure, plusieurs heuristiques et une application de l'algorithme génétique présenté dans le chapitre 5, ainsi que les résultats obtenus afin d'analyser les performances de nos méthodes approchées.

Chapitre 6
Méthode de résolution exacte

6. 1. Introduction

Dans ce chapitre, nous allons déterminer une solution optimale pour résoudre le problème de minimisation du makespan d'un Flow-Shop hybride avec des contraintes de blocage différentes entre les étages. Nous avons proposé un modèle mathématique linéaire en nombres entiers. Cette modélisation est inspirée du modèle présenté par (Yuan *et al*., 2009) pour minimiser le makespan dans un Flow-Shop hybride avec uniquement le blocage *RCb* entre tous les étages du problème. Nous avons transformé ce modèle afin de pouvoir considérer différentes contraintes de blocage entre les différents étages du problème.

6. 2. Modèle mathématique

Dans ce paragraphe, nous commençons par présenter les paramètres, variables et contraintes du modèle. Nous présentons ensuite des résultats numériques afin d'analyser les performances de ce modèle. Ces calculs ont été réalisés sur un PC Core 2 Duo' 3.16 GHz en utilisant le logiciel d'optimisation Xpress-MP.

6. 2. 1. Paramètres

Les paramètres utilisés pour la formulation mathématique du problème d'ordonnancement de type Flow-Shop hybride avec blocage mixte sont les suivants :

- N : Nombre de jobs.
- i, j : Indices sur les jobs.
- K : Nombre d'étages.
- k : Indice sur les étages.
- M_k : Nombre de machines à l'étage k.

- m : Indice sur les machines à chaque étage.
- $O_{i,k}$: Opération du job J_i sur l'étage k.
- $P_{i,k}$: Temps d'exécution de l'opération $O_{i,k}$.
- $B_{h,k}$ = 1 si il y a un blocage de type h entre l'étage k et l'étage k+1 et 0 sinon. Avec :
  - h=1 : si il n'y a pas de blocage entre l'étage k et l'étage k+1 (*Wb*).
  - h=2 : si il y a un blocage de type *RSb* entre l'étage k et l'étage k+1.
  - h=3 : si il y a un blocage de type *RCb** entre l'étage k et l'étage k+1.
  - h=4 : si il y a un blocage de type *RCb* entre l'étage k et l'étage k+1.
- H : Constante suffisamment grande.

6. 2. 2. Variables

Les variables de décision du modèle permettant d'obtenir l'ordre de passage des jobs sur les étages sont les suivantes :

- x_{ikm} = 1, si l'opération $O_{i,k}$ est affectée à la machine m de l'étage k et 0 sinon.
- y_{ijkm} = 1, si les opérations $O_{i,k}$ et $O_{j,k}$ sont affectées à la même machine m de l'étage k et 0 sinon.
- u_{ijkm} = 1, si l'opération $O_{i,k}$ précède l'opération $O_{j,k}$ (pas nécessairement immédiatement) sur la machine m de l'étage k et 0 sinon.
- z_{ijkm} = 1, si l'opération $O_{i,k}$ précède immédiatement l'opération $O_{j,k}$ sur la machine m de l'étage k et 0 sinon.
- $S_{i,k}$ = Date de début du job J_i sur l'étage k.
- $C_{j,k}$ = Date de fin du job J_i sur l'étage k.

6. 2. 3. Modèle

En utilisant les paramètres et les variables décrits précédemment, le problème peut être modélisé de la façon suivante :

$$Min\ C_{\max} \qquad 6.1$$

Sous les contraintes :

$$C_{\max} \geq C_{iK}, \forall i \in \{1,..,N\} \quad 6.2$$

$$C_{ik} = S_{ik} + P_{ik}, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\} \quad 6.3$$

$$S_{ik} \geq C_{i(k-1)}, \forall i \in \{1,..,N\}, \forall k \in \{2,..,K\} \quad 6.4$$

$$S_{jk} + H(1 - z_{ijkm}) \geq C_{ik}.B_{1k} + S_{i(k+1)}.B_{2k} + C_{i(k+1)}.B_{3k} + S_{i(k+2)}.B_{4k}, \quad \forall i, j \in \{1,..,N\}, i \neq j, \forall k \in \{1,..,K-2\}, \forall m \in \{1,..,M_k\} \quad 6.5$$

$$S_{j(K\text{-}1)} + H(1 - z_{ij(K-1)m}) \geq C_{i(K\text{-}1)}.B_{1(K\text{-}1)} + S_{i(K\text{-}1)}.B_{2(K\text{-}1)} + C_{i(K\text{-}1)}.B_{3(K\text{-}1)}, \quad \forall i, j \in \{1,..,N\}, i \neq j, \forall m \in \{1,..,M_k\} \quad 6.6$$

$$S_{jK} + H(1 - z_{ijKm}) \geq C_{iK}, \forall i, j \in \{1,..,N\}, i \neq j, \forall m \in \{1,..,M_k\} \quad 6.7$$

$$\sum_{m=1}^{M_k} x_{ikm} = 1, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\} \quad 6.8$$

$$2y_{ijkm} \leq x_{ikm} + x_{jkm} \leq 2y_{ijkm} + 1, \forall i, j \in \{1,..,N\}, i \neq j, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.9$$

$$y_{ijkm} = y_{jikm}, \forall i, j \in \{1,..,N\}, i \neq j, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.10$$

$$u_{ijkm} + u_{jikm} = y_{ijkm}, \forall i, j \in \{1,..,N\}, i \neq j, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.11$$

$$\sum_{j=1,\ j\neq i}^{N} z_{ijkm} \leq 1 \quad et \quad \sum_{j=1,\ j\neq i}^{N} z_{jikm} \leq 1, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.12$$

$$2x_{ikm} - 2 \leq \sum_{j=1,\ j\neq i}^{N} (z_{ijkm} + z_{jikm}) \leq 2x_{ikm}, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.13$$

$$u_{ijkm} - z_{ijkm} \geq 0, \forall i, j \in \{1,..,N\}, i \neq j, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.14$$

$$\sum_{i=1}^{N} \sum_{j=1,\ j\neq i}^{N} z_{ijkm} \geq \sum_{i=1}^{N} x_{ikm} - 1 \ \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.15$$

$$S_{ik} \geq 0, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\} \quad 6.16$$

$$x_{ikm}, y_{ijkm}, u_{ijkm}, z_{ijkm} \in \{0,1\}, \forall i \in \{1,..,N\}, \forall k \in \{1,..,K\}, \forall m \in \{1,..,M_k\} \quad 6.17$$

6. 2. 4. Signification des équations

Nous pouvons donner les explications suivantes sur les contraintes du modèle que nous proposons.

- Équation 6.1 : La fonction objectif de notre problème est de minimiser la date de fin de l'ordonnancement, appelée C_{max} ou encore makespan.

- Équation 6.2 : Cette équation impose que la valeur du makespan soit supérieure ou égale à la date de fin de tous les jobs sur le dernier étage.
- Équation 6.3 : Cette équation calcule la date de fin de chaque job sur chaque étage.
- Équation 6.4 : Cette équation représente la contrainte de précédence entre deux opérations successives d'un même job. Un job doit finir son opération sur l'étage *k*-1, avant de pouvoir démarrer son opération sur l'étage suivant *k*.
- Équation 6.5 : Cette équation modélise la contrainte de précédence en prenant en considération les différentes contraintes de blocage représentées par le paramètre $B_{h,k}$. Par exemple, s'il y a un blocage de type *RSb* entre l'étage *k* et l'étage *k*+1, l'équation 6.5 devient :

$$S_{jk} + H(1 - z_{ijkm}) \geq S_{i(k+1)}$$

- Équation 6.6 : Cette équation est l'application de l'équation précédente au cas particulier de l'avant dernier étage.
- Équation 6.7 : Cette équation est l'application de l'équation 6.5 au cas particulier du dernier étage, qui ne peut être que sans blocage.
- Équation 6.8 : Cette équation garantit que chaque opération est affectée à exactement une machine à chaque étage.
- Équations 6.9 et 6.10 : Ces équations définissent que $y_{ijkm} = y_{jikm} = 1$, si et seulement si, $x_{ikm} = x_{jkm} = 1$, *i.e.* O_{ik} et O_{jk} sont toutes les deux affectées à la même machine *m* de l'étage *k*.
- Équation 6.11 : Cette équation garantit que un et un seul de u_{ijkm} ou u_{jikm} est égal à 1 si $y_{ijkm} = 1$. Si $y_{ijkm} = 0$, alors $u_{ijkm} = u_{jikm} = 0$.
- Équation 6.12 : Cette équation garantit que chaque opération a au maximum un prédécesseur et un successeur sur une machine *m*.
- Équation 6.13 : Cette équation garantit que les opérations sont traitées dans des séquences bien définies.
- Équation 6.14 : Lorsque l'opération O_{ik} précède immédiatement l'opération O_{jk} (z_{ijkm} est égal à 1), alors u_{ijkm} est forcément égal à 1. De même, lorsque l'opération O_{ik} ne précède pas l'opération O_{jk} (u_{ijkm} est égal à 0), alors z_{ijkm} est forcément égal à 0.
- Équation 6.15 : Cette équation garantit la cohérence entre le nombres de précédences sur une machine *m* de l'étage *k* et le nombre d'opérations exécutées sur cette machine. On a, le nombre de précédences = (nombre d'opérations sur cette machine) -1.
- Équation 4.16 : La date de début des opérations de chaque job ne peut pas être négative.

- Équation 4.17 : Les variables de décision *x*, *y*, *u* et *z* sont binaires.

6. 2. 5. Résultats expérimentaux

Nous présentons ici les résultats numériques obtenus pour des problèmes de différentes dimensions qui se trouvent dans les benchmarks de Vignier (Néron *et al*., 2001) résolus par une méthode Branch and Bound.

Pour chaque dimension fixée de *N* jobs, *K* étages et M_k machines, 6 instances différentes existent. Les durées des opérations ont été générées uniformément dans l'intervalle [0, 20]. Le Tableau 6.1 présente une comparaison des temps moyens de calcul utilisés par notre modèle pour résoudre les problèmes de type Flow-Shop hybride sans blocage (*Wb*) et avec blocage mixte. Ces calculs ont été réalisés sur un PC Core 2 Duo' 3.16 GHz en utilisant le logiciel d'optimisation Xpress-MP.

Pour les problèmes avec blocage mixte, nous avons choisi arbitrairement la séquence de contraintes de blocage suivante : *V* = (*RCb**, *RSb*, *RCb*, *Wb*). Cette séquence de contraintes sera adaptée en fonction du nombre d'étages du système considéré. Ce vecteur contient *K*-1 éléments, c'est-à-dire autant que de transitions entre les étages.

Étages	M_k	*j* 5		*j* 6	
		(*Wb*)	(*mixte*)	(*Wb*)	(*mixte*)
K =2	(1 3)	0.51	0.48	272,48	26,77
	(3 1)	0.49	0.75	478,14	175,29
	(3 3)	0.41	2.46	2,69	2,36
K =3	(1 3 3)	1.26	0.87	>3600	700,67
	(3 3 1)	0.76	1.64	>3600	532,54
	(3 3 3)	0.69	0.78	6,00	6,03
	(3 1 3)	1.19	1.21	1884,85	110,37
K =5	(1 3 3 2 3)	46.32	3.44	>3600	>3600
	(3 3 2 3 1)	12.04	7.55	>3600	>3600
	(3 3 3 3 3)	76.09	17.68	>3600	746,76
	(3 3 1 2 2)	516.36	1.83	>3600	>3600

Tab. 6.1 : Comparaison des temps d'exécution moyens entre les problèmes de 5 et 6 jobs sans blocage et avec blocage mixte (en secondes)

Dans le Tableau 6.1, nous remarquons que pour les problèmes à 5 jobs et 2 ou 3 étages, le temps d'exécution moyen pour le cas du blocage mixte est très proche de celui des problèmes classiques (*Wb*). Il devient nettement plus rapide pour les problèmes à 5 étages. Cela peut s'expliquer par la complexité des problèmes de type Flow-Shop où l'on considère différentes contraintes de blocage entre les machines, qui a été étudiée dans le chapitre 3. Nous y avons montré que certains cas avec blocage mixte sont plus faciles à traiter que ceux avec une seule contrainte de blocage entre toutes les machines, pour le même nombre de machines.

Nous constatons aussi que les temps d'exécution augmentent avec le nombre d'étages (k=5) et le nombre de jobs (j=6) et que pour les problèmes de taille importante, il devient plus difficile d'obtenir une solution exacte à ces problèmes en un temps raisonnable. En effet, comme nous le montre le Tableau 6.1, le temps de calcul des problèmes à 6 jobs et 5 étages devient prohibitif même dans les cas de blocage mixte.

Nous remarquons aussi que lorsqu'on a le même nombre de machines sur tous les étages, le temps de calcul dans certains cas, en particulier pour les problèmes avec 6 jobs, devient nettement plus rapide.

6. 3. Conclusion

Dans ce chapitre, nous avons présenté un modèle linéaire en nombres entiers pour résoudre le problème d'ordonnancement d'un Flow-Shop hybride avec blocage mixte.

Ce modèle a été validé avec le logiciel d'optimisation XPress-MP. Les résultats obtenus sur des problèmes de petites tailles montrent que les temps moyens d'exécution pour les problèmes avec blocage mixte sont nettement plus rapides que ceux des problèmes sans contrainte de blocage (essentiellement pour les problèmes 5jobs 5étages et pour les problèmes avec 6 jobs).

Néanmoins, le temps de calcul pour résoudre des problèmes de plus grandes tailles devient de plus en plus prohibitif. Ceci nous a encouragés à proposer, dans le chapitre suivant, des méthodes approchées pour résoudre les problèmes de taille importante dans un temps raisonnable. Le modèle mathématique développé dans ce chapitre va nous permettre d'évaluer les performances des heuristiques et métaheuristiques que nous présentons dans le chapitre suivant, pour les problèmes de petites et moyennes tailles.

Ce modèle mathématique ainsi que les bornes inférieures que nous allons présenter dans le chapitre suivant ont été présentés à la conférence ***Information Control Problems in Manufacturing 'INCOM12'*** *(Trabelsi et al., 2012b).*

Chapitre 7
Méthodes approchées

7. 1. Introduction

Dans le chapitre précédent, nous avons constaté que, pour les problèmes de taille importante, il n'est pas possible d'obtenir une solution exacte en un temps raisonnable. Il est donc nécessaire de développer des méthodes approchées et de les évaluer en relevant le pourcentage d'erreur entre la solution obtenue et la solution optimale pour les problèmes de petites tailles. Pour les problèmes de moyennes et grandes tailles que nous proposons, nous évaluerons la qualité de la solution obtenue par les bornes inférieures.

Dans ce chapitre, nous présentons différentes méthodes de résolution approchées pour résoudre les problèmes de type Flow-Shop hybride avec blocage mixte. Nous présentons tout d'abord une borne inférieure. Ensuite, nous proposons différentes méthodes heuristiques. Enfin, nous proposons comme métaheuristique un algorithme génétique.

7. 2. Bornes inférieures

Dans ce paragraphe, nous commençons par rappeler la borne inférieure classique basée sur le temps total d'exécution du plus long job puis nous proposons une nouvelle borne inférieure pour notre problème. Cette borne inférieure est basée sur le temps maximal d'occupation des étages. Ces bornes sont ensuite comparées à la solution optimale obtenue avec notre modélisation présentée dans le chapitre précédent pour les problèmes d'ordonnancement de type Flow-Shop hybride avec blocage mixte et de petites tailles. Nous comparons aussi ces bornes avec les solutions optimales données pour les benchmarks de Vignier (Néron *et al.*, 2001) pour les problèmes d'ordonnancement de type Flow-Shop hybride sans aucune contrainte de blocage entre les différents étages.

7. 2. 1. Borne inférieure 1

La première borne, que nous appellerons *LB*1, est basée sur le temps total d'exécution du plus long job. Cette borne est indépendante du nombre de machines par étage et signifie que l'ordonnancement ne peut se terminer avant que le plus long job soit terminé.

Soit un Flow-Shop hybride composé de *N* jobs et *K* étages soumis à différentes contraintes de blocage. P_{ik} est le temps d'exécution du job J_i à l'étage *k* et on suppose qu'à chaque étage *k*, il existe une ou plusieurs machines identiques ($M_k \geq 1$). Évidemment, aucun job ne peut finir avant son temps total d'exécution. Par conséquent, l'ensemble des jobs ne peuvent finir avant le temps total d'exécution du plus long job. Une borne inférieure classique du makespan du problème d'ordonnancement de ce système est donc donnée par :

$$LB1 = \max_{i=1,\dots N}\left[\sum_{k=1}^{K} P_{ik}\right] \qquad 7.1$$

7. 2. 2. Borne inférieure 2

Dans le but d'estimer le pourcentage d'erreur des heuristiques que nous présentons par la suite, nous proposons une deuxième borne inférieure basée sur le temps maximal d'occupation des étages. Cette proposition est inspirée de la borne inférieure utilisée pour les problèmes d'ordonnancement de type Flow-Shop hybride sans contrainte de blocage (Santos *et al.*, 1995) et avec la contrainte de blocage *RCb* entre tous les étages (Yuan *et al.*, 2009). Nous avons adapté ces bornes inférieures afin de considérer différentes contraintes de blocage entre les différents étages du même problème.

Théorème 8. Soit un Flow-Shop hybride composé de *N* jobs, *K* étages et M_k machines à chaque étage, soumis à différentes contraintes de blocage. Une borne inférieure du makespan du problème d'ordonnancement de ce système est donnée par :

$$LB2 = \max_{k=1,\dots,K} \frac{1}{M_k} \cdot \left[\sum_{i=1}^{N}\left[P_{ik} + P_{i(k+1)} \cdot \psi_k\right] + \sum_{y=1}^{M_k} LS(a_y, k) + \sum_{y=1}^{M_k} RS(b_y, k)\right] \qquad 7.2$$

Avec :

- $\psi_k = \begin{cases} 1 & \textit{si } k < K \textit{ et s' il existe une contrainte de blocage RCb ou RCb * entre les étages } k \textit{ et } k+1 \\ 0 & \textit{sinon} \end{cases}$

- $LS(i,k)$ (left side sum) : Somme des temps opératoires du job J_i, jusqu'à l'étage *k*-1 :

$$LS(i,k) = \begin{cases} \sum_{k'=1}^{k-1} P_{ik'} & \text{si } k > 1 \\ 0 & \text{si } k = 1 \end{cases} \quad 7.3$$

$LSA(k)$: Liste triée dans l'ordre croissant de $LS(i,k)$:

$LSA(k) = \{LS(a_1,k), LS(a_2,k),\ldots, LS(a_N,k)\}$ avec $LS(a_r,k) \leq LS(a_{r+1},k)$.

- $RS(i,k)$ (right side sum) : Somme des temps opératoires du job J_i, à partir de l'étage k+1 :

$$RS(i,k) = \begin{cases} \sum_{k'=k+1}^{K} P_{ik'} - P_{i(k+1)} \cdot \psi_k & si\ k < K \\ 0 & si\ k = K \end{cases} \quad 7.4$$

$RSA(k)$: Liste triée dans l'ordre croissant de $RS(i,k)$:

$RSA(k) = \{RS(b_1,k), RS(b_2,k),\ldots, RS(b_N,k)\}$ avec $RS(b_r,k) \leq RS(b_{r+1},k)$.

Preuve :

Soit $F(m,k)$, où $m \in \{1,..,M_k\}$, l'ensemble des jobs qui sont affectés à la machine m de l'étage k dans un ordonnancement qui donne le makespan optimal. Pour chaque machine m de l'étage k , il existe un job (noté $J_{f(m,k)}$) qui arrive en premier sur cette machine et un job qui est le dernier à passer sur cette machine (noté $J_{l(m,k)}$) pour l'ordonnancement optimal. Chaque job affecté à une machine de l'étage k a une date de fin associée. La date de fin du job $J_{l(m,k)}$ est alors une borne inférieure pour le maximum des dates de fin des jobs de l'ensemble $F(m,k)$.

Soit $C(m,k)$ la date de fin du job $J_{l(m,k)}$ pour l'ordonnancement optimal. Nous allons associer une borne inférieure à chaque $C(m,k)$.

Tous les jobs de l'ensemble $F(m,k)$ doivent être traités sur la machine m à l'étage k avant que le job $J_{l(m,k)}$ débute sur l'étage suivant dans la gamme.

Dans le cas sans blocage (*Wb*) ou avec blocage *RSb* entre les étages k et k+1, ce temps est équivalent à la somme des temps opératoires de ces jobs à l'étage k :

$$T1.a = \sum_{i' \in F(m,k)} [P_{i'k}] \quad 7.5$$

S'il existe une contrainte de blocage *RCb* ou *RCb** entre les étages k et k+1, ce temps sera défini par $T1.b$ (Figure 7.1):

$$T1.b = \sum_{i' \in F(m,k)} \left[P_{i'k} + P_{i'(k+1)} \right] - P_{l(m,k)k+1} \qquad 7.6$$

A partir des équations 7.5 et 7.6, nous obtenons donc le temps $T1$ défini par :

$$T1 = \sum_{i' \in F(m,k)} \left[P_{i'k} + P_{i'(k+1)} \cdot \psi_k \right] - P_{l(m,k)k+1} \cdot \psi_k \qquad 7.7$$

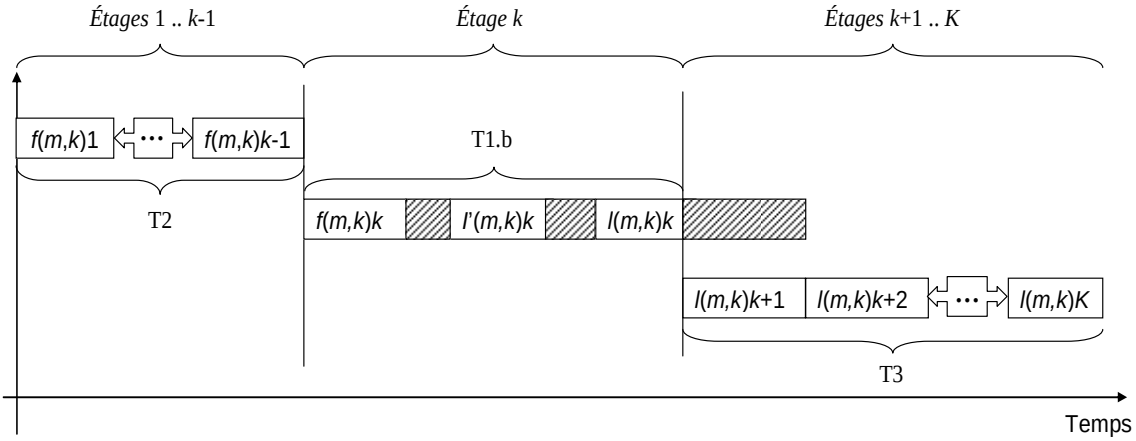


Figure 7.1. Temps d'attente avant et après l'exécution des opérations affectées à la machine m de l'étage k

Avant de pouvoir débuter l'exécution de l'opération du premier job $J_{f(m,k)}$ sur la machine m de l'étage k, il faut que toutes ses opérations sur les étages précédents soient terminées (Figure 7.1). Par conséquent, la machine m ne peut pas débuter une opération avant un temps nécessaire $T2$ défini par :

$$T2 = \sum_{k'=1}^{k-1} P_{f(m,k)k'} \qquad 7.8$$

De la même façon, après l'exécution de l'opération du dernier job $J_{l(m,k)}$ sur la machine m de l'étage k, il faut que toutes ses opérations sur les étages suivants se terminent (Figure 7.1). Ce temps est défini par :

$$T3 = \sum_{k'=k+1}^{K} [P_{l(m,k)k'}] \qquad 7.9$$

Une borne inférieure peut donc être associée à chaque $C(m,k)$ comme suit :

$$C(m,k) \geq T1 + T2 + T3 \qquad 7.10$$

L'inégalité 7.10 établit une borne inférieure sur les dates de fin de l'ensemble des jobs qui sont affectés à chaque machine m de l'étage k dans un ordonnancement optimal. Une borne pour l'étage k est alors le maximum des dates de fin, établi sur toutes les machines M_k à cet étage. Or, nous ne savons pas exactement quelles sont les tâches affectées à chaque $F(m,k)$ dans un ordonnancement optimal. Donc, nous ne pouvons pas déterminer le maximum des dates de fin de cet ensemble. Cependant, nous savons que ce maximum doit être supérieur ou égal à la moyenne des M_k makespan à cet étage. Ainsi, une borne sur le makespan total peut être établie dans l'inégalité suivante :

$$\max_{k=1..K} C(m,k) \geq \frac{1}{M_k} \cdot \sum_{m=1}^{M_k} C(m,k) \qquad 7.11$$

À partir des équations 7.3 - 7.10, le second terme de l'inégalité 7.11 peut être écrit comme suit :

$$\frac{1}{M_k} \cdot (X + Y + Z) \qquad 7.12$$

Où :

- $X = \sum_{m=1}^{M_k} \sum_{i' \in F(m,k)} \left[P_{i'k} + P_{i'(k+1)} \cdot \psi_k \right]$
- $Y = \sum_{m=1}^{M_k} \sum_{k'=1}^{k-1} P_{f(m,k)k'}$
- $Z = \sum_{m=1}^{M_k} \left[\sum_{k'=k+1}^{K} [P_{l(m,\mathrm{k})\mathrm{k}'}] - P_{l(m,\mathrm{k})\mathrm{k}+1} \cdot \psi_k \right]$

Or, un job est nécessairement affecté à une et une seule machine de l'étage k. Par conséquent :

$$X = \sum_{m=1}^{M_k} \sum_{i' \in F(m,k)} \left[P_{i'k} + P_{i'(k+1)} \cdot \psi_k \right] = \sum_{i=1}^{N} \left[P_{ik} + P_{i(k+1)} \cdot \psi_k \right] \qquad 7.13$$

De plus, si l'on considère les M_k $LS(i,k)$ temps les plus court, nous avons :

$$Y = \sum_{m=1}^{M_k} \sum_{k'=1}^{k-1} P_{f(m,k)k'} \geq \sum_{y=1}^{M_k} LS(a_y, k) \qquad 7.14$$

De même, nous obtenons :

$$Z = \sum_{m=1}^{M_k} \left[\sum_{k'=k+1}^{K} [P_{l(m,\mathrm{k})\mathrm{k}'}] - P_{l(m,\mathrm{k})\mathrm{k}+1} \cdot \psi_k \right] \geq \sum_{y=1}^{M_k} RS(b_y, k) \qquad 7.15$$

D'où, la borne inférieure du makespan du problème, notée *LB*2 (Équation 7.2) :

$$LB2 = \max_{k=1,\ldots,K} \frac{1}{M_k} \cdot \left[\sum_{i=1}^{N} \left[P_{ik} + P_{i(k+1)} \cdot \psi_k \right] + \sum_{y=1}^{M_k} LS(a_y, k) + \sum_{y=1}^{M_k} RS(b_y, k) \right]$$

CQFD

Nous avons proposé deux bornes inférieures du makespan et nous retenons le maximum des deux solutions comme la meilleure borne inférieure. Par conséquent, la borne inférieure du makespan du problème d'ordonnancement d'un Flow-Shop hybride soumis à différentes contraintes de blocage, notée B_{inf}, est comme suit :

$$B_{inf} = \max\{LB1; LB2\} \qquad 7.16$$

Exemple :

On considère un problème d'ordonnancement d'un Flow-Shop hybride constitué de 5 jobs, 5 étages et au plus 3 machines par étage.

Les temps opératoires ainsi que le nombre de machines par étage sont donnés dans le tableau suivant :

Étages	1	2	3	4	5
Machines/Étage	1	3	3	2	3
*J*1	12	15	8	9	30
*J*2	37	19	28	17	8
*J*3	17	18	2	56	19
*J*4	79	8	18	29	38
*J*5	28	18	28	18	27

Tab. 7.1 : Temps opératoires des jobs par étage pour un FSH à 5 jobs / 5 étages

La gamme de fabrication est fixée d'avance, telle que chaque job passe par tous les étages dans le même ordre, de l'étage 1 jusqu'à l'étage 5. Nous considérons le vecteur de contraintes de blocage $V = (RCb^*, RSb, RCb, Wb)$. Ce vecteur indique le type de contrainte entre les étages du problème (*i.e*. *RCb** est la contrainte de blocage entre les étages 1 et 2, *RSb* est celle entre les étages 2 et 3, etc.).

D'après l'équation 7.1, la valeur de la première borne inférieure est la suivante :

$$LB1 = \max_{i=1,\dots N}\left[\sum_{k=1}^{K} P_{ik}\right] = \max(74;109;112;172;119) = 172ut$$

Pour obtenir la valeur de la deuxième borne inférieure, on effectue les calculs suivants :

- La somme des temps opératoires du job J_i, jusqu'à l'étage k-1 :

$$LS(i,k) = \begin{cases} \sum_{k'=1}^{k-1} P_{ik'} & k > 1 \\ 0 & k = 1 \end{cases}$$

$LS(i,k)$	1	2	3	4	5
$J1$	0	12	27	35	44
$J2$	0	37	56	84	101
$J3$	0	17	35	37	93
$J4$	0	79	87	105	134
$J5$	0	28	46	74	92

- La somme des temps opératoires du job J_i, à partir de l'étage k+1 :

$$RS(i,k) = \begin{cases} \sum_{k'=k+1}^{K} P_{ik'} - P_{i(k+1)} \cdot \psi_k & k < K \\ 0 & k = K \end{cases}$$

Il y a une contrainte de blocage de type *RCb** entre les deux premiers étages, alors $RS(i,1)$ est comme suit :

$$RS(i,1) = \sum_{k'=2}^{K} P_{ik'} - P_{i2}$$

Alors qu'entre les étages 2 et 3, il existe une contrainte de blocage de type *RSb*. Donc *RS*(*i*,2) est comme suit :

$$RS(i,2) = \sum_{k'=3}^{K} P_{ik'}$$

On applique pour chaque type de blocage la formule correspondante et on obtient ainsi :

RS(*i*,*k*)	1	2	3	4	5
*J*1	47	47	30	30	0
*J*2	53	53	8	8	0
*J*3	77	77	19	19	0
*J*4	85	85	38	38	0
*J*5	73	73	27	27	0

- On fait le tri pour chaque liste :

LSA(1) ={0,0,0,0,0} *RSA*(1) ={47,53,73,77,85}

LSA(2) ={12,17,28,37,79} *RSA*(2) ={47,53,73,77,85}

LSA(3) ={27,35,46,56,87} *RSA*(3) ={8,19,27,30,38}

LSA(4) ={35,37,74,84,105} *RSA*(4) ={8,19,27,30,38}

LSA(5) ={44,92,93,101,134} *RSA*(5) ={0,0,0,0,0}

- On peut maintenant calculer la borne inférieure du makespan pour chaque étage :

$$C(m,k) \geq \frac{1}{M_k} \cdot \left[\sum_{i=1}^{N} \left[P_{ik} + P_{i(k+1)} \cdot \psi_k \right] + \sum_{y=1}^{M_k} LS(a_y,k) + \sum_{y=1}^{M_k} RS(b_y,k) \right]$$

$C(m,1) \geq$ [(12+...+28) + (15+…+18) + 0 + 47] / 1 = **298 ut**

$C(m,2) \geq$ [(15+…+18) + (12+17+28) + (47+53+73)] / 3 = **102,67 ut**

$C(m,3) \geq$ [(8+…+28) + (9+…+18) + (27+35+46) + (8+19+27)] / 3 = **125 ut**

$C(m,4) \geq$ [(9..18) + (35+37) + (8+19)] / 2 = **114 ut**

$C(m,5) \geq$ [(30+…+27) + (44+92+93) + 0] / 3 = **117 ut**

D'où, $LB2 = \max_{k=1,...K} C(m,k) =$ **298 ut**

- Finalement, on garde la meilleure borne inférieure :

  B_{inf} = max{$LB1$; $LB2$} = max{172 ; 298} = 298 ut.

Nous avons calculé la solution optimale de cet exemple avec le modèle mathématique présenté dans le chapitre précédent et nous avons obtenu la valeur 298 ut comme makespan optimal.

Ainsi, sur cet exemple, la borne inférieure B_{inf} donne la solution optimale.

7. 2. 3. Résultats expérimentaux

Pour mieux évaluer les bornes inférieures proposées, nous avons utilisé les mêmes six instances pour chaque dimension fixée de N jobs, K étages et M_k machines (une seule instance pour le problème à 15 jobs / 5 étages marqué par un astérisque) que celles utilisées dans le chapitre 6. Nous y avions précisé, dans le Tableau 6.1, les temps d'exécution moyens pour obtenir la solution optimale des problèmes de type Flow-Shop hybride sans blocage et avec blocage mixte.

Le tableau 7.2 donne le pourcentage d'erreur des deux bornes inférieures proposées par rapport au makespan optimal ainsi que le nombre de fois où la meilleure de ces bornes est égale à la solution optimale pour les problèmes d'ordonnancement de type Flow-Shop hybride sans blocage. Le pourcentage d'erreur est défini par la formule suivante :

$$\%err = \frac{S_{opt} - B_{\inf}}{S_{opt}} \times 100 \qquad 7.17$$

Étages	Machines par étage : M_k	j 5				j 10				j 15			
		$LB1$ (%)	$LB2$ (%)	$Binf$ (%)	$Binf$ = $Cmax$	$LB1$ (%)	$LB2$ (%)	$Binf$ (%)	$Binf$ = $Cmax$	$LB1$ (%)	$LB2$ (%)	$Binf$ (%)	$Binf$ = $Cmax$
K=2	(1 3)	54,62	0	0	6	74,53	0	0	6	81,02	0	0	6
	(3 1)	53,66	0	0	6	78,14	0,19	0,19	5	82,32	0	0	6
	(3 3)	18,66	14,38	10,92	1	36,86	4,37	4,37	0	50,44	1,15	1,15	3
K=3	(1 3 3)	43,60	1,26	1,26	5	68,03	0	0	6	74,93	0	0	6
	(3 3 1)	46,90	0,48	0,48	5	64,42	0,21	0,21	5	75,04	0	0	6
	(3 3 3)	11,04	13,75	6,44	2	25,97	6,95	3,75	1	41,18	1,92	1,92	1
	(3 1 3)	44,52	0,96	0,96	5	73,67	0	0	6	74,23	0	0	6
K=5	(1 3 3 2 3)	30,95	2,48	2,48	5	53,43	0,36	0,36	5	64,84	0	0	6
	(3 3 2 3 1)	28,79	1,06	1,06	5	52,49	0	0	6	64,37	0,55	0,55	5
	(3 3 3 3 3)	6,58	12,88	3,47	3	20,08	7,75	4,59	1	22,90	2,30	0,97	4
	(3 3 1 2 2)	28,55	2,32	2,32	3	54,00	1,08	1,08	4	70,83	0	0	1*

Tab. 7.2 : Pourcentage d'erreur des bornes inférieures et le nombre de fois où $B_{inf} = C_{max}$ (sans blocage)

Les résultats des tests réalisés dans le cas du blocage mixte, sont donnés dans le Tableau 7.3.

Étages	Machines par étage : M_k	*j 5*			
		$LB1$ (%)	$LB2$ (%)	$Binf$ (%)	$Binf = Cmax$
K=2	(1 3)	74,44	0	0	6
	(3 1)	53,66	0	0	6
	(3 3)	38,86	16,12	16,12	0
K=3	(1 3 3)	64,75	0	0	6
	(3 3 1)	47,31	0,93	0,93	5
	(3 3 3)	24,16	12,58	11,96	2
	(3 1 3)	44,52	0,96	0,96	5
K=5	(1 3 3 2 3)	53,23	0	0	6
	(3 3 2 3 1)	30,79	3,54	3,54	3
	(3 3 3 3 3)	13,95	10,52	6,73	1
	(3 3 1 2 2)	53,05	0,17	0,17	5

Tab. 7.3 : Pourcentage d'erreur des bornes inférieures et le nombre de fois où $B_{inf} = C_{max}$ (blocage mixte)

À partir de ces deux tableaux, nous pouvons analyser les résultats selon un paramètre important qui est le nombre de machines par étage. Nous observons que pour les problèmes uniformes (le nombre de machines est le même sur tous les étages), les bornes inférieures sont moins efficaces que pour les problèmes non-uniformes.

En effet, dans le cas des problèmes non-uniformes, la borne inférieure *LB*2 est souvent très proche de la solution optimale voire égale à S_{opt} pour quelques instances : l'erreur maximale est de 2,48% pour les problèmes sans blocage et 3,54% pour les problèmes avec blocage mixte. Quant à la borne inférieure *LB*1, elle est nettement moins efficace.

Dans le cas des problèmes uniformes, la borne inférieure *LB*2 devient moins efficace et on obtient plus rarement la solution optimale : le pourcentage d'erreur avoisine les 14 % pour les problèmes sans blocage et peut atteindre 16,12% pour les problèmes avec blocage mixte. Par contre, c'est dans ce type de problèmes qu'intervient la borne inférieure *LB*1. En effet, le pourcentage d'erreur donné par la borne *LB*1 pour les problèmes uniformes devient beaucoup plus petit par rapport aux problèmes non-uniformes, mais surtout elle est parfois meilleure que la borne inférieure *LB*2. C'est ce qui explique l'amélioration de la borne inférieure maximale B_{inf}.

De même que pour les heuristiques, la multiplication des bornes inférieures est une très bonne chose pour l'amélioration de la précision des méthodes que nous développons.

7. 3. Heuristiques

Comme nous l'avons remarqué dans le paragraphe précédent, les bornes inférieures proposées peuvent atteindre la valeur optimale du makespan ou sont souvent très proches de la solution optimale. Cela nous permet d'évaluer les heuristiques que nous présentons dans ce chapitre.

Pour tenir compte des différentes contraintes de blocage, nous avons reprogrammé en C++ trois heuristiques simples bien connues (SPT, LPT et RAND) en ajoutant quelques modifications sur les deux premières méthodes que nous précisons dans le paragraphe suivant. Nous avons aussi programmé l'heuristique NH présentée dans (Thornton et Hunsucker, 2004) qui a prouvé son efficacité pour la résolution des problèmes de type Flow-Shop hybride avec la contrainte de blocage *RSb*.

Dans ce paragraphe, nous proposons une description de chacune de ces heuristiques, ainsi qu'une comparaison des résultats que nous avons obtenus.

7. 3. 1. Heuristique MSPT (Modified Shortest Processing Time)

La méthode SPT (Shortest Processing Time) est l'une des règles de priorité les plus connues utilisées dans l'ordonnancement de la fabrication. Elle consiste à traiter les opérations/commandes par ordre croissant des temps opératoires.

Cette procédure d'ordonnancement est simple mais souvent efficace. Sa façon d'ordonner les jobs a montré sa supériorité pour les problèmes de Flow-Shop hybride classique par rapport aux autres règles simples d'ordonnancement. C'est la raison pour laquelle nous avons choisi de l'utiliser dans notre travail, tout en ajoutant quelques modifications pour tenir compte du nombre d'étages par problème.

En effet, l'heuristique MSPT génère *K* ordonnancements (*K* étant le nombre d'étages), calcule la valeur de la fonction objectif pour chacun et prend celui donnant la meilleure solution (le Cmax le plus faible).

Le principe consiste à traiter en premier les jobs ayant les plus petits temps opératoires. Le premier ordonnancement proposé va être généré en tenant compte seulement des temps

opératoires du premier étage. La séquence d'ordonnancement est la liste croissante par rapport aux temps opératoires de tous les jobs. Les autres ordonnancements sont générés aussi selon la méthode SPT, mais la différence repose sur les temps opératoires qui sont pris en compte. Le deuxième ordonnancement va être engendré en calculant la somme des temps opératoires du premier et du deuxième étage pour chaque job. Suivant la même logique, on continue jusqu'au $K^{ème}$ ordonnancement, qui va être produit en calculant la somme des temps opératoires de chaque job sur tous les étages.

L'ordre de passage sur le premier étage se fait en utilisant l'ordonnancement défini. Lorsque plusieurs jobs attendent le traitement d'une machine dans un étage postérieur, une fois que cette machine est libérée, le choix sur le job suivant à traiter est fait selon la règle, premier arrivant premier sortant.

7. 3. 2. Heuristique MLPT (Modified Longest Processing Time)

La méthode LPT (Longest Processing Time) est aussi une règle classique de priorité. Elle procède de la même manière que la méthode SPT sauf qu'elle donne la priorité, lors de la génération de l'ordre de passage, aux jobs ayant le plus grand temps opératoire.

Les résultats de la méthode LPT sont souvent moins bons que ceux qu'offre la méthode SPT, mais quand elle est combinée avec d'autres algorithmes ou utilisée dans des procédures plus complexes, elle peut donner de bons résultats. D'ailleurs, le fameux algorithme de Johnson (Johnson, 1954) qui est optimal pour les problèmes de type Flow-Shop à 2 machines est une combinaison des deux méthodes SPT et LPT.

Nous avons donc choisi d'utiliser cette méthode dans notre heuristique pour sa simplicité et sa rapidité d'exécution. L'heuristique MLPT procède alors de la même manière que MSPT, c'est-à-dire elle génère *K* ordonnancements (*K* étant le nombre d'étages), calcule la valeur de la fonction objectif pour chacun et prend celui fournissant la meilleure solution.

7. 3. 3. Heuristique RAND (Aléatoire)

Cette procédure consiste à générer *K* ordonnancements quelconques. L'intérêt d'avoir choisi cette méthode dans notre comparaison est le même que celui de la méthode LPT : simplicité à programmer, rapidité d'exécution et possibilité de donner de bons résultats en combinant avec d'autres méthodes.

7. 3. 4. Heuristique NH

L'heuristique NH, présentée dans (Thornton et Hunsucker, 2004) a prouvé son efficacité pour la résolution des problèmes de type Flow-Shop hybride uniforme (même nombre de machines dans tous les étages) avec la contrainte de blocage *RSb*. Nous avons donc choisi de l'utiliser, en l'adaptant à notre problème pour respecter les différentes contraintes de blocage entre les étages et dans le cas où le FSH n'est pas uniforme.

L'heuristique NH est composée de deux parties importantes : le calcul du « pseudo temps des jobs » que nous allons décrire par la suite et l'application de l'algorithme de Johnson. La méthode du pseudo temps a été présentée par (Deal et Parks, 1999) et elle vise à améliorer l'approximation faite lorsqu'on ramène un problème de *K* étages à un problème à 2 étages.

Un autre point de NH est le concept qui concerne la génération de l'espace de solutions à évaluer. Les auteurs proposent de réaliser la méthode des pseudos temps pour plusieurs scénarios dans lesquels on va oublier l'existence de certains étages. Le but de cette méthode est l'élimination des anomalies que peuvent contenir les pseudos temps.

Pour expliquer les différentes étapes de l'heuristique, nous allons reprendre l'exemple donné dans (Thornton et Hunsucker, 2004) où ils considèrent un problème d'ordonnancement d'un Flow-Shop hybride constitué de 4 jobs et 5 étages.

Les temps opératoires des jobs pour chaque étage $P_{i,k}$ sont donnés dans le tableau suivant :

Étages	1	2	3	4	5
J1	7	7	5	5	10
J2	7	6	3	6	8
J3	6	9	3	2	7
J4	3	7	10	5	5

Tab. 7.4 : Temps opératoires des jobs par étage

Étape 1 : On commence par le calcul des pseudo-temps :

- Calcul du temps total d'occupation pour chaque étage. Dans notre exemple, la somme des temps de tous les jobs sur l'étage 1 est égale à 23 ut, sur l'étage 2 est égale à 29 ut etc. (Tableau 7.5)

- Calcul du temps d'occupation cumulé. Le temps cumulé pour l'étage 2 est égal à la somme des temps totaux d'occupation des étages 1 et 2, ce qui donne comme résultat 23+29 = 52 ut.
- Calcul du point central *D*. C'est la division du temps total d'occupation cumulé du dernier étage par 2. Dans notre cas, *D* est égal à 121/2 = 60,5 ut.
- Sélection de l'étage *k*. C'est l'étage ayant un temps d'occupation cumulé le plus proche du point central *D*, par valeur supérieure ou égale à *D*. Dans notre cas, cet étage *k* est le $3^{ème}$ étage, car son temps d'occupation cumulé est de 73 ut (*D* = 60,5 ut). On peut remarquer que le temps d'occupation cumulé à l'étage 2 est de 52 ut ce qui est plus proche de la valeur de *D*, mais comme 52 < 60,5 on sélectionne l'étage 3.
- Calcul de la valeur de proximité *Q*. Cette valeur représente la distance existant entre la valeur *D* et l'étage *k*–1. Pour notre exemple, on a *Q* = *D*–52 = 8,5 ut.
- Calcul du coefficient de proximité *P*. Ce coefficient est égal à la valeur de *Q* divisée par le temps d'occupation de l'étage *k*. Pour l'exemple, on trouve un *P* de 8,5/21 = 0,4048.
- Calcul des pseudos temps « *PT*(*i*) ». Ces temps vont ramener le problème de *K* étages à un problème à 2 étages (pseudo-étages) :
  - Pseudo étage 1 :

$$PT1(i) = P_{i,k} \times P + \sum_{s=1}^{k-1} P_{i,s}, \forall\, i \in \{1,..,N\}$$

  - Pseudo étage 2 :

$$PT2(i) = P_{i,k} \times (1-P) + \sum_{s=k+1}^{K} P_{i,s}, \forall\, i \in \{1,..,N\}$$

Étape 2 : On applique la règle de Johnson pour trouver la séquence optimale. La méthode de Johnson consiste à sélectionner les temps opératoires les plus courts parmi les temps de traitement sur les deux étages. Puis on applique le critère SPT (Shortest Processing Time) sur les temps opératoires du premier étage et le critère LPT (Longest Processing Time) sur les temps opératoires du deuxième étage. Dans cet exemple, la séquence est donc : {*J4, J2, J1, J3*}.

Un des points forts de cette approche est l'introduction du coefficient de proximité *P* qui vise à éviter les égalités entre les temps opératoires des 2 pseudo-étages.

Temps Total par étage	23	29	21	18	30
Temps cumulé	23	52	73	91	121

$D = 121 / 2 = 60,5$ $Q = D - 52 = 8,5$ $P = Q / 21 =$ **0,4048**

Jobs	Pseudo Étage 1	Pseudo Étage 2
J1	$7 + 7 + 5.P =$ **16,02**	$5.(1 - P) + 5 + 10 = 17,98$
J2	$7 + 6 + 3.P =$ **14,21**	$3.(1 - P) + 6 + 8 = 15,79$
J3	$6 + 9 + 3.P = 16,21$	$3.(1 - P) + 2 + 7 =$ **10,79**
J4	$3 + 7 + 10.P =$ **14,05**	$10.(1 - P) + 5 + 5 = 15,95$
Séquence de Johnson	*J4 J2 J1 J3*	*Cmax* = 56

Tab. 7.5 : Étapes de calcul du Cmax par la méthode NH

Étape 3 : On génère l'espace de solutions pour plusieurs scénarios dans lesquels on va oublier l'existence de certains étages : 2, 3 , 4, (2-3), (3-4) et (2-3-4). On recalcule les pseudos temps et on reprend la même démarche des deux premières étapes. Enfin, on garde la meilleure séquence qui donne le makespan le plus petit.

Scénario lorsque l'on ignore	Séquence de Johnson	*Cmax*
Le 2ème étage	{*J2*, *J1*, *J4*, *J3*}	55
Le 3ème étage	{*J4*, *J2*, *J1*, *J3*}	56
Le 4ème étage	{*J4*, *J1*, *J2*, *J3*}	55
Les étages 2 et 3	{*J4*, *J3*, *J1*, *J2*}	55
Les étages 3 et 4	{*J4*, *J1*, *J3*, *J2*}	55
Les étages 2, 3 et 4	{*J4*, *J1*, *J2*, *J3*}	55

7. 3. 5. Résultats expérimentaux

Dans cette partie, nous présentons les résultats que nous avons obtenus en utilisant les différentes heuristiques ainsi qu'une comparaison des différentes méthodes.

Pour valider la qualité de ces heuristiques, nous avons utilisé les mêmes 6 instances pour chaque dimension fixée de N jobs, K étages et M_k machines (une seule instance pour le problème à 15 jobs / 5 étages marqué par un astérisque) que celles utilisées dans les chapitres 6 et 7.

Le tableau 7.6 donne le pourcentage d'erreur entre la solution optimale S_{opt} et le makespan C_{max} trouvé en utilisant les différentes heuristiques pour les problèmes d'ordonnancement de type Flow-Shop hybride sans blocage. Le tableau 7.7 donne le pourcentage d'erreur entre la solution optimale S_{opt} ou la borne inférieure B_{inf} et le makespan C_{max} trouvé en utilisant les différentes heuristiques pour les problèmes d'ordonnancement de type Flow-Shop hybride avec blocage mixte. Le pourcentage d'erreur est défini par la formule suivante :

$$\%err = \frac{C_{\max} - S_{opt}}{S_{opt}} \times 100 \qquad 7.18$$

La colonne *Best* dans les deux tableaux, représente une heuristique qui combine les quatre méthodes et qui donne la meilleure solution possible fournie par les heuristiques MSPT, MLPT, RAND et NH.

Étages	Machines par étage : M_k	*j* 5					*j* 10					*j* 15				
		MSPT (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best* (%)**	*MSPT* (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best* (%)**	*MSPT* (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best* (%)**
K=2	(1 3)	4,47	2,33	5,97	2,47	**0,00**	3,69	1,11	3,97	0,00	**0,00**	3,40	0,29	3,99	0,13	**0,00**
	(3 1)	0,00	12,18	7,69	3,70	**0,00**	0,00	4,57	3,42	0,19	**0,00**	0,00	5,36	3,46	0,53	**0,00**
	(3 3)	8,29	3,88	6,54	7,80	**3,35**	21,34	15,07	19,41	11,31	**6,41**	11,27	10,92	15,21	19,95	**8,49**
K=3	(1 3 3)	4,54	3,41	2,39	0,93	**0,00**	2,79	1,41	6,42	0,00	**0,00**	2,15	1,65	6,40	0,00	**0,00**
	(3 3 1)	0,95	13,81	9,80	2,86	**0,95**	0,00	8,12	7,66	0,00	**0,00**	0,00	7,82	8,07	0,26	**0,00**
	(3 3 3)	12,78	1,82	5,66	1,82	**1,32**	21,38	17,48	24,93	9,99	**9,62**	13,42	19,83	21,18	6,40	**6,40**
	(3 1 3)	2,17	11,70	4,90	0,71	**0,00**	0,18	0,71	1,24	0,00	**0,00**	3,75	7,39	6,11	0,77	**0,77**
K=5	(1 3 3 2 3)	5,12	2,15	10,46	4,16	**0,45**	7,98	2,10	8,66	3,87	**1,81**	2,70	1,12	7,05	0,70	**0,36**
	(3 3 2 3 1)	0,53	15,54	8,06	2,08	**0,53**	1,12	9,44	6,83	4,61	**1,12**	0,28	8,88	7,96	2,82	**0,28**
	(3 3 3 3 3)	4,03	1,39	5,20	2,78	**1,39**	13,32	12,97	21,64	10,14	**9,61**	17,53	22,68	22,85	13,52	**13,29**
	(3 3 1 2 2)	6,89	8,65	15,82	0,00	**0,00**	1,40	8,47	11,59	0,33	**0,33**	0,69*	2,78*	2,78*	0*	**0***

Tab. 7.6 : Pourcentage d'erreur entre la solution optimale (ou la B_{inf}) et le makespan trouvé en utilisant différentes heuristiques (sans blocage)

Étages	Machines par étage : M_k	*j* 5					*j* 10					*j* 15				
		MSPT (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best*** **(%)**	*MSPT* (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best*** **(%)**	*MSPT* (%)	*MLPT* (%)	*RAND* (%)	*NH* (%)	***Best*** **(%)**
K=2	(1 3)	0,00	0,00	0,00	0,00	**0,00**	0,00	0,00	0,00	0,00	**0,00**	0,00	0,00	0,00	0,00	**0,00**
	(3 1)	0,00	13,57	8,80	3,70	**0,00**	0,19	4,77	4,19	0,37	**0,19**	0,41	5,50	4,75	1,07	**0,28**
	(3 3)	4,19	0,00	8,15	0,54	**0,00**	17,25	12,23	14,10	13,60	**10,68**	3,63	0,47	4,56	1,73	**0,47**
K=3	(1 3 3)	3,53	3,09	3,86	0,00	**0,00**	0,95	0,84	1,62	0,25	**0,25**	0,13	0,73	1,09	0,00	**0,00**
	(3 3 1)	1,85	13,08	7,97	2,31	**1,85**	0,22	8,79	6,54	0,22	**0,22**	0,15	7,97	8,10	0,26	**0,00**
	(3 3 3)	16,44	0,37	6,76	3,13	**0,00**	20,27	10,73	17,53	11,28	**10,18**	7,06	4,72	10,34	3,47	**2,78**
	(3 1 3)	2,88	11,70	10,54	0,71	**0,71**	0,18	0,71	1,24	0,00	**0,00**	5,17	7,93	9,07	2,75	**2,75**
K=5	(1 3 3 2 3)	2,36	4,19	7,51	1,26	**1,26**	4,85	2,08	2,39	0,41	**0,20**	1,23	0,67	1,26	0,07	**0,00**
	(3 3 2 3 1)	3,88	16,31	12,44	3,08	**2,18**	8,39	15,74	17,19	7,83	**7,22**	12,75	19,29	19,03	12,92	**12,31**
	(3 3 3 3 3)	6,20	3,54	7,27	2,56	**0,64**	20,24	13,57	23,35	10,78	**10,35**	9,31	10,17	12,37	3,67	**3,67**
	(3 3 1 2 2)	3,52	6,27	3,89	1,15	**0,33**	1,07	4,73	6,31	0,30	**0,30**	0,39*	1,17*	2,33*	0*	**0***

Tab. 7.7 : Pourcentage d'erreur entre la solution optimale (ou la B_{inf}) et le makespan trouvé en utilisant différentes heuristiques (blocage mixte)

À partir de ces deux tableaux, nous remarquons, que le pourcentage d'erreur dans la colonne *Best* est considérablement réduit par rapport aux quatre heuristiques, ce qui s'explique par le fait que les meilleures solutions trouvées à partir de ces différentes heuristiques sont souvent différentes. Elles sont donc complémentaires entre elles. L'intérêt de les combiner apparait ici clairement, d'autant plus que leurs temps de calcul sont négligeables.

Nous constatons que la méthode MSPT est la meilleure pour les problèmes où il n'existe qu'une seule machine sur le deuxième étage. MLPT est souvent efficace pour les problèmes où il n'existe qu'une seule machine dans le premier étage. RAND est l'heuristique la moins efficace des 4 heuristiques, mais parfois elle peut donner (par chance) des solutions optimales pas nécessairement obtenues par les autres méthodes. Enfin, nous pouvons remarquer que l'heuristique NH est la plus performante, en particulier pour les problèmes uniformes où le nombre de machines est identique sur tous les étages.

Nous constatons globalement que les pourcentages d'erreur des deux tableaux sont comparables : l'erreur moyenne de tous les problèmes en utilisant l'heuristique *Best* pour le cas sans blocage est de 2,01% contre 2,09% pour le pourcentage moyen des problèmes avec blocage mixte.

Nous notons aussi que les temps de calcul en utilisant ces heuristiques sont négligeables. C'est la raison pour laquelle nous ne les avons pas indiqués dans les tableaux.

7. 4. Métaheuristique

Afin d'améliorer davantage les solutions obtenues par les méthodes heuristiques, nous proposons d'utiliser l'algorithme génétique dont nous avons présenté le fonctionnement au chapitre cinq.

7. 4. 1. Algorithme génétique

Dans l'algorithme génétique conçu pour résoudre des problèmes de type Flow-Shop non hybride, les chromosomes sont une suite de nombres qui correspond à l'ordre dans lequel les jobs vont passer sur les machines. Ainsi, la fonction d'évaluation, pour un Flow-Shop, consiste simplement à calculer le makespan correspondant à l'ordre de jobs passé en référence :

Makespan = fonction_Évaluation (ordre de job)

Afin de pouvoir utiliser l'algorithme génétique pour le Flow-Shop hybride, nous avons adapté notre fonction d'évaluation.

L'ordre des jobs qui est passé en référence correspond maintenant à l'ordre dans lequel nous posons les jobs sur les différents étages. En effet, le premier job pose toutes ses opérations en premier sur chacun des étages. Ensuite, chaque job vient poser ses opérations sur chacun des étages au plus tôt possible sur les machines disponibles de l'étage. Cette façon de faire nous a donné les résultats présentés dans les tableaux 7.8 et 7.9.

7. 4. 2. Résultats expérimentaux

Dans notre algorithme génétique, nous pouvons imposer le pourcentage gardant les meilleurs individus (*pc_best*), le pourcentage d'insertion d'une nouvelle population générée de façon aléatoire (*pc_new*) et le pourcentage de croisement (*pc_cross*).

Pour effectuer les travaux que nous présentons pour le cas du Flow-Shop hybride, nous avons défini pour nos expériences, les mêmes pourcentages utilisés pour le cas du Flow-Shop présenté dans le chapitre cinq : *pc_best* = 0,1 ; *pc_new* = 0,1 ; *pc_cross* = 0,6. La valeur de *ctbvi* que nous avons choisie pour arrêter nos simulations est *ctbvi* = 500. En effet, après des tests, ces paramètres semblent efficaces.

Pour mieux constater les améliorations apportées par l'utilisation de l'algorithme génétique, nous présentons dans les tableaux 7.8 et 7.9 une comparaison des résultats trouvés avec les meilleures solutions données par l'heuristique *Best*. Les temps de calcul engendrés par l'utilisation de l'algorithme génétique sont donnés pour chaque dimension de problèmes.

Le pourcentage d'erreur est toujours défini par la formule suivante :

$$\%err = \frac{C_{\max} - S_{opt}}{S_{opt}} \times 100 \qquad 7.19$$

Étages	Machines par étage : M_k	*j* 5			*j* 10			*j* 15		
		Best (%)	*AG* (%)	*AG* (s)	*Best* (%)	*AG* (%)	*AG* (s)	*Best* (%)	*AG* (%)	*AG* (s)
K=2	(1 3)	0,00	0,00	3	0,00	0,00	7	0,00	0,00	11
	(3 1)	0,00	0,00	3	0,00	0,00	7	0,00	0,00	11
	(3 3)	3,35	**0,88**	4	6,41	**2,92**	7	8,49	**1,58**	12
K=3	(1 3 3)	0,00	0,00	5	0,00	0,00	11	0,00	0,00	17
	(3 3 1)	0,95	0,95	5	0,00	0,00	11	0,00	0,00	17
	(3 3 3)	1,32	**0,00**	5	9,62	**5,87**	14	6,40	**3,75**	26
	(3 1 3)	0,00	0,00	5	0,00	0,00	11	0,77	**0,00**	17
K=5	(1 3 3 2 3)	0,45	**0,23**	9	1,81	**0,18**	18	0,36	**0,00**	58
	(3 3 2 3 1)	0,53	0,53	9	1,12	**0,00**	18	0,28	**0,14**	29
	(3 3 3 3 3)	1,39	**0,69**	9	9,61	**4,15**	25	13,29	**9,90**	33
	(3 3 1 2 2)	0,00	0,00	9	0,33	**0,00**	48	0*	0*	28*

Tab. 7.8 : Pourcentage d'erreur entre la solution optimale (ou la B_{inf}) et le makespan trouvé en utilisant l'algorithme génétique (sans blocage)

Étages	Machines par étage : M_k	*j* 5			*j* 10			*j* 15		
		Best (%)	*AG* (%)	*AG* (s)	*Best* (%)	*AG* (%)	*AG* (s)	*Best* (%)	*AG* (%)	*AG* (s)
K=2	(1 3)	0,00	0,00	2	0,00	0,00	5	0,00	0,00	9
	(3 1)	0,00	0,00	2	0,19	0,19	5	0,28	**0,00**	8
	(3 3)	0,00	0,00	2	10,68	**6,21**	5	0,47	**0,00**	8
K=3	(1 3 3)	0,00	0,00	3	0,25	**0,00**	7	0,00	0,00	11
	(3 3 1)	1,85	**0,46**	3	0,22	0,22	13	0,00	0,00	12
	(3 3 3)	0,00	0,00	4	10,18	**6,62**	9	2,78	**0,00**	12
	(3 1 3)	0,71	**0,35**	3	0,00	0,00	7	2,75	**0,00**	12
K=5	(1 3 3 2 3)	1,26	**0,00**	6	0,20	**0,00**	13	0,00	0,00	20
	(3 3 2 3 1)	2,18	**1,46**	6	7,22	**1,46**	18	12,31	**3,90**	28
	(3 3 3 3 3)	0,64	**0,00**	6	10,35	**6,77**	14	3,67	**1,28**	27
	(3 3 1 2 2)	0,33	**0,00**	6	0,30	**0,00**	13	0*	0*	20*

Tab. 7.9 : Pourcentage d'erreur entre la solution optimale (ou la B_{inf}) et le makespan trouvé en utilisant l'algorithme génétique (blocage mixte)

Nous constatons que l'algorithme génétique améliore considérablement les meilleurs résultats donnés par les heuristiques (pourcentages mis en gras dans les tableaux) et cela dans un temps de calcul raisonnable (moins d'une minute en moyenne pour tous les problèmes testés).

Pour la plupart des problèmes à 10 jobs et 15 jobs 5 étages dans le cas sans blocage, et tous les problèmes à 10 et 15 jobs dans le cas du blocage mixte, nous n'avons pas les solutions optimales. Nous avons donc comparé les solutions trouvées avec les bornes inférieures que nous avons développées dans ce chapitre. Ceci explique le fait d'obtenir les pourcentages les plus élevés pour ces problèmes.

Nous remarquons aussi que les temps de calcul pour le cas du blocage mixte sont plus courts que ceux où on ne considère aucune contrainte de blocage.

Bien que cet algorithme génétique donne de bons résultats, il faut noter que la façon de son utilisation ne permet de trouver des solutions que dans un sous espace de solutions. En effet, l'adaptation de cet algorithme génétique nous fait traiter le problème hybride comme un problème non hybride, dans la mesure où on traite chaque job de façon séquentielle. Cette manière d'utilisation de l'algorithme génétique nous interdit ainsi les solutions telles que l'ordre des jobs varie d'un étage à un autre.

7. 5. Conclusion

Dans ce chapitre, nous avons présenté différentes méthodes de résolution approchées pour résoudre les problèmes de type Flow-Shop hybride avec blocage mixte. Nous avons, tout d'abord, présenté deux bornes inférieures : l'une est classique basée sur le temps total d'exécution du plus long job et une nouvelle borne basée sur le temps maximal d'occupation des étages. Puis, nous avons proposé quatre heuristiques, trois d'entre elles sont des règles simples bien connues (SPT, LPT et RAND) en ajoutant quelques modifications sur les deux premières méthodes, ainsi qu'une adaptation de l'heuristique NH développée dans (Thornton et Hunsucker, 2004) et qui a prouvé son efficacité pour la résolution des problèmes de type Flow-Shop hybride avec la contrainte de blocage *RSb*. Ensuite, nous avons combiné ces quatre heuristiques pour obtenir la meilleure solution possible. Enfin, nous avons utilisé l'algorithme génétique proposé au chapitre cinq pour améliorer davantage nos solutions.

Les deux bornes inférieures proposées dans ce chapitre sont efficaces et surtout complémentaires : dans le cas des problèmes non-uniformes, la borne inférieure *LB*2 est souvent

très efficace, alors que dans le cas des problèmes uniformes, la borne inférieure *LB*1 est plus performante.

Les quatre heuristiques utilisées dans ce chapitre traitent différemment les problèmes d'ordonnancement d'où l'intérêt de les combiner, d'autant plus que les temps de calcul sont négligeables. Cette méthode a permis de réduire considérablement le pourcentage d'erreur et de donner de bons résultats. Mais l'heuristique NH reste la plus performante, en particulier, pour les problèmes uniformes où le nombre de machines est identique sur tous les étages.

En utilisant l'algorithme génétique, nous avons encore amélioré considérablement les solutions et cela dans un temps de calcul raisonnable (moins d'une minute en moyenne pour tous les problèmes testés).

Conclusions et perspectives

Depuis longtemps, plusieurs auteurs se sont intéressés aux problèmes d'ordonnancement de type Flow-Shop classique où les espaces de stockage entre les machines sont considérés comme infinis. D'autres problèmes sont caractérisés par une seule contrainte de blocage classique (*RSb*) où les capacités de stockage sont limitées ou nulles et d'autres encore par une contrainte de blocage particulière (*RCb* ou *RCb**). Dans le cadre de cette thèse, nous avons présenté un cas général qui peut être utilisé dans l'industrie et modélisé sous forme de systèmes de type Flow-Shop ou Flow-Shop hybride soumis simultanément à plusieurs types de blocage en considérant le makespan comme critère d'optimisation. A notre connaissance, très rares sont les travaux qui ont traité ce genre de problème.

Dans la première partie de la thèse, nous avons donné tout d'abord une présentation générale du problème. Pour cela, nous avons présenté les caractéristiques d'un système de type Flow-Shop et Flow-Shop hybride, ainsi qu'une description détaillée des différentes contraintes de blocage et quelques exemples d'applications ont été présentés. Ensuite, nous avons présenté un état de l'art des principaux travaux qui ont été développés pour les systèmes de type Flow-Shop et Flow-Shop hybride, avec et sans contraintes de blocage. Nous avons ainsi pu constater que différents cas classiques de ces systèmes ont été largement étudiés mais que les travaux sont moins nombreux pour les cas avec blocage mixte.

La deuxième partie a été consacrée à l'étude des systèmes de production de type Flow-Shop soumis simultanément à plusieurs types de blocage.

Nous avons d'abord présenté des résultats de complexité pour les problèmes soumis à la contrainte de blocage *RCb**, ainsi que pour des problèmes où différentes contraintes de blocage sont mélangées. Nous avons ainsi pu montrer que le problème d'ordonnancement de type Flow-Shop avec la contrainte de blocage *RCb** est polynomial pour $M \leq 3$ machines et devient NP-Difficile à partir de 4 machines. Pour les problèmes qui contiennent des contraintes de blocage différentes entre les machines, certains cas à 3 et 4 machines ont été prouvés polynomiaux. Ceci montre qu'en général, les problèmes qui considèrent le blocage mixte sont plus faciles à résoudre que ceux qui ne considèrent qu'une seule contrainte de blocage entre toutes les machines du même problème.

Ensuite, pour résoudre le problème, nous avons présenté une méthode de résolution exacte basée sur la modélisation mathématique en nombres entiers. Cette modélisation est une adaptation du modèle proposé par (Martinez, 2005) afin de considérer le blocage mixte. Ce modèle a été validé avec le logiciel d'optimisation XPress-MP. Les résultats obtenus sur des problèmes de différentes tailles montrent que des problèmes de taille inférieure à 12 jobs et 100 machines sont résolus dans un temps raisonnable. Nous avons également constaté que les temps moyens de résolution

pour les problèmes avec blocage mixte sont nettement inférieurs à ceux des problèmes avec une seule contrainte de blocage (*RCb* ou *Wb*).

Nous avons ensuite présenté différentes méthodes de résolution approchées pour résoudre ce problème. Nous avons, tout d'abord, présenté une borne inférieure basée sur le temps maximal d'occupation des machines. Cette borne inférieure est assez performante pour les instances de petite taille, mais devient moins efficace pour les problèmes où le nombre de machines est supérieur à 10.

Ensuite, nous avons proposé une nouvelle heuristique TSS, ainsi qu'une adaptation de l'heuristique NEH. Nous avons apporté successivement deux améliorations locales à ces heuristiques. La première est basée sur la méthode itérative NEH et la seconde est basée sur la permutation du job qui a le plus grand temps de blocage différentiel. Les résultats ont montré que ces heuristiques, quand elles sont combinées, améliorent remarquablement et rapidement les solutions. En effet, l'utilisation de toutes les heuristiques suivies de toutes les améliorations locales donne moins de 1,61% d'erreurs entre la solution optimale et le makespan obtenu pour un temps inférieur à 1 seconde pour tous les problèmes testés.

Enfin, nous avons proposé un algorithme génétique qui a encore amélioré les solutions (l'erreur maximale n'est plus que de 0,34%), tout en gardant un temps de calcul raisonnable : jusqu'à 11,15 secondes pour les tailles les plus grandes des problèmes testés.

Dans la troisième et dernière partie, nous avons étudié les systèmes de production de type Flow-Shop hybride avec blocage mixte.

Nous avons tout d'abord présenté une méthode de résolution exacte basée sur la modélisation mathématique en nombres entier pour des instances de petite taille. Cette modélisation est une adaptation du modèle proposé dans (Yuan *et al.*, 2009) afin de considérer le blocage mixte. Les résultats obtenus montrent que les temps moyens de résolution pour les problèmes avec blocage mixte sont nettement plus rapides que ceux des problèmes sans contrainte de blocage (essentiellement pour les problèmes 5 jobs / 5 étages et pour les problèmes avec 6 jobs). Néanmoins, le temps de calcul pour résoudre des problèmes de plus grandes tailles devient de plus en plus prohibitif. Pour cela, nous avons présenté différentes méthodes de résolution approchées.

Nous avons, tout d'abord, présenté deux bornes inférieures : l'une est classique, basée sur le temps total d'exécution du plus long job et une nouvelle borne est basée sur le temps maximal d'occupation des étages. Les résultats ont montré que ces deux bornes inférieures sont complémentaires : la borne inférieure *LB*2 est souvent très efficace dans le cas des problèmes

non-uniformes, alors que la borne inférieure *LB*1 est plus performante dans le cas des problèmes uniformes.

Ensuite, nous avons proposé quatre heuristiques, dont trois d'entre elles sont des heuristiques simples bien connues (SPT, LPT et RAND). Nous avons ajouté quelques modifications sur les deux premières méthodes, ainsi qu'une adaptation de l'heuristique NH développée dans (Thornton et Hunsucker, 2004) qui a prouvé son efficacité pour la résolution des problèmes de type Flow-Shop hybride avec la contrainte de blocage *RSb*. Nous avons ensuite combiné ces quatre heuristiques pour obtenir la meilleure solution possible. Cette méthode a permis de réduire considérablement le pourcentage d'erreur et de donner de bon résultats. L'heuristique NH reste la plus performante, en particulier pour les problèmes uniformes où le nombre de machines est identique sur tous les étages.

Enfin, l'utilisation d'un algorithme génétique nous a encore permis d'améliorer considérablement les solutions et cela dans un temps de calcul raisonnable (moins d'une minute en moyenne pour tous les problèmes testés).

En ce qui concerne les perspectives de recherche, nous proposons quelques axes d'amélioration.

Pour les problèmes de type Flow-Shop hybride avec blocage mixte, la modélisation mathématique en nombres entiers que nous avons proposée n'étant pas suffisamment satisfaisante, il serait intéressant de développer une autre méthode de résolution exacte plus performante. Un Branch and Bound serait une solution intéressante pour essayer de résoudre de façon optimale des problèmes de tailles plus importantes.

Concernant la métaheuristique que nous avons utilisé pour résoudre les problèmes de type Flow-Shop hybride avec blocage mixte, il serait intéressant d'apporter des améliorations sur sa façon de résoudre le problème et l'adapter au cas hybride, dans la mesure où on peut accepter des solutions telles que l'ordre des jobs varie d'un étage à un autre.

Toujours dans le cas des Flow-Shop hybrides, nous pouvons nous intéresser aussi aux problèmes avec des machines parallèles non identiques ou considérer d'autres paramètres comme la disponibilité des machines.

Enfin, nous pouvons aussi proposer un autre axe de recherche qui traite les problèmes où l'on considère différentes contraintes de blocage simultanément pour les systèmes de productions de type Job-Shop.

Références

Abadi, K., N.G. Hall et C. Sriskandarajah. Minimizing cycle time in a blocking flowshop. Operations Research, vol. 48, pp. 177-180, 2000.

Allaoui, H. et A. Artiba. Scheduling two stage hybrid Flow-Shop with availability constraints. Computers and Operations Research, vol. 33, pp. 1399-1419, 2006.

Arthanary, T.S. et K.G. Ramaswamy. An extension of two machine sequencing problem. Operations Research, vol. 8, pp. 10-22, 1971.

Averbakha, I., O. Berman et I. Chernykh. The m-machine flowshop problem with unit-time operations and intree precedence constraints. Operations Research Letters, vol. 33, pp. 263-266, 2005.

Bagchi, T. P., J. N.D. Gupta et C. Sriskandarajah. A review of TSP based approaches for flowshop scheduling. European Journal of Operational Research, vol. 169, pp. 816-854, 2006.

Baptiste, P. et V. G. Timkovsky. Shortest path to nonpreemptive schedules of unit-time jobs on two identical parallel machines with minimum total completion time. Mathematical Methods of Operations Research, vol. 60, pp.145-153, 2004.

Bellman, R.. Dynamic Programming. Princeton University Press, 1957.

Brah, S. A. et J. L. Hunsucker. Branch and bound algorithm for the flowshop with multiple processors. European Journal of Operational Research, vol. 51, pp. 88-99, 1991.

Buten, R.E. et V.Y. Shen. A scheduling model for computer systems with two classes of processors. Proceedings of the Sagamore Computer Conference on Parallel Processing, pp. 130-138, 1973.

Campbell, H.G., R.A. Dudek et M.L. Smith. A heuristic algorithm for the n job m machine sequencing problem. Management Science, vol.16, pp. B630-637, 1970.

Caraffa, V., S. Ianes, T.P. Bagchi et C. Sriskandarajah. Minimizing makespan in a blocking flow-shop using genetic algorithms. International Journal Production Economics, vol. 70, pp. 101-115, 2001.

Carlier, J. et P. Chrétienne. Les problèmes d'ordonnancement: modélisation, complexité, algorithmes. Masson, Paris, France, 1988.

Carlier, J. et I. Rebaï. Two branch and bound algorithms for the permutation flow. European Journal of Operational Research, vol. 90, pp. 238-251, 1996.

Carlier, J. et E. Néron. An exact method for solving the multiprocessor flowshop. RAIRO - Operations Reseach, vol. 34(1), pp. 1-25, 2000.

Carpaneto, G. et P. Toth. Some new branching and bounding criteria for the asymmetric traveling salesman problem. Management Science, vol. 26, pp. 736-743, 1980.

Carpaneto, G., M. Dell'Amico et P. Toth. Exact solution of large-scale the asymmetric traveling salesman problems. ACM Transactions on Mathematical Software, vol. 21, pp. 394-409, 1995.

Cho, Y. et S. Sahni. Preemptive Scheduling of Independent Jobs with Release and Due Times on Open, Flow and Job Shops. Operations Research, vol. 29, pp. 511-522, 1981.

Cirasella, J., D.S. Johnson, L.A. McGeoch et W. Zhang. The asymmetric travelling salesman problem: Algorithms, instance generator, and tests. In: Buchsbaum, A.L., Snoeyink, J. (Eds.), Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Lecture Notes in Computer Science, 2153. Springer-Verlag, Berlin, pp. 32-59, 2001.

Dantzig, G., R. Fulkerson et S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. Journal of the Operations Research Society of America, vol. 2(4), pp. 393-410, 1954.

Dauzère-Pérès, S., C. Pavageau et N. Sauer. Modélisation et résolution par PLNE d'un problème réel d'ordonnancement avec contraintes de blocage. 3ème congrès ROADEF, Nantes, pp. 216-217, 2000.

Deal, D.E. et D.R. Parks. A heuristic algorithm for job shop scheduling. Energy Sources Technology Conference and Exhibition, Houston, Texas. 1999.

Dréo, J., A. Petrowski, P. Siarry et E. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, 2003.

Dutta, S.K. et A.A. Cunningham. Sequencing two machine Flow-Shops with finite intermediate storage. Management Science, vol. 21, pp. 989-996, 1975.

Garey, M.R., D.S. Johnson et R. Sethi. The complexity of job-shop and flow-shop scheduling. Mathematics of Operations Research, vol. 1, 1976.

Gicquel, C., L. Hege, M. Minoux et W. Van Canneyt. A discrete time exact solution approach for a complex hybrid Flow-Shop scheduling problem with limited-wait constraints. Computers and Operations Research, vol. 39, pp. 629-636, 2012.

Gilmore, P.C. et R.E. Gomory. Sequencing a one state-variable machine : A solvable case of the traveling salesman problem. Operations Research, vol.12, pp. 655-679, 1964.

Gonzalez, T. et Sahni, S. Flow-shop and job-shop scheduling: Complexity and approximation. Operations Research, vol. 26, pp. 36-52, 1978.

Gorine, A. et N. Sauer. Méthode exacte et bornes inférieures pour le Job Shop avec contrainte de blocage particulière. Conference Internationale Francophone d'Automatique, CIFA, Bucarest, 2008.

Gourgand, M., N. Grangeon et S. Norre. Metaheuristics for the deterministic hybrid flow shop problem. Proceeding of the International Conference on Industrial Engineering and Production Management, Glasgow, United Kingdom, pp. 136-145, 1999.

Grabowski, J., E. Skubalska, et C. Smutnicki. On flow shop scheduling with release and due dates to minimize maximum lateness. Journal of Operational Research Society, vol. 34, pp. 615-620, 1983.

Grabowski, J. et J. Pempera. Sequencing of jobs in some production systems. European Journal of Operational Research, vol. 125, pp. 535-550, 2000.

Graham, R.L., E.L. Lawler, J.K. Lenstra et A.H.G. Rinnooy Kan. Optimisation and approximation in deterministic machine scheduling : Survey. Annals of Discrete Mathematics, vol. 5, pp. 287-326, 1979.

Gupta, J.N.D. A general algorithm for the n × m flowshop scheduling problem. The International Journal of Production Research 7, 241–247, 1969.

Gupta, J.N.D. M-stage flowshop by branch and bound. Operations Research, vol. 9, pp. 37-43, 1970.

Gupta, J.N.D. Two-stage hybrid flowshop scheduling problem. Journal of Operational Research Society, vol. 39(4), pp. 359-364, 1988.

Gupta, J.N.D. et E.A. Tunc. Schedules for a two-stage hybrid flow shop with parallel machines at the second stage. International Journal of Production Research, vol. 29, pp. 1489-1502, 1991.

Hall, N.G. et C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research, vol. 44, pp. 510-525, 1996.

Hall, N.G., H. Kamoun et C. Sriskandarajah. Scheduling in robotic cells : Complexity and steady state analysis. European Journal of Operational Research, vol. 109, pp. 43-65, 1998.

Haouari, M. et R. M'Hallah. Heuristic algorithms for the two-stage hybrid flowshop problem. operations research letters, vol. 21, pp. 43-53, 1997.

Haouari, M.et T. Ladhari. A branch-and-bound-based local search method for the Flow-Shop problem. Journal of the Operational Research Society, vol. 54, pp. 1076-1084, 2003.

Haouari, M., L. Hidri, et A. Gharbi. Optimal Scheduling of a two stage hybrid Flow-Shop. Mathematical Methods of Operations Research, vol. 64, pp.107-124, 2006.

Holger, H.H. et T. Stützle. Stochastic Local Search. Foundations and Applications, Morgan Kaufmann, San Francisco, CA, USA, 2005.

Holland, J. H. Outline for logical theory of adaptive systems. Journal of the association of computing machinery, vol. 3, pp. 297-314, 1962.

Hoogeveen, J.A., J.K. Lenstra, and B. Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. European Journal of Operational Research, vol. 89, pp. 172-175, 1996.

Hunsucker, J.L. et S.A. Brah. Optimal scheduling in a flow shop with mutliple processors. Paper presented at the TIMS/ORSA Joint National Meeting in New Orleans, pp. 4-6, 1987.

Hunsucker, J.L. et J.R.Shah. Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. European Journal of Operational Research, vol. 72, pp. 102-114, 1994.

Ishibuchi, H., S. Misaki et H. Tanaka. Modified simulated annealing algorithms for the Flow-Shop sequencing problem. European Journal of Operational Research, vol. 81, pp. 388-399, 1995.

Iyer, S. K. et B. Saxena. Improved genetic algorithm for the permutation flow-shop scheduling problem. Computers and Operations Research, vol. 31, pp. 593–606, 2004.

Jin, Z., Z. Yang et T. Ito. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. International Journal of Production Economics, vol. 100(2), pp. 322-334, 2006.

Johnson, S.M. Optimal Two and Three Stage Production Schedules with Setup Times Included. Naval Research Logistics Quarterly, vol. 1, pp. 61-68, 1954.

Johnson, D.S., G. Gutin, L.A. McGeoch, A. Yeo, W. Zhang et A. Zverovich. Experimental analysis of heuristics for the ATSP. In: Gutin, G., Punnen, A. (Eds.), The Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers, Dordrecht, pp. 445-487, 2002.

Kahraman, C., O. Engin, I. Kaya et M.K. Yilmaz. An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. International Journal of Computational Intelligence Systems, vol. 1(2), pp. 134-147, 2008.

Karp, R.M. Reducibility among combinatorial problems in complexiy of computer computations. R.E. Miller and J.W. Tatcher, Eds, Plenum Press, New York, pp. 85-103, 1972.

Langston, M.A. Interstage transportation planninf in deterministic flow shop environment. Operation Research, vol. 35, pp. 556-564, 1987.

Lawler, E.L. Optimal Sequencing of a Single Machine Subject to Precedence Constraints. Management Science, vol. 19(5), pp. 544-546, 1973.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan et D.B. Shmoys. The Traveling Salesman Problem. Wiley, New York, 1985.

Lee, C.Y. et R.G. Vairaktarakis. Minimizing makespan in hybrid flowshops. Operation Research Letters, pp. 149-158, 1994.

Leisten, R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research, vol. 28, pp. 2085-2100, 1990.

Lenstra, J.K, A.H.G. Rinnooy Kan et P. Brucker. Complexity of machine scheduling problems. Annals of Discrete Mathematics, vol. 1, pp. 343-362, 1977.

Levner, E.M. Optimal planning of parts machining on a number of machines. Automat Remote Control, vol. 12, pp. 1972-1978, 1969.

Liao CL et C.T. You. An improved formulation for the job-shop scheduling problem. Journal of the Operational Research Society, vol. 43, pp. 1047-1054, 1992.

Linn, R. et W. Zhang. Hybrid Flow-Shop scheduling : A survey. Computers & Industrial Engineering, vol. 37, pp. 57-61, 1999.

Liu, Y. et I.A. Karimi. Scheduling multistage batch plants with parallel units and no interstage storage. Computers and Chemical Engineering, vol. 32, pp. 671-693, 2008.

Lomnicki, Z.A. A branch and bound algorithm for the exact solution of the three machine scheduling problem. Operational Research Quarterly, vol. 16, pp. 89-100, 1965.

Manne, A. On the job-shop scheduling problem. Operations Research, vol. 8, pp. 219-223, 1960.

Martinez, S. Ordonnancement de systèmes de production avec contraintes de blocage. Thèse, Université de Nantes, 2005.

Martinez, S., S. Dauzière-Pérès, C. Guèret, Y. Mati et N. Sauer. Complexity of Flow-Shop scheduling problems with a new blocking constraint. European Journal of Operational Research, vol. 169(3), pp. 855-864, 2006.

Mati, Y. Les problèmes d'ordonnancement dans les systèmes de production automatisés : Modèles, complexité et approches de résolution. Thèse, Université de Metz, 2002.

Nawaz, M., E. Enscore et I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science, vol. 11(1), pp. 91-95, 1983.

Neron, E., Ph. Baptiste et J.N.D. Gupta. Solving hybrid flow shop problem using energetic reasoning and global operations. OMEGA, The International Journal of Management Science, vol. 29, pp. 501-511, 2001.

Nowicki, E. et C. Smutnicki. A fast tabu search algorithm for the permutation Flow-Shop problem. European Journal of Operational Research, vol. 91, pp. 160-175, 1996.

Nowicki, E. et C. Smutnicki. The flow shop with parallel machines: A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research, vol. 106, pp. 226-253, 1998.

Osman, I.H., et Potts, C.N. Simulated annealing for permutation flow-shop scheduling, OMEGA, vol. 17(6), pp. 551-557, 1989.

Pan, C.H. A study of integer programming formulations for scheduling problems. International Journal of Systems Science, vol. 28, pp. 33-41, 1997.

Papadimitriou, C. et P. Kanellakis. Flow-shop scheduling with limited temporary storage. Journal of the Association for Computing Machinery, vol. 27, pp. 533-549, 1980.

Perregaard M. Branch-and-Bound methods for the multi-Processor Job Shop and Flow Shop scheduling problems. Master's Thesis, Datalogisk institute Kubenhavns Universitet, 1995.

Portmann, M. C., A. Vignier, D. Dardilhac et D. Dezalay. European Journal of Operational Research, vol. 107, pp. 389-400, 1998.

Potts, C.N. An adaptive branching rule for the permutation flow-shop problem. European Journal of Operational Research, vol. 5, pp. 19-25, 1980.

Reddi, S.S. et C.V. Ramamoorthy. On the flow-shop sequencing problem with no-wait in process. Operational Research Quarterly, vol. 23, pp. 323-330, 1972.

Reddi, S.S. On the Flow-Shop sequencing problem with no-wait in process. Management Science, vol. 23, pp. 216-217, 1976.

Riane, F., A. Artiba, et S.E. Elmaghraby. A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. European Journal of Operational Research, vol. 109, pp. 321-329, 1998.

Ribas, I., R. Companys et X. Tort-Martorell. An iterated greedy algorithm for the Flow-Shop scheduling problem with blocking. OMEGA, vol. 39, pp. 293-301, 2011.

Röck, H. et G. Schmidt. Machine aggregation heuristics in shop scheduling. Methods of Operational Research, vol. 45, pp. 303-314, 1983.

Röck, H. The three machine no-wait flowshop problem is NP-complete. Journal of the Association of Computer Machinery, vol. 31, pp. 336-345, 1984.

Ronconi, D.P. A Branch-and-Bound Algorithm to Minimize the Makespan in a Flow-Shop with Blocking. Annals of Operations Research, pp. 53-65, 2005.

Roy, B. et B. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical report, SEMA, Paris, France. 1964.

Ruiz, R. et C. Marotto. A comprehensive review and evaluation of permutation Flow-Shop heuristics. European Journal of Operational Research, vol. 165, pp. 479-494, 2005.

Ruiz, R., C. Maroto et J. Alcaraz. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research, vol. 165(1), pp. 34-54, 2005.

Ruiz, R. et T. Stützle. A simple and effective iterated greedy algorithm for the permutation Flow-Shop scheduling problem. European Journal of Operational Research, vol. 177, pp. 2033-2049, 2007.

Ruiz, R. et J. A. Vázquez-Rodríguez. The hybrid Flow-Shop scheduling problem. European Journal of Operational Research, vol. 205, pp.1-18, 2010.

Santos, D.L., J.L. Hunsucker et D.E. Deal. Global lower bounds for flow shops with multiple processors. European Journal of Operational Research, vol. 80, pp. 112-120, 1995.

Sauvey, C. et N. Sauer. Initial populations tests for genetic algorithm Flow-Shop scheduling problems solving with a special blocking Information Control Problems in Manufacturing. 13th IFAC Symposium on Information Control problems in manufacturing, vol. 13, pp. 1961-1966, 2009.

Sauvey, C. et N. Sauer. A genetic algorithm with genes-association recognition for flowshop scheduling problems. Journal of Intelligent Manufacturing, vol. 23, pp. 1167- 1177, 2012.

Sawik, T.J. A schedule algorithm for flexible flow lines with limited intermediate buffers. Applied Stochastic Models and Data Analysis, vol. 9, pp. 127-138, 1993.

Sawik, T.J. Scheduling flexible flow lines with no in-process buffers. Int. J. Prod. Res, vol. 33, pp. 1357-1367, 1995.

Sawik, T.J. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. Mathematical and Computer Modelling, vol. 31, pp. 39-52, 2000.

Shen, V.Y. et Y.E. Chen. A scheduling strategy for the flow shop problem in a system with two classes of processors. Proceedings of the Conference on Information and Systems Science, pp. 645-649, 1972.

Siarry, P. et Z. Michalewicz. Advances in Metaheuristics for Hard Optimization. Springer-Natural Computing Series, 2007.

Taillard, E. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research, vol. 47, pp. 65-74, 1990.

Thornton, H. W. et J. L. Hunsucker. A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. European Journal of Operational Research, vol. 152, pp. 96-114, 2004.

Trabelsi, W., C. Sauvey et N. Sauer. Heuristic methods for problems with blocking constraints solving Job-Shop scheduling. International Conference on Modelling and Simulation, Hammamet, Tunisia, 2010.

Trabelsi, W., C. Sauvey et N. Sauer. Complexity and Mathematical Model for Flowshop Problem Subject to Different Types of Blocking Constraint. IFAC, Milan, Italy. 2011a.

Trabelsi, W., C. Sauvey et N. Sauer. Mathematical Model and Lower Bound for Flowshop Problem With Mixed Blocking Constraints. IESM, Metz, France. 2011b.

Trabelsi, W., C. Sauvey et N. Sauer. Heuristics and metaheuristics for solving mixed blocking constraints flowshop scheduling problems. Computers & Operations Research, vol. 39 (11), pp. 2520-2527, 2012a.

Trabelsi, W., C. Sauvey et N. Sauer. Mathematical Model and Lower Bound for Hybrid Flowshop Problem With Mixed Blocking Constraints. INCOM, Bucharest, Romania. 2012b.

Tseng, F. T., E. F. Stafford Jr. et J.N.D. Gupta. An empirical analysis of integer programming formulations for the permutation flowshop. OMEGA, vol. 32, pp. 285-293, 2004.

Vignier, A., J. Billaut et C. Proust. Les problèmes d'ordonnancement de type Flow-Shop Hybride : Etat de l'art. RAIRO - Recherche Opérationnelle, vol. 33, pp. 117-183, 1999.

Wagner, H.M. An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly, vol. 6, pp. 131-140, 1959.

Wang, L., L. Zhang et D.Z. Zheng. An effective hybrid genetic algorithm for flow-shop scheduling with limited buffers. Computers & Operations Research, vol. 33, pp. 2960-2971, 2006.

Wardono, B. et Y. Fathi. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. European Journal of Operational Research, vol. 155, pp. 380-401, 2004.

Wilson, JM. Alternative formulations of a Flow-shop scheduling problem. Journal of the Operational Research Society, vol. 40, pp. 395-399, 1989.

Yuan, K. et N. Sauer. Application of EM algorithm to flow-shop scheduling problems with a special blocking. ISEM. 2007.

Yuan, K., N. Sauer et C.Sauvey. Application of EM algorithm to hybrid flow shop scheduling problems with a special blocking. Emerging Technologies and Factory Automation. 2009.

Zobolas, G. I., C. D. Tarantilis et G. Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Computers and Operations Research, vol. 36, pp. 1249-1267, 2009.