Ordonnancement des systèmes flexibles avec contrainte de blocage

Ali Gorine

UNIVERSITÉ PAUL-VERLAINE DE METZ

IAEM Lorraine

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ Paul Verlaine-Metz

Mention : Automatique, Traitement du Signal, Génie Informatique

Ali GORINE

ORDONNANCEMENT DE SYSTEMES FLEXIBLES AVEC CONTRAINTE DE BLOCAGE.

Soutenue le 13-09-2011 devant la commission d'examen :

M. YALAOUI Farouk	Président
Mme GUERET Christelle	Rapporteur
M. SARI Zaki	Rapporteur
M. AGGOUNE Riad	Examinateur
M. SAUVEY Christophe	Examinateur
Mme SAUER Nathalie	Directrice de thèse

Laboratoire de Génie Industriel et Production de Metz.

À mes parents,
À mes frères et sœurs.

REMERCIEMENTS

Le présent travail a été réalisé au laboratoire LGIPM (Laboratoire de Génie Industriel et Production de Metz) à l'Université Paul Verlaine – Metz.

Pour commencer, je tiens à remercier tous les membres du jury pour l'intérêt qu'ils ont porté à mes travaux.

Merci à Nathalie SAUER, directrice de ma thèse, pour m'avoir accueilli au laboratoire LGIPM durant les années de ma thèse, pour toute l'aide et la disponibilité que vous m'avez offert, pour votre générosité, pour votre écoute patiente et vos conseils précieux. Veuillez trouver l'expression de ma sincère reconnaissance.

Merci à Christelle GUERET et Zaki SARI, rapporteurs de ma thèse, pour l'intérêt qu'ils ont porté à mes travaux. Veuillez trouver l'expression de ma sincère gratitude.

Merci à Riad AGGOUNE et Christophe SAUVEY, examinateurs de ma thèse, pour m'avoir fait l'honneur de participer au jury de ma soutenance. Veuillez trouver l'expression de ma sincère gratitude.

Merci à Farouk YALAOUI pour m'avoir fait l'honneur d'accepter la présidence de ce jury.

Merci à mes proches : mes parents, mes frères et sœurs, mon cousin Kamel.

Table des matières

7

CHAPITRE I

INTRODUCTION

I.1. Présentation générale

Le problème d'ordonnancement est l'un des problèmes les plus étudiés dans le domaine de la recherche opérationnelle. Ce problème consiste à trouver une séquence optimale pour l'exécution de *n* jobs sur *m* machines afin d'optimiser une fonction objectif et de déterminer également les dates de début et de fin d'exécution de chaque tâche (job). L'intérêt porté à cette thématique est largement motivé par l'apparition de ces problèmes dans des domaines aussi variés que nombreux. On peut citer les problèmes en temps réel tels que le partage de ressources dans les systèmes informatiques ou l'organisation des différentes tâches d'un robot autonome. Parmi les problèmes en temps décalé, on trouve la gestion de projet, la conception des emplois du temps, les chaines logistiques, les problèmes d'ordonnancement des systèmes de production, … etc.

Parmi les problèmes d'ordonnancement, on trouve également les problèmes dits d'ateliers, ces derniers appartiennent aux problèmes d'ordonnancement de la production. Dans ces types

de problèmes, chaque job est caractérisé par une séquence qui définit l'ordre de passage des opérations sur les machines. Ces séquences sont appelées dans la littérature gamme opératoire. On trouve principalement trois types d'atelier :

- Les ateliers de type flow-shop. Dans ce type d'atelier, la gamme opératoire est la même pour tous les jobs.
- Les ateliers de type job-shop. Dans ce type d'atelier, les gammes opératoires des jobs sont fixées et peuvent être différentes d'un job à un autre.
- Les ateliers de type open-shop. Dans ce type d'atelier, les gammes opératoires des jobs ne sont pas fixées.

Dans cette thèse, nous nous sommes intéressées aux problèmes de type job-shop.

Afin d'augmenter la capacité de production, il est possible de mettre en parallèle des machines qui peuvent exécuter les mêmes opérations. Dans ce cas, on parle de systèmes hybrides. Nous nous sommes également intéressés aux problèmes de job-shop hybride avec des machines parallèles identiques.

L'intérêt majeur des entreprises est d'augmenter la productivité tout en réduisant les coûts au maximum. Pour cela, l'une des solutions les plus utilisées est de limiter voir annuler l'espace de stockage entre les machines.

L'absence d'espace de stockage entre les machines implique qu'une machine qui finit l'exécution d'un job ne peut pas libérer la machine immédiatement si la machine sur laquelle nous exécutons l'opération suivante dans la gamme opératoire n'est pas disponible. Ces situations sont appelées dans la littérature situations de blocage.

Parmi les situations de blocage existantes, on trouve la contrainte de blocage classique dite *RSb (Release when Starting)* dans laquelle une machine reste bloquée par un job jusqu'à ce que le job qu'elle vient de terminer démarre sur la machine suivante. La deuxième situation de blocage particulière, que nous avons étudie dans cette thèse dite *RCb (Release when Completing*), une machine reste bloquée par un job jusqu'à ce que l'opération du job sur la machine suivante soit terminée et qu'elle quitte la machine. Les premiers travaux dans le cas des problèmes d'ordonnancement avec la contrainte de blocage de type *RCb* ont été ceux de Dauzère-Pérès et *al*. (Dauzère-Pérès et *al*. 2000a) où les auteurs proposent une modélisation PLNE pour la résolution d'un problème industriel d'ordonnancement.

I.2. Objectif de la thèse

Cette thèse concerne de manière générale l'évaluation des performances et l'ordonnancement dans des systèmes (au sens large : production ou service) flexibles, principalement les problèmes de type job-shop, soumis à des contraintes de blocage particulières. Il s'agit d'un problème d'ordonnancement composé de tâches regroupées en jobs et utilisant un ensemble de ressources (machines, opérateurs, …). Les machines sont disponibles en un ou plusieurs exemplaires. Chaque job doit passer sur un ensemble de types de machines (sur l'ensemble des machines de ce type ou sur un sous-ensemble). L'ordre de passage sur les ressources dans la gamme opératoire peut différer d'un produit à l'autre (tous les produits ne passent pas nécessairement sur toutes les ressources). Nous avons considéré le problème de minimisation de la date de fin de toutes les opérations, aussi appelée *Makespan*.

I.3. Plan de thèse

Le manuscrit de cette thèse contient six chapitres organisés de la manière suivante : dans le premier chapitre, nous donnons une description générale des problèmes d'ordonnancement ainsi que les différentes représentations pour ces problèmes.

Dans le deuxième chapitre, nous présentons un état de l'art porté sur les différents problèmes d'ordonnancement, notamment les problèmes à cheminement unique (Flow-shop) et les problèmes à cheminement multiple (Job-shop) avec les contraintes de blocage de type *RSb* et *RCb*.

Dans le troisième chapitre, nous avons modélisé le problème d'ordonnancement de type job shop avec la contrainte de blocage particulière afin d'obtenir une solution exacte. Nous avons proposé un modèle PLNE (Programmation Linéaire en Nombres Entiers). Ce modèle est basé sur la formulation proposée dans le cas d'un job-shop sans blocage. Pour les problèmes de taille plus importante, il n'était pas possible d'obtenir une solution exacte à ce problème en un temps raisonnable. Il était 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. Nous avons donc proposé deux bornes inférieures. La première est basée sur le temps maximal d'occupation des machines. La deuxième borne inférieure est basée sur la position de chaque opération dans la gamme opératoire des jobs et le temps nécessaire pour le débloquer la machine sur laquelle l'opération est exécutée.

Dans le quatrième chapitre, nous avons développé une méta-heuristique basée sur l'algorithme de recuit simulé pour résoudre le problème étudié. Pour développer un voisinage efficace, nous avons proposé une méthode qui permet de détecter les conflits qui peuvent survenir après la modification des séquences d'entrée des machines et ainsi obtenir plus facilement les solutions admissibles du voisinage.

Le cinquième chapitre est consacré à l'étude de la nouvelle contrainte de blocage dans les systèmes de production de type Job-Shop hybride. Nous avons développé une méthode de résolution exacte pour ce problème qui nous a permis de trouver des résultats optimaux pour des instances de petite taille. Ensuite, nous avons proposé trois bornes inférieures pour ce problème. La première borne est la borne classique qui correspond au maximum du temps total des jobs, la deuxième est basée sur le temps maximal d'occupation des machines, enfin la troisième borne inférieure est basée sur le temps moyen d'occupation de chaque étage.

Finalement le dernier chapitre est consacré à la présentation d'une conclusion résumant les travaux développés, les résultats obtenus et montrant les perspectives de recherche ouvertes par ces travaux.

CHAPITRE II

DESCRIPTION DU PROBLEME

II.1. Introduction

La résolution des problèmes d'ordonnancement est une branche de la recherche opérationnelle qui consiste à trouver une séquence optimale pour l'exécution de *n* tâches sur *m* ressources afin de minimiser une fonction objectif. Il s'agit également de calculer les dates de début et de fin d'exécution des tâches.

Dans ce chapitre, nous présentons plus en détails les systèmes de production de type job shop et job shop hybride qui font l'objet de cette étude. Ensuite, nous décrivons les contraintes de blocage : la contrainte de blocage classique *RSb* et la contrainte de blocage spécifique *RCb* qui est appliquée aux problèmes étudiés dans cette thèse. La quatrième partie de ce chapitre est consacrée à la représentation des ordonnancements par le diagramme de Gantt et le graphe disjonctif.

II.2. Système de production de type Job-Shop

II.2.1. Job-Shop classique

Le problème de type job-shop est l'un des problèmes les plus étudiés dans la littérature de l'ordonnancement. Son importance théorique ainsi que la modélisation de nombreuses applications industrielles sous forme de systèmes de type job-shop le rendent très intéressant.

La particularité des problèmes de type job shop par rapport aux problèmes de production de type flow shop est la gamme opératoire qui n'est pas fixe (atelier à cheminement libre), ainsi que le nombre d'opérations qui n'est pas forcément le même pour tous les jobs. Ces systèmes (problèmes de type job-shop) sont considérés comme des systèmes fortement combinatoires. Ils sont composés d'un ensemble de n jobs, chacun composé d'un ensemble de n_i opérations qui peuvent être exécutées sur m machines en respectant les contraintes suivantes :

- L'ordre de passage des opérations d'un job sur les différentes machines est fixe pour chaque job et peut être différent d'un job à un autre.
- Une machine ne peut exécuter qu'une seule opération à la fois et à l'instant initial toutes les machines sont disponibles.
- Une opération ne peut être exécutée que sur une seule machine et sans interruption.
- La capacité de stockage entre les machines est considérée comme infinie.
- La préemption des opérations n'est pas autorisée.

La figure suivante représente un problème type de job shop classique composé de 3 jobs et 5 machines dont les gammes opératoires sont :

- J1 : M1, M2, M3, M4, M5.
- J2 : M1, M5, M4.
- J3 : M2, M3, M4.

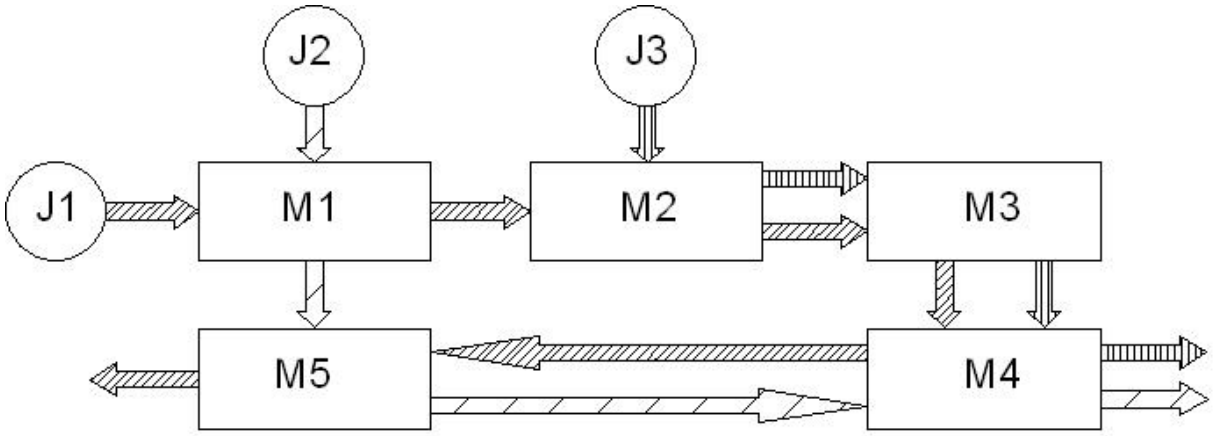

Figure 1 Problème de job shop classique composé de 3 jobs et 5 machines

L'objectif du problème d'ordonnancement consiste à trouver le meilleur enchaînement des opérations afin de minimiser ou maximiser un certain critère, ainsi que les instants de début des opérations sur chaque machine. Parmi les critères à optimiser, la minimisation de la date de fin de toutes les opérations sur toutes les machines est considérée dans la majorité des études de la littérature. Ce critère est appelé dans la littérature C_{max} ou *makespan*.

II.2.2. Job-Shop Hybride

Une façon d'augmenter la productivité d'un atelier est d'augmenter sa flexibilité. Pour cela, nous pouvons multiplier le nombre de machines qui réalisent la même tache. Le modèle résultant est connu dans la littérature comme un job-shop hybride. Dans ce modèle, les machines qui effectuent la même opération sont groupées dans un même étage.

Les problèmes de type job-shop hybride (*JSH*) sont une extension de deux problèmes d'ordonnancement : le problème de job-shop et le problème d'ordonnancement des machines parallèles. Le *JHS* peut être vu comme un problème de job-Shop qui possède une ou plusieurs machines parallèles par étage. La machine nécessaire pour exécuter une opération n'est pas connue a priori. Par conséquent, le problème se compose de deux parties dont la première est de trouver la séquence des jobs sur les étages, la deuxième partie est de trouver la machine nécessaire à l'exécution d'un job tout on respectant les contraintes du job-shop.

Un exemple de ce problème à *m* étages et trois machines maximum par étage, est donné dans la Figure 2.

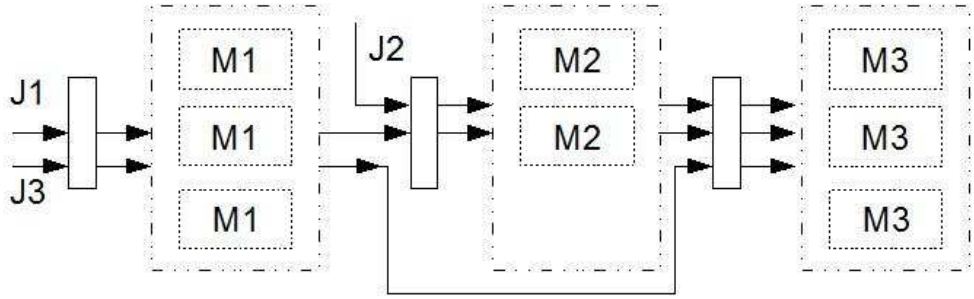

Figure 2 Représentation d'un système de type Job-shop hybride à trois étages

Une remarque importante dans les problèmes de type job-shop hybride est que le nombre de machines peut varier d'un étage à l'autre, ainsi que les performances des machines qui ne sont pas forcément les mêmes pour toutes les machines d'un même étage. Ces performances nous permettent de classer les systèmes avec machines parallèles en trois groupes, à savoir :

- Machines identiques : La durée d'exécution d'une tâche est la même sur toutes les machines.
- Machines uniformes : La durée d'exécution d'une tâche est similaire sur toutes les machines.
- Machines indépendantes : La durée d'exécution d'une opération dépend de la machine sur laquelle elle est exécutée.
- Dans ce travail, nous considérons le cas où les machines d'un étage sont identiques.

II.2.3. Notations

Dans cette partie, nous présentons quelques-unes des notations que nous adoptons dans la suite de ce manuscrit (Tableau 1.1).

n	: Nombre de jobs.
m	: Nombre de machines.
$J = \{J_1, J_2, \ldots, J_n\}$	: Ensemble des travaux ou jobs à réaliser.
$M = \{M_1, M_2, \ldots, M_m\}$	: Ensemble des machines.
n_i	: Nombre d'opérations du job J_i.
O_{ij}	: j^{eme} opération du job i.
P_{ij}	: Durée d'exécution de l'opération O_{ij}.
M_{ij}	: Machine sur laquelle l'opération O_{ij} est exécutée.
$O_i = \{O_{i1}, O_{i2}, \ldots, O_{ini}\}$	: Gamme opératoire du job J_i.
S_{ij}	: Date de début de l'opération O_{ij}.
C_{ij}	: Date de fin de l'opération O_{ij}.
C_i	: Date de fin de job J_i.
C_{max}=max $\{C_i, i = 1,\ldots,n\}$	: Makespan ou date de fin de tous les jobs

II.3. Description des contraintes de blocage

Un des principaux objectifs d'une entreprise est d'augmenter sa productivité en réduisant ses coûts. Il est donc important qu'elle réduise ses stocks et éventuellement qu'elle les supprime. Le stock est considéré comme l'un des facteurs importants dans l'étude de la complexité des problèmes d'ordonnancement.

La capacité de stockage entre les machines est supposée illimitée dans la conception classique des problèmes de type job-shop. Cette hypothèse est non réaliste dans les problèmes réels, à cause du problème d'espace entre les machines dans les lignes de production, ainsi que le coût élevé des espaces de stockage. Une solution pour réduire les coûts est de limiter l'espace de stockage ou annuler les stocks entre les machines.

Dans la suite de ce manuscrit, nous considérons que la capacité de stockage est nulle, ce qui conduit à des situations dites de blocage. Des exemples de différentes contraintes de blocage sont donnés dans la suite.

II.3.1. Contrainte de blocage de type *RSb*

Dans les situations de blocage classique, une machine reste bloquée par un job jusqu'à la fin d'exécution de l'opération et la libération de la machine suivante, c'est à dire le démarrage de l'opération suivante dans la gamme opératoire. Dans ce manuscrit, cette contrainte de blocage classique est appelée contrainte de blocage de type *RSb* (*Release when Starting blocking*).

II.3.2. Contrainte de blocage de type *RCb*

Dans cette partie, nous présentons la contrainte de blocage spécifique qui est utilisée dans plusieurs domaines industriels. Dans cette contrainte de blocage notée *RCb* (*Release when Completing blocking*), une machine reste bloquée par un job jusqu'à ce que son opération sur la machine suivante soit finie et qu'il quitte la machine. Cette contrainte de blocage a été définie pour la première fois dans (Dauzère-Pérès et *al*. 2000a)

Plusieurs applications industrielles utilisent cette contrainte de type *RCb*, on peut citer deux exemples (Martinez, 2005) : le premier concerne le traitement de déchets industriels et le deuxième la fabrication de pièces métalliques. Dans le premier exemple qui concerne le

traitement de déchets industriels, une compagnie reçoit différents types de déchets industriels et agricoles à traiter avant de les enfouir. Les déchets sont apportés par camions, chaque camion doit décharger sa cargaison dans un seul silo. Les silos peuvent être vus comme des machines parallèles. 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 contient (contrainte de blocage type *RCb*). La deuxième application concerne une entreprise qui réalise différentes pièces métalliques pour l'industrie aéronautique. Ces pièces doivent être chauffées à bonne température dans un ou plusieurs fours qui peuvent être vus comme des machines parallèles. Ensuite, les pièces chauffées doivent passer sur une presse unique pour être embouties. Un four qui traite un lot est bloqué jusqu'à ce que toutes les pièces qu'il contient soient embouties par la presse (contrainte de blocage type *RCb*).

II.4. Représentation des ordonnancements

Nous rappelons deux manières de représenter les solutions des problèmes d'ordonnancement : le diagramme de Gantt et le graphe conjonctif.

Dans ce qui suit, nous exposons ces deux méthodes de représentations, ainsi que leurs applications sur les problèmes d'ordonnancement avec contrainte de blocage de type *RSb* et *RCb*.

II.4.1. Diagramme de Gantt

Le diagramme de Gantt, développé par Henry Gantt (1861-1919), est le moyen le plus simple de représenter les problèmes d'ordonnancement, à la fois dans la littérature et dans l'entreprise.

Pour les problèmes d'atelier, chaque machine est représentée par une ligne horizontale, les opérations qui sont exécutées sur cette machine sont représentées sous forme des barres ayant des longueurs proportionnelle à leurs durées d'exécution. Des barres sont positionnées à la date de début d'exécution des opérations. L'axe des abscisses représente le temps. Ainsi que le diagramme de Gantt nous montre l'occupation des machines dans le temps, la séquence de traitement sur les machines et le temps de fin de traitement des jobs sur toutes les machines.

Considérons le problème de minimisation de *makespan* d'un job shop à 4 jobs et 4 machines avec les gammes opératoires suivantes :

- $J1 = \{(M1,2), (M2,1), (M4,2)\}$.

- *J2* = {(*M2*,2), (*M3*,1), (*M4*,1)}.
- *J3* = {(*M3*,1), (*M4*,2)}.
- *J4* = {(*M1*,1), (*M2*,1), (*M3*,2)}.

Où (*M1*,2) signifie que l'opération sera exécutée sur la machine *M*1 avec une durée de 2 unités de temps. L'ordonnancement sur les différentes machines choisi est le suivant :

- M1 : J1, J4
- M2 : J2, J1, J4
- M3 : J3, J2, J4
- M4 : J3, J2, J1

Le diagramme de Gantt de cet ordonnancement sans contrainte de blocage est représenté par le graphe de la figure 3.

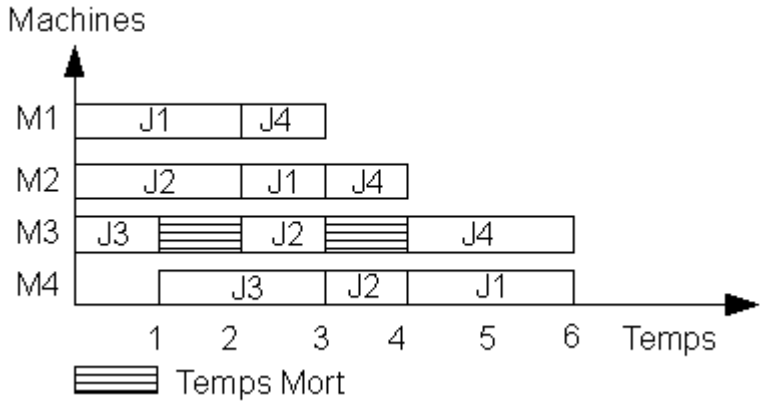

Figure 3 Diagramme de Gantt d'un problème sans contrainte de blocage

Le diagramme de Gantt pour le même problème avec la contrainte de blocage de type *RSb* est représenté sur la figure 4.

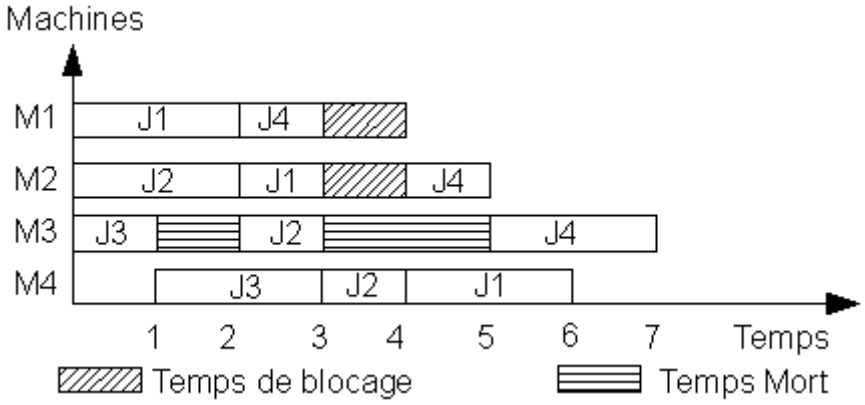

Figure 4 Diagramme de Gantt d'un problème avec contrainte de blocage type *RSb*

Dans la figure 4, nous constatons qu'un job ne peut démarrer sur une machine jusqu'à ce que la machine soit disponible, c'est-à-dire le début d'exécution de l'opération suivante du dernier job exécuté sur cette même machine. Par exemple, le job *J3* ne peut commencer sur la machine *M1* avant le début de job *J4* sur la machine *M2* (contrainte de blocage de type *RSb*). Une solution de ce même problème avec la contrainte de blocage de type *RCb* est représentée sur la figure 5.

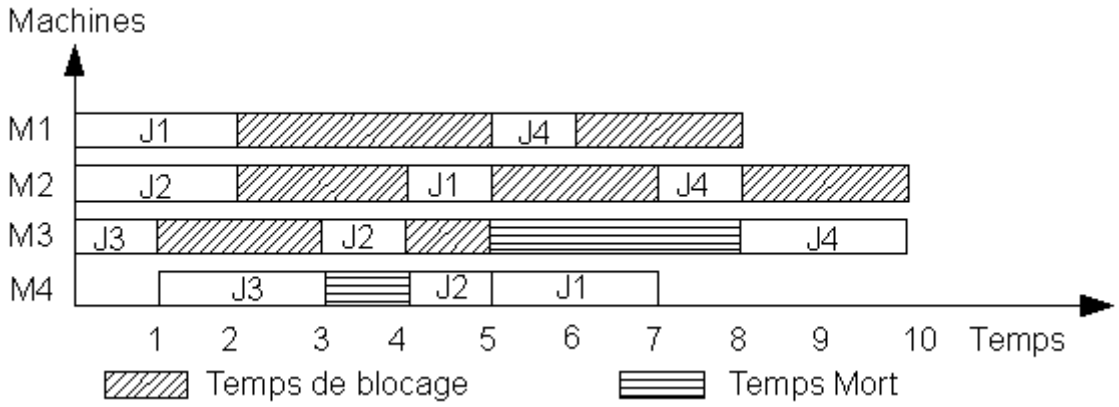

Figure 5 Diagramme de Gantt d'un problème avec contrainte de blocage type *RCb*

Sur cette figure, nous pouvons vérifier que la contrainte de blocage de type *RCb* est bien appliquée, le job J_3 ne peut démarrer sur la machine *M3* avant la fin du job J_2 sur la machine *M4* et le début de ce même job J_2 sur la machine *M1*.

II.4.2. Graphe disjonctif

Le graphe disjonctif est une autre façon de représenter les problèmes d'ordonnancement d'atelier. L'intérêt pour lequel le graphe disjonctif est abondamment utilisé est la modélisation des problèmes d'ateliers multi-machines.

Le graphe disjonctif a été proposé par Roy et Susmann en 1964 pour l'ordonnancement des job-shop.

Le graphe disjonctif est composé de sommets qui représentent les différentes opérations des jobs et de deux sommets pour indiquer le début (source) et la fin des jobs. Les sommets sont reliés entre eux par deux types d'arcs : arc conjonctif et arc disjonctif.

Les arcs conjonctifs relient deux opérations consécutives d'un même job, décrivant ainsi la contrainte de précédence entre les opérations. Les sources sont connectées aux premières opérations de chaque job par un arc conjonctif.

Les arcs disjonctifs relient les opérations qui s'exécutent sur la même machine et qui n'appartiennent pas au même job.

La solution réalisable est obtenue en orientant les arcs disjonctifs de façon à avoir un graphe acyclique (sans circuit). Le graphe disjonctif acyclique est appelé un graphe arbitré (Carlier et Chrétienne 1988).

Les figures suivantes représentent le graphe disjonctif des solutions d'un problème de job shop à 4 jobs et 4 machines avec les gammes opératoires suivantes :

- *J1* = {*M1*, *M2*, *M3*, *M4*}.
- *J2* = {*M2*, *M3*, *M4*, *M1*}.
- *J3* = {*M3*, *M4*, *M1*, *M2*}.
- *J4* = {*M4*, *M1*, *M2*, *M3*}.

La figure 6 présente le graphe disjonctif du problème sans blocage.

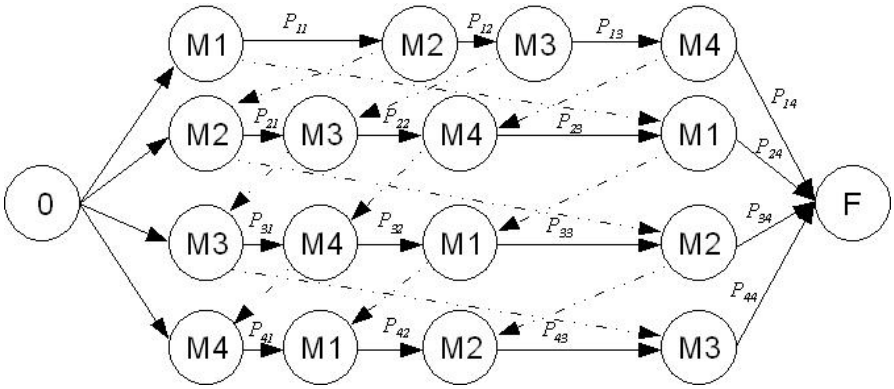

Figure 6 Graphe disjonctif du problème de type job-shop sans contrainte de blocage

La figure 7 représente le graphe disjonctif du problème avec contrainte de blocage de type *RSb*.

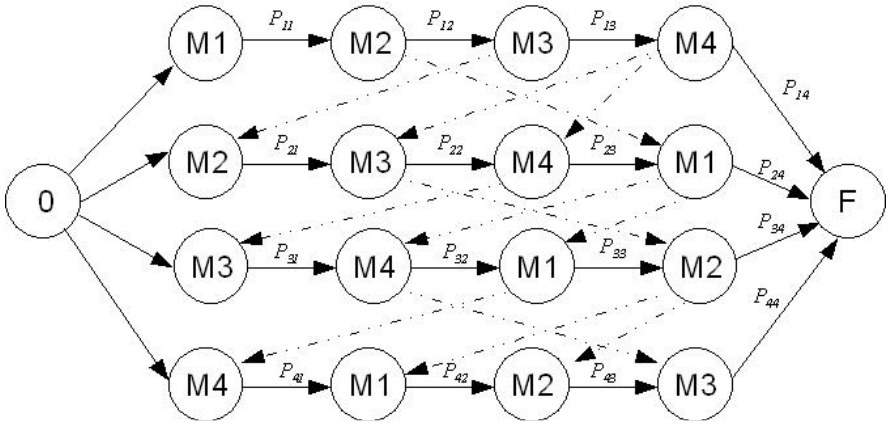

Figure 7 Graphe disjonctif du problème de type job-shop avec contrainte de blocage de type *RSb*

La figure 8 représente le graphe disjonctif du problème avec contrainte de blocage de type *RCb*. Dans ce cas, nous avons dû ajouter des nœuds fictifs w_i pour chaque job afin de pouvoir représenter la fin du dernier job de chaque gamme opératoire. Plus de détails sont disponibles dans (Martinez, 2005).

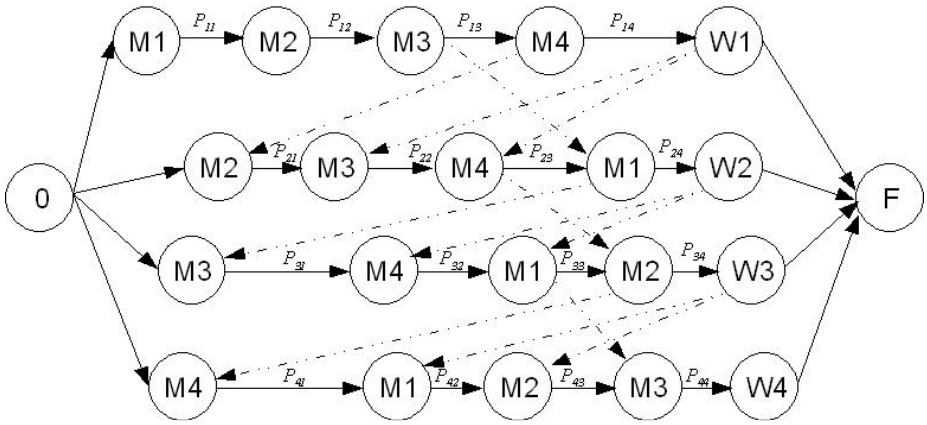

Figure 8 Graphe disjonctif du problème de type job-shop avec contrainte de blocage de type *RCb*

II.5. Conclusion

Dans ce chapitre, nous avons présenté le cadre dans lequel est réalisée cette thèse. Nous avons tout d'abord défini les systèmes de production de type Flow-Shop et Flow-Shop hybride. Nous avons ensuite présenté la contrainte de blocage classique ainsi que la contrainte de blocage spécifique que nous étudions dans cette thèse. Nous avons illustré cette contrainte par deux applications industrielles et nous montré comment représenter les différentes solutions d'un ordonnancement.

Dans le chapitre suivant, nous nous consacrons à l'état de l'art des systèmes de type Flow-Shop et Flow-Shop hybride avec et sans contraintes de blocage.

CHAPITRE III

ETAT DE L'ART

III.1. Introduction

Dans ce chapitre, nous dressons un état de l'art portant sur les problèmes d'ordonnancement avec et sans contrainte de blocage. Il est organisé en quatre parties. Les différents types de méthodes de résolution utilisées pour résoudre les problèmes d'ordonnancement sont présentés en première partie. La deuxième partie contient quelques notations que nous allons utiliser dans ce manuscrit. Dans la troisième et la quatrième partie, nous présentons un état de l'art des travaux existants sur la complexité et les approches développées pour résoudre les problèmes d'ordonnancement, dans le cas des problèmes sans contrainte de blocage puis des problèmes d'ordonnancement avec contrainte de blocage. Chacune des deux dernières parties se divise en deux sous parties, la première contient le problème d'atelier à cheminement unique (flow-shop) et la deuxième partie contient les problèmes d'atelier à cheminements multiples (job-shop).

III.2. Méthodes de résolution

Dans cette partie, nous présentons quelques méthodes d'optimisation combinatoire couramment utilisées pour la résolution des problèmes d'ordonnancement d'atelier. Nous entamons cette partie par les méthodes exactes, parmi ces méthodes nous décrivons la procédure par séparation et évaluation, la programmation dynamique et la programmation linéaire. Nous passons ensuite à la présentation des méthodes approchées, notamment les méthodes constructives et les méthodes d'amélioration.

III.2.1. Méthodes exactes

La programmation dynamique, les méthodes par séparation et évaluation ("Branch-and-Bound") et la programmation linéaire en nombres entiers, sont les principales méthodes de résolution exacte. Elles nous permettent de trouver une solution optimale grâce à une exploration intelligente de l'espace des solutions.

- **Procédure par Séparation et Evaluation (PSE)**

La procédure par séparation et évaluation aussi appelée "Branch and Bound" suivant le terme anglo-saxon, est une méthode d'optimisation combinatoire très utilisée. Cette méthode est basée sur la recherche par numération implicite des solutions. La numération concerne toutes les solutions possibles du problème mais l'analyse des propriétés du système permet d'éviter la numération des mauvaises solutions. Dans le cas de résolution des problèmes de minimisation, la première étape est de calculer une borne supérieure de la valeur de la fonction objective.

Comme son nom l'indique, la séparation consiste à séparer l'espace de solutions en plusieurs sous-ensembles (phase de branchement) de taille inférieure. L'évaluation consiste à calculer une borne inférieure (dans le cas des problèmes de minimisation) pour chaque sous-ensemble ou branche, afin d'éliminer les branches qui ne nous permettent pas d'obtenir une solution optimale.

La procédure d'élimination peut être résumée par les étapes suivantes :

- On calcule une borne inférieure pour chaque sommet d'un problème.
- Le sommet est éliminé si le minorant (la borne inférieure calculée dans l'étape précédente) dépasse la meilleure solution obtenue jusqu'à présent qui est une borne supérieure du problème.

L'exploration s'arrête si nous ne pouvons plus effectuer de séparation, c'est-à-dire lorsque l'arbre des solutions a été complètement exploré.

Cette méthode a été introduite pour la première fois par Dantzig et *al*. (1954) pour résoudre le problème de voyageur de commerce. Elle a été également appliquée à des problèmes d'ordonnancement dont on peut citer, par exemple, Carlier et Pinson (1989) appliquent la méthode PSE pour résoudre le problème de minimisation du makespan pour les problèmes de type job-shop. Cette méthode leurs permet de résoudre une instance à 10 jobs et 10 machines proposée par Fisher et Thompson (1963).

- **Programmation dynamique**

La programmation dynamique est une méthode d'optimisation développée par Bellman (1957) pour résoudre les problèmes de recherche du plus court chemin d'un graphe. Cette méthode nous permet de résoudre les problèmes avec contraintes dont la fonction objectif doit être décomposable (Gondran & Minoux 1984).

Le principe de cette méthode est basé essentiellement sur la décomposition du problème en une série de sous-problèmes reliés entre eux par une relation de récurrence et résolus localement de façon optimale. 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 de l'ensemble.

L'application de la programmation dynamique dans le domaine de l'ordonnancement reste limitée. Elle est utilisée pour développer des algorithmes polynomiaux et pseudo polynomiaux pour résoudre les problèmes à une machine et à machines parallèles (Lawler 1973), (Schrage & Baker 1978), (Potts & Van Wassenhove 1982b), (Carlier & Chrétienne 1988).

- **Programmation linéaire**

La programmation linéaire (PL) est un outil très puissant d'optimisation qui permet de résoudre un grand nombre des modèles linéaires. Une fois un problème modélisé sous forme d'équations linéaires, cette méthode assure la résolution optimale du problème.

Suivant l'ensemble de définition des variables, on distingue trois types pour cette méthode :

- La programmation linéaire **continue** pour laquelle les variables appartiennent à l'ensemble des réels. Ces problèmes peuvent être résolus par l'algorithme du Simplexe.

- La programmation linéaire **en nombres entiers** (PLNE) pour laquelle les variables sont des entiers. Les solutions de ces problèmes sont obtenues en utilisant la PSE (Gomory 1958).
- La programmation linéaire **mixte** pour laquelle il existe des variables réelles et des variables entières.

Parmi les logiciels de résolution des problèmes de PLNE on peut citer LP-Solve, Cplex, Mosel-Xpress (qui est aussi un langage de modélisation et résolution des problèmes d'optimisation).

Dans notre travail, nous avons utilisé le logiciel Mosel-Xpress pour la résolution des problèmes d'une façon optimale.

III.2.2. Méthodes approchées

La taille des problèmes influe de façon importante sur les temps de calcul des méthodes exactes. Ce temps de calcul est raisonnable pour les problèmes de petite taille, mais il devient vite prohibitif si la taille des problèmes augmente, et par conséquent il est utile de développer des méthodes approchées. Ces méthodes nous permettent de trouver une solution acceptable en un temps de calcul raisonnable. La qualité de la solution trouvée peut être donnée par l'estimation du pourcentage d'erreur entre la solution optimale et la solution trouvée par les méthodes approchées.

Il existe deux types de méthodes approchées : les méthodes constructives et les méthodes d'amélioration.

- Les méthodes constructives : Ce sont des méthodes très utilisées pour obtenir très rapidement une première solution pour les problèmes d'ordonnancement. La recherche d'une solution par ces méthodes (dites aussi méthodes heuristiques) commence à partir du zéro et à chaque itération une solution pour le problème est élaborée en utilisant, par exemple, des règles de priorité. Ces méthodes sont généralement très rapides.
- Les méthodes d'amélioration : Le principe de ces méthodes est d'améliorer une solution initiale trouvée généralement par une méthode constructive. Pour la plupart de ces méthodes, la recherche de la nouvelle solution est basée sur la notion de voisinage. Ces méthodes sont appelées aussi des méta-heuristiques. La recherche locale et la recherche globale sont les deux types de méthodes d'amélioration. La

recherche locale permet de sélectionner une solution meilleure que la solution courante à chaque itération. Par contre, la nouvelle solution obtenue par recherche globale n'améliore pas forcement la solution courante. Parmi ces méthodes, nous trouvons les méta-heuristiques telles que la recherche tabou, le recuit simulé et l'algorithme génétique. Dans ce qui suit, nous présentons brièvement la recherche tabou et l'algorithme génétique, le recuit simulé sera présenté en détail dans le chapitre 4.

- La recherche Tabou : La méthode tabou est une méta-heuristique d'optimisation combinatoire présentée par Glover (Glover 1989), (Glover 1990). Cette méthode appartient à l'ensemble des méthodes d'amélioration. Elle est basée principalement sur la notion de voisinage et de mouvements interdits. Le voisinage d'une solution est un ensemble des solutions réalisables que l'on peut obtenir en effectuant un mouvement prédéfini (permutation de deux opérations par exemple). Chaque mouvement est ajouté dans une liste dite la liste tabou de taille fixée à l'avance et qui contient donc les mouvements interdits. Il est à noter que la notion de voisinage, la taille de la liste tabou et les conditions d'arrêt peuvent être très différentes suivant le type de problème considéré. La méthode tabou est plus efficace pour les problèmes de taille réduite à cause du nombre de solutions voisines qui reste limité. Par contre, l'augmentation de la taille des problèmes peut fortement ralentir cette méthode. En effet, l'augmentation de la taille de problème élargit l'espace de voisinage ce qui fait exploser le nombre de solutions voisines. Dans ce dernier cas, il est intéressant de diminuer la taille du voisinage.

- L'algorithme génétique : cette méthode a été introduite par Rechenberg (1973), Holland (1975), et Schwefel (1977). Cette méthode s'appuie sur des techniques inspirées de la génétique et des mécanismes d'évolution naturelle. Contrairement aux méthodes d'amélioration par les algorithmes de recherche locale qui manipulent une seule solution réalisable, les algorithmes génétiques considèrent un ensemble de solutions réalisables (population) appelées individus. Le but de la méthode est d'évaluer la population courante, puis la faire évoluer en effectuant des mutations, des croisements et une sélection des individus. La mutation est une opération qui consiste à modifier la structure d'une solution (un individu). Le croisement est une opération binaire qui permet de produire deux nouveaux individus à partir de deux autres individus. Parmi les nouvelles solutions obtenues, la sélection nous permet, par exemple, de supprimer les individus les moins forts afin de garder les meilleures solutions.

III.3. Notations

La littérature portant sur les problèmes d'ordonnancement est très riche. Ces problèmes d'ordonnancement peuvent être notés de différentes façons. On peut citer celle introduite par Conway et *al*. (1967) qui contient quatre champs A|B|C|D dont les significations sont les suivantes :

- Le champ A contient les caractéristiques des machines.
- Le champ B contient les caractéristiques des opérations.
- Le champ C contient les contraintes de précédence.
- Le champ D contient les critères objectifs.

Cette notation a été utilisée pour la représentation des problèmes d'ordonnancement de base. Elle a été améliorée par Mac Carthy et Liu (1993) afin d'être utilisée pour la représentation de problèmes plus généraux.

Dans la suite de ce manuscrit, nous adoptons la notation proposée par Graham et *al*. (1979) qui utilise trois champs A|B|C pour la description des problèmes d'ordonnancement. Les significations de ces trois champs sont les suivantes :

- Le champ A est décomposé en a1 a2 où a1 représente le type d'atelier considéré et a2 le nombre des machines.
- Le champ B présente l'ensemble des contraintes sur les jobs.
- Le champ C indique le critère à optimiser.

Par exemple, J2 | *RCb* | C_{max} désigne la minimisation du *Makespan* pour un problème de type job-shop à deux machines avec contrainte de blocage de type *RCb*.

III.4. Problèmes d'ordonnancement sans blocage

Les problèmes d'ordonnancement classiques (sans contrainte de blocage) s'intéressent à ordonnancer des tâches dans les problèmes d'atelier pour lesquels la capacité de stockage entre les machines est suffisamment grande afin d'assurer un bon fonctionnement du système. Une solution au problème d'ordonnancement classique consiste à associer à chaque tâche une date de début et/ou de fin d'exécution qui respecte les contraintes temporelles.

Un état de l'art des travaux existant dans la littérature pour les problèmes le flow-shop et le job-shop sans blocage est présenté dans les paragraphes suivants.

III.4.1. Problèmes d'ateliers à cheminement unique

Dans cette partie, nous exposons les travaux existant dans la littérature concernant les problèmes de flow-shop classique et hybride. L'organisation de ce paragraphe est comme suit. Tous d'abord, les résultats de complexité sont présentés. Nous donnons ensuite les résultats utilisant des méthodes de résolution exacte et enfin, nous terminons par la présentation des travaux concernant les méthodes de résolution approchées.

a. Complexité

FS à deux machines

Pour les problèmes à deux machines, Johnson (1954) a montré que la minimisation du *makespan* était polynomiale. De plus, le problème *F*2| no-wait |*Cmax* peut être résolu polynomialement par l'algorithme proposé dans Gilmore et Gomory (1964), qui transforme le problème en un problème de voyageur de commerce avec des distances particulières.

Toujours pour la complexité des problèmes de type Flow-Shop à 2 machines, on trouve le cas du problème avec les dates de disponibilité pour les jobs et le makespan comme critère d'optimisation ($F2|ri|C_{max}$), qui est un problème NP-difficile au sens fort (Lenstra et *al*. 1977c). Cho et Sahni (1981) montrent également que le cas préemptif de ce problème ($F2|ri, pmtn|C_{max}$) est NP-difficile au sens fort.

FS à trois machines et plus

Pour les problèmes de flow-shop à trois machines Garey et *al*. (1976b) ont montré que le problème $F3|\ |C_{max}$ est NP-difficile au sens fort. Pour ce même problème, 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 (Johnson 1954).

Le problème du flow-shop préemptif à trois machines ($F3|pmtn|C_{max}$) est aussi un problème NP-difficile au sens fort (Gonzalez & Sahni 1976).

FS hybrides

Le cas du flow-shop hybride à deux étages avec le makespan comme critère d'optimisation est largement étudié. La complexité de ce problème avec plus d'une machine sur au moins un étage est NP-difficile (Gupta 1988).

Hoogeveen et *al*. (1996b) montrent que le cas préemptif d'un Flow-Shop contenant plusieurs machines parallèles identiques sur l'un des deux étages est NP-difficile au sens fort.

b. Méthodes de résolution exactes

Problèmes de FS

La résolution des problèmes flow-shop par les méthodes exactes fait référence à plusieurs auteurs. Parmi les techniques utilisées, on trouve la procédure par Séparation et Evaluation qui a été appliquée sur le problème $Fm| |C_{max}$ par plusieurs auteurs dont les premiers Ignall et Schrage (1965). Potts (1980a) a également proposé une méthode "Branch-and-Bound" avec laquelle il est possible de résoudre des instances jusqu'à 15 jobs et 4 machines. Aujourd'hui, on peut résoudre des instances jusqu'à 50 jobs et 10 machines en utilisant la procédure par Séparation et Evaluation proposés par Carlier et Rebaï (1996).

Problèmes de FS hybride

Pour le flow-shop hybride Arthanary et Ramaswamy (1971) proposent un "Branch-and-Bound" pour le problème $HF(a = 2, b = 1)| |C_{max}$.

Hunsucker et Brah (1987) proposent une méthode exacte basée sur la modélisation du problème par un programme linéaire mixte pour résoudre les problèmes de flow-shop hybride à m étages.

Brah et Hunsucker (1991) proposent une méthode de type "Branch-and-Bound" pour résoudre des problèmes $HFm| |C_{max}$ de petite taille.

Pour le problème de flow shop hybride où $m_1 = 1$, $m_2 = 2$, $m_3 = 1$, deux méthodes ont été développées par Riane *et al.* (1998). La première repose sur la programmation linéaire et la deuxième est basée sur le "Branch-and-Bound".

En 2001, Néron *et al.* (2001) ont proposé une nouvelle procédure de "Branch-and-Bound" pour le problème de Flow-shop hybride. Cette procédure repose principalement sur l'ajustement systématique des dates de disponibilité et des durées de latence des tâches.

c. Méthodes de résolution approchées

Les méthodes de résolution exactes perdent leurs efficacités dès que la taille du problème devient importante. Pour cela afin de trouver une solution acceptable en un temps polynomial, nous utilisons les méthodes approchées.

Problèmes de FS

Parmi les méthodes trouvées dans la littérature pour résoudre des problèmes de type flow-shop, nous pouvons citer Campbell et Dudek (1970), Dannenbring (1977) et Rock et Schmidt (1983), et Nawaz et *al*. (1983). Campbell et Dudek (1970), Dannenbring (1977) et Rock et Schmidt (1983) qui proposent des méthodes constructives basés sur une adaptation de l'algorithme de Johnson (1954) pour le problème $Fm|\ |C_{max}$. Un autre algorithme pour ce même problème $Fm|\ |C_{max}$ appelé *NEH* à été proposé par Nawaz et *al*. (1983). Cet algorithme est considéré comme le meilleur algorithme constructif dans l'étude réalisée par Leisten (1990).

Nowicki et Smutnicki (1998b) et Reeves (1993) proposent une recherche tabou pour ce même problème ($Fm|\ |C_{max}$). Anderson et *al*. (1997) ont réalisé une étude qui a permis de conclure que la méthode de Nowicki et Smutnicki (1996a) est considérée comme la meilleure méthode de recherche globale pour résoudre ce problème.

En 2003, Haouari et Ladhari (2003) ont proposé une méthode d'amélioration basée sur un "Branch-and-Bound" tronqué, pour minimiser le makespan.

Problèmes de FS hybride

Pour le problème de flow-shop hybride, on peut citer Gupta (1988) qui propose une heuristique pour le problème $HF(a = m, b = 1)|\ |C_{max}$ avec machines identiques.

Rao (1970), et Narasimhan et Panwalker (1984) proposent des heuristiques pour le problème à une machine sur le premier étage et deux machines sur le deuxième étage avec le makespan comme critère d'optimisation.

Pour le cas avec plus de 2 machines sur le deuxième étage, ($HF(a = 1, b = m)||C_{max}$), des bornes supérieures et inférieures ont été proposées par Gupta et Tunc (1991). Ces bornes sont utilisées dans une heuristique basée sur une méthode de "Branch-and-Bound".

Rao (1970) propose une approche logique pour résoudre des instances où un des deux étages contient plusieurs machines.

Pour le problème à 2 étages, avec le même nombre de machines parallèles identiques sur chaque étage ($HF(a = b = m)||C_{max}$), des méthodes heuristiques gloutonnes ont été proposées

par Langston (1987) et Shen et Chen (1972), ainsi que dans le cas préemptif pour ce problème, des heuristiques ont aussi été proposées par Buten et Shen (1973).

Pour les problèmes généraux avec plusieurs machines sur les deux étages ($HF(a = m_1, b = m_2)||C_{max}$), Lee et Vairaktarakis (1994) proposent une heuristique gloutonne et des bornes inférieures pour résoudre ce problème.

Toujours pour le problème de flow shop hybride Negenman (2001) a proposé une méthode de recherche locale basée sur un recuit simulé, une recherche Tabou ainsi qu'une méthode connue sous le nom Variable-depth Search (Kernighan & Lin 1970), (Marsland & Björnsson 2000). Le principe de ces méthodes est basé sur la recherche d'une solution dans le voisinage de la solution courante. Ce voisinage est constitué de toutes les solutions obtenues en effectuant une seule transition sur la solution courante.

Lee et Vairaktarakis (1994) proposent des heuristiques pour le $HFm||C_{max}$. Pour ce problème, on trouve aussi les heuristiques proposées par Nowicki et Smutnicki (1998b) qui sont basées sur la recherche tabou avec une notion de voisinage basée sur la notion de blocs proposée par (Grabowski et *al*. 1983).

Des heuristiques basées sur le recuit simulé et la recherche tabou ont été proposées par Haouari et M'Hallah (1997). Les résultats expérimentaux montrent que ces heuristiques sont plus performantes que celles proposées par Lee et Vairaktarakis. Plusieurs heuristiques ont été également été proposées pour le problème à deux étages avec différents critères d'optimisation (Narasimhan & Mangiameli 1987), (Sherali et *al*. 1990).

III.4.2. Problèmes d'ateliers à cheminements multiples

Les problèmes d'ordonnancement de type job-shop font partie des problèmes les plus étudiés dans la littérature (Jain & Meeran 1999). Cela s'explique par deux raisons : l'importance théorique du modèle, et de nombreuses applications pratiques qui peuvent être modélisées de cette façon.

Le job shop est une généralisation directe du problème d'ordonnancement d'atelier de type flow-shop, pour lequel la gamme opératoire des jobs peut varier d'un job à l'autre. Le problème est donc fortement combinatoire. Les chercheurs ont entamé ce domaine par des cas particuliers avec un nombre fixé de machine et de jobs.

a. Problèmes de type job-shop classique

Complexité

Parmi les travaux existant sur la complexité des problèmes de type job-shop, on trouve Garey et Johnson (1979) et Gonzalez et Sahni (1978). Les auteurs ont classé le problème de job-shop parmi les problèmes NP-difficiles au sens fort. Une démonstration de ce résultat de complexité peut être trouvée dans Rinnooy Kan (1976). Cette démonstration repose principalement sur la réduction polynômiale d'un problème de flow-shop vers un problème de knapsack.

Il existe des cas particuliers de problèmes de job-shop qui peuvent être classés parmi les problèmes polynômiaux. Parmi ces problèmes on peut citer deux cas qui sont faciles à résoudre.

Le premier est un problème à deux jobs. Ce type de problème fait l'objet des premiers efforts des chercheurs dans le domaine de la complexité. Les résultats de complexité pour ce cas sont dus à Sotskov (1985) et Brucker (1988). Ces derniers ont développé un algorithme polynomial qui permet de trouver la solution optimale pour le problème de job-shop à deux jobs. Cet algorithme est basé sur la représentation géométrique développée par Akers et Friedman (1955), Szwarc (1960), Hardgrave et Nemhauser (1963).

Le deuxième cas est le job-shop à deux machines pour lesquels une généralisation des règles de Jackson (1956) a été proposée pour résoudre des cas particuliers à deux machines $J2||C_{max}$ par un algorithme polynomial.

Méthodes de résolution exactes

La majorité des chercheurs dans le domaine de l'ordonnancement de job-shop utilise le "Branch-and-Bound" comme méthode de résolution exacte. Depuis 1965, les efforts des chercheurs portent sur la résolution optimale des problèmes d'ordonnancement. Parmi les travaux dans ce domaine, on peut citer (Giffler & Thompson 1960), (Ashour et *al*. 1973), (Brooks & White 1965), (Florian et *al*. 1971). Les auteurs ont proposé des algorithmes de résolution basés sur le principe de génération d'ordonnancements actifs. Des méthodes plus efficaces ont été développées plus récemment afin de trouver la solution optimale en un temps modéré ((Carlier & Pinson 1990), (Appelgate & Cook 1991), (Brucker et *al*. 1994)).

Méthodes de résolution approchées

Les méthodes exactes perdent leur efficacité dès que la taille des problèmes devient importante. Donc, pour résoudre les problèmes de moyennes et grandes tailles, il est

nécessaire de développer des heuristiques. Dans ce qui suit, nous citons les travaux existant sur les heuristiques constructives et d'améliorations pour les problèmes de type job-shop.

Les méthodes constructives consistent à construire une solution pour le problème à partir des données initiales. Deux méthodes sont utilisées dans ce contexte. Pour la première, il s'agit de l'approche par opération qui repose principalement sur les règles de priorité, cette méthode basée sur l'ordonnancement des opérations l'une après l'autre selon des règles de priorité. Pour la deuxième, il s'agit de l'approche par machine. Cette méthode consiste à trouver un séquencement sur les machines les unes après les autres et à ordonnancer les opérations sur les machines.

Plusieurs travaux ont été réalisés sur l'approche par opération. On peut citer (Panwalkar & Iskander 1977) où les auteurs ont proposé une centaine de règles de priorité, ainsi qu'une classification de ces règles. Vancheeswaran et Townsend ont proposé d'autres règles de priorité afin de trouver une valeur minimale du makespan (Vancheeswaran & Townsend 1993).

Une méthode basée sur la technique d'insertion des opérations a été proposée par Werner et Winkler (1991). Cette méthode consiste à insérer les opérations par ordre décroissant de leur durée opératoire dans un ordonnancement partiel.

Ces méthodes constructives permettent de trouver une solution initiale qui sera utilisée dans les méthodes d'améliorations telles que les algorithmes génétiques, la recherche Tabou et le recuit simulé. De nombreux travaux ont été réalisés sur la résolution du problème de job-shop par les algorithmes génétiques, parmi ceux-ci nous pouvons citer (Davis 1985), (Falkenauer & Bouffouix 1991), (Dorndorf & Pesch 1992) et (Nakano & Yamada 1991).

Van Laarhoven et *al*. (1992) a présenté des résultats sur la résolution des problèmes de job-shop par la méthode de recuit simulé. Lourenço (1994) a comparé le recuit simulé avec d'autres méthodes de voisinage telles que la recherche tabou.

La notion de voisinage fait l'objet de plusieurs travaux dans le domaine du job-shop, parmi lesquelles on peut citer (Van Laarhoven et *al*. 1992). Les auteurs montrent qu'il existe toujours une séquence de mouvements qui mène à la solution optimale. Matsuo et *al*. (1988) ont proposé un voisinage qui consiste à permuter deux opérations dans le chemin critique à condition que ces opérations appartiennent à la même machine et que l'un des arcs entrant de la première opération ne soit pas dans le chemin critique et de même pour les arcs sortant.

Dell'Amico et Trubian (1993) proposent une généralisation du premier voisinage dans lequel plusieurs permutations sont possibles en même temps.

b. Problèmes de type job-shop flexible

Le job-shop hybride est une extension du problème d'ordonnancement de type job-shop classique pour lequel une opération peut être traitée par l'une des machines d'un étage, ce qui crée une complexité supplémentaire. Les travaux portant sur les problèmes de type job-shop hybrides ne sont pas nombreux. Dans ce qui suit nous présentons quelques travaux existants dans la littérature pour ce problème.

Comme précédemment, nous commençons par présenter les travaux existants sur la complexité des problèmes de job shop hybride puis les méthodes de résolution.

Complexité

Pour le problème de job shop flexible, Jurisch (1995) a proposé un algorithme permettant de résoudre d'une façon polynomiale la relaxation du problème à deux jobs afin de trouver des bornes inferieures. Le problème de job-shop flexible devient NP-difficile à partir trois jobs et deux machines.

Méthodes de résolution

Pour la résolution optimale du cas général du job shop flexible à plusieurs jobs, Jurisch (1992) a proposé une méthode de séparation et évaluation**.**

Gomes et *al*. (2005) présentent un modèle de programmation linéaire en nombres entiers (PLNE) pour le problème de type job shop flexible avec la minimisation du makespan comme critère.

Toujours pour le problème de job-shop hybride, Penz (1996) a proposé une heuristique basée sur une procédure itérative.

Dans l'article présenté par Dauzère-Pérès et Pavageau (1998), les auteurs ont proposé une représentation par graphe disjonctif ainsi qu'un voisinage. Ce voisinage est utilisé dans une recherche locale pour le problème de job-shop avec flexibilité de ressources.

Chen et *al*. (1999) présentent un algorithme génétique pour résoudre ce problème. Une approche basée sur l'algorithme génétique est également proposée dans (Chan et *al*. 2006).

Scrich et *al*. (2004) ont développé deux heuristiques basées sur la recherche Tabou pour les problèmes de type job-shop hybride. Les procédures d'utilisation des règles de répartition sont utilisées pour obtenir une solution initiale. Ils cherchent ensuite des solutions dans les voisinages générés par les chemins critiques des jobs, à l'aide d'une représentation par graphe disjonctif.

Alvarez-Valdes et *al*. (2005) a présenté une heuristique pour la conception et la mise en œuvre d'un système d'ordonnancement dans une usine de verre dans le but de minimiser le retard. La structure correspond à un problème d'ordonnancement de type job-shop flexible.

Loukil et *al*. (2007) ont proposé un algorithme de recuit simulé pour minimiser les retards dans un problème de job-shop hybride. Un algorithme génétique pour minimiser le retard total d'un problème de job-shop hybride a été développé par Nait Tahar et *al*. (2004). Les résultats obtenus ont montré que l'algorithme génétique est efficace pour la résolution de ce problème. Plus tard, un algorithme d'optimisation par colonie de fourmis a été développé par Nait Tahar et *al*. (2005) afin de minimiser ce critère.

Finalement, Xia et *al* (2005) ont proposé une approche hybride pour le problème de job-shop flexible multi-objectif.

III.5. Problèmes d'ordonnancement avec blocage

Le problème de blocage est présent dans tous les systèmes de production avec des capacités de stockage entre les machines limitées ou nulles. Dans ce qui suit nous présentons quelques travaux sur les différents ateliers de production avec contrainte de blocage de type *RSb*. A notre connaissance, il existe très peu de travaux concernant la contrainte de blocage de type *RCb*. Ces travaux sont présentés en fin de chapitre.

III.5.1. Problèmes d'ateliers à cheminement unique avec blocage *RSb*

Les systèmes de production de type flow-shop avec contrainte de blocage ont été rencontrés dans différentes compagnies industrielles par plusieurs auteurs ((Reklaitis 1982), (Grabowski & Pempera 2000), (Sawik 2000) entre autres). Une étude sur différentes approches ainsi que des résultats de complexité pour les systèmes sans attente avec blocage a également été proposée par Hall et Sriskandarajah (1996).

Complexité

Gilmore et Gomory (1964) ont montré que le problème de flow-shop à deux machines avec contrainte sans attente noté $F2|Sans\ attente|C_{max}$ est polynomial. Papadimitriou et Kanellakis (1980) prouvent que le problème $F2|b\ finie|C_{max}$ est NP-difficile au sens fort.

Dans le cas à 3 machines, Hall et Sriskandarajah (1996) montrent que le problème $F3|b = 0|C_{max}$ est NP-Difficile au sens fort.

Pour le problème de Flow-Shop à *m* machines avec capacité de stockage limitée, $Fm|b\ finie|PW_iC_i$, Khosla (1989) a prouvé que sa complexité est NP-difficile.

Méthodes de résolution exactes

Pour les problèmes $F2|b\ finie|C_{max}$, une solution optimale pour les instances de petite taille a été trouvée par la méthode de programmation dynamique proposée par Dutta et Cunningham (1975). Pour ce même problème, une méthode de programmation dynamique avec un espace d'états limité a été proposée par Dutta et Cunningham (1975).

Pour le $F2|b = 1|C_{max}$, Papadimitriou et Kanellakis (1980) proposent une heuristique qui utilise l'algorithme de Gilmore et Gomory (1964).

La programmation dynamique a été utilisée aussi pour résoudre le problème $Fm|b\ finie|C_{max}$ par Reddi (1976), dans le cas où il y a une faible différence entre le nombre de jobs à ordonnancer et la capacité de stockage.

Pour le problème $F3|b = 0|C_{max}$, Suhami et Mah (1981) ont développé une heuristique, ainsi qu'une procédure pour l'évaluation d'une borne inférieure dans un "Branch-and-Bound".

Pour la résolution des problèmes de flow-shop sans attente, Levner (1969) propose un "Branch-and-Bound" pour résoudre le problème $Fm|b = 0|C_{max}$. Il a montré que ce problème est équivalent au problème qui consiste à trouver un plus long chemin dans un graphe orienté.

Khosla (1989) propose une borne inférieure ainsi que des règles de priorité pour minimiser la somme pondérée de dates de fin pour le problème $Fm|b\ finie|PW_iC_i$.

Pour les problèmes de flow-shop hybrides avec blocage, plusieurs auteurs ont résolus le problème par des méthodes exactes. On peut citer Sawik (2000) qui a résolu les problèmes $HFm|b=0|C_{max}$ et $HFm|b\ finie|Cmax$ par la programmation linéaire en nombres entiers. Il a présenté des résultats expérimentaux pour des instances jusqu'à 10 jobs, 3 étages avec 2 et 3 machines par étage et des capacités de stockage entre les étages de $b = 0$ et $b = 3$ jobs.

Wittrock (1985) a proposé une méthode "Branch-and-Bound" et un modèle mathématique pour le problème $HFm|b = 0|C_{max}$. Pour le même problème une description mathématique ainsi qu'un algorithme itératif ont été proposés par Sawik (1995). Des résultats expérimentaux pour des problèmes de 100 jobs et 5 étages avec 4 machines parallèles par étage sont présentés.

Méthodes de résolution approchées

Pour la résolution des problèmes de flow-shop avec contrainte par les méthodes approchées Nowicki (1999) a proposé une méthode de recherche tabou pour le problème $Fm|b\ finie,\ prmu|C_{max}$.

Pour les problèmes de type flow-shop avec capacité de stockage limitée ($Fm|bfinie|C^t$) avec la minimisation du temps de cycle comme critère, McCormick et *al*. (1989) ont fait une étude qui consiste à transformer le problème en un problème sans capacité de stockage. Pour cela, ils considèrent que le stockage entre les machines est une opération de durée opératoire nulle.

Caraffa, Ianes, Bagchi et Sriskandarajah (2001), proposent un algorithme génétique pour résoudre le problème $Fm|b = 0|C^t$.

Pour le flow-shop hybride, Wittrock (1988) a proposé un algorithme constructif pour le problème $HFm|b\ finie|C_{max}$. Sawik (1993) propose un algorithme constructif pour résoudre le même problème. L'algorithme proposé par Sawik a donné de meilleures performances que celui de Wittrock.

Pour le même problème $HFm|b\ finie|C_{max}$, une méthode de recherche tabou a été présentée par Wardono et Fathi (2004).

Pour le problème $HF(a = m,\ b = 1)|b\ finie|\mathrm{P}W_iC_i$, Khosla (1995) propose 4 heuristiques ainsi que des bornes inférieures. Les résultats expérimentaux pour des instances jusqu'à 50 jobs, 3 machines et $b = 12$ ont été présentés.

III.5.2. Problèmes d'ateliers à cheminement multiples avec blocage *RSb*

Très peu de travaux ont été réalisés en ordonnancement d'atelier de type job-shop notamment le job-shop avec blocage *RSb*. Dans ce qui suit, nous allons citer quelques travaux qui ont été réalisés dans la littérature en ordonnancement de type job-shop et job-shop hybride.

a. Problèmes de type job-shop classique

Banaszak et Krogh (1990) ont proposé un système de production de type job-shop pour lequel la machine reste occupée par un job jusqu'au passage à l'opération suivante. Les auteurs ont proposé une politique de contrôle en temps réel basée sur l'algorithme de Banker. Ensuite, des améliorations de cet algorithme sont proposées dans (Hsieh & Cheng 1994), (Xing et *al*. 1996).

Wysk et *al* (1991) ont proposé une modélisation basée sur la théorie des graphes et des techniques d'évitements de blocages. Pour l'évitement de blocage, Fanti et *al*. (1996, 1997, 1998) ont proposé deux graphes orientés pour décrire les relations entre les routages de jobs et les machines. Ces graphes ont été utilisés pour formuler une stratégie de contrôle permettant d'éviter le blocage.

b. Problèmes de type job-shop flexible

Parmi les quelques travaux portant sur le job-shop flexible avec blocage, on peut citer (Ezpeleta et *al*. 1995) , les auteurs ont généralisé le modèle proposé par Banaszak et Krogh (1990) en introduisant la flexibilité de routage.

Dans (Mati et *al*. 2002b), les auteurs ont proposé un algorithme polynomial pour le job-shop à deux jobs avec flexibilité des ressources, ainsi qu'une méta-heuristique basée sur la recherche tabou et une autre méta-heuristique basée sur les algorithmes génétiques pour ce même problème.

III.5.3. Problème avec contrainte de blocage de type *RCb*

Comme nous l'avons cité plus tôt, il existe peu de travaux portant sur la contrainte de blocage de type *RCb*. Les premiers travaux ont été ceux de Dauzère-Pérès et *al*. (2000a) où les auteurs proposent une modélisation PLNE pour la résolution d'un problème réel d'ordonnancement avec la contrainte de blocage de type *RCb.*

Dans le cas des problèmes d'ordonnancement de type flow-shop avec contrainte de blocage de type *RCb*, une étude de complexité a été réalisé par Martinez et *al*. (2006b) et une méthode exacte, des bornes inférieures et un recuit simulé ont été proposés dans (Martinez 2005).

Dans l'étude de la complexité, les auteurs ont prouvé que le problème F2|*RCb*|C_{max} est polynomial et qu'il est équivalent au problème 1||C_{max}. Ils ont également prouvé que le problème F3|*RCb*(1,2),*RCb*(2,3)|C_{max} et le problème *F*4|*RCb*(1,2),*RCb*(2,3),*RCb*(3,4)|C_{max}

sont polynomiaux. Par contre, le problème $F5|RCb(1,2),RCb(2,3),RCb(3,4),RCb(4,5)|C_{max}$ est NP-Difficile.

Concernant la résolution du problème par la méthode optimale, les auteurs ont proposé deux modèles linéaires en nombres entiers. Les résultats obtenus pour le premier modèle, montrent que des problèmes jusqu'à 12 jobs et 20 machines sont résolus dans un temps raisonnable. Le deuxième modèle est basé sur la discrétisation de l'horizon, les auteurs trouvent que ce dernier modèle est nettement moins performant que le premier modèle.

Enfin, les auteurs ont développé une méta-heuristique basée sur le recuit simulé. Les auteurs ont testé plusieurs voisinages. Parmi les voisinages testés, les auteurs ont montré que le meilleur voisinage consiste à déplacer un job dans la séquence de la solution courante.

Pour le même problème, Sauvey et Sauer (2011d) ont proposé une méta-heuristique basée sur les algorithmes génétiques. Les résultats obtenus par cette méthode ont montré l'efficacité de la méthode proposée.

L'étude de (Martinez et *al*. 2004c) sur le flow-shop hydride avec la contrainte *RCb* concerne la modélisation en PLNE d'un problème de flow-shop hybride à deux étages avec une seule machine au second étage. Les résultats obtenus montrent que les problèmes jusqu'à 15 jobs et 3 machines parallèles au premier étage sont résolus en un temps raisonnable. Dans (Martinez 2005), l'auteur propose également des heuristiques et une méta-heuristique pour résoudre ce même problème.

Yuan et Sauer (2009) ont proposé des bornes inférieures pour les problèmes de grande taille pour les problèmes de type Flow-shop hybride. Dans ce même article, les auteurs ont proposé une heuristique basée sur l'algorithme dit *EM-algorithm* (électromagnétisme algorithme) pour les problèmes de type flow shop hybride avec contrainte de blocage de type *RCb*.

III.6. Conclusion

Dans ce chapitre, nous avons présenté les travaux existants portant sur les problèmes d'ordonnancement de la production. Nous avons détaillé les principales méthodes de résolution des problèmes d'ordonnancement. Nous avons décrit les différentes notations utilisées dans la littérature pour exprimer les problèmes d'ordonnancement.

Ensuite, nous nous sommes intéressés aux problèmes de type job-shop et flow-shop. Nous avons présenté les différentes études qui ont été réalisées pour résoudre ce type de problème d'ordonnancement sans les contraintes de blocage.

Nous avons présenté les travaux existants portants aussi sur les problèmes d'ordonnancement avec la contrainte de blocage de type *RSb*. Les problèmes avec contraintes de blocage sont moins étudiés, en particulier dans le cas d'utilisation de la contrainte de blocage de type *RCb*.

Dans le chapitre suivant nous abordons la présentation de notre contribution, à savoir la modélisation du problème de job-shop par modèle PLNE avec la contrainte de blocage de type *RCb*, et les bornes inférieures que nous avons proposées.

CHAPITRE IV JOB SHOP CLASSIQUE AVEC LA CONTRAINTE *RCB*

IV.1. Introduction

Dans ce chapitre nous présentons une méthode exacte et trois bornes inférieures pour résoudre le problème d'ordonnancement d'un système de production de type Job-Shop avec la contrainte de blocage *RCb*.

Ce chapitre contient deux parties. Dans la première partie, nous présentons plus en détail le problème, nous décrivons la méthode de résolution exacte, qui est basée sur le modèle linéaire

en nombres entiers (PLNE). Le modèle mathématique, la signification des variables, des contraintes ainsi que des résultats expérimentaux sont présentés.

Dans la deuxième partie, nous présentons les bornes inférieures. Pour chaque borne, nous donnons un exemple d'application, une démonstration ainsi que les résultats expérimentaux.

IV.2. Description du problème

Dans cette section, nous introduisons le modèle d'ordonnancement de type job shop avec la contrainte de blocage de type *RCb* et nous donnons les notations utilisées.

Dans ce modèle, on considère un ensemble de n jobs exécutés sur un ensemble de m machines. Chaque job J_i est constitué d'un ensemble de n_i opérations noté O_{ij}. Les contraintes qui relient les opérations aux machines sont :

- Chaque opération ne peut être exécutée que sur une seule machine.
- Chaque machine ne peut exécuter qu'une seule opération simultanément.
- La préemption des opérations n'est pas autorisée.
- La capacité de stockage entre les machines est nulle.
- La machine reste bloquée par une opération O_{ij} jusqu'à la fin d'exécution de l'opération suivante O_{ij+1} et le début de O_{ij+2} si elles existent (contrainte *RCb*).

Le problème est de déterminer le meilleur ordonnancement des jobs sur les machines afin de minimiser le *makespan*, c'est-à-dire le temps total d'exécution.

Dans ce qui suit, nous allons présenter un modèle mathématique basé sur la programmation linéaire en nombre entier (PLNE) ainsi que des bornes inferieures pour résoudre ce problème. Des résultats expérimentaux pour le modèle PLNE et les bornes inférieures sont également présentés.

IV.3. Méthode de résolution exacte

Dans cette section, nous présentons une modélisation du problème du type Job-Shop avec blocage *RCb* pour des problèmes de n jobs et m machines.

Le problème revient donc à déterminer pour chaque machine, l'ordre dans lequel les opérations vont passer sur la machine afin de minimiser la date de fin de l'ordonnancement.

Le modèle que nous donnons est basé sur la formulation proposée dans Mati et *al.* (2002) dans le cas d'un Job Shop multi-ressources sous contrainte *RCb*.

IV.3.1. Paramètres

Dans cette partie nous présentons les différents paramètres et données utilisés pour la formulation mathématique du problème afin de calculer la solution optimale des problèmes de job-shop classique avec la contrainte de blocage de type *RCb*.

Les paramètres et données du problème sont :

- n : nombre de jobs.
- m : nombre de machines.
- n_i : nombre d'opérations du job i.
- P_{ij} : durée d'exécution d'opération O_{ij}.
- M_{ij} : machine sur laquelle nous exécutons l'opération O_{ij}.
- $O_i = \{O_{i1}, O_{i2}, \ldots, O_{ini}\}$: gamme opératoire du job J_i.
- λ : constante positive très grande. Dans notre travail, λ est définie comme suit :

  $\lambda = \sum_{i=1}^{n} \sum_{j=1}^{n_i} P_{ij}$.

- $\delta_{ij}^{0} = \begin{cases} 1 \; si \; j \leq \; n_i - 2 \\ 0 \; si \; non \end{cases}$: signifie que l'opération O_{ij} n'est ni la dernière ni l'avant dernière opération de la gamme opératoire du job i.
- $\delta_{ij}^{1} = \begin{cases} 1 \; si \; j \leq \; n_i - 1 \\ 0 \; si \; non \end{cases}$: signifie que l'opération O_{ij} est l'avant dernière opération de la gamme opératoire du job i.
- $\delta_{ij}^{2} = \begin{cases} 1 \; si \; j \leq \; n_i \\ 0 \; si \; non \end{cases}$: signifie que l'opération O_{ij} est la dernière opération de la gamme opératoire du job i.

IV.3.2. Variables de décision

Les variables de décision du modèle permettent d'obtenir, d'une part, la date de début de chaque opération et d'autre part, l'ordre de passage des opérations sur chaque machine. Nous avons donc :

- S_{ij} = date de début d'exécution de l'opération O_{ij}.

- $y_{ij,i'j'}$ = 1 si l'opération O_{ij} est placée avant l'opération $O_{i'j'}$ et les opérations O_{ij} et $O_{i'j'}$ sont exécutées sur la même machine ($M_{ij}= M_{i'j'}$), et 0 sinon.

$y_{ij,i'j'}$ est définie pour tous les couples d'opérations (O_{ij}, $O_{i'j'}$) partageant la même ressource.

IV.3.3. Modèle mathématique

En utilisant les variables de décision présentées dans le paragraphe précédent, le problème peut être modélisé de la façon suivante :

$$\min C_{max} \quad (1)$$

Sous les contraintes suivantes :

$$S_{ij} \geq S_{i(j-1)} + P_{i(j-1)}, \forall i \in [1,n], \forall j \in [1,n_i] \quad (2)$$

$$\begin{cases} S_{ij} \geq (S_{i'(j'+2)}).\delta^0_{i'j'} + (S_{i'(j'+1)}+P_{i'(j'+1)}).\delta^1_{i'j'} + (S_{i'j'}+P_{i'j'}).\delta^2_{i'j'} - \lambda.y_{ij,i'j'} \\ S_{i'j'} \geq (S_{i(j+2)}).\delta^0_{ij} + (S_{i(j+1)}+P_{i(j+1)}).\delta^1_{ij} + (S_{ij}+P_{ij}).\delta^2_{ij} - \lambda.(1-y_{ij,i'j'}) \\ \forall \left(O_{ij}, O_{i'j'}\right)/ i < i' et\ M_{ij} = M_{i'j'} \end{cases} \quad (3)$$

$$C_{max} \geq S_{in_i} + P_{in_i}, \forall i \in [1,n] \quad (4)$$

$y_{ij,i'j'}$ est binaire, $\forall \left(O_{ij}, O_{i'j'}\right)/\ M_{ij} = M_{i'j'}$ (5)

$$S_{ij} \geq 0, \forall i \in [1,n], \forall j \in [1,n_i] \quad (6)$$

IV.3.4. Signification des équations

Dans ce paragraphe nous donnons une explication pour les contraintes du modèle mathématique présenté au paragraphe précédent.

- Contrainte 2 : cette équation assure la contrainte de précédence entre les opérations d'un même job. Une opération ne peut commencer à être exécutée avant la fin du traitement de l'opération précédente du même job.
- Contraintes 3 : Elles expriment le partage de ressource entre deux opérations O_{ij} et $O_{i'j'}$ passant sur la même machine. Il y a plusieurs cas possibles en fonction de la position de l'opération O_{ij} dans la gamme opératoire O_i :
  - Si $\delta^0_{ij} = 1$, et par conséquent $\delta^1_{ij} = \delta^2_{ij} = 0$, et si l'opération $O_{i'j'}$ précède O_{ij}, i.e. $y_{ij,i'j'}$ = 0, la première contrainte (3) implique que $S_{ij} \geq S_{i'(j+2)}$. En effet, la machine est bloquée par le job $J_{i'}$ jusqu'à ce que ce job ait terminé son opération $O_{i'(j'+1)}$ et

qu'il ait quitté la machine, i.e. jusqu'à l'instant où il peut commencer son opération $O_{i'(j+2)}$. Par contre, la seconde contrainte (3) nous donne $S_{i'j'} \geq S_{i(j+2)} - \lambda$, ce qui est toujours vérifié si λ est une constante très grande. Dans le cas où O_{ij} précède $O_{i'j'}$, i.e. $y_{ij,i'j'} = 1$, la première contrainte (3) est toujours vérifiée et la seconde implique que $S_{i'j'} > S_{i(j+2)}$.

- Si $\delta_{ij}^1 = 1$ ou $\delta_{ij}^2 = 1$, i.e. O_{ij} est l'avant dernière ou la dernière opération du job J_i, l'équation (3) est adaptée pour prendre en compte le fait qu'il n'y ait plus qu'une seule ou aucune opération derrière O_{ij} dans la gamme O_i.

- Contrainte 4 : Cette contrainte impose que la valeur du makespan doit être supérieure ou égale aux dates de fin des dernières opérations pour tous les jobs.
- Contrainte 5 : $y_{ij,i'j'}$ est une variable binaire. Pour deux opérations O_{ij} et $O_{i'j'}$ telles que $M_{ij} = M_{i'j'}$. la variable $y_{ij,i'j'}$ prend la valeur 1 si l'opération $O_{i'j'}$ précède O_{ij} sur la machine M_{ij} et 0 sinon.
- Contrainte 6 : cette équation signifie que la date de début de chaque opération doit être positive ou nulle.

IV.3.5. Résultats expérimentaux

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

Pour un problème de taille $n \times m$ (n jobs et m machines), nous avons généré 100 instances différentes. Pour chacune de ces instances, le nombre d'opérations par job est choisi aléatoirement dans l'intervalle [*m/2, m*]. Les gammes opératoires ont été générées aléatoirement, les durées d'exécution des opérations ont été générées uniformément dans l'intervalle [0, 9]. Le Tableau 1 indique les temps de calcul moyen en secondes obtenus par le logiciel d'optimisation *Mosel Xpress* pour trouver la solution optimale. Tous les calculs ont été réalisés sur un PC Intel Pentium (R) 3GHz, 1Go de RAM.

Les instances à 9 jobs et 100 machines, 10 jobs et 20, 50 et 100 machines n'ont pas été testés car les temps de calcul sont très élevés (plus de 24 heures par problème).

Pour les problèmes de tailles 8 jobs et 50, 100 machines, 9 jobs et 20, 50 machines et 10 jobs et 15 machines, le temps de calcul est très important (plus de 9 heures par problème). Nous

avons donc calculé la solution optimale pour seulement 10 problèmes différents pour chacune de ces tailles.

Jobs	Machines							
	5	6	7	10	15	20	50	100
5	0,065	0,15	0,17	0,39	0,64	0,9	2,22	9,15
6	0,14	0,27	0,3	0,94	2,37	7,32	29,84	75,69
7	0,24	0,49	0,57	3,24	29,19	131,83	1085,38	7330.96
8	0,49	1,07	1,4	19,31	346,71	2707.07	35667*	27007*
9	1,07	3,24	4,05	102,19	4531,63	24345*	88249*	> 24 h
10	5,39	12,92	16,85	1160,56	58149*	> 24 h	> 24 h	> 24 h

Tableau 1. Temps de calcul moyen (en secondes) de la solution optimale pour 100 instances différentes pour chaque taille de problème (* 10 instances pour ces problèmes).

Nous pouvons constater que les temps de calcul moyen restent raisonnables pour les problèmes dont le nombre de machines varie entre 5 et 10 machines, pour ces problèmes le temps de calcul est inférieur à 20 minutes.

Pour les problèmes de taille moyenne (15 et 20 machines), le temps de calcul est raisonnable pour les problèmes avec le nombre de job inférieur ou égal à 7, mais il devient clairement très important pour les problèmes à plus de 7 jobs. Ce temps atteint 16 heures pour les problèmes de 10 jobs 15 machines.

Pour les problèmes de grande taille (50 et 100 machines), le temps d'exécution est très élevé à partir de 7 jobs.

IV.4. Bornes inférieures

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 d'estimer le pourcentage d'erreur avec la solution optimale (si elle est calculable) ou des bornes inférieures de qualité. Nous proposons dans cette section trois bornes inférieures pour notre problème. Ces bornes sont ensuite comparées à la solution optimale obtenue avec la modélisation présentée dans le paragraphe précédent.

IV.4.1. Borne inférieure 1

La première borne inférieure, que nous appellerons *Binf1*, est basée sur le temps maximal d'occupation des machines.

Cette borne est donnée dans le théorème suivant.

Théorème 1 : Soit un Job-Shop composé de *m* machines et *n* jobs soumis à la contrainte de blocage de type *RCb*. Une borne inférieure du makespan du problème d'ordonnancement de ce système est donnée par :

$$Binf1 = \max_{k=1,\dots m}\left[\sum_{\substack{i=1,\dots,n/\\ \exists O_{il}\in O_i \text{ et}\\ M_{il}=M_k}}\left[P_{il}+P_{i(l+1)}.\psi_{il}\right]+\min_{\substack{i=1,\dots,n/\\ \exists O_{il}\in O_i \text{ et}\\ M_{il}=M_k}}\left[\sum_{s=0}^{l-1}P_{is}\right]+\min_{\substack{i=1,\dots,n/\\ \exists O_{il}\in O_i \text{ et}\\ M_{il}=M_k}}\left[\sum_{s=l+2}^{n_i}P_{is}\right]\right]$$

Où $\psi_{il}=\begin{cases}1 & \text{si } l<n_i\\ 0 & \text{sinon}\end{cases}$.

Démonstration de théorème 1.

Chaque machine M_k peut avoir trois états :

- Machine disponible.
- Machine en cours d'exécution d'une opération.
- Machine bloquée par un job : si une opération O_{ij} du job *i* qui est effectuée sur cette machine M_k n'est pas la dernière opération dans la gamme opératoire du job *i*, et si elle a fini de s'exécuter sur la machine M_k et que l'opération O_{ij+1} qui suit l'opération O_{ij} sur le job *i* n'a pas encore quitté la machine.

Le temps total d'exécution d'une opération plus le temps de blocage est supérieur ou égal à un temps *T1* :

$$T1=\sum_{\substack{i=1,\dots,n/\\ \exists O_{il}\in O_i \text{ et}\\ M_{il}=M_k}}\left[P_{il}+P_{i(l+1)}.\psi_{il}\right]$$

Le temps $T1$ peut être représenté par la figure suivante :

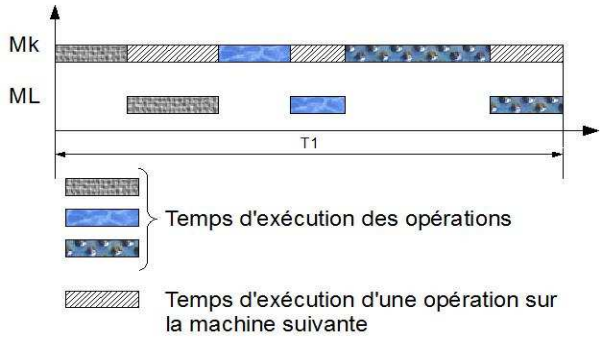

Figure 9 : Diagramme de Gantt représentant la période *T1* pour la machine M_k.

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

$$T2 = \min_{\substack{i=1,\dots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=0}^{l-1} P_{is} \right]$$

De même, on peut représenter le temps $T2$ par la figure 2. Dans la figure 2, le temps $T2$ = min $\{T21, T22, T23\}$.

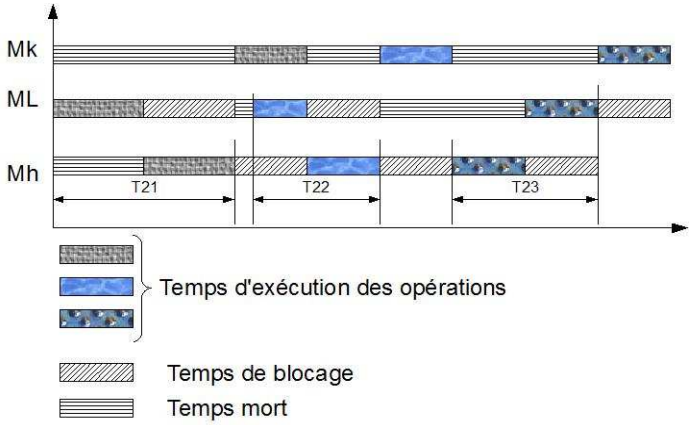

Figure 10 : Diagramme de Gantt représentant la période *T2* pour la machine M_k.

De la même façon, après l'exécution d'une opération O_{il} sur M_k, il faut que toutes les opérations qui suivent O_{il} dans la gamme de J_i se terminent. Par conséquent, après toutes les opérations O_{il} et $O_{i(l+1)}$ qui sont réalisées sur M_k, il faut attendre au moins le temps $T3$ avant la fin de l'ordonnancement.

Le temps $T3$ est donné par : $T3 = \min_{\substack{i=1,\dots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=l+2}^{n_i} P_{is} \right]$

La figure 3 représente le temps $T3$. A partir de cette figure le temps $T3 = min\ \{T31, T32, T33\}$.

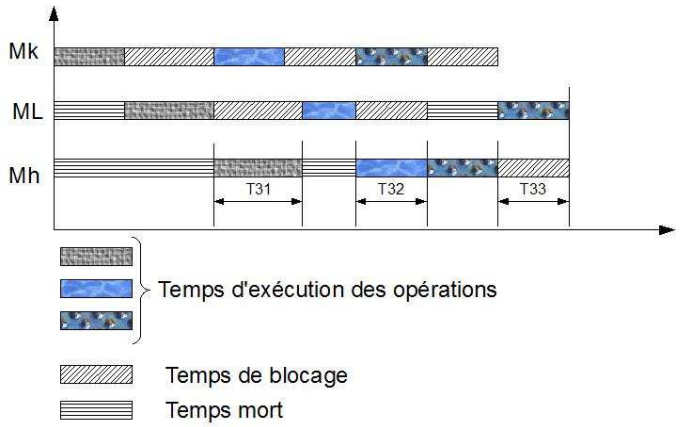

Figure 11 : Diagramme de Gantt représentant la période *T3* pour la machine M_k.

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

Si on considère toutes les machines, on obtient :

$$C_{max} \geq Binf1 = \max_{k=1,\dots m} \left[\sum_{\substack{i=1,\dots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[P_{il} + P_{i(l+1)} . \psi_{il} \right] + \min_{\substack{i=1,\dots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=0}^{l-1} P_{is} \right] + \min_{\substack{i=1,\dots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=l+2}^{n_i} P_{is} \right] \right]$$

CQFD

Exemple : On considère un problème d'ordonnancement de type job-shop constitué de 3 jobs et de 4 machines soumis à la contrainte *RCb* dont les gammes opératoires sont les suivantes :

- $J_1 = \{(M_3, 18)\}$.
- $J_2 = \{(M_1, 86)\}$.
- $J_3 = \{(M_2, 7), (M_1, 51), (M_3, 31)\}$.

Dans cet exemple, une opération est représentée comme suit : (M_i, P_{ij}), avec M_i la machine sur laquelle l'opération O_{ij} est exécutée, et P_{ij} le temps d'exécution de l'opération O_{ij}.

Nous avons calculé la solution optimale pour cet exemple et elle est égale à 168 ut. Calculons maintenant la borne inférieure.

Pour $k = 1,\ldots, m$, on note :

$$D_k = \sum_{\substack{i=1,\ldots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[P_{il} + P_{i(l+1)}.\psi_{il}\right] + \min_{\substack{i=1,\ldots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=0}^{l-1} P_{is}\right] + \min_{\substack{i=1,\ldots,n/ \\ \exists O_{il} \in O_i \text{ et} \\ M_{il}=M_k}} \left[\sum_{s=l+2}^{n_i} P_{is}\right]$$

Nous allons calculer D_k pour chacune des machines.

- Machine M_1 : les opérations O_{21} et O_{32} sont exécutées sur M_1 donc :

$$D_1 = [P_{21} + P_{32} + P_{33}] + \min(0, P_{31}) + \min(0,0) = 168ut .$$

- Machine M_2 : l'opération O_{31} est exécutée sur cette machine, donc :

$$D_2 = [P_{31} + P_{32}] + 0 + [P_{33}] = 153ut .$$

- Machine M_3 : les opérations O_{11} et O_{33} sont exécutées sur M_3 donc :

$$D_3 = [P_{11} + P_{33}] + \min(0, P_{31} + P_{32}) + \min(0,0) = 49ut .$$

Par conséquent :

$$Binf1 = max(D_1, D_2, D_3) = 168ut .$$

Dans cet exemple, la borne inférieure *Binf1* égale à 168 ut et est égale à la solution optimale.

IV.4.2. Borne inférieure 2

La deuxième borne inférieure que nous appellerons *Binf2*, est basée sur les jobs. Elle est donnée par le théorème suivant.

Théorème 2 : Soit un Job-Shop composé de *m* machines et *n* jobs soumis à la contrainte de blocage de type *RCb*. Une borne inférieure du makespan du problème d'ordonnancement de ce système est donnée par :

$$Binf2 = \max_{\substack{(O_{ij}, O_{kl}) \\ / M_{ij}=M_{kl}}} \left(\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right)\right)$$

Où :

- $A_{ij}^{kl} = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=1}^{\min(n_k, l+1)} P_{ks} - \sum_{s=1}^{j-1} P_{is}\right]$

- $B_{ij}^{kl} = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=l}^{n_k} P_{ks} - \sum_{s=j+2}^{n_i} P_{is} + \varphi_{ij}^{kl}.\left(P_{k(l-1)}. + P_{i(j+2)}\right)\right]$

- $\varphi_{ij}^{kl} = \begin{cases} 1 & \text{si } j < n_i\text{-}1, l > 1 \text{ et } M_{i(j+1)} = M_{k(l-1)} \\ 0 & \text{sinon} \end{cases}$

Remarque :

- Pour un job J_i, on pose $P_{ij} = 0$ pour tout $j > n_i$.
- Cette borne permet d'améliorer la borne inférieure classique donnée par :

$$Binf\ * = \max_{i=1,\ldots,n} \sum_{s=1}^{n_i} P_{is}\ .$$

- Pour le calcul de cette borne inférieure, il est nécessaire de considérer les couples : (O_{ij}, O_{kl}) et (O_{kl}, O_{ij}) car $A_{ij}^{kl} \neq A_{kl}^{ij}$ et $B_{ij}^{kl} \neq B_{kl}^{ij}$.

Avant de démontrer le théorème, nous présentons le calcul de cette borne inférieure sur l'exemple précédent.

Exemple : Nous prenons les mêmes données que l'exemple précédent. Pour le calcul de cette borne, nous considérons uniquement les opérations exécutées sur les mêmes machines. Les opérations O_{11} *et* O_{33} doivent être exécutées sur la machine M_3. Nous allons donc calculer les valeurs A_{11}^{33} *et* B_{11}^{33} puis A_{33}^{11} et B_{33}^{11}.

$$\begin{aligned} A_{11}^{33} &= \sum_{s=1}^{1} P_{1s} + \max\left[0, \sum_{s=1}^{3} P_{3s} - \sum_{s=1}^{0} P_{1s}\right] \\ &= 18 + \max[0, 71 + 51 + 31 - 0] = 171ut \end{aligned}$$

$$\begin{aligned} B_{11}^{33} &= \sum_{s=1}^{1} P_{1s} + \max\left[0, \sum_{s=3}^{3} P_{3s} - \sum_{s=1+2}^{1} P_{1s} + 0\right] \\ &= 18 + \max[0, 31 - 0] = 49ut \end{aligned}$$

Donc le minimum entre A_{11}^{33} et B_{11}^{33} est 49 ut.

$$A_{33}^{11} = \sum_{s=1}^{3} P_{3s} + \max\left[0, \sum_{s=1}^{1} P_{1s} - \sum_{s=1}^{3-1} P_{3s}\right]$$
$$= 71+51+31+\max[0,18-71-51] = 153ut$$

$$B_{33}^{11} = \sum_{s=1}^{3} P_{3s} + \max\left[0, \sum_{s=1}^{1} P_{1s} - \sum_{s=3+2}^{3} P_{3s} + 0\right]$$
$$= 71+51+31+\max[0,18] = 171ut$$

Donc le minimum entre A_{33}^{11} et B_{33}^{11} est 153 ut.

On passe ensuite aux opérations O_{21} et O_{22} qui sont exécutées sur la machine M_1. En suivant le même raisonnement, on obtient :

$$A_{21}^{32} = \sum_{s=1}^{1} P_{2s} + \max\left[0, \sum_{s=1}^{3} P_{3s} - \sum_{s=1}^{0} P_{2s}\right] = 239ut$$

$$B_{21}^{32} = \sum_{s=1}^{1} P_{2s} + \max\left[0, \sum_{s=2}^{3} P_{3s} - \sum_{s=1+2}^{1} P_{2s}\right] = 168ut$$

Donc le minimum entre A_{21}^{32} et B_{21}^{32} est 168 ut.

$$A_{32}^{21} = \sum_{s=1}^{3} P_{3s} + \max\left[0, \sum_{s=1}^{1} P_{2s} - \sum_{s=1}^{1} P_{3s}\right] = 168ut$$

$$B_{32}^{21} = \sum_{s=1}^{3} P_{3s} + \max\left[0, \sum_{s=1}^{1} P_{2s} - \sum_{s=2+2}^{3} P_{3s}\right] = 239ut$$

Donc le minimum entre A_{32}^{21} et B_{32}^{21} est 168 ut.

La borne inférieure est donnée par :

$$Binf2 = \max_{\substack{(O_{ij},O_{kl}) \\ / M_{ij}=M_{kl}}} \left(\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right)\right) = 168ut$$

Pour cet exemple, la deuxième borne inférieure *Binf2* est aussi égale à la solution optimale qui est 168 ut.

Démonstration du Théorème 2

Nous allons considérer chaque couple d'opérations réalisées sur la même machine. Soient la paire d'opération (O_{ij}, O_{kl}) qui s'exécute sur la même machine. On note que O_{ij} est la $j^{ème}$

opération de la gamme du job J_i et O_{kl} la $l^{ème}$ opération de la gamme du job J_k. Nous avons donc $M_{kl} = M_{ij}$. La démonstration se décompose en deux parties :

- Premier cas : Si O_{kl} est planifiée avant O_{ij}, nous démontrons que $A_{ij}^{kl} \le C_{\max}$. Par conséquent, quelle que soit la valeur de B_{ij}^{kl}, nous aurons bien $\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right) \le C_{\max}$.
- Deuxième cas : Si O_{kl} est planifiée après O_{ij}, nous démontrons que $B_{ij}^{kl} \le C_{\max}$. De même, quelle que soit la valeur de A_{ij}^{kl}, nous aurons bien $\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right) \le C_{\max}$.

Par conséquent dans tous les cas : $\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right) \le C_{\max}$.

Premier cas : Montrons que si O_{kl} est planifiée avant O_{ij} alors dans tous les cas $A_{ij}^{kl} \le C_{\max}$.

a. <u>O_{kl} *est la dernière opération du job* J_k *i.e.* $l = n_k$</u>

Si il n'y a pas d'attente entre les opérations de la gamme O_k, le temps minimum nécessaire avant le début d'exécution de l'opération O_{ij} est $\gamma = \max\left(\sum_{s=1}^{j-1} P_{is}, \sum_{s=1}^{l=n_l} P_{ks}\right)$ et le temps minimum après la fin de O_{kl} pour terminer le job J_i est supérieur ou égal à $\alpha = \sum_{s=j}^{n_i} P_{is}$ (voir Figure 12).

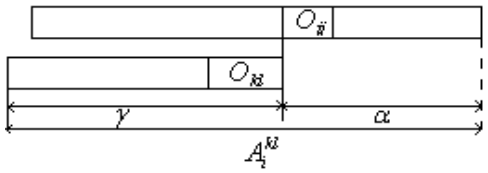

Figure 12. Cas où O_{kl} est planifiée avant O_{ij} et $l = n_k$

Donc le job J_i ne peut pas se terminer avant l'instant :

$$T = \alpha + \gamma = \sum_{s=j}^{n_i} P_{is} + \max\left(\sum_{s=1}^{j-1} P_{is}, \sum_{s=1}^{l=n_l} P_{ks}\right)$$
$$= \sum_{s=1}^{n_i} P_{is} + \max(0, \sum_{s=1}^{l} P_{ks} - \sum_{s=1}^{j-1} P_{is})$$

Comme $l = n_l = \min(n_l, l+1)$, nous avons bien :

$$T = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=1}^{\min(n_l, l+1)} P_{ks} - \sum_{s=1}^{j-1} P_{is}\right] = A_{ij}^{kl}$$

D'où : $A_{ij}^{kl} \le C_{\max}$.

b. *O_{kl} n'est pas la dernière opération du job J_k i.e. $l < n_k$*

Dans ce cas, le temps minimum nécessaire avant le début d'exécution de l'opération O_{ij} est $\gamma = \max\left(\sum_{s=1}^{j-1} P_{is}, \sum_{s=1}^{l+1} P_{ks}\right)$ car la machine $M_{kl} = M_{ij}$ est bloquée par l'opération O_{kl} au moins jusqu'à la fin du traitement de l'opération suivante de la gamme O_k (voir Figure 13). Le temps minimum à partir du début O_{ij} de pour terminer le job J_i est toujours .

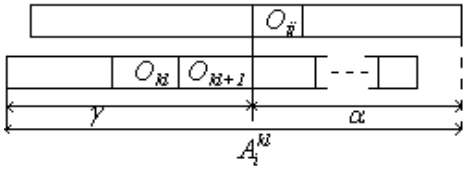

Figure 13. Cas où O_{kl} est planifiée avant O_{ij} et $l < n_k$

Donc le job J_i ne peut pas se terminer avant l'instant :

$$T = \alpha + \gamma = \sum_{s=j}^{n_i} P_{is} + \max\left(\sum_{s=1}^{j-1} P_{is}, \sum_{s=1}^{l+1} P_{ks}\right)$$

D'où :

$$T = \sum_{s=1}^{n_i} P_{is} + \max(0, \sum_{s=1}^{l+1} P_{ks} - \sum_{s=1}^{j-1} P_{is})$$

Comme $l+1 = \min(n_l, l+1)$, nous avons bien :

$$T = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=1}^{\min(n_l, l+1)} P_{ks} - \sum_{s=1}^{j-1} P_{is}\right] = A_{ij}^{kl} \le C_{\max}.$$

Par conséquent, si O_{kl} est planifiée avant O_{ij}, $A_{ij}^{kl} \le C_{\max}$ et donc $\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right) \le C_{\max}$.

Deuxième cas : Montrons que si O_{kl} est planifiée après O_{ij}, alors dans tous les cas $B_{ij}^{kl} \le C_{\max}$.

a. *O_{ij} est la dernière opération du job J_i i.e. $j = n_i$*

S'il n'y a pas d'attente entre les opérations de la gamme O_i, le temps minimum nécessaire avant le début d'exécution de l'opération O_{ij} est $\gamma = \sum_{s=1}^{j} P_{is} = \sum_{s=1}^{n_i} P_{is}$ (voir Figure 14) et le temps minimum nécessaire après la fin de O_{ij} pour terminer le job J_k est $\alpha = \sum_{s=l}^{n_k} P_{ks}$.

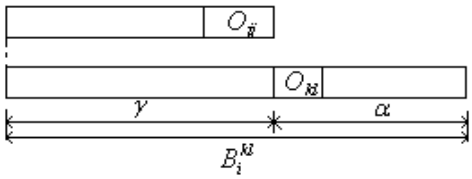

Figure 14. Cas où O_{kl} est planifiée après O_{ij} et $j= n_i$

Donc le job J_k ne peut pas se terminer avant l'instant :

$$T = \alpha + \gamma = \sum_{s=l}^{n_k} P_{ks} + \sum_{s=1}^{n_i} P_{is}$$

Comme $j= n_i$, $\varphi_{ij}^{kl} = 0$ et $\sum_{s=j+2}^{n_i} P_{is} = 0$. Nous avons bien :

$$T = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=l}^{n_k} P_{ks} - \sum_{s=j+2}^{n_i} P_{is} + \varphi_{ij}^{kl}.\left(P_{k(l-1)}. + P_{i(j+2)}\right)\right]$$

Donc

$$T = B_{ij}^{kl} \le C_{\max}.$$

b. *O_{ij} n'est pas la dernière opération du job J_i i.e. $j< n_i$*

d.1 $M_{i(j+1)} \neq M_{k(l-1)}$ i.e. $\varphi_{ij}^{kl} = 0$

Le temps minimum nécessaire avant le début d'exécution de l'opération O_{kl} est $\gamma = \sum_{s=1}^{j+1} P_{is}$ car la machine M_{ij} est bloquée par l'opération O_{ij} jusqu'à la fin du traitement de l'opération suivante de la gamme O_i (voir Figure 15). Le temps minimum après la fin de $O_{i(j+1)}$ pour terminer le job J_k est $\alpha = \max\left[\sum_{s=j+2}^{n_i} P_{is}, \sum_{s=l}^{n_k} P_{ks}\right]$.

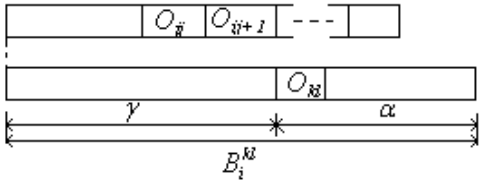

Figure 15. Cas où O_{kl} est planifiée après O_{ij}, $l+1 \le nk$ et $M_{i(j+1)} \neq M_{k(l-1)}$

Donc le job J_k ne peut pas se terminer avant l'instant :

$$T = \alpha + \gamma = \max\left[\sum_{s=j+2}^{n_i} P_{is}, \sum_{s=l}^{n_k} P_{ks} \right] + \sum_{s=1}^{j+1} P_{is}.$$

$$T = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=l}^{n_k} P_{ks} - \sum_{s=j+2}^{n_i} P_{is} \right] = B_{ij}^{kl} \le C_{\max}.$$

d.2. $\underline{M_{i(j+1)} = M_{k(l-1)}}$ i.e. $\varphi_{ij}^{kl} = 1$

Ce qui change par rapport au cas d.1, c'est que l'opération $O_{k(l-1)}$ ne peut pas débuter avant la fin de l'opération $O_{i(j+2)}$ car la machine $M_{k(l-1)} = M_{i(j+1)}$ est bloquée par l'opération $O_{i(j+1)}$ jusqu'à la fin du traitement de l'opération suivante de la gamme O_i (voir Figure 16). En effet, si $O_{k(l-1)}$ avait débuté nous aurions été dans une situation de blocage : $O_{k(l-1)}$ ne pouvait pas libérer $M_{k(l-1)}$ avant que O_{kl} ne termine sur M_{kl} mais $M_{kl} = M_{ij}$ ne peut pas être libre tant que $O_{i(j+1)}$ n'a pas terminé sur $M_{i(j+1)} = M_{k(l-1)}$.

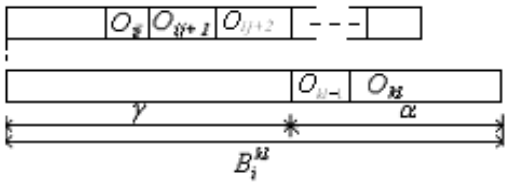

Figure 16. Cas où O_{kl} est planifiée après O_{ij}, $l+1 \le nk$ et $M_{ij+1}=M_{lk-1}$

Par conséquent :

$$\gamma = \sum_{s=1}^{j+1} P_{is} + \varphi_{ij}^{kl} . \left(P_{k(l-1)} . + P_{i(j+2)} \right).$$

D'où :

$$T = \sum_{s=1}^{n_i} P_{is} + \max\left[0, \sum_{s=l}^{n_k} P_{ks} - \sum_{s=j+2}^{n_i} P_{is} + \varphi_{ij}^{kl} . \left(P_{k(l-1)} . + P_{i(j+2)} \right) \right]$$

Donc :

$$T = B_{ij}^{kl} \le C_{\max}$$

Par conséquent, si O_{kl} est planifiée après O_{ij}, nous avons $B_{ij}^{kl} \le C_{\max}$ donc :

$$\min\left(A_{ij}^{kl}, B_{ij}^{kl} \right) \le C_{\max}.$$

Comme le job O_{kl} est planifié avant ou après O_{ij}, nous avons dans tous les cas :

$$C_{\max} \ge Min\left(A_i^{kl}, B_i^{kl} \right)$$

Si nous considérons toutes les opérations O_{ij} et O_{kl} effectuées sur une même machine, nous obtenons :

$$Binf = \max_{\substack{(O_{ij},O_{kl}) \\ / M_{ij}=M_{kl}}} \left(\min\left(A_{ij}^{kl}, B_{ij}^{kl}\right)\right)$$

Cqfd

IV.4.3. Borne inférieure 3

La troisième borne inférieure *Binf3* que nous avons implémentée est basée sur le fait que chacune des deux bornes précédentes *Binf1* et *Binf2* est efficace pour des problèmes bien définis. Donc la borne inférieure *Binf3* est le maximum entre les deux bornes.

Binf3= max (*Binf1*, *Binf2*).

IV.4.4. Résultats numériques

Les Tableau 2 et Tableau 3 nous donnent les pourcentages d'erreur moyens entre les différentes bornes inférieures et la solution optimale $C_{\max}$, calculée dans le chapitre précédent, ainsi que le nombre de fois où les bornes inférieures sont égales à la solution optimale (nombre entre parenthèses). Le pourcentage d'erreur est calculé de la façon suivante :

$$\%err = \frac{C_{\max} - Binfi}{C_{\max}} \times 100 .$$

Jobs	Machines											
	5			6			7			10		
	Binf1	*Binf2*	*Binf3*	*Binf1*	*Binf2*	*Binf3*	*Binf1*	*Binf2*	*Binf3*	*Binf1*	*Binf2*	*Binf3*
5	**6,61**	30,07	**6,6**	15,18	32,69	**14,9**	19,32	31,60	**18,75**	25,71	33,18	**25,18**
6	6,07	33,82	**6,05**	**14,45**	39,59	**14,45**	17,21	36,37	**17,09**	28,24	39,09	**27,98**
7	**4,6**	40,26	**4,6**	15,43	43,08	**15,32**	15,79	41,10	**15,07**	32,74	37,98	**31,81**
8	**5,41**	43,24	**5,41**	**13,34**	47,52	**13,34**	17,31	45,6	**17,26**	**27,49**	47,79	**27,49**
9	**4,16**	47,27	**4,16**	**10,89**	51,43	**10,89**	**15,17**	49,4	**15,17**	**28,34**	51,37	**28,34**
10	**3,72**	51,06	**3,72**	**10,58**	54,62	**10,58**	**14,59**	51,68	**14,59**	**26,59**	53,7	**26,59**

Tableau 2 Le pourcentage d'erreur moyen entre les trois bornes inférieure et la solution optimale pour les problèmes de moins de 10 machines.

Le Tableau 2 nous montre que le pourcentage d'erreur pour *Binf1* varie de 3,72 à 19,32 selon la taille des problèmes alors que pour ces mêmes problèmes, la borne inférieure *Binf2* varie entre 30,07 et 51,68.

Nous pouvons constater aussi que le pourcentage d'erreur de *Binf1* diminue lorsque le nombre de jobs augmentent. En effet, la borne inférieure 1 se compose de 3 paramètres dont l'un devient plus important lorsque le nombre de jobs augmente et par conséquent améliore la borne inférieure.

Jobs	Machines											
	15			20			50			100		
	Binf1	*Binf2*	*Binf3*	*Binf1*	*Binf2*	*Binf3*	*Binf1*	*Binf2*	*Binf3*	*Binf1*	Binf2	Binf3
5	27,48	26,12	**23,8**	24,41	20,84	**19,58**	**8,92**	**4,7**	**4,67**	3,88	**1,49**	**1,49**
6	30,22	31,63	**27,77**	29,1	26,18	**24,39**	**15,43**	**7,21**	**7,19**	6,88	**1,46**	**1,46**
7	28,6	43,52	**28,51**	33,03	32,54	**30,32**	20,36	**9,57**	**9,57**	14,49	**1,74**	**1,74**
8	33,59	41,5	**33,27**	34,12	37,08	**32,93**	26,76	**12,76**	**12,76**	17,04	**2,77**	**2,77**
9	33,55	45,24	**33,49**	39,39	41,26	**38,20**	35,58	**20,43**	**20,43**	NC	NC	NC
10	**32,93**	50,92	**32,93**	NC	NC	NC	NC	NC	NC	NC	NC	NC

Tableau 3 Le pourcentage d'erreur moyen entre les trois bornes inférieure et la solution optimale pour les problèmes de 15 à 100 machines.

À partir du tableau 3, nous pouvons constater que l'écart entre la solution optimale et *Binf2* est nettement inférieur à celui avec la borne *Binf1*. Cet écart est inférieur à 2,77 pour les problèmes à 100 machines. La deuxième borne inférieure contient deux paramètres qui dépendent tous les deux du nombre de machines, ce qui conduit à l'amélioration de la borne inférieure 2 lorsque le nombre de machines augmente.

En conclusion, la borne inférieure 1 nous donne de bons résultats pour les problèmes avec un nombre de machines réduit, et la borne inférieure 2 donne de bons résultats pour les problèmes de grande taille (50 et 100 machines). Pour ces problèmes, le temps de calcul de la solution optimale est très important sachant que le temps de calcul des bornes inférieures est de moins d'une seconde par problème.

IV.5. Conclusion

Nous avons présenté dans ce chapitre un modèle linéaire en nombres entiers pour résoudre le problème d'ordonnancement de type job-shop avec la contrainte de blocage *RCb*. Ce modèle a été validé avec le logiciel d'optimisation Xpress-Mosel. Les résultats obtenus pour les problèmes de différentes tailles générés de façon aléatoire, montrent que des problèmes jusqu'à 10 jobs et 10 machines, ainsi que 15 et 20 machines et 7 jobs maximum sont résolus dans un temps raisonnable, inférieur à 20 minutes.

Nous avons présenté également deux bornes inférieures. La première est basée sur le temps maximal d'occupation des machines. Elle nous donne de bons résultats pour les problèmes contenant un nombre limité de machines.

La seconde borne est basée sur les jobs et la particularité de la contrainte considérée. Elle donne de meilleurs que la première borne pour les problèmes dont le nombre de machines est plus élevé. Ces deux bornes sont donc complémentaires.

Dans le chapitre suivant, nous allons analyser plus en détail le problème de conflit et proposer une méthode pour résoudre ce problème. Nous donnerons également une méta-heuristique de type recuit simulé.

CHAPITRE V **METHODES DE RESOLUTION APPROCHEES**

V.1. Introduction

Dans le chapitre précédent, nous avons proposé une méthode de résolution exacte pour le problème de job-shop avec contrainte de blocage de type *RCb*, ainsi que des bornes inférieures pour ce problème. Dans ce chapitre, nous donnons des conditions mathématiques pour représenter les cas de conflits, ainsi qu'une méthode heuristique basée sur le recuit simulé pour résoudre le problème d'ordonnancement de type job-shop soumis à la contrainte *RCb*.

Ce chapitre est organisé de la façon suivante. La première partie est consacrée à l'étude du problème de conflit. Dans cette partie, nous proposons des conditions mathématiques définissant les situations de conflit. Dans la deuxième partie, nous proposons des heuristiques de liste et nous donnons des résultats expérimentaux. La troisième partie présente une méthode approchée basée sur l'algorithme de recuit simulé pour résoudre le problème de job-shop avec la contrainte de blocage de type *RCb*. La dernière partie fournit des résultats expérimentaux qui permettent de comparer la solution obtenue par le recuit simulé, la solution optimale et les borne inférieures. Nous terminons ce chapitre par une conclusion.

V.2. Problème de conflit

Le calcul de la solution optimale pour les problèmes de moyenne et grande taille exige un temps de calcul très élevé, d'où la nécessité d'utiliser des heuristiques ou des méta-heuristiques utilisant souvent la notion de voisinage. Ces heuristiques appartiennent aux méthodes d'amélioration, c'est-à-dire qu'à partir d'une solution réalisable, nous calculons une nouvelle solution en perturbant la solution courante.

L'introduction de la nouvelle contrainte de blocage de type *RCb* rend les applications des modèles d'ordonnancement classiques très difficiles. En particulier, la perturbation de la solution courante avec la présence de la combinaison des deux contraintes (de précédence et de blocage) conduit souvent à des situations de blocage de tout le système. Ces situations sont appelées dans la littérature « situation d'inter-blocage » ou situation de conflit. Dans ce cas, plus aucune opération ne peut être traitée.

Dans cette section, nous présentons des conditions mathématiques qui nous permettent de détecter les situations de conflit et la méthode utilisée pour résoudre ce problème.

V.2.1. Définition du conflit

Dans ce paragraphe, nous donnons une définition de la notion de conflit. Pour cela, nous commençons par définir les différents états de la machine.

- Machine occupée : dans ce cas, la machine est indisponible car elle est en cours d'exécution d'une opération.
- Machine bloquée : la machine est indisponible, mais elle n'est pas en cours d'exécution d'une opération. Elle est bloquée par un job jusqu'à la fin de traitement de l'opération suivante dans la gamme opératoire du job ou le début d'exécution de la

deuxième opération qui suit l'opération considérée si elles existent (contrainte de blocage de type *RCb*).

- Machine en attente : la machine est disponible, mais l'opération qui doit être exécutée sur la machine considérée, ne peut pas débuter car les opérations précédentes dans la gamme opératoire du job ne sont pas terminées.

On dit qu'il y a un conflit si aucune opération ne peut avancer à cause des deux raisons suivantes :

- Blocage d'un ensemble de machines.
- Les opérations qui doivent être exécutées pour libérer une machine sont affectées à des machines bloquées.

On peut expliquer ces situations à travers l'exemple suivant.

Exemple 1 : On considère un problème de type job-shop avec contrainte de blocage *RCb* composé de 4 jobs et 13 machines dont les gammes opératoires sont les suivantes :

- J_1: (M_1, O_{11}), (M_5, O_{12}), (M_6, O_{13}), (M_2, O_{14}).
- J_2: (M_7, O_{21}) (M_8, O_{22}) , (M_9, O_{23}), (M_{10}, O_{24}).
- J_3: (M_{11}, O_{21}) (M_{12}, O_{22}) , (M_2, O_{23}), (M_3, O_{25}).
- J_4: (M_3, O_{31}), (M_{13}, O_{32}), (M_4, O_{34}).
- J_4: (M_4, O_{41}), (M_{14}, O_{42}) , (M_{15}, O_{43}), (M_1, O_{44}).

On considère l'ordonnancement réalisable donné par les séquences suivantes :

M_1: J_1 (O_{11}), J_5 (O_{54}) M_2: J_1 (O_{14}), J_3 (O_{33})

M_3: J_3 (O_{34}), J_4 (O_{41}) M_4: J_4 (O_{43}), J_5 (O_{51})

Les autres machines ne réalisent qu'un seul job.

Le graphe conjonctif de cette solution réalisable est donné par la figure 17.

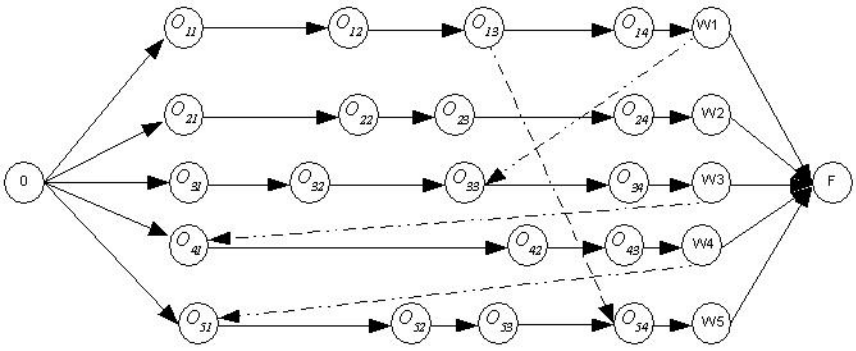

Figure 17 Graphe conjonctif pour un problème avec blocage *RCb* sans conflit

Nous constatons que le graphe conjonctif pour cette solution réalisable ne possède aucun cycle (il n'y a pas de conflit). Si on réalise une permutation sur la machine M_1 de la façon suivante : M_1: J_5 (O_{54}), J_1 (O_{11}) on obtient la solution représentée par la figure 18.

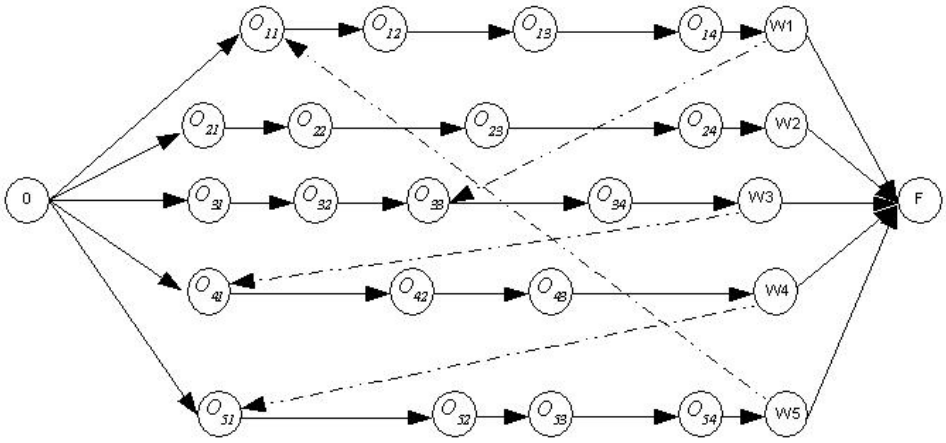

Figure 18 Graphe conjonctif pour un problème avec blocage *RCb* avec conflit

Dans cet exemple, nous constatons qu'il y a un cycle composé des opérations suivantes :

$$\{O_{11}, O_{12}, O_{13}, O_{14}, w_1, O_{33}, O_{34}, w_3, O_{41}, O_{42}, O_{43}, w_4, O_{51}, O_{52}, O_{53}, O_{54}, w_5\}.$$

Les jobs sur lesquels le conflit intervient sont J_1, J_3, J_4 puis J_5.

Nous remarquons aussi que cet état de conflit est le résultat d'une simple permutation de deux opérations à partir d'une solution réalisable ce qui nous pose problème lors de l'application des méta-heuristiques.

Dans la partie suivante, nous présentons des conditions mathématiques qui nous permettent de détecter les situations de conflit afin de les corriger. Nous commençons par donner les notations utilisées dans cette partie.

Soit une opération O_{ij} appartenant à un cycle. Dans ce cycle, cette opération peut être notée comme suit :

- Ci : numéro du i^{eme} job qui contient des opérations appartenant au cycle.
- $O_{ci,fci}$: une opération appartenant au cycle pour le job Ci. $O_{ci,fci}$ peut être w_i.
- $O_{ci,hci}$: une opération du job Ci appartenant au cycle et tel que : $1 \leq hci \leq fCi$.

Suite de l'exemple 1.

À partir de l'exemple précédent, nous avons le cycle composé des opérations suivantes :

$$\{O_{11}, O_{12}, O_{13}, O_{14}, w_1, O_{33}, O_{34}, w_3, O_{41}, O_{42}, O_{43}, w_4, O_{51}, O_{52}, O_{53}, O_{54}, w_5\}.$$

En appliquant la notation présentée ci-dessus, ces opérations peuvent être notées comme suit :

$$\{O_{c1,1}, O_{c1,2}, O_{c1,3}\, O_{c1,4}, O_{c1,5}, O_{c2,3}, O_{c2,4}, O_{c2,5}, O_{c3,1}, O_{c3,2}, O_{c3,3}, O_{c3,4}, O_{c4,1}, O_{c4,2}, O_{c4,3}, O_{c4,4,}\, O_{c4,5}\}.$$

Une condition de blocage est donnée par le théorème suivant :

Théorème 1 : On considère un problème de type job-shop avec contrainte de blocage de type *RCb*. Une condition nécessaire et suffisante pour avoir un conflit est donnée par : Nous avons un conflit si et seulement si il existe un ensemble de m_{clt} machines telles que

$$\forall i \in \{1, \dots, m_{clt} - 1\}, \exists\, O_{ci,hci}\ et\ O_{ci,fci} \,/\, M_{c(i+1),hc(i+1)} = M_{ci,fci}$$

$$et\ \ M_{c(mclt),fc(mclt)} = M_{c(1),hc1}$$

Dans ce cas, le cycle créé est le suivant :

$$(O_{C1,hc1}, O_{C1,(hc1)+1}, \dots., O_{C1,fc1}, O_{C2,hc2}, O_{C2,(hc2)+1}, \dots., O_{C2,fc2}, \dots, O_{Cmclt,hCmclt}, \dots., O_{Cmclt,fCmclt}, O_{C1,hc1})$$

Démonstration du Théorème 1 :

Cette démonstration se divise en deux parties. Nous commençons par prouver que pour avoir un conflit la condition donnée dans le théorème 1 est nécessaire puis nous passons à la deuxième partie dans laquelle nous prouvons que la condition donnée dans le théorème 1 est suffisante.

- **Condition nécessaire : dans cette partie nous montrons que si nous avons un conflit alors la condition 1 est vérifiée.**

Si nous avons un conflit, alors aucune opération ne peut être exécutée, c'est-à-dire :

(i) Soit la machine qui doit exécuter l'opération O_{zj} n'a pas terminé l'opération précédente sur cette machine.

(ii) Soit parce que le job qui doit passer sur la machine n'a pas encore fini son opération précédente dans la gamme.

Considérons une opération O_{zj} qui ne peut pas débuter sur la machine M_k.

Si elle se trouve dans le cas (ii), alors la machine est bloquée par l'opération précédente dans la gamme. Ce cas correspond à un arc en trait plein dans notre exemple. On a donc un arc

(O_{zj-1} O_{zj}).

Si la machine se trouve dans le cas (i), l'opération O_{zj} attend la fin d'une opération $O_{l,s}$ d'une autre gamme opératoire. Dans ce cas, pour avoir un arc conjonctif (traits en pointillés sur l'exemple), il faut que :

- Soit $O_{l,s+1}$, n'existe pas et on a $M_{l,s-1} = M_k$ *ou* $M_{l,s} = M_k$
- Soit $O_{l,s+1}$, existe et on a $M_{l,s-1} = M_k$

Dans ce cas, il existe un chemin orienté qui relie O_{ls} à O_{zj} ou $O_{l,s-1}$ à O_{zj}. La machine M_k est donc ajoutée à l'ensemble des machines considérée pour le blocage.

On peut ensuite considérer l'opération $O_{l,s}$ ou $O_{l,s-1}$ et ainsi de suite.

Comme le nombre d'opérations est fini, on retombera forcément sur une opération déjà visitée et nous aurons donc un ensemble de *mclt* machines en conflit. Nous avons donc bien un cycle dans notre graphe qui peut s'écrire ($O_{C1,hc1}$, $O_{C1,(hc1)+1}$, ……, $O_{C1,fc1}$, $O_{C2,hc2}$, $O_{C2,(hc2)+1}$, ……, $O_{C2,fc2}$, …, $O_{Cmclt,hCmclt}$, ……, $O_{Cmclt,fCmclt}$, $O_{C1,hc1}$).

Si on considère la $i^{ème}$ machine parmi les *mclt* machines appartenant à l'ensemble des machines en conflit et qu'on la note M_k. Cette machine se trouve dans le (i) de blocage. Les opérations qui s'exécutent sur cette machine sont donc O_{zj} et O_{ls} ou $O_{l,s-1}$ avec $M_{zj}=M_{ls}$ ou $M_{zj}=M_{l,s-1}$. Nous avons donc $ci = l$ et $ci+1 = z$. Par conséquent, nous avons bien une opération qui appartient au cycle pour le job *Ci* telle que $fCi \geq hci$ et qui vérifie

$M_{ci,s}=M_{c(i+1),j}$ ou $M_{ci,s-1}=M_{c(i+1),j}$. , c'est-à-dire $M_{c(i+1),hc(i+1)} = M_{ci,fci}$.

Et ainsi de suite pour les autres machines jusqu'à la machine M_{mclt} pour laquelle nous avons l'opération $O_{c(mclt),fc(mclt)}$ qui s'exécute sur la même machine (M_{mclt}) qui est la première opération dans le cycle noté $O_{c1,hc1}$. Donc $M_{c(mclt),fc(mclt)}= M_{c1,hc1}$.

Par conséquent, si nous avons un cycle avec la condition suivante :

$$\forall i \in \{1, \dots, m_{clt} - 1\}, \exists\, O_{ci,hci}\ et\ O_{ci,fci}\ /\ M_{c(i+1),hc(i+1)} = M_{ci,fci}$$

$$et\ \ M_{c(mclt),fc(mclt)} = M_{c(1),hc1}$$

Le cycle crée sera le suivant :

($O_{C1,hc1}$, $O_{C1,(hc1)+1}$, ……, $O_{C1,fc1}$, $O_{C2,hc2}$, $O_{C2,(hc2)+1}$, ……, $O_{C2,fc2}$, …, $O_{Cmclt,hCmclt}$, ……, $O_{Cmclt,fCmclt}$, $O_{C1,hc1}$)

- **Condition suffisante : Dans cette partie, nous montrons que si la condition 1 est vérifiée, alors nous avons un conflit (blocage).**

Si la condition 1 est vérifiée, alors nous avons un circuit ($O_{C1,hc1}$, $O_{C1,(hc1)+1}$, ……, $O_{C1,fc1}$, $O_{C2,hc2}$, $O_{C2,(hc2)+1}$, ……, $O_{C2,fc2}$, …, $O_{Cmclt,hCmclt}$, ……, $O_{Cmclt,fCmclt}$, $O_{C1,hc1}$) dans le graphe conjonctif. Or, il a été démontré qu'un graphe conjonctif qui possède un cycle conduit à une solution non admissible avec blocage.

Par conséquent, si la condition 1 est vérifiée nous avons forcément un conflit (blocage).

CQFD

V.2.2. Résolution du problème de conflit

L'idée principale de la méthode qui nous permet de résoudre le problème de cycle est basée sur le ré-ordonnancement du job J_i afin d'obtenir une solution réalisable en imposant les opérations permutées.

Pour cette méthode, nous avons utilisé les sémaphores pour indiquer l'état de la machine après le placement de chaque opération.

a. Notion des sémaphores

Le sémaphore a été inventé par Edsger Dijkstra et utilisé pour la première fois dans le système d'exploitation (Jones 2005). Le sémaphore est un outil informatique qui permet de contrôler l'accès à des ressources partagées. Ils fournissent une solution à plusieurs problèmes d'inter-blocage tel qu'il existe dans le problème du dîner des philosophes par exemple.

Le sémaphore utilise trois opérations : Prendre, vendre, initialiser.

- L'opération ***Prendre*** permet de prendre une ressource disponible, s'il n'y a pas de ressources disponibles, le sémaphore est en attente jusqu'à disponibilité d'une ressource qui sera immédiatement allouée au processus courant.
- ***Vendre*** permet de libérer une ressource après avoir terminé de l'utiliser.
- L'opération ***Initialiser*** permet d'initialiser le sémaphore. Cette opération ne doit être utilisée qu'une seule et unique fois.

b. Algorithme de correction des solutions non réalisables

Après la permutation de deux opérations, la solution obtenue peut être non réalisable. Pour corriger cette solution, nous avons suivi les étapes suivantes :

- Détection des opérations qui appartiennent au cycle en utilisant la condition présentée ci-dessus.
- Affectation des valeurs du sémaphore (***Prendre-sémaphore et vendre-sémaphore***) à chacune des opérations des différents jobs pour indiquer l'état des machines.
- Ré-ordonnancement : la permutation des opérations peut conduire à un état de conflit, donc l'étape de ré-ordonnancement consiste à faire d'autres permutations des opérations afin de débloquer toutes les machines bloquées (ouverture de cycle).

Exemple 2

On considère un problème d'ordonnancement de type job-shop avec une contrainte de blocage particulière dont les séquences sont données comme suit :

$M_1 : O_{12}, O_{25}, O_{31}, O_{45}, O_{51}$ $\quad$ $M_2 : O_{13}, O_{22}, O_{32}, O_{41}$

$M_3 : O_{11}, O_{21}, O_{43}, O_{53}$ $\quad$ $M_4 : O_{15}, O_{24}, O_{44}, O_{54}$

$M_5 : O_{14}, O_{23}, O_{42}, O_{52}$

La figure 3 représente l'affectation des valeurs de sémaphore sur les machines pour chaque opération de cette solution réalisable.

Après la permutation de l'opération O_{13} sur la machine M_2, nous obtenons la séquence suivante :

$$M_2 : O_{22}, O_{32}, O_{13}, O_{41}$$

Comme l'opération O_{11} prend le sémaphore de la machine M_3, ce sémaphore ne peut pas être vendu ce qui indique une situation de blocage.

Donc cette séquence, nous conduit au cycle donné par : $O_{13}, O_{22}, O_{23}, O_{14}$. Pour corriger cette situation, en appliquant la méthode présentée dans le paragraphe 3, on doit placer l'opération O_{14} après O_{23} sur la machine M_5.

Nous obtenons la séquence sur cette machine M_5 : $\boldsymbol{O_{23}, O_{14},}$ O_{42}, O_{52}

Et ainsi de suite pour les autres opérations. Finalement, nous obtenons une solution réalisable qui est la suivante :

$M_1 : O_{25}, O_{31}, O_{12}, O_{45}, O_{51}$ $\quad$ $M_2 : O_{22}, O_{32}, O_{13}, O_{41}$

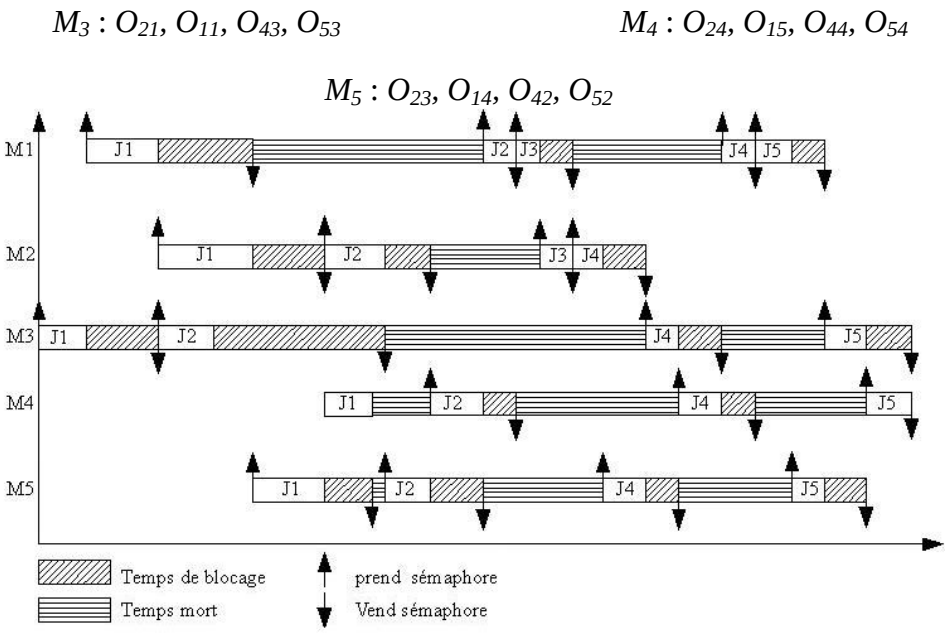

Figure 19. Diagramme de Gantt représentant l'affectation des valeurs de sémaphore

V.3. Heuristiques

Pour avoir très rapidement une première solution admissible, nous avons testé des heuristiques pour résoudre le problème d'ordonnancement décrit précédemment. Nous avons utilisé des heuristiques de liste classiques telles que :

- FIFO (First in first out) : ordonnance les jobs dans l'ordre dans lequel ils apparaissent dans les données.
- LPT (Longuest Processing Time) : le job le plus prioritaire est celui qui a la somme des durées la plus longue.
- SPT (Shortest Processing Time) : le job le plus prioritaire est celui qui a la somme des durées la plus petite.

L'avantage de ces heuristiques est qu'elles nous donnent des solutions réalisables très rapidement, mais ces solutions sont dans la plupart des cas des solutions de mauvaise qualité. Nous avons choisi celle qui nous donne la meilleure solution comme solution de départ pour l'algorithme de recuit simulé présenté par la suite.

V.3.1. Évaluation des performances des heuristiques

Nous avons comparé les trois heuristiques de liste à la solution optimale trouvée par le modèle *PLNE*. Ces trois heuristiques ont été testées sur des problèmes avec 5, 6, 7, 8, 9 et 10 jobs, et

5, 6, 7 et 10 machines. 100 problèmes ont été générés pour chaque configuration. Ces problèmes ont été générés de la façon suivante :

Pour chaque problème comportant *n* jobs et *m* machines, le nombre d'opérations par job est choisi aléatoirement dans l'intervalle [*m*/2, *m*] et les machines sont ensuite choisies aléatoirement. Les temps opératoires sont générés suivant une loi uniforme entre 1 et 9.

Le tableau 4 contient le pourcentage d'erreur entre chacune de ces heuristiques et la solution optimale calculée par le *PLNE*. Ce pourcentage d'erreur noté % *err* entre la solution optimale noté S^* et la solution obtenue par l'une de ces heuristiques noté S_h est calculé comme suit :

$$\%\ err = \frac{S_h - S^*}{S_h} \times 100$$

	5 Machines			6 Machines			7 Machines			10 Machines		
	FIFO	SPT	LPT	FIFO	SPT	LPT	FIFO	SPT	LPT	FIFO	SPT	LPT
5 Jobs	**20,28**	20,43	21,43	27,62	**26,04**	27,1	30,22	**29,35**	31,40	**40,92**	41,59	41,14
6 Jobs	23,81	**24,69**	24,75	**35,1**	35,44	35,92	34,06	**34,04**	34,55	43,1	**42,85**	43,79
7 Jobs	25,73	**24,01**	27,03	34,1	**33,15**	34,66	**25,3**	36,48	35,99	46,81	**46,42**	47,06
8 Jobs	29,99	**28,3**	29,18	37,01	**35,85**	35,98	32,22	**30,84**	31,2	47,61	47,52	**46,65**
9 Jobs	**30,08**	30,4	30,5	38,34	37,08	**36,72**	33,68	32,25	**31,85**	50,31	**49,16**	49,96
10 Jobs	31,89	**31,44**	33,72	38.85	**37,22**	39,03	**41,9**	42,35	42,37	51,08	50,95	**50,74**

Tableau 4. Pourcentage d'erreur entre la solution obtenue par les différentes heuristiques et la solution optimale.

Le tableau 4 nous montre que le pourcentage d'erreur entre les différentes heuristiques et la solution optimale varie très peu d'une heuristique à une autre (de 0 à 2 pourcent maximum pour chaque taille de problème). Ce pourcentage d'erreur varie entre 20 et 40 pourcent pour les problèmes de petite et moyenne taille et de 30 à 51 pourcent pour les problèmes de grande taille. Nous pouvons conclure que ces résultats sont très insuffisants et qu'il est donc indispensable de développer des méta-heuristiques plus performantes.

Dans ce qui suit, nous essayons d'améliorer ces résultats en utilisant une méta-heuristique basée sur le recuit simulé.

V.4. Recuit simulé

Le recuit simulé que nous décrivons dans cette section a été présenté lors de la conférence ETFA 2010.

Le recuit simulé est une méta-heuristique inspirée du processus utilisé en métallurgie. Elle a été mise au point par Kirkpatrick (1983) puis Cerny (1985).

Cette méthode une méthode de recherche globale qui accepte, avec une certaine probabilité, des solutions qui dégradent la solution courante afin d'éviter de rester bloqué sur un optimum local de mauvaise qualité.

Nous présentons dans cette section le principe général de cette méthode pour notre problème, puis nous proposons les différents voisinages que nous avons utilisés.

V.4.1. Algorithme général

Le recuit simulé (RS) est un algorithme itératif stochastique qui progresse vers la solution optimale par échantillonnage d'une fonction objectif. L'application de cette méthode dans des problèmes d'optimisation combinatoire commence par la sélection d'une solution initiale X_0. Cette solution peut être choisie aléatoirement dans l'espace des solutions admissibles, ou construite en utilisant des méthodes constructives. Nous associons à cette solution une énergie initiale $F(X_0)$ calculée par le critère que nous voulons optimiser. Cette solution sera considérée comme la solution courante X_n (avec $n = 0$). Nous sélectionnons ensuite aléatoirement une solution voisine X^* de l'ensemble des voisins de la solution actuelle X_n.

- si $F(X^*) \leq F(X_n)$ alors X^* devient la nouvelle solution courante noté X_{n+1}.
- Sinon nous décidons à l'aide d'une variable de décision générée aléatoirement si X^* devient la nouvelle solution courante.

Cet algorithme reste difficile à adapter aux problèmes d'ordonnancement de type Job-Shop, malgré son efficacité dans le domaine de l'optimisation combinatoire. Cette difficulté est liée aux différents paramètres. Dans ce qui suit, nous adaptons les paramètres de cet algorithme afin de résoudre notre problème d'ordonnancement de type Job-Shop avec la contrainte de blocage de type *RCb*.

L'algorithme du recuit simulé est représenté par la structure suivante :

Algorithme

$S^* \leftarrow S$; (S est la solution initiale) ;

$k \leftarrow 0$; (nombre global d'itérations) ;

$T \leftarrow T_0$ (T_0 : température initiale du système) ;

Tant que ($T > Epsilon$) **faire**

Tant que ($k <$ nombre total d'itérations) **faire**

Générer aléatoirement une solution $S' \in N(S)$; ($N(S)$ est le voisinage de la solution S)

$\Delta F \leftarrow F(S')-F(S)$;

Si ($\Delta F <0$) **alors** $S \leftarrow S'$;

Sinon

Prob($\Delta F,T$) $\leftarrow \exp(-\Delta F/T)$;

Générer U_k uniformément dans l'intervalle [0,1[;

Si ($U_k <$ Prob($\Delta F,T$)) **alors** $S \leftarrow S'$;

Fin si

Si ($F(S)<F(S^*)$) **alors** $S^* \leftarrow S$;

$k = k+1$;

Fin Tant que

$T=\alpha T$; ($0 < \alpha < 1$ coefficient de refroidissement)

Fin Tant que

V.4.2. Solution initiale

L'efficacité de cet algorithme dépend du choix de la solution initiale. Ce choix peut être fait par des règles de priorité en respectant les contraintes du problème, ou en utilisant les algorithmes gloutons. Dans ce travail, nous utilisons des algorithmes de liste pour obtenir la solution initiale.

V.4.3. Voisinages

La fonction de voisinage est obtenue en faisant une mutation d'une opération qui appartient au chemin critique. Cette mutation consiste à choisir aléatoirement une machine et deux

opérations qui appartiennent au chemin critique sur la machine choisie, puis l'opération qui a l'ordre de passage sur la machine considérée le plus petit sera placé juste après la deuxième opération choisie. Cette modification doit assurer la faisabilité de la nouvelle solution. Comme notre problème a deux types de contraintes (contrainte de précédence et contrainte de blocage), le déplacement d'une des opérations peut conduire à un conflit qui a été présenté précédemment. Pour éviter cette situation, nous utilisons le processus de correction présenté dans le paragraphe V.2.2. Cette correction consiste à faire d'autres permutations des opérations qui n'appartiennent pas à la même machine afin d'éviter de retourner à la solution de départ.

V.4.4. Probabilité d'acceptation d'une moins bonne solution

Nous acceptons une solution *S'* qui est moins bonne que la solution courante, à l'aide d'une probabilité *P*. Cette probabilité est une fonction *P(ΔF, T)* dépendant d'une variable *T*, cette variable *T* décroissant avec le temps. La forme de la loi de probabilité *P* est inspirée de la distribution de Boltzmann, la valeur numérique de *P(ΔF, T)* appartient à l'intervalle [0,1].

La loi de probabilité choisie est :

$$P(T, \Delta F) = \exp(\frac{-\Delta F}{T})$$

V.4.5. Paramètres de recuit simulé

Afin d'optimiser la solution obtenue par la méthode de recuit simulé, nous devons fixer les paramètres à utiliser par cette méthode : T_0, α, P_0. Ces paramètres doivent être déterminés expérimentalement en tenant compte la taille du problème.

T_0 est choisie de telle sorte qu'au début d'exécution de l'algorithme de nombreuses solutions soient acceptées. Nous avons choisi :

$$T_0 = \frac{-C_{max0}}{\log(P_0)}$$

Avec :

- C_{max0} : la solution initiale.
- P_0 : est choisie expérimentalement dans l'intervalle [0,1 ; 0,9].

Après chaque pallier de température, nous multiplions la température par le facteur α.

Le critère d'arrêt est lorsque la température atteint la valeur ε (ε est une petite valeur proche de 0).

Dans la partie suivante, nous faisons varier la valeur de ces paramètres afin de déterminer les valeurs les plus efficaces.

V.4.6. Résultats expérimentaux

Nous avons utilisé les exemples numériques présentés au chapitre précédent. Dans le tableau 5, nous avons utilisé les paramètres suivants :

- P_0=0.998
- α =0.5
- Nombre d'itérations =250.

Ce tableau nous donne les temps d'exécution pour obtenir la solution optimale S* et la solution approchée RS obtenue avec l'algorithme du recuit simulé, ainsi que le pourcentage d'erreur entre la solution optimale et la solution obtenue par l'algorithme de recuit simulé.

Jobs	Machines					
	3			5		
	Temps d'exécution (en min)		% erreur	Temps d'exécution (en min)		% erreur
	S*	RS		S*	RS	
5	0,577	0,812	0	0,715	43,368	10,507
6	0,795	0,949	0,246	1,32	57,74	13,126
7	2,61	1,5	1,269	3,72	67,02	15,52
8	13,45	1,66	2,92	22,396	92,379	15,701
9	103,063	9,475	2,174	131,27	119,97	17,94
10	1298,86	10,4	3,772	456,48	232,56	19,57

Tableau 5. Temps de calcul et pourcentage d'erreur entre la solution optimale et la solution obtenue par le recuit simulé pour les problèmes de type job-shop avec *RCb.*

Dans le tableau 5, nous pouvons constater que le pourcentage d'erreur est très réduit pour les problèmes à trois machines, et il varie entre 10,5 et 19,5 pour les problèmes à 5 machines.

Pour les problèmes de grande taille, nous avons essayé de déterminer les paramètres optimaux. Pour cela, nous avons fait varier les différents paramètres de l'algorithme de recuit

simulé afin de trouver des paramètres qui nous donnent le meilleur compromis entre le temps d'exécution et la meilleure solution obtenue par le recuit simulé.

Pour un problème avec 10 jobs et 10 machines, nous avons fait varier les trois paramètres suivants : nombre d'itérations, P_0 et α. Pour le tableau 6, les paramètres de départ sont P_0 = 0,7 et α = 0,8 et nous avons fait varier le nombre d'itérations (250, 500 et 1000). Dans ce tableau, nous avons le pourcentage d'erreur de la solution obtenue par recuit simulé par rapport à la solution optimale, et le temps d'exécution de l'algorithme du recuit simulé. Comme prévu, le temps de calcul augmente avec le nombre d'itérations, mais les résultats sont également améliorés de (3%).

Nombres d'itérations	Temps de calcul (en min)	% erreur
250	19,342	25,183
500	42,291	25,04
1000	76,106	22,086

Tableau 6. Temps de calcul et pourcentage d'erreur entre la solution optimale et la solution obtenue par le recuit simulé pour les problèmes de type job-shop (10 jobs, 10 machines) avec *RCb* pour différentes valeurs du nombre d'itérations.

Pour le nombre d'itérations égal à 1000, nous avons fait varier le paramètre P_0, ce qui nous permet d'accepter un plus grand nombre de solutions. Nous avons fait également varier le paramètre α qui nous permet de faire varier le nombre de palier de température. Dans le tableau 7, nous présentons les résultats obtenus pour les problèmes de grande taille (10 jobs et 10 machines) avec une valeur différente de paramètres P_0 et α.

α	P_0			
	0,6	0,8	0,9	0,998
0,7	23,44 (55,2)	23,26 (59,36)	23,17 (62,19)	22,77 (73,36)
0,8	22,768 (73,11)	22,31 (80,16)	22,32 (149,31)	22,07 (115,32)
0,9	22,98 (73.47)	22,53 (83,52)	22,15 (163,31)	**21,82** (215,24)

Tableau 7. Temps de calcul et pourcentage d'erreur entre la solution optimale et la solution obtenue par le recuit simulé pour les problèmes de type job-shop avec *RCb* pour différentes valeurs de P_0, α.

La meilleure solution est obtenue avec les paramètres α = 0,9 et P_0 = 0,998, mais le temps de calcul est important. Il est donc nécessaire de faire un compromis entre la qualité de la solution et le temps de calcul.

V.5. Conclusion

Dans ce chapitre, nous avons présenté une condition qui nous permet de détecter les inter-blocages ainsi qu'une méthode pour corriger les solutions non réalisables.

Nous avons également présenté des heuristiques de listes qui sont utilisées comme solutions initiales pour l'algorithme de recuit simulé.

Les résultats de recuit simulé montrent une excellente performance pour des problèmes à petit nombre de machine (3 machines, de 5 à 10 jobs), où la valeur optimale est trouvée presque dans tous les cas. Par contre, les résultats pour des instances de tailles plus grandes sont moins performants.

Dans le chapitre suivant, nous abordons le problème de Job-Shop hybride avec des machines identiques parallèles.

CHAPITRE VI **JOB-SHOP HYBRIDE AVEC LA CONTRAINTE *RCB***

VI.1. Introduction

Ce chapitre concerne des systèmes de production de type job-shop hybride (aussi appelés job-shop flexibles) soumis à la contrainte de blocage particulière de type *RCb*. La résolution de ce problème est plus difficile que les problèmes de type job-shop classique car, en plus du séquencement des jobs sur les machines, il faut affecter les tâches sur les différentes machines de chaque étage. Nous nous concentrons dans cette partie sur les problèmes comportant des machines parallèles identiques par étage.

Ce chapitre est composé de trois parties. La première contient une description des problèmes de job-shop hybride avec la contrainte *RCb*. Dans la deuxième partie, nous présentons un modèle mathématique basé sur la programmation linéaire en nombres entiers (PLNE). La

troisième partie est réservée à la présentation de quatre bornes inférieures que nous avons proposées pour le problème d'ordonnancement de type job-shop hybride soumis à la contrainte de blocage de type *RCb*.

VI.2. Description du problème

Le problème d'ordonnancement de type job-shop hybride noté *JSH* est défini de la manière suivante :

Un ensemble de n jobs doit être exécuté sur m étages. Sur chaque étage r, il existe m_r machines identiques. Chaque job J_i est composé d'un ensemble de n_i opérations. Chaque opération ne peut s'exécuter que sur une seule machine et sans interruption, chaque machine ne peut exécuter qu'une seule opération à la fois.

L'application de la contrainte de blocage de type *RCb* impose que chaque opération O_{ij} retient la machine sur laquelle elle est exécutée jusqu'à la fin du traitement de l'opération qui suit O_{ij} (i.e. O_{ij+1}) et le début de l'opération qui suit O_{ij+1} (i.e. O_{ij+2}) si elles existent.

Le modèle de job-shop hybride fournit une représentation générique pour résoudre un nombre important des problèmes rencontrés dans les systèmes de production. Ce problème est une extension de deux problèmes d'ordonnancement :

- le problème de job-shop
- le problème d'ordonnancement de machines parallèles.

Ce problème peut donc être vu comme un problème de job-Shop qui possède une ou plusieurs machines parallèles sur chaque étage et qui peuvent effectuer les mêmes opérations. Dans notre cas, nous considérons que les machines parallèles sur chaque étage sont identiques.

Dans ce qui suit, nous allons présenter un modèle mathématique basé sur la programmation linéaire en nombres entiers (PLNE) ainsi que plusieurs bornes inférieures pour résoudre ce problème.

VI.3. Méthode de résolution exacte

Dans cette partie, nous présentons un modèle mathématique pour le problème du Job-Shop hybride soumis à la contrainte de blocage *RCb* pour des problèmes de tailles n jobs et m étages.

Le problème est de déterminer la meilleure affectation des opérations sur les machines de chaque étage, ainsi que le meilleur ordonnancement des opérations affectées à chaque machine afin de minimiser la date de fin de l'ordonnancement (minimisation du makespan).

VI.3.1. Paramètres

Les différents paramètres et données utilisés pour la formulation mathématique du problème *JSH* sont :

- n_i nombres d'opérations du job *i*.
- $n_{op} = \sum_{i=1}^{n} n_i$: Nombre total d'opérations.
- $O_i = \{O_{i1}, O_{i2}, \ldots, O_{ini}\}$: gamme opératoire du job J_i.
- m : nombre d'étages.
- m_r : nombre d'exemplaires de machines identiques à l'étage *r*.
- M_R : ensemble des machines tout type confondu.
- γ : constante positive très grande, dans notre travail, nous avons choisi :

$$\gamma = \sum_{i=1}^{n} \sum_{j=1}^{n_i} P_{ij}.$$

- $\delta_{ij}^0 = \begin{cases} 1 & \text{si } j \le n_i - 2 \\ 0 & \text{sinon} \end{cases}$: signifie que l'opération O_{ij} n'est ni la dernière ni l'avant dernière opération de la gamme opératoire du job *i*.
- $\delta_{ij}^1 = \begin{cases} 1 & \text{si } j = n_i - 1 \\ 0 & \text{sinon} \end{cases}$: signifie que l'opération O_{ij} est l'avant dernière opération de la gamme opératoire du job *i*.
- $\delta_{ij}^2 = \begin{cases} 1 & \text{si } j = n_i \\ 0 & \text{sinon} \end{cases}$: signifie que l'opération O_{ij} est la dernière opération de la gamme opératoire du job *i*.

VI.3.2. Variables de décision

Les variables de décision que nous avons utilisés sont :

- S_{ij} = date de début d'exécution de l'opération O_{ij} sur une machine de type M_{ij}.
- $U(i,j,k,r) = \begin{cases} 1 \; si \; l'opération \; O_{i,j} \; s'exécute \; sur \; la \; ressource \; r \; de \; l'étage \; k \\ 0 \; si \; non \end{cases}$
- $Y(ij,i'j') = \begin{cases} 1 \; si \; l'opération \; O_{i,j} \; précède \; O_{i',j'} \\ 0 \; si \; non \end{cases}$

VI.3.3. Modèle mathématique

En utilisant les variables présentées dans les paragraphes précédents, le problème peut être modélisé comme suit :

$$\min C_{max} \tag{1}$$

Sous les contraintes suivantes :

$$S_{ij} \geq S_{ij-1} + P_{ij-1} \qquad \forall i = 1..n, \forall j = 2..n_i \tag{2}$$

$$S_{ij} \geq 0 \qquad \forall i = 1..n, \forall j = 1..n_i \tag{3}$$

$$C_{max} \geq S_{in_i} + P_{in_i} \qquad \forall i = 1..n \tag{4}$$

$$\begin{cases} S_{ij} \geq \delta^0_{i'j'} \times S_{i'j'+2} \\ +\delta^1_{i'j'} \times \left(S_{i'(j'+1)} + P_{i'(j'+1)}\right) + \delta^1_{i'j'} \times \left(S_{i'j'} + P_{i'j'}\right) \\ -\gamma \times \left[2 - \left(U(i,j,k,r) + U(i',j',k,r)\right) + Y(ij,i'j')\right] \\ S_{i'j'} \geq \delta^0_{ij} \times S_{i(j+2)} \\ +\delta^1_{ij} \times \left(S_{i(j+1)} + P_{i(j+1)}\right) + \delta^1_{ij} \times \left(S_{ij} + P_{ij}\right) \\ -\gamma \times \left[3 - \left(U(i,j,k,r) + U(i',j',k,r) + Y(ij,i'j')\right)\right] \end{cases} \quad \begin{matrix} \forall i=1..n, i'=1..n, \\ \forall j=1..n_i, j'=1..n_{i'} \end{matrix} \tag{5}$$

$$\sum_{k=1}^{m_r} U(i,j,k,r) = 1 \;\; \forall\, i = 1..n, \forall j = 1..n_i, \forall r \tag{6}$$

$U(i,j,k,r)$ est binaire $\forall\, i = 1..n, \forall j = 1..n_i, \forall k \in M_R, \forall r$ (7)

$Y\left(O_{ij}, O_{i'j'}\right)$ est binaire (8)

VI.3.4. Signification des équations

Pour le modèle mathématique qui représente le problème de *JSH* donné dans le paragraphe précédent, nous pouvons donner la signification de chaque équation comme suit :

- L'équation 1 : critère à minimiser il s'agit de la minimisation du makespan.
- Contrainte 2 : aucune opération ne peut commencer avant la fin d'exécution de l'opération qui la précède dans la gamme opératoire du job. Cette contrainte assure la contrainte de précédence entre les opérations d'un même job.
- Contrainte 3 : c'est une contrainte de causalité, la date de début de chaque opération doit être positive ou nulle.
- Contrainte 4 : cette contrainte indique que le makespan doit être supérieur ou égal à la date de fin de toutes les opérations.

- Les contraintes 5 : ces contraintes expriment le partage des ressources entre les opérations qui utilisent la même machine à l'aide des variables *Y* et *U*. Ceci permet d'assurer que la contrainte de blocage de type *RCb* est bien respectée.
- Contrainte 6 : une opération ne peut être exécutée que sur une seule ressource.
- Contrainte 7 : $U(i,j,k,r)$ est une variable binaire.
- Contrainte 8 : $Y(ij, i'j')$ est une variable binaire.

VI.3.5. Résultats expérimentaux

Dans cette partie, nous présentons les résultats expérimentaux pour le problème du job-shop hybride avec contrainte de blocage de type *RCb*.

Pour un problème de taille $n \times m$ (n jobs et m étages), nous avons généré 50 instances différentes. Chacune de ces instances est générée de la même manière que le problème du job-shop classique présenté dans le chapitre 3 :

- Le nombre d'opérations par job est choisi aléatoirement dans l'intervalle [$m/2$, m].
- Les gammes opératoires ont été générées aléatoirement.
- Les durées d'exécution des opérations ont été générées uniformément dans l'intervalle] 0, 9].
- Le nombre d'exemplaires de machines par étage est fixé à 3 puis à 5.

La solution optimale pour ces problèmes a été obtenue par le logiciel d'optimisation Mosel Xpress. Tous les calculs ont été réalisés sur un PC Intel Pentium (R) 3GHz, 1Go de RAM.

a. Cas où chaque machine existe en trois exemplaires

Le tableau suivant présente le temps de calcul moyen nécessaire pour obtenir la solution optimale en utilisant le modèle mathématique de programmation linéaire en nombre entier pour le problème de JSH avec 3 machines identiques pour chaque étage avec la contrainte de blocage de type *RCb*.

Jobs	Nombre d'étages					
	5	6	7	10	15	20
5	0,087	0,21	0,07	20,02	0,17	0,21
6	0,62	3,10	1,12	397,60	0,33	0,35
7	4,34	579,79	1368,98	525,60	0,57	0,69
8	118,24	10826,80	11694,88	3038,96	*341,6 (1,43)	*209,5(1,13)
9	14959,44	38089,85	38777,89	2430,08	2,88	6,354

Tableau 8. Temps de calcul moyen (en seconde) de la solution optimale dans le cas où chaque machine existe en trois exemplaires.

D'après le tableau 8 nous pouvons déduire que les problèmes jusqu'à 7 jobs peuvent être résolus en un temps raisonnable. Nous pouvons également constater que le temps de calcul augmente avec l'augmentation de nombre de jobs. Les problèmes à 8 jobs et 15 machines possèdent un seul problème très difficile pour lequel le temps de calcul est de 34021,7 secondes, ce qui nous donne un temps moyen de 341,6 secondes. En ignorant ce problème le temps de calcul moyen devient 1,43 secondes. Idem pour les problèmes à 8 jobs et 20 machines qui possèdent un seul problème difficile qui nécessite un temps de calcul de 20838,7 secondes, ce qui nous donne un temps de calcul moyen de 209,5 secondes. En supprimant ce problème, nous obtenons un temps de calcul de 1,13 secondes. Par conséquent, les problèmes à 15 et 20 étages restent des problèmes en général faciles même pour les problèmes à 9 jobs.

b. Cas où chaque machine existe en cinq exemplaires

Le tableau 9 présente le temps de calcul moyen afin d'obtenir la solution optimale pour le même problème que celui présenté dans le paragraphe précédent mais avec 5 machines identiques par étage.

Jobs	Nombre d'étages					
	5	6	7	10	15	20
5	0,04	0,07	0,07	0,15	0,19	0,24
6	0,09	0,13	0,13	0,27	0,33	0,35
7	0,10	0,19	0,21	0,41	0,49	0,54
8	0,16	0,27	0,29	0,55	0,71	0,73
9	1246,94	1672,21	0,43	0,81	1,01	1,10

Tableau 9 Temps de calcul moyen de la solution optimale dans le cas où chaque machine existe en cinq exemplaires pour chaque machine.

Les conclusions qui se dégagent du tableau 9 sont analogues à celles mises en évidence dans le cas du job-shop avec 3 exemplaires par machine, sauf que dans ce cas à 5 nous pouvons trouver la solution optimale en un temps raisonnable pour les problèmes jusqu'à 8 jobs au lieu de 7 jobs pour les problèmes précédents. Même avec 9 jobs, les temps de calcul ne sont pas trop importants. Ceci peut s'expliquer par le fait que plus on a de machines identiques pour traiter des jobs à chaque étages, plus il est facile de le faire.

VI.4. Bornes inférieures

Dans cette partie, nous présentons quatre bornes inférieures pour résoudre le problème du job-shop hybride avec la contrainte de blocage de type *RCb*. La première borne, notée *Binf1* est la

borne classique qui concerne le temps total d'exécution des jobs, la deuxième borne, notée *Binf2*, est liée au temps d'occupation des machines et la troisième borne noté *Binf3* est une amélioration de la borne inférieure donnée par (Derbala et *al* 2006***)*** pour les machines parallèles et la quatrième borne inférieure est le maximum des deux bornes *Binf2* et *Binf3*.

VI.4.1. Notations

Les notations utilisées dans cette section sont les suivantes :

- nop_r: Nombre d'opérations à effectuer sur l'étage *r*.
- m_r : Nombre de ressources sur l'étage *r*.
- $E(O_{ij})$ = Etage sur lequel est réalisée l'opération O_{ij}.
- $\lambda_r = \left\lceil \frac{nop_r}{m_r} \right\rceil$ est le nombre minimal d'opérations qui doivent s'effectuer sur au moins une des ressources de l'étage *r*. Par exemple, si il y a 3 ressources à l'étage *r* et que 10 opérations doivent être effectuées sur cet étage alors au moins 4 des opérations devront être exécutées sur une même ressource.
- T_occ_{ij} : temps d'occupation des ressources par l'opération O_{ij}. C'est la somme de durées d'exécution des opérations et des temps de blocage.

$$T\_occ_{ij} = \sum_{s=j}^{\min(n_i, j+1)} P_{is}$$

- T_prc_{ij} : Temps minimum nécessaire pour pouvoir débuter l'opération O_{ij}. C'est la somme des temps d'exécutions des opérations qui précèdent l'opération O_{ij} dans la gamme opératoire du job *i*.

$$T\_prc_{ij} = \sum_{s=1}^{j-1} P_{is}$$

- T_suiv_{ij} : Temps minimum nécessaire pour le terminer l'exécution du job *i*.

$$T\_suiv_{ij} = \sum_{s=j+2}^{n_i} P_{is}$$

$Tocc_r$: Somme des λ_r premiers T_occ_{ij} classés dans l'ordre croissant sur l'étage *r*.

Le temps d'exécution de chaque job (la somme des durées d'exécution de toutes les opérations d'un job) peut donc être décomposé en trois temps :

- Le temps minimum nécessaire pour le début d'une opération O_{ij}, appelé T_prc_{ij}.

- Le temps minimum nécessaire pour terminer un job après l'exécution d'une opération O_{ij} et de l'opération qui suit, appelé T_suiv_{ij}.
- La somme des temps d'exécution de l'opération O_{ij} et de l'opération suivante si elle existe, appelé T_occ_{ij}.

VI.4.2. Borne inférieure 1

La première borne inférieure, que nous appellerons *Binf1*, est la borne classique basée sur le temps maximal d'exécution des jobs.

Soit P_i le temps total d'exécution d'un job *i* :

$$P_i = \sum_{s=1}^{n_i} P_{is}$$

La borne inférieure *Binf1* est donnée par :

$$Binf1 = \max_{i=1\dots n}[P_i].$$

VI.4.3. Borne inférieure 2

La deuxième borne inférieure, que nous appellerons *Binf2*, est basée sur le temps maximal d'occupation des machines. Cette borne est donnée dans le théorème suivant.

Théorème 2 : Soit un problème de Job-Shop hybride composé de *n* jobs, *m* étages, chaque étage comportant m_r machines identiques parallèles, et soumis à la contrainte de blocage de type *RCb*. Une borne inférieure du makespan pour ce problème est donnée par :

$$Binf2 = \max_{r=1,\dots m}\left[Tocc_r + \min_{\substack{i=1\dots n \\ /\exists O_{il}\in O_i \\ \text{et } E(O_{il})=\text{r}}} T\_prc_{il} + \min_{\substack{i=1\dots n \\ /\exists O_{il}\in O_i \\ \text{et } E(O_{il})=\text{r}}} T\_suiv_{il}\right]$$

Preuve :

Comme nous avons précisé dans le chapitre 3, chaque machine peut avoir trois états :

- Machine disponible.
- Machine en cours d'exécution d'une opération.
- Machine bloquée par un job.

Chaque étage *r* est composé de m_r ressources. On considère l'étage *r*.

Sur au moins une machine de l'étage *r*, nous devons exécuter au moins λ_r opérations. Nous notons M_k cette machine. Par conséquent, le temps total d'exécution des opérations sur cet étage plus le temps de blocage est supérieur ou égal à un temps *T*1 où

$T1 = Tocc_r$.

Une opération O_{ij} qui assure la contrainte E(O_{ij}) = *r* (c'est-à-dire qui passe sur l'étage *r*) ne peut démarrer sur une machine de l'étage *r* que si toutes les opérations qui précèdent O_{ij} dans la gamme de J_i sont terminées. Par conséquent, pour démarrer l'opération O_{ij} sur une machine de l'étage *r*, il faut attendre un temps :

$$T\_prc_{ij} = \sum_{s=1}^{l-1} P_{is}$$

Si on considère toutes les opérations qui assure la contrainte E(O_{ij}) = *r*, on aura :

$$T2 = \min_{\substack{i=1..n \\ /\exists O_{il} \in O_i \\ et\ E(O_{il})=\mathrm{r}}} T\_prc_{il}$$

Enfin, un job J_i ne peut être terminé avant la fin du traitement de toutes les opérations O_i. Par conséquent, si O_{il} est réalisée sur M_k, le job n'est pas fini avant la fin des exécutions des opérations O_{il} et $O_{i(l+1)}$. Il faut donc attendre au moins le temps *T*3 avant la fin de l'ordonnancement où le temps *T*3 est donné par :

$$T3 = \min_{\substack{i=1\ldots n \\ /\exists O_{il} \in O_i \\ \text{et } E(O_{il})=\mathrm{r}}} \sum_{s=l+2}^{n_i} P_{is}$$

En conclusion, C_{max} est supérieur ou égal à *T*1+*T*2+*T*3.

Si on considère tous les étages, on obtient :

$$C_{max} \geq \max_{r=1,\ldots m} \left[Tocc_r + \min_{\substack{i=1\ldots n \\ /\exists O_{il} \in O_i \\ \text{et } E(O_{il})=\mathrm{r}}} \sum_{s=1}^{l-1} P_{is} + \min_{\substack{i=1\ldots n \\ /\exists O_{il} \in O_i \\ \text{et } E(O_{il})=\mathrm{r}}} \sum_{s=l+2}^{n_i} P_{is} \right]$$

$$C_{max} \geq Binf2 = \max_{r=1,\ldots m} \left[Tocc_r + \min_{\substack{i=1\ldots n \\ /\exists O_{il} \in O_i \\ \text{et } E(O_{il})=\mathrm{r}}} T\_prc_{il} + \min_{\substack{i=1\ldots n \\ /\exists O_{il} \in O_i \\ \text{et } E(O_{il})=\mathrm{r}}} T\_suiv_{il} \right]$$

CQFD

Exemple : On considère un problème d'ordonnancement de type job-shop hybride constitué de 5 jobs et de 5 étages et sur chaque étage il existe en 3 exemplaires de la même machine, soumis à la contrainte *RCb* dont les gammes opératoires sont les suivantes :

- J1 = {(E2, 5), (E1, 5), (E5, 8)}.
- J2 = {(E5, 4), (E1, 6), (E4, 4)}.
- J3 = {(E4, 7), (E2, 6), (E1, 3)}.
- J4 = {(E3, 6), (E2, 8), (E4, 6)}.
- J5 = {(E2, 8), (E3, 2)}.

Dans cet exemple, une opération est représentée comme suit : (E_i, P_{ij}), avec E_i l'étage sur lequel l'opération O_{ij} est exécutée, et P_{ij} le temps d'exécution de l'opération O_{ij}.

Nous avons calculé la solution optimale pour cet exemple et elle est égale à 20 ut.

Les opérations passent sur chacun des étages comme suit :

- E1 = {(O12, 5), (O22, 6), (O33, 3)}. ➔$\lambda_1 = 1$
- E2 = {(O11, 5), (O32, 6), (O42, 8), (O51, 8)}. ➔$\lambda_2 = \left\lceil \frac{4}{3} \right\rceil = 2$
- E3 = {(O41, 6), (O52, 2)}. ➔ $\lambda_3 = 1$
- E4 = {(O23, 4), (O31, 7), (O43, 6)}. ➔ $\lambda_4 = 1$
- E5 = {(O13, 8), (O21, 4)}. ➔ $\lambda_5 = 1$

Le tableau suivant nous donne les valeurs de $Tocc_e$, T_prc_{ij}, T_suiv_{ij} ainsi que la $Binf2_r$ (borne inférieure calculée pour l'étage r)

	$Tocc_e$	T_prc_{ij}	T_suiv_{ij}	$Binf2_r$
E1	3	4	0	7
E2	19	0	0	19
E3	2	0	0	2
E4	4	0	0	4
E5	8	0	0	8

Donc la borne inferieure est :

$$Binf2 = \max_{r=1,\dots 5}[Binf2_r] = 19\ ut$$

VI.4.4. Borne inférieure 3

La troisième borne inférieure Binf3 est une amélioration de la borne inférieure donnée par Boumédiène-Merouane Hocine et Derbala (2006) pour les machines parallèles. Cette borne

est basée sur la répartition uniforme de temps d'exécution des opérations sur les ressources identiques sur lesquelles ces opérations sont exécutées.

Théorème 3 : Soit un problème de Job-Shop hybride composé de *n* jobs, *m* étages, chaque étage contient m_r machines parallèles identiques et soumis à la contrainte de blocage de type *RCb*. Une borne inférieure du makespan pour ce problème est donnée par :

$$Binf3 = \max_{r=1,\dots m}\left[\frac{\sum_{E(O_{ij})=r} Tocc_{ij}}{m_r} + \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T\_prc_{il} + \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T\_suiv_{il}\right]$$

Preuve :

On suit le même raisonnement que pour la borne *Binf2*.

Le temps total d'exécution des opérations sur un étage *r* plus le temps de blocage est supérieur ou égal à un temps *T*1 où :

$$T1 = \frac{\sum_{E(O_{ij})=r} Tocc_{ij}}{m_r}$$

Comme pour la borne 2, une opération exécutée sur l'étage r, ne peut démarrer sur une machine de l'étage *r* avant l'instant *T*2 suivant :

$$T2 = \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T\_prc_{il}$$

De la même façon, pour terminer l'exécution d'un job *i* pour lequel nous avons une opération O_{ij} qui s'exécute sur l'étage *r*, il faut attendre au moins un temps *T3* donné par :

$$\mathrm{T3} = \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T\_suiv_{il}$$

En conclusion, C_{max} est supérieur ou égal à *T*1+*T*2+*T*3.

Si on considère tous les étages, on obtient :

$$C_{max} \geq Binf3 = \max_{r=1,\dots m}\left[\frac{\sum_{E(O_{ij})=r} Tocc_{ij}}{m_r} + \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T_{prc_{il}} + \min_{\substack{i=1..n \\ /\exists O_{il}\in O_i \\ et\ E(O_{il})=\mathrm{r}}} T_{suiv_{il}}\right]$$

CQFD.

Exemple : On considère l'exemple précédent, nous calculons les temps d'exécutions pour chaque étage, T_prc_{ij} et T_suiv_{ij} , ainsi que la *Binf3r*

Le tableau suivant nous donne les valeurs de $Tocc_e$, T_prc_{ij}, T_suiv_{ij} ainsi que la *Binf3r*.

	$T_{moyenne}$	T_prc_{ij}	T_suiv_{ij}	*Binf3r*
E1	9	4	0	13
E2	15	0	0	15
E3	6	0	0	6
E4	8	0	0	8
E5	6	0	0	6

Donc la borne inferieure est :

$$Binf3 = \max_{r=1,\dots5}[Binf3_r] = 15\ ut$$

VI.4.5. Borne inférieure 4

Afin de comparer la complémentarité des bornes, nous avons utlisé comme quatrième borne inférieure *Binf4* le maximum entre les deux bornes *Binf2* et *Binf3*.

Donc :

$$Binf4 = \max(Binf2, Binf3)$$

VI.4.6. Résultats numériques

Dans cette partie nous présentons les résultats expérimentaux. Nous donnons les pourcentages d'erreur moyens entre les différentes bornes inférieures et la solution optimale $C_{\max}$ pour les problèmes de *JSH* pour les trois cas suivants :

- Les problèmes à trois machines identiques par étage.
- Les problèmes à cinq machines identiques par étage.
- Les problèmes dont le nombre de machines identiques varie d'un étage à un autre.

Les tests sont effectués sur 50 instances de chaque taille de problème. Le pourcentage d'erreur est calculé de la façon suivante :

$$\%err_i = \frac{C_{max} - Binf_i}{C_{max}} \times 100, i \in \{1,2,3,4\}$$

a. Problèmes à trois machines identiques par étage

Dans cette partie nous présentons les résultats obtenus concernant les quatre bornes inférieures dans le cas où chaque étage contient trois machines identiques. Les tableaux 10 et 11 présentent le pourcentage d'erreur entre les bornes inférieures *Binf1*, *Binf2*, *Binf3*, *Binf4* et la solution optimale.

Jobs	Étages											
	5				6				7			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	1,2	26,9	36,8	23,55	0,519	26,12	35,45	24,11	0,24	25,42	38,81	23,59
6	3,1	30,1	30,5	23,97	1,52	30,2	33,6	25,76	0,53	31,30	35,39	27,38
7	4,3	29,2	26,3	10,64	2.18	36,07	32,03	28,63	1,57	35,54	35,63	30,58
8	4,9	32,2	23,1	10,04	3,53	39,27	25,25	7,53	1,68	39,61	31,76	11,37
9	4,8	34,8	20,1	18,2	5,15	37,79	27,93	26,99	3,28	36 ,11	28,67	26,81

Tableau 10 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec 3 exemplaires de machine pour chaque étage.

Dans le tableau 10 nous pouvons constater que la borne inférieure *Binf1* varie entre 0,24 et 5,15 pour les instances jusqu'à 7 machines, pour ces mêmes instances la borne inférieure *Binf2* varie entre 25,42 et 39,61 et *Binf3* varie entre 20,1 et 38,81et la borne *Binf4* varie entre 7,53 et 30,58. Nous pouvons constater que la borne inférieure *Binf1* est nettement la meilleure dans ce cas.

Pour les problèmes à 5 et 6 jobs, la borne *Binf2* est meilleure que *Binf3*. Par contre pour les problèmes à 7, 8 et 9 jobs, nous remarquons *Binf*3 est nettement meilleure que *Binf*2 sauf pour le cas à 7 jobs et 7 machines. Les bornes 2 et 3 sont donc très complémentaires dans certains cas, mais toujours moins bonnes que la première borne.

Le tableau suivant indique le pourcentage d'erreur entre la solution optimale et les bornes inférieures pour les instances entre 10 et 20 étages.

Jobs	Étages											
	10				15				20			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	0,07	25,03	33,62	23,95	0	17,32	31,39	16,6	0	14,81	25,98	14,42
6	0,29	31,36	37,17	14,57	0	27,27	36,98	13,11	0	22,45	31,77	10,88
7	0,71	36,17	35,81	15,9	0	36,53	39,38	34,92	0	30,71	34,22	29,76
8	0,46	34,95	32,56	8,84	0	38,85	39,92	18,15	0,1	34,07	36,06	31,97
9	0,71	44,34	37,87	37,63	0	40,96	38,32	37,44	0	34,74	34,73	33,14

Tableau 11 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec 3 exemplaires de machine pour chaque étage.

Du tableau 11, nous constatons que pour les instances à 10 étages la borne inférieure *Binf1* est globalement bonne. Elle devient optimale pour les instances à 15 et 20 étages. Cela peut être justifié par le fait que le nombre d'opérations par job est compris entre le nombre d'étages divisé par deux et le nombre d'étages. Donc pour les problèmes à 20 étages nous avons en moyennes 15 opérations par job et si on considère 9 jobs on aura donc 135 opérations ordonnancées sur 20 ressources ce qui nous donne en moyenne de 6 opérations par étages. Comme nous avons trois machines parallèles identiques par étage donc sur chaque machine

deux opérations sont affectées en moyenne, et par conséquent nous trouvons que la solution optimale est proche de la durée du job le plus long (la borne *Binf1*). Les bornes *Binf2* et *Binf3* gardent le même comportement pour les instances à 10 étages, mais au-delà de ces tailles la borne inférieure *Binf2* devient la meilleure dans tous les cas.

b. Problèmes à cinq machines identiques par étages

Le tableau 12 présente le pourcentage d'erreur entre la solution optimale et les bornes inférieures dans le cas du *JSH* avec 5 machines par étage.

Jobs	Étages											
	5				6				7			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	0	0,96	2,81	0 ,32	0	3,04	0.25	0	0	2,59	0,31	0,24
6	0,11	1,61	1,42	0,65	0	3,83	1,13	0 ,53	0	3,43	0	0
7	0	1,09	2,77	0	0	3,47	0.31	0	0	4,18	0.81	0 ,38
8	0	3,37	1.73	0,44	0	2,78	0.63	0.63	0	4,84	0.05	0
9	0,55	3,06	2,58	0,89	0	4,03	0.33	0,16	0	3,78	0	0

Tableau 12 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec 5 machines identique dans chaque étage.

Du tableau 12, on peut constater que la borne inférieure *Binf1* est optimale dans la majorité des cas, par contre le pourcentage d'erreur des deux bornes inférieures *Binf2* et *Binf3* varie entre 0,96 et 4,84 pour *Binf2* et de 0 à 2,81 pour *Binf3*. Pour la borne *Binf4*, le pourcentage d'erreur appartient à l'intervalle [0, 0,89]. On peut donc constater que la *binf4* est nettement meilleure que dans le cas avec 3 machines par étages.

Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* à 5 machines identiques par étage avec nombre d'étages varie entre 10 et 20 est donné dans le tableau 13.

Jobs	Etages											
	10				15				20			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	0	3,07	0	0	0	0,56	0	0	0	0,6	0	0
6	0	3,07	0	0	0	2,13	0	0	0	1,11	0	0
7	0	7,01	0,15	0,15	0	4,73	0	0	0	2,78	0	0
8	0	5,96	0,515	0	0	6,08	0,08	0	0	2,18	0	0
9	0	7,02	0,12	0	0	6,08	0,12	0	0	3,79	0	0

Tableau 13 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec cinq exemplaires de machine pour chaque étage.

Le tableau 13 nous indique que la borne inférieure *Binf1* est optimale dans tous des cas, la borne *Binf3* est globalement bonne. Elle a un pourcentage d'erreur qui ne dépasse pas 0,515

% pour les problèmes à 10 machines et nombre de jobs supérieur à 7 jobs. Un maximum de 0,12% est atteint pour les problèmes à 15 étages et nombre de jobs supérieur à 8 jobs. La borne *Binf3* est optimale pour les autres cas. Par contre, *Binf2* à un pourcentage d'erreur qui varie entre 0,56 et 7,02. Elle est donc moins bonne que *Binf1* et *Binf3*.

c. Problèmes dont le nombre de machines identiques varie d'un étage à l'autre

Dans cette partie, nous avons généré aléatoirement le nombre de machines parallèles pour chaque étage. Le nombre de machines parallèles varie d'un étage à un autre entre 1 et 3 machines. Le pourcentage d'erreur entre la solution optimale et les quatre bornes inférieures sont donnés dans les tableaux 14 et 15.

Jobs	Étages											
	5				6				7			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	41,26	12,05	11,77	11,77	34,98	8,11	8,11	8,11	30,97	9,68	9,68	9,68
6	33,39	15,2	8,04	8,04	47,79	10,47	10,47	10,47	38,38	10,15	10,15	10,15
10	65,24	3,5	3,5	3,5	60,25	16,52	11,04	11,04	55,88	10,12	10,12	10,12

Tableau 14 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec le nombre de machines parallèles variable d'un étage à un autre.

Du tableau 14, on peut déduire que contrairement au cas précédent, la borne inférieure *Binf1* est la plus mauvaise avec un pourcentage d'erreur qui varie entre 30,97 et 47,79 pourcent. Par contre la borne *Binf3* est la meilleure pour les problèmes à 5 étages avec une erreur maximale de 11,77 pourcent. La borne *Binf2* à un pourcentage d'erreur qui varie entre 8,11 et 15,2 pourcent.

Jobs	Étages											
	10				15				20			
	Binf1	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*	*Binf1*	*Binf2*	*Binf3*	*Binf4*
5	27,1	15,9	15,9	15,9	13,62	22,48	22,48	22,48	12,33	14,05	14,05	14,05
6	29,73	14,96	14,96	14,96	18,57	17,95	17,95	17,95	14,34	23,16	23,16	23,16
10	53,33	13	13	13	40,48	23,03	23,03	23,03	36.34	28.8	28.8	28.8

Tableau 15 Le pourcentage d'erreur moyen entre les bornes inférieures et la solution optimale pour le problème de *JSH* avec le nombre de machines parallèles variable d'un étage à un autre.

Pour les problèmes à 10 étages les deux bornes *Binf2* et *Binf3* sont meilleures que la première borne inférieure. Par contre, pour les problèmes à 15 et 20 étages, la première borne devient la meilleure.

d. Comparaison entre les deux bornes inferieures *Binf1* et *Binf4*

Nous avons décidé d'étudier les performances des bornes *Binf4* et *Binf1* pour les problèmes de grande taille. Pour cela, nous avons généré des instances de 15 à 50 jobs et de 5 à 20 machines. Chaque job contient un nombre d'opérations égal au nombre de machines avec une gamme opératoire générée aléatoirement. Nous avons testé les bornes inférieures sur les instances générées, nous avons calculé le pourcentage d'erreur entre les deux bornes de la façon suivante :

$$\%\, err = \frac{Binf4 - Binf1}{Binf4} \times 100$$

La comparaison entre les deux bornes dans le cas du *JSH* à trois machines par étage est présentée dans le tableau 16.

Jobs	Étages					
	5	6	7	10	15	20
15	29,81	22 ,29	11,63	-12,75	-14,16	-66,28
20	47,84	39,01	31,71	9,74	-21,39	-45,81
30	62,36	57,6	52,14	36,89	12,94	-9,96
40	70,8	67,36	63,34	51,41	32,28	13,48
50	76,31	73,55	70,14	60,39	44,40	29,73

Tableau 16 Pourcentage d'erreur entre la première borne inférieure *Binf1* et la quatrième borne inférieure *Binf4* pour les problèmes de *JSH* avec 3 exemplaires de machine pour chaque étage.

Du tableau 16, nous pouvons conclure que la borne inférieure *Binf1* est meilleure que la borne *Binf4* pour les problèmes à 15 jobs et nombre d'étages supérieur à 10, et pour les problèmes à 20 jobs et 15 et 20 étages. Pour tous les autres cas la borne inférieure *Binf4* est la meilleure. Globalement pour les problèmes de taille importante, la borne 4 est la meilleure.

VI.5. Conclusion

Ce chapitre a été consacré à la présentation d'un modèle de programmation linéaire en nombres entiers pour résoudre le problème de job-shop hybride avec la contrainte de blocage de type *RCb.* Ce modèle a été validé avec le logiciel d'optimisation XPress-MP. Les résultats obtenus sur des problèmes de différentes tailles générés aléatoirement montrent que les problèmes jusqu'à 7 jobs sont résolus en un temps raisonnable dans le cas de *JSH* à 3 machines par étage et 8 jobs pour le *JSH* à 5 machines identiques par étage. Pour les

problèmes de plus de 8 jobs pour le cas à 3 machines identiques par étages et plus que 9 jobs dans le deuxième cas, les temps d'exécutions deviennent très importants.

Des bornes inférieures ont été développées pour ce problème. Ces bornes inférieures ont été testées sur différents problèmes (des problèmes à 3, 5 machines identiques par étages puis des problèmes avec un nombre de machines identiques variable pour chaque étage). Nous avons constaté que chacune de ces bornes donne de bons résultats dans des cas bien précis. La borne *Binf1* est globalement bonne pour les problèmes dont le nombre de machines parallèle est fixe pour tous les étages. Par contre, dans le cas où le nombre de machines parallèles est variable d'un étage à un autre, les bornes *Binf2*, *Binf3* sont très complémentaires et par conséquent *Binf4* devient la meilleurs. Pour les problèmes de grande taille (entre 15 et 50 jobs et 3 machines identiques pour chaque étage), la borne *Binf1* est meilleure dans les cas 5 jobs et 10, 15 et 20 étages et 6 jobs et 15 et 20 étages, la borne inférieure *Binf4* est la meilleure dans tous les autres cas.

CHAPITRE VII CONCLUSIONS ET PERSPECTIVES

VII.1. Conclusions

Dans cette thèse, nous nous sommes intéressés aux problèmes d'ordonnancement des ateliers flexibles avec une contrainte de blocage particulière rencontrée dans plusieurs environnements industriels, cette contrainte est de type *RCb* (Release when Completing blocking). Parmi les problèmes d'ordonnancement existants, nous nous sommes intéressés aux problèmes de job-shop et job-shop hybride. Nous avons considéré dans notre travail le cas non-préemptif, c'est-à-dire qu'une fois commencée, l'exécution d'une opération ne peut en aucun cas être interrompue ni pour des raisons de maintenance, ni pour l'exécution d'une autre opération sur la même machine.

Dans ce rapport, nous avons présenté dans un premier temps le cadre dans lequel est réalisée cette thèse. Nous avons abordés notre travail par les définitions des ateliers flexibles à cheminement unique (flow-shop) et à cheminements multiples (job-shop) pour les deux cas classique et hybride. Puis les contraintes de blocage *RSb* et *RCb* ont été présentées. Nous avons cité deux moyens de représentation des ordonnancements, le diagramme de Gantt et le graphe disjonctif.

Dans le deuxième chapitre, nous avons mené une étude bibliographie portant sur les travaux existant concernant les problèmes d'ordonnancement, notamment les problèmes sans contrainte de blocage. Cette étude comprend les différents problèmes d'atelier (à cheminement unique et multiple). Nous avons ensuite présenté dans ce chapitre les problèmes avec contrainte de blocage de type *RSb*. À la fin de ce chapitre, nous avons présenté les quelques travaux existants qui concernent les problèmes avec la contrainte *RCb*.

Nous sommes ensuite passé à la présentation d'un modèle linéaire en nombres entiers utilisé pour trouver la solution optimale des problèmes d'ordonnancement de type job-shop avec la contrainte de blocage particulière *RCb*. Les résultats de calculs de la solution optimale nous ont montré que le temps de calcul pour les problèmes jusqu'à 7 jobs avec nombre de machines supérieur ou égal à 20 machines, ainsi que les problèmes dont le nombre de jobs supérieur à 7 avec un nombre de machines inférieur à 10 machines reste raisonnable (temps de calcul inférieur à 20 minutes). Pour les problèmes de tailles supérieures le temps de calcul devient très important.

Nous avons présenté dans la deuxième partie de ce chapitre des bornes inférieures. La première borne inférieure est basée sur le temps maximal d'occupation des machines, la deuxième borne inférieure est basée sur les jobs. Nous avons constaté que la première borne nous donne de bons résultats pour les problèmes dont le nombre de machines est très réduit. La deuxième borne n'est pas très bonne pour ces mêmes instances. Pour les problèmes dont le nombre de machines est très élevé, la deuxième borne inférieure devient la plus efficace.

Des heuristiques de liste ainsi qu'une méta-heuristique basée sur le recuit simulé pour le problème de job-shop avec contrainte de blocage *RCb* ont été présentées dans le chapitre 4. Nous avons choisi comme solution initiale pour l'algorithme de recuit simulé la meilleure solution obtenue par les heuristiques de liste. La solution voisine est obtenue en permutant deux opérations. Cette permutation nous conduit dans presque tous les cas à une situation dite de conflit. Pour résoudre le problème de conflit, nous avons proposé une condition permettant de détecter l'inerte-blocage (conflit) ainsi qu'une méthode permettant de corriger le conflit.

Pour les problèmes de petites tailles l'erreur entre la solution optimale et la solution obtenue par le recuit simulé est très faible (pour problèmes de 5 et 10 jobs et 3 machines, la solution optimale a été trouvée dans la majorité des cas). Par contre, les résultats ne sont pas très efficaces pour des instances de tailles importantes.

Dans la dernière partie de ce rapport, nous avons proposé un modèle linéaire en nombres entiers pour les problèmes de type job-shop hybride soumis à la contrainte de blocage de type *RCb*. Nous avons également développé trois bornes inférieures, la première basée sur les jobs, la deuxième basée sur le temps maximum d'occupation des machines (étages) et la troisième est basée sur le temps moyen d'occupation de chaque étage.

Des expérimentations ont été réalisées afin de valider le modèle PLNE ainsi que les bornes inférieures. Ces expérimentations nous ont montré que pour les problèmes à trois machines identiques par étages, le temps de calcul reste raisonnable jusqu'à 7 jobs et pour les problèmes à cinq étages le temps de calcul est raisonnable pour les instances jusqu'à 8 jobs. Au-delà de ces tailles, le temps de calcul devient très important.

Les bornes inférieures ont été testées sur les problèmes à trois puis à cinq machines identiques par étages. Les résultats de calcul ont montré que ces bornes ont les mêmes caractéristiques pour les deux cas à trois et cinq machines identiques par étages. La borne *Binf1* est globalement bonne pour les problèmes de petit taille (jusqu'à 9 jobs), les autres bornes inférieures sont la plus efficaces pour les problèmes de grande taille (entre 15 et 50 jobs). Ces bornes sont donc très complémentaires.

VII.2. Perspectives

Pour le problème que nous avons traité, nous proposons quelques axes de recherche concernant les problèmes de job-shop avec contrainte de blocage *RCb* et les job-shop hybrides avec blocage *RCb*.

En ce qui concerne les problèmes de job-shop avec blocage *RCb*, il sera très intéressant de faire une étude de complexité, ainsi qu'une amélioration des résultats de bornes inférieures en faisant une recherche d'autres bornes inférieures plus performantes.
Il faudrait également développer des heuristiques plus performantes afin d'obtenir rapidement une première solution de meilleure qualité. D'autres types de méta-heuristiques peuvent également être développés, comme par exemple un algorithme génétique.

Pour le problème de job-shop hybride avec la contrainte de blocage de type *RCb*, nous proposons un axe de recherche qui consiste à développer des heuristiques, ainsi que d'adapter des méta-heuristiques pour résoudre le problème.

Dans les deux cas, il serait peut être intéressant de développer une nouvelle méthode exacte basée sur une procédure par séparation et évaluation pour essayer de résoudre de façon optimale des problèmes de tailles plus importantes.

Liste des figures

Liste des tableaux

Bibliographie

Aarts, E.H.L. & Laarhoven, V., 1985. Statistical cooling: A general approach to combinatorial optimization problems. *Philips J. Res.*, 40, 4, p. 193-226.

Akers, S.B. & Friedman, J., 1955. A Non-numerical Approach to Production Scheduling Problems. *Operations Research*, 3, p. 429-442.

Alvarez-Valdes, R. et al., 2005. A heuristic to schedule flexible job shop in a glass factory. *European Journal of Operational Research*, 165, 2, p. 525-534.

Anderson, E.J., Glass, C.A. & Potts, C.N., 1997. Machine scheduling, In: Aarts E and Lenstra JK(eds). *Local Search in Combinatorial Optimization, Wiley: Chichester*, p. 361-414.

Applegate, D. and W. Cook (1991). "A Computational Study of the Job-Shop Scheduling Problem." ORSA Journal on Computing **3**(2): 149-156.

Arthanary, T.S. & Ramaswamy, K.G., 1971. An extension of two machine sequencing problem. *Opsearch,*, 8, p. 10-22.

Ashour, S., Chiu, K.Y. & Moore, T.E., 1973. An optimal schedule time of a job-shop like disjunctive graph. *Networks*, 3.

Banaszak, Z.A. & Krogh, B.H., 1990. Deadlock Avoidance in flexible Manufacturing Systems with concurrently Cometing Process flows. *IEEE Transactions on Robotics and automation*, 6, p. 724-734.

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

Boumédiène-Merouane Hocine et Derbala Ali. Les Problèmes d'Ordonnancement à Machines Parallèles de Tâches Dépendantes : une Evaluation de Six Listes et d'un Algorithme Génétique. COSI06, colloque international sur l'optimisation et les systèmes d'information, Alger, Algérie, 11-13 Juin 2006, pp. 279-289.

Brah, S.A. & Hunsucker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51, p. 88-99.

Brooks, G.H. & White, C.R., 1965. An algorithm for finding optimal or near optimal solution to the production scheduling problem. *Journal of Industrial Ingineering*, 16, 1.

Brucker, P., 1988. An efficient algorithm for the job-shop problem with tow jobs. *Computing*, 40, p. 353-359.

Brucker, P., Jurisch, B. & Sievers, B., 1994. A Beanch and Bound Algorithm for the Job-shop Scheduling Problem. *Discrete Applied Mathematic*, 49, p. 109-127.

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

Campbell, H.G. & Dudek, R.A., 1970. A heuristic algorithm for the *n* job *m* machine sequencing problem. *Management Science*, 16, p. B 630-637.

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

Carlier, J. & Pinson, E., 1990. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26, pp. 269-287.

Carlier, J. & Pinson, E., 1989. An Algorithm for Solving the Job-Shop Problem. *Management Science*, 35(2), p. 164-176.

Carlier, Jacques & Chrétienne, P., 1988. *Les problèmes d'ordonnancement: modélisation, complexité, algorithmes*, Masson, Paris, France.

Carlier, Jacques & Rebaï, I., 1996. Two branch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90, p. 238-251.

CERNY, V., 1985. A thermodynamical approach to travelling salesman problem: an efficient simulated annealing algorithm. *Journal of Optimization Theory and Applications*, 41, p. 41-51.

Chan, F., Wong, T. & Chan, L., 2006. Flexible job-shop scheduling problem under resource constraints. *International journal of production research*, 44, 11, p. 2071-2089.

Chen, H., Ihlow, J. & Lehmann, C., 1999. A genetic algorithm for flexible job shop scheduling. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2, p. 1120-1125.

Cho, Y. & Sahni, Sartaj, 1981. Preemptive Scheduling of Independent Jobs with Release and Due Times on Open, Flow and Job Shops. *Operations Research*, 29, p. 511-522.

Conway, R.W., Maxwell, W.L. & Miller, L.W., 1967. *Theory of Scheduling*, Addison Wesley, 1st edn Reading, Mass., USA.

Dannenbring, D.G., 1977. An evaluation of flow-shop sequencing heuristics. *Management Science*, 23, p. 1174-1182.

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

Dauzère-Pérès, S. & Pavageau, C., 1998. Multi-ressource shop scheduling whith ressource flexibility. *European Journal of Operational Research*, 107, p. 289-305.

Dauzère-Pérès, S., Pavageau, C. & Sauer, N., 2000a. Modélisation et résolution par PLNE d'un problème réel d'ordonnancement avec contraintes de blocage. *3éme congrès de la ROADEF*, p. 216-217.

Davis, L., 1985. Job shop scheduling with genetic algorithms. Dans International conference on Genetic Algorithms and their Applications, éd par Grefenstette (J.). Carnegie Mellon University.

Dell'Amico, M. & Trubian, M., 1993. Applying tabu-search to the job shop scheduling problem. *Annals of Operations Research*, 41: 231-252.

Dorndorf, U. & Pesch, E., 1992. Evolution based learning in a Job shop scheduling environment. *Research Memorandum* 92-019, University of Limburg, Belgium.

Dutta, S.K. & Cunningham, A.A., 1975. Sequencing two machine flow shops with finite intermediate storage. *Management Science*, 21, p. 989-996.

Ezpeleta, J., Colom, J.M. & Martinez, J., 1995. A Petri net Based Deadlock prevention Policy for Flexible Manufacturing Systems. *IEEE Transactions on Robotics and automation*, 11, p. 173-184.

Falkenauer, E. & Bouffouix, S., 1991. A genetic algorithm for job shop. Dans IEEE International Conference on Robotics and Automation. Sacramento, California.

Fanti, M.P., Maione, B., et al., 1997. Event-Based Feedback control for Deadlock Avoidance in Flexible Production Systems. *IEEE Transactions on Robotics and automation*, 13, p. 347-363.

Fanti, M.P., Maione, B. & Turchiano, B., 1998. Deadlock Avoidance in flexible Production Systems with Multiple Capacity Resource. *informatics and control*, 7, 4, p. 343-364.

Fanti, M.P., Maione, B. & Turchiano, B., 1996. Digraph-theoritic Approach for Deadlock Detection and recovery in Flexible Procudtion Systems. *informatics and control*, 5, 4, p. 373-383.

Fisher, H. & Thompson, G., 1963. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling. Prentice-Hall, Englewood Cliffs, NJ*, p. 225–251.

Florian, M., McMahon, G. & Trepant, P., 1971. An implicit enumeration algorithm for the machine sequencing problem. *Management Science*, 17, 12.

Garey, M.R., Johnson, D.S. & Sethi, R., 1976. The Complexity of Flowshop and Jobshop Scheduling. *MATHEMATICS OF OPERATIONS RESEARCH*, 1, p. 117-129.

Garey, M.R. & Johnson, M.R., 1979. Computers and intractability: A guide to the theory of NP-completeness. *San Francico: Freeman.*

Giffler, R. & Thompson, G.L., 1960. Algorithms for solving production scheduling problems. *Operations Research*, 8, 4.

Gilmore, P.C. & Gomory, R. E., 1964. Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *OPERATIONS RESEARCH*, 12(5), p. 655-679.

Glover, F., 1989. Tabu Search-Part I. *ORSA JOURNAL ON COMPUTING*, 1(3), p. 190-206.

Glover, F., 1990. Tabu Search-Part II. *ORSA JOURNAL ON COMPUTING*, 2(1), p. 4-32.

Gomes, M., Barbosa-Póvoa, A. & Novais, A., 2005. Optimal scheduling for flexible job shop operation. *International journal of production research*, 43, 11, p. 2323-2353.

Gomory, Ralph E., 1958. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64, p. 275-278.

Gondran, M. & Minoux, M., 1984. *Graphs and algorithms*, John Wiley and Sons, New York. 21.

Gonzalez, T. & Sahni, S., 1978. Flow-shop and job-shop scheduling: Complexity and approximation. *Operations Research*, 26, p 36-52.

Gonzalez, Teofilo & Sahni, Sartaj, 1976. Open Shop Scheduling to Minimize Finish Time. *Journal of the Association for Computing Machinery*, 23, p. 665–679.

Grabowski, J. & Pempera, J., 2000. Sequencing of jobs in some production systems. *European Journal of Operational Research*, 125, p. 535-550.

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

Graham, R.L. et al., 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Dans *Annals Discrete Math*. p. 287-326.

Gupta, J.N.D. & Tunc, E.A., 1991. Schedules for a two-stage hybrid flow shop with parallel machines at the second stage. *International journal of production research*, 29, p. 1489-1502.

Gupta, Jatinder N. D., 1988. Two-Stage, Hybrid Flowshop Scheduling Problem. *The Journal of the Operational Research*, 39, p. 359-364.

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

Haouari, M. & Ladhari, T., 2003. A branch-and-bound-based local search method for the flow shop problem. *Journal of the Operational Research Society*, 54, p. 1076-1084.

Hardgrave, W.H. & Nemhauser, G.L., 1963. A geometric model and a graphical algorithm for a sequencing problem. *Operations Research*, 11, p. 889-900.

Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.

Hoogeveen, J.A., Lenstra, J.K. & Veltman, B., 1996. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *European Journal of Operational Research*, 89, p. 172-175.

Hsieh, F.S. & Cheng, S.C., 1994. Dispatching-driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Ssystems. *IEEE Transactions on Robotics and automation*, 10, 2, p. 196-209.

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

Ignall, E. & Schrage, L., 1965. Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Operations Research*, 13, p. 400-412.

Jackson, J.R., 1956. An extension of Johnson's results on job lot scheduling. *Naval Research Logistics Quarterly*, 3, 201-203.

Jain, A.S. & Meeran, S., 1999. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113, 2, p. 390-434.

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

Jones, M., 2005. *GNU/Linux Application Programming*, Charles River Media.

Jurisch, B., 1995. Lower bounds for the job shop scheduling problem on multi-purpose machines. *Discrete Applied Mathematic*, 58, p. 145-156.

Jurisch, B., 1992. *Scheduling Jobs in Shops with multi-purpose machines*. P.h.D Thesis. University of Osnabück, Germany.

Kernighan, B.W., Lin, S., (1970). An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 49, 291-307.

Khosla, I., 1995. The scheduling problem where multiple machines compete for a common local buffer. *European Journal of Operational Research*, 84, p. 330-342.

Khosla, I.S., 1989. Scheduling of job shops with finite-capcity local buffers. *Working paper, The Curtis L. Carlson School of Management, University of Minnesota*.

Kirkpatrick, S., 1983. Optimization by Simulated Annealing. *Science*, 220, p. 671-680.

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

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

Lee, C.Y. & Vairaktarakis, R.G., 1994. Minimizing makespan in hybrid Flowshops. *Operation Research Letters*, p. 149-158.

Leisten, R., 1990. Flowshop sequencing problems with limited buffer storage. *International journal of production research*, 28, p. 2085-2100.

Lenstra, J.K., Rinnooy Kan, A.H.G. & Brucker, P., 1977. Complexity of machines scheduling problems. Dans *Annals of Operations Research*. p. 343-362.

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

Loukil, T., Teghem, J. & Fortemps, P., 2007. A multi-objective production scheduling case study solved by simulated annealing. *European Journal of Operational Research*, 179, 3, p. 709-722.

Lourenço, H.R., 1994. Local optimization and the job shop scheduling problem. *Communication Personnelle*.

MAC CARTHY, B.L. & LIU, J., 1993. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International journal of production research*, (31), p. 59-79.

Marsland T.A., Björnsson Y., (2000). Variable Depth Search, Advances in Computer Games 9, University of Maastricht p. 5-20.

Martinez, S., 2005. *Ordonnancement de systèmes de production avec contraintes de blocage. PhD thesis.*

Martinez, S. et al., 2006b. Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, 163, 3, p. 855-864.

Martinez, S., Guèret, C. & Sauer, N., 2004c. Heuristic algorithms for the two-stage flow-shop with a new blocking constraint. Dans MOSIM'04.

Mati, Y., XIE, X. & Rezg, N., 2002b. *Les problèmes d'ordonnancent dans les systèmes de production Automatisés: modèle, complexité*. PhD THESIS. University DE Metz.

Matsuo, H., Suh, C.J. & Sullivan, R.S., 1988. *A Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem*, Graduate School of Business, University of Texas, Austin.

McCormick, S.T. et al., 1989. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37, p. 925-936.

M. Haouari & R. M'Hallah. Heuristic algorithms for the two-stage hybrid flow-shop problem. operations research letters, 21 :43-53, 1997.

Nait, T. et al., 2005. An ant colony system minimizing total tardiness for hybrid job shop scheduling problem with sequence dependent setup times and release dates. Dans *International Conference on Industrial Engineering and Systems Management*. Marrakech.

Nait, T. et al., 2004. Hybrid job-shop scheduling in a printing workshop. Dans *MOSIM'04*. Ecole des Mines de Nantes-France, p. 775-762.

Nakano, R. & Yamada, T., 1991. Conventional genetic algorithm for job-shop problems. Proceedings of the 4th International *Conference on Genetic Algorithm* and their Applications, San Diego, USA, pp. 474-479.

Narasimhan, S.L. & Mangiameli, P.M., 1987. A comparetion of scheduling rules for a two stage hybrid flow shop. *Decision Sciences*, 18, p. 250-265.

Narasimhan, S.L. & Panwalker, S.S., 1984. Scheduling in a two-stage manufacturing process. *International journal of production research*, 22, p. 555-264.

Nawaz, M., Enscore, E. & Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA, The International Journal of Management Science*, 11/1, p. 91-95.

Negenman, E.G., 2001. Local Search Algorithms for the Multiprocessor Flow Shop Scheduling Problem. *European Journal of Operational Research*, 128, p. 147-158.

Néron, E., Baptiste, P. & Gupta, J.N.D., 2001. Solving hybrid flow shop problem using the energetic reasoning and global operations. *Omega*, 29, p. 501-511.

Nowicki, E., 1999. The permutation flow shop with buffers: A tabu search approach. *European Journal of Operational Research*, 116, p. 205-219.

Nowicki, E. & Smutnicki, C., 1996. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91, p. 160-175.

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

Panwalkar, S.S. & Iskander, W., 1977. A survey of scheduling rules. *Operations Research*, 25, 1.

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

Penz, B., 1996. Une méthode heuristique pour la résolution des problèmes de job-shop flexible. *Journal européen des systèmes automatisés*, 30, 6, p. 767-780.

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

Potts, C.N. & Van Wassenhove, L.N., 1982. A decomposition algorithm for the single machine tardiness problem. *Operations Research Letters, 32*, p. 177–181.

Rao, T.B.K., 1970. Sequencing in the order a? b with multiplicity of machines for a single operation. *Opsearch*, 7, p. 135-144.

Rechenberg, I., 1973. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, PhD thesis*, Reprinted by Fromman-Holzboog.

Reddi, S.S., 1976. On the flow shop sequencing problem with no-wait in process. *Managment Science*, 23, p. 216-217.

Reeves, C.R., 1993. Improving the effciency of tabu search for machine sequencing problems. *Journal of the Operational Research Society*, 44, p. 375-382.

Reklaitis, G.V., 1982. Review of Scheduling of Process Operations, *AIChE Symp. Ser., 78*, No. 214, 119-133.

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

Rinnooy Kan, A.H.G., 1976. *Machine Scheduling Problem: Classification, Complexity and Computations*. Nyhoff.

Rock, H. & Schmidt, G., 1983. Machine aggregation heuristics in shop scheduling. *Methods of Operational Research*, 45, p. 303-314.

Sauvey, C. & Sauer, N., 2011d. A genetic algorithm with genes-association recognition for flowshop scheduling problems. *Journal of Intelligent Manufacturing*.

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

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

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

Schrage, L. & Baker, K.R., 1978. Dynamic Programming Solution of Sequencing Problems with Precedence Constraints. *Operations Research*, 26(3), p. 444-449.

Schwefel, H.-P., 1977. *Numerische Optimierung Von Computer-Modellen Mittels Der Evolutions- Strategie*, Birkhauser Verlag.

Scrich, C., Armentano, V. & Laguna, M., 2004. Tardiness minimization in a flexible job shop: A tabu search approach. *Journal of Intelligent Manufacturing*, 15, 1, p. 130-115.

Shen, V.Y. & Chen, Y.E., 1972. 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*, p. 645-649.

Sherali, H.D., Sarin, S.C. & Kodialam, M.S., 1990. Models and algorithms for a two-stage production process. *Production Planning and Control*, 1, p. 27-39.

Sostkov, Y.N., 1985. Optimal serving tow jobs with a regular criterion, In Automation of Designing Processes. *Minsk*, p. 86-95.

Suhami, I. & Mah, R.S.H., 1981. An implicit enumeration scheme for the flow shop problem with no intermediate storage. *Computers and Chemical Engineering*, 5, p. 83-91.

Szwarc, W., 1960. Solution of the Akers-Friedman scheduling problem. *Operations Research*, 8, 6, p. 782-788.

Van Laarhoven, P.J.M., Aarts, E.H.L. & Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. *Operations Research*, 40, 1.

Vancheeswaran, R. & Townsend, M.A., 1993. Two-stage heuristic procedure for scheduling job shops. *Journal of Manufacturing Systems*, 12, 4.

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

Werner, F. & Winkler, A., 1991. Insertion techniques for the heuristic solution of the job shop problem. *Communication Personnelle*.

Wittrock, R.J., 1988. An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 36, p. 445-453.

Wittrock, R.J., 1985. Scheduling algorithms for flexible flow lines. *IBM Journal of Research and Development*, 29, p. 401-412.

Wysk, R.A., Yang, N.S. & Joshi, S., 1991. Detection of Deadlocks in Flexible Manufacturing Cells. *IEEE Transactions on Robotics and automation*, 7, 6, p. 853-859.

Xia, W. & Wu, Z., 2005. An effective hybrid optimization approach for multi-objective flexible job shop scheduling problems. *Computers & Industrial Engineering*, 48, 2, p. 409 – 425.

Xing, K.Y., Hu, B.S. & Chen, H.X., 1996. Deadlock Avoidance Policy for Petri Net Modeling of flexible manufacturing Systems with Shared Resources. *IEEE Transactions on Robotics and automation*, 41, 2, p. 289-295.

Yuan, K., Sauvey, C. & Sauer, N., 2009e. Application of EM algorithm to hybrid flow shop scheduling problems with a special blocking. *ETFA*.