



Gestion de production sous incertitudes

Mourad Benttaieb

► To cite this version:

Mourad Benttaieb. Gestion de production sous incertitudes. Recherche opérationnelle [math.OC]. Université de Technologie de Troyes, 2018. Français. NNT : 2018TROY0028 . tel-03215278

HAL Id: tel-03215278

<https://theses.hal.science/tel-03215278>

Submitted on 3 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse
de doctorat
de l'UTT

Mourad BENTTALEB

Gestion de production sous incertitudes

Spécialité :
Optimisation et Sûreté des Systèmes

2018TROY0028

Année 2018

THESE

pour l'obtention du grade de

DOCTEUR de l'UNIVERSITE DE TECHNOLOGIE DE TROYES

Spécialité : OPTIMISATION ET SURETE DES SYSTEMES

présentée et soutenue par

Mourad BENTTALEB

le 24 septembre 2018

Gestion de production sous incertitudes

JURY

M. S. DAUZÈRE-PÉRÈS	PROFESSEUR ENSM SAINT-ETIENNE	Président
Mme M.-L. ESPINOUSE	PROFESSEUR DES UNIVERSITES	Rapporteur
M. F. HNAIEN	MAITRE DE CONFERENCES - HDR	Directeur de thèse
M. B. IUNG	PROFESSEUR DES UNIVERSITES	Examineur
M. N. REZG	PROFESSEUR DES UNIVERSITES	Rapporteur
M. F. YALAOUI	PROFESSEUR DES UNIVERSITES	Directeur de thèse

Avant-propos

Le travail présenté dans ce mémoire a été effectué dans le laboratoire d'optimisation des systèmes industriels (LOSI) à l'université de technologie de Troyes (UTT).

Je tiens tout d'abord et particulièrement, à remercier mes directeurs de thèse, Farouk Yalaoui et Faicel Haien, pour m'avoir accordé leur confiance et permis d'effectuer cette thèse dans les meilleures conditions. J'exprime ma profonde gratitude pour l'intérêt qu'ils ont porté aux travaux, les conseils éclairés qu'ils m'ont prodigués et leurs encouragements. Mon profond respect pour leur rigueur scientifique et leur disponibilité permanente.

Je ne peux manquer de remercier Madame Marie-Laure Espinouse, Professeur des Universités à l'université Grenoble Alpes et Monsieur Nidhal Rezg, Professeur des Universités à l'Université de Lorraine en leur qualité de rapporteurs de cette thèse. Un grand merci à Monsieur Stéphane Dauzère-Pérès, professeur à l'école des mines de Saint-Étienne et Monsieur Benoit Iung, Professeur des universités à l'université de Lorraine qui ont accepté d'être examinateurs de ma thèse.

J'adresse bien un vif remerciement aux enseignants chercheurs. Particulièrement, je remercie Alice Yalaoui, Yassine Ouazene pour l'expérience d'enseignement que nous avons partagée ensemble.

Enfin, j'adresse un grand salut à tous mes collègues, du LOSI et de l'UTT de manière générale.

Mourad Banttaleb

Résumé

Dans un contexte de production en perpétuelle évolution, l'entreprise est tenue à s'adapter en permanence aux fluctuations des marchés (demandes aléatoires) et aux perturbations internes de ses systèmes (pannes, absentéisme, ...). Face à ces incertitudes, la gestion de production permanente bénéficiant également de la puissance de la technologie informatique est indispensable. L'ordonnancement des ateliers de production est l'un des axes d'optimisation contribuant à l'amélioration de la productivité face aux aléas. Il s'agit dans ce contexte de définir les politiques d'ordonnancement avec la prise en compte de l'indisponibilité des ressources (gestion de la maintenance, de ressources humaines ...). Dans cette thèse, nous nous intéressons plus particulièrement aux problèmes d'ordonnancement d'atelier à chemins multiples (Job shop) à deux machines avec des ressources non disponibles en permanence. Le but de nos travaux est de proposer des méthodes de résolution exactes et approchées pour ce type de problèmes, exploitant autant que possible des propriétés théoriques pour la minimisation du makespan qui revient à maximiser la productivité.

Mots-clés : Recherche opérationnelle, Optimisation combinatoire, Ordonnancement (gestion), Programmation linéaire, Algorithmes d'approximation, Métaheuristiques.

Abstract

Production is always in perpetual evolution. Indeed, company needs to adapt its performance to market fluctuations (random demand) and internal perturbations within its systems (breakdowns, absenteeism, etc.). To cope with these uncertainties, permanent production management benefiting from the power of computer technology is fundamental. The scheduling of production is one of optimization's tools contributing to the improvement of productivity under constraints. In this context, the objective is to define scheduling policies, taking into account the unavailability of resources (management of maintenance, human resources, etc.). In this thesis, we are particularly interested in the study of the two-machine job shop scheduling problems with resources not always available. The aim of our work is to propose methods for exact and approximate resolution for this type of problem. These methods exploit as much as possible theoretical properties for the minimization of the total completion time (makespan) which is to maximize the productivity.

Keywords : Operations research, Combinatorial optimization, Production scheduling, Linear Programming, Approximation algorithms, Metaheuristics.

Table des matières

Remerciements	iii
Résumé	v
Abstract	vii
Table des matières	xii
Liste des figures	xiv
Liste des tableaux	xvii
1 Généralités sur l’ordonnancement	5
1.1 Introduction	6
1.2 Ordonnancement : Formulation, ateliers, complexité et méthodes de résolution	6
1.2.1 Formulation d’un problème d’ordonnancement	6
1.2.2 Notations	9
1.2.3 Les ateliers	11
1.2.4 La complexité	13
1.2.5 Les méthodes de résolution	14
1.3 Conclusion	19
2 Problèmes d’ordonnancement avec contraintes de disponibilité : État de	

l'art	21
2.1 Introduction	22
2.2 Définitions	22
2.3 Problèmes d'ordonnancement avec contraintes de disponibilité	23
2.3.1 Problème d'ordonnancement à une seule machine	23
2.3.2 Problème d'ordonnancement à machines parallèles	30
2.3.3 Problème d'ordonnancement de type Flow shop	34
2.3.4 Problème d'ordonnancement de type Job shop	40
2.3.5 Problème d'ordonnancement de type Open shop	42
2.4 Conclusion	44
3 Le problème d'ordonnancement de type Job shop à deux machines avec contraintes de disponibilité sur une machine	45
3.1 Introduction	47
3.2 Description du problème	47
3.3 Modélisation mathématique	49
3.3.1 Formulation 1 : MILP1	49
3.3.2 Formulation 2 : MILP2	51
3.4 Propriétés	52
3.5 Bornes supérieures et inférieures	59
3.5.1 Bornes supérieures	59
3.5.2 Bornes inférieures	64
3.6 Procédure par séparation et évaluation	65
3.6.1 Initialisation	66
3.6.2 Branchement et séparation des noeuds	67
3.6.3 Évaluation des noeuds	67

3.6.4	Exemple illustratif	69
3.7	Résultats expérimentaux	70
3.7.1	Description des instances	71
3.7.2	Analyse des résultats expérimentaux	71
3.8	Conclusion	81
4	Le problème d'ordonnancement de type Job shop à deux machines avec une contrainte de disponibilité sur chaque machine	83
4.1	Introduction	85
4.2	Description du problème	85
4.3	Modélisation mathématique : MILP3	86
4.4	Propriétés	87
4.5	Bornes supérieures et inférieures	92
4.5.1	Bornes supérieurs	92
4.5.2	Bornes inférieures	94
4.6	Procédure par séparation et évaluation	94
4.6.1	Le premier schéma de branchement	95
4.6.2	Le second schéma de branchement	97
4.6.3	Exemple illustratif	98
4.7	Modèle de programmation linéaire en nombres entiers mixtes (MILP4) . . .	101
4.8	Résultats expérimentaux	103
4.8.1	Description des paramètres	103
4.8.2	Analyse des résultats expérimentaux	105
4.9	Conclusion	121
5	Méthodes approchées pour le problème d'ordonnancement de type Job shop à deux machines avec contraintes de disponibilité sur une machine	123

5.1	Introduction	125
5.2	Propriétés	125
5.3	Heuristique Constructive (H_const)	127
5.4	Recherche locale itérée (Iterated local search algorithm)	127
5.4.1	La solution initiale	129
5.4.2	La recherche locale	129
5.4.3	La perturbation	129
5.4.4	Le critère d'acceptation	130
5.4.5	Le critère d'arrêt	131
5.5	Résultats expérimentaux	131
5.5.1	Génération d'instances	131
5.5.2	Paramétrage et analyse de sensibilité	131
5.5.3	Analyse des résultats expérimentaux	136
5.6	Conclusion	139

Table des figures

3.1	Impact de la période d'indisponibilité à $s_1 = 0$, sur M_1 si $Idle_1^{JK} \neq 0$	53
3.2	Impact de la période d'indisponibilité à $s_1 = 0$, sur M_1 , si $Idle_1^{JK} = 0$	54
3.3	Impact de la période d'indisponibilité à $s_1 = 0$, sur M_2	55
3.4	Résolution du problème avec période d'indisponibilité sur chaque machine $s_k = 0$	55
3.5	Ordonnancement des tâches l'ordre optimal " S^* " pour $J_2, h_{11} a C_{max}$	56
3.6	Ordonnancement optimal avant la période d'indisponibilité	57
3.7	Ordonnancement optimal après la période d'indisponibilité	57
3.8	Ordonnancement des tâches selon l'ordre optimal " S^* " pour $J_2, h_{12} a C_{max}$	58
3.9	Ordonnancement optimal des tâches après la deuxième période d'indisponibilité	58
3.10	Ordonnancement des tâches par l'heuristique H1	60
3.11	Solution obtenue par l'algorithme de Jackson (sans période d'indisponibilité)	69
3.12	Solution trouvée par les heuristiques $H1$ et $H2$	69
3.13	L'arborescence du PSE pour l'exemple présenté	70
3.14	La solution optimale obtenue par la PSE	70
3.15	$\frac{BS - BI}{BI}$ (%) pour chaque instance ($N = 15$, $U\{2, N/4, N/2, 3N/4\}$)	76
3.16	$\frac{BS - BI}{BI}$ (%) pour chaque instance ($N = 20$, $U\{2, N/4, N/2, 3N/4\}$)	76
4.1	Illustration du contre-exemple	88

4.2	Illustration du cas $J_i = J_{A^{k'}}$	89
4.3	Illustration du cas $J_j = J_{A^k}$	89
4.4	Illustration du cas $J_i = J_{A^k}$	89
4.5	Ordonnancement des tâches selon l'ordre optimal " S^* " pour $J_2, h_{k1} a, d_{t_k, s_{\bar{k}}} C_{max}$	91
4.6	Ordonnancement des tâches selon l'ordre optimal " S^* " avant la période d'indisponibilité sur la machine M_1	91
4.7	Ordonnancement des tâches selon l'ordre optimal " S^* " après la période d'indisponibilité sur la machine M_1	92
4.8	Ordonnancement des tâches par l'heuristique H3	93
4.9	(a) Solution obtenu par l'algorithme de Jackson (sans contraintes de disponibilité), (b) Solution trouvée par les heuristiques $H3$ et $H4$	99
4.10	L'arborescence obtenue par $PSE1$	100
4.11	L'arborescence obtenue par $PSE2$	101
4.12	La solution optimale obtenue	101
5.1	Diagramme des interactions pour Avg	136
5.2	Diagramme des interactions pour $CPU(s)$	136
5.3	Diagramme des interactions pour Dev	137
5.4	Graphe des effets principaux pour Avg	137
5.5	Graphe des effets principaux pour $CPU(s)$	138
5.6	Graphe des effets principaux pour Dev	138

Liste des tableaux

3.1	Notations	48
3.2	Paramètres de l'exemple	69
3.3	Comparaison des modèles MILP1 et MILP2 ($N = \{15, 20\}$, $U = 2$)	74
3.4	Pourcentage des problèmes résolus optimalement par la PSE (%)	74
3.5	Le gap (%) entre la borne supérieure (BS) et la borne inférieure (BI) ($N = \{15, 20\}$, $U = \{2, N/4, N/2, 3N/4\}$)	75
3.6	Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = 2$)	77
3.7	Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = N/4$)	78
3.8	Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = N/2$)	79
3.9	Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = 3N/4$)	80
4.1	Temps d'exécution Lee (1999)	88
4.2	Paramètres de l'exemple considéré	99
4.3	Les configurations considérées	104
4.4	Temps d'exécution moyen (en secondes) of PSE1, PSE2 et MILP4 (Instances de Taillard)	107
4.5	Résultats obtenus par MILP3 (instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$)	108
4.6	Résultats obtenus par PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$	109

4.7	Résultats obtenus par PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$, $P4$ et $P5$	110
4.8	Résultats obtenus pa PSE2 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$	111
4.9	Résultats obtenus pa PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$, $P4$ et $P5$	112
4.10	Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$)	113
4.11	Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$)	114
4.12	Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P3$)	115
4.13	Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances aléatoires, $P3$)	116
4.14	Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P1$)	117
4.15	Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P2$)	118
4.16	Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P3$)	119
4.17	Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances aléatoires , $P3$)	120
5.1	Valeurs des paramètres	133
5.2	Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 15$	134
5.3	Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 20$	135
5.4	Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 30$	135
5.5	Comparaison des différentes méthodes approchées $U = N/4$, $N = 15, 20, 30$)	140
5.6	Comparaison des différentes méthodes approchées $U = N/2$, $N = 15, 20, 30$)	141

5.7	Comparaison entre ILS_LT et PSE_Stop ($U = N/4$, $N = 15, 20, 30$)	142
5.8	Comparaison entre ILS_LT et PSE_Stop ($U = N/2$, $N = 15, 20, 30$)	143

Introduction générale

L'entreprise d'aujourd'hui vit dans un environnement caractérisé par un marché de plus en plus concurrentiel et dans le quel le client est de plus en plus averti et exigeant sur la qualité, le coût et le délai. L'entreprise est également obligée de s'adapter en permanence au fluctuation de ses marchés (demandes aléatoires) et aux perturbations internes de ses systèmes (incertitudes, pannes des machines, absentéisme ...). Par ailleurs, le monde actuel est défini par une croissance des nouvelles technologies en perpétuelle évolution. Les décideurs se trouvent donc obligés et motivés d'optimiser en permanence leurs systèmes profitant également de la puissance de la technologie informatique, du progrès scientifique et de l'innovation.

Parmi les axes d'amélioration de la compétitivité des entreprises, nous trouvons l'optimisation des systèmes (industriels, services, chantiers, transport ...). Particulièrement, l'ordonnancement des ateliers de production qui dépend étroitement de l'amélioration des performances de ces systèmes. Il s'agit dans ce contexte de définir la règle d'ordonnancement et d'affectation de tâches aux ressources avec la prise en compte de la disponibilité de ces ressources à travers une politique de gestion de leurs disponibilités (gestion de la maintenance, de ressources humaines ...), dans la perspective de minimiser les coûts et les délais d'exploitation des systèmes de production.

Nos travaux présentés dans cette thèse concernent la résolution du problème d'ordonnancement avec la prise en compte des contraintes de disponibilité des machines. En effet, une majeure partie des travaux dédiés à l'étude des problèmes d'ordonnancement se place dans le contexte où les ressources sont disponibles de façon permanente. En réalité, les différents ressources matérielles et humaines peuvent être indisponibles pour différentes raisons. D'ailleurs, la date de début et la durée d'une indisponibilité peut être imprévisibles dans des cas tels que les pannes machines ou absences non programmée du personnel (non déterministe). Elles sont connues à l'avance ou planifiées dans certains cas notamment les opérations de maintenance préventive ou toute intervention planifiée nécessitant l'arrêt d'une ou plusieurs machines ou lors d'une absence planifiée du personnel telle que les congés.

Nous traitons dans cette thèse le contexte déterministe où l'ordonnancement prends en compte les arrêts des machines planifiés à l'avance pour une meilleurs gestion et exploitation de la disponibilité des machines et aussi une meilleure anticipation et prévention de toute incertitude due aux arrêts imprévisibles pouvant engendrer plus de coûts.

Nous nous intéressons plus particulièrement aux problèmes d'ordonnancement de type Job shop à deux machines avec des ressources non disponibles en permanence. Le but de nos travaux est de proposer des méthodes de résolution pour ce type de problèmes dans la perspective de développer un outil d'aide à la décision visant à maitriser l'incertitude due aux indisponibilités.

Ce type d'atelier existe dans l'industrie et répond éventuellement à un besoin de plusieurs types d'industries. Nous citons à titre d'exemple, l'industrie de fabrication additive (impression 3D) et soustractive impliquant l'utilisation des machines-outils ou d'ateliers d'usinage et de fabrication. Dans ce genre d'industrie, le processus de fabrication pourrait être modélisé comme problème d'ordonnancement de type job shop à deux machines. L'application est également possibles dans le transport, le robotique, la manutention, l'administration et également dans l'ordonnancement des chantiers de travaux publics et bien d'autres applications.

Dans le chapitre 1, nous situons notre travail dans le cadre de l'ordonnancement des systèmes de production. Nous donnons en premier lieu une vue globale sur l'ordonnancement et sur la complexité des problèmes. Nous présentons ensuite les notations utilisées permettant de caractériser le problème ainsi que le différents types d'ateliers d'ordonnancement. Dans un deuxième temps, nous focalisons notre attention sur la présentation des différentes méthodes d'optimisation exactes et approchées.

Le chapitre 2 présente un état de l'art couvrant les travaux de recherche menés sur les problèmes d'ordonnancement statiques avec contraintes de disponibilité qui considèrent que toutes les données sont connues à l'avance. Nous considérons par ailleurs le contexte déterministe.

Le chapitre 3 est consacré à l'étude de l'ordonnancement de type job shop à deux machines avec contraintes de disponibilité sur une seule machine. En particulier, nous présentons une procédure par séparation et évaluation pour la résolution exacte du problème. Les bornes inférieures et les heuristiques employées sont basées sur des propriétés de dominance que nous avons établies.

Dans le chapitre 4, nous nous intéressons au problème d'ordonnancement de type Job shop à deux machines avec une contrainte de disponibilité sur chaque machine. Des propriétés

d'optimalité et autres traitant les ordonnancements de permutation sont démontrées. Nous proposons ainsi des bornes inférieurs et deux procédures par séparation et évaluation ainsi qu'un modèle mathématique considérant les propriétés démontrées dans ce chapitre.

Le chapitre 5 concerne les méthodes de résolution approchées. Nous développons les heuristiques et une méta-heuristique de recherche locale itérée (Iterated local search) que nous avons construites. Ces approches constituent une alternative aux méthodes exactes pour la résolution du problème d'ordonnancement de type job shop à deux machines avec contraintes de disponibilités sur une machine.

Le rapport se termine par des conclusions ainsi que des perspectives de recherche envisagées.

Chapitre 1

Généralités sur l'ordonnancement

Résumé :

Ce chapitre est consacré à la présentation des notions de base nécessaires à la description du sujet d'étude de cette thèse. Nous introduisons les différentes définitions et formulations propres à l'ordonnancement. Également, nous présentons brièvement les types d'ateliers d'ordonnancement et aussi un rappel de la théorie de complexité. Nous finissons le chapitre par une présentation de quelques méthodes de résolution exacte et approchée du problème.

1.1 Introduction

La théorie de l'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales des tâches, après affectation des ressources nécessaires à l'exécution de ces tâches. Un problème d'ordonnancement peut être considéré comme un sous-problème de planification dans lequel il s'agit de décider de l'ordre d'exécution opérationnelle des tâches planifiées.

D'une manière précise, ordonnancer consiste à organiser dans le temps l'exécution des opérations en leur allouant les ressources requises. Il permet de fixer leurs dates de début de réalisation compte tenu des contraintes temporelles et de disponibilité des ressources requises, pour répondre au mieux aux besoins (objectifs, critères).

À la suite de l'étape de planification qui vise à déterminer les différentes opérations à réaliser, les dates correspondantes et les moyens matériels et humains à y affecter. L'ordonnancement se déroule en trois étapes :

- l'affectation qui attribue les ressources nécessaires aux tâches.
- le séquençement qui indique l'ordre de passage des tâches sur les ressources.
- la détermination des dates de début et de fin d'exécution des tâches sur les ressources.

1.2 Ordonnancement : Formulation, ateliers, complexité et méthodes de résolution

1.2.1 Formulation d'un problème d'ordonnancement

Un problème d'ordonnancement est composé de façon générale d'un ensemble de tâches soumises à certaines contraintes, et dont l'exécution nécessite des ressources. Résoudre un problème d'ordonnancement, consiste à organiser ces **tâches**, c'est à dire à déterminer leur dates de démarrage et à leur attribuer des **ressources**, de telle sorte que les **contraintes** soient respectées, afin d'optimiser un ou plusieurs **critères**.

1.2.1.1 Les tâches

Une tâche est un travail (ou job) constitué d'un ensemble d'opérations qui requiert, pour son exécution, un certain nombre d'unités de temps (sa durée) et d'unités de ressources.

Certaines contraintes peuvent associer aux tâches des dates de début au plus tôt r_i ou des dates de fin au plus tard d_i .

On distingue deux types de tâches :

- les tâches préemptives (morcelables) divisibles ou exécutables par morceaux.
- les tâches non-préemptives (indivisibles) exécutables en une seule fois et ne peuvent être interrompues jusqu'à leur fin d'exécution.

1.2.1.2 Les ressources

Une ressource est un moyen technique ou humain permettant la réalisation des tâches et dont la disponibilité est limitée ou non. On distingue deux types de ressources :

- Les ressources renouvelables, qui, après avoir été allouées à une tâche, redeviennent disponibles (machines, personnel, etc).
- Les ressources consommables, qui, après avoir été allouées à une tâche, ne sont plus disponibles pour les tâches restant à exécuter (argent, matières premières, etc).

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. Par ailleurs, dans le cas des ressources renouvelables, on distingue principalement :

- Les ressources disjonctives qui ne peuvent exécuter qu'une tâche à la fois.
- Les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité.

1.2.1.3 Les contraintes

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre les variables liant les tâches aux ressources. Deux types de contraintes peuvent être distinguées : contraintes de ressources et contraintes temporelles.

- Contraintes de ressources : Elles traduisent le fait que les ressources sont disponibles en quantité limitée en l'occurrence, la capacité limitée d'une ressource implique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cette ressource.

Les contraintes relatives aux ressources peuvent être :

- Disjonctives, engendrant une contrainte de réalisation des tâches sur des intervalles de temps pour une même ressource. C'est à dire, une ressource ne peut être utilisée que par une tâche à la fois.
- Cumulatives, impliquant la limitation du nombre de tâches à réaliser en parallèle.
- Contraintes temporelles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement.

Ces contraintes peuvent être :

- Contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (délais de livraison, disponibilité des approvisionnements) ou à la durée totale d'un projet.
- Contraintes de précédence, une tâche J_i doit précéder la tâche J_j , et plus généralement les contraintes de cohérence technologique, qui décrivent le positionnement relatif de certaines tâches par rapport à d'autres.
- Contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches.

1.2.1.4 Les critères

La résolution de problèmes d'ordonnancement se fait, soit en cherchant à atteindre un optimal par rapport à un ou plusieurs critères, soit en cherchant une solution réalisable vis à vis de contraintes répondant au mieux à ces critères.

L'approche par optimisation cherchera alors, soit à minimiser ou à maximiser un critère correspondant à une amélioration liée au temps, par exemple la minimisation du temps total d'exécution des tâches ($\min C_{max}$) ou encore la minimisation de la somme des durées des retards ($\min \sum_i T_i$). Les critères d'optimisation concernent aussi l'utilisation des ressources ; à titre d'exemple, le nombre de ressources nécessaires pour réaliser un ensemble de tâches, ou leur charge.

Les critères sont classés en critères réguliers et irréguliers. Un critère est régulier, si la valeur à minimiser est une fonction non décroissante des dates de fin des produits. Un critère est non régulier s'il peut augmenter lorsqu'une tâche se termine au plus tôt. En particulier, si une tâche se terminant avant sa date de fin souhaitée implique un coût, alors il n'est pas régulier.

1.2.2 Notations

Nous considérerons les notations suivantes, pour définir les critères les plus utilisés dans un problème d'ordonnancement.

N : nombre de tâches,

m : nombre de machines,

$J = \{J_1, J_2, \dots, J_N\}$: ensemble de tâches à réaliser,

$M = \{M_1, M_2, \dots, M_m\}$: ensemble des machines de l'atelier,

r_i : date de disponibilité (*release date or ready date*) de la tâche J_i ,

d_i : date de fin souhaitée (*due date*) de J_i ,

w_i : coefficient de pondération (*weight*) associé à J_i ,

C_i : date de fin (*completion time*) de J_i ,

$L_i = C_i - d_i$: écart par rapport à la fin souhaitée ou retard algébrique (*lateness*) de J_i ,

$E_i = \max(d_i - C_i, 0)$: l'avance (*earliness*) de la tâche J_i ,

$T_i = \max(C_i - d_i, 0)$: le retard (*tardiness*) de la tâche J_i ,

U_i : indicateur de retard (*unit penalty*) de la tâche J_i . $U_i = 1$ si $T_i > 0$, $U_i = 0$ sinon,

O_{ij} : $j^{\text{ème}}$ opération de la tâche J_i ,

t_{ij} : date de début (*starting date*) de O_{ij} ,

p_{ij} : durée opératoire (*processing time*) de O_{ij} ,

C_{ij} : date de fin (*completion date*) de O_{ij} ,

Il existe une très grande variété de problèmes d'ordonnancement. Pour leur identification et leur classification nous adoptons la notation proposée par Graham *et al.* (1979) et par Blazewicz *et al.* (1983). Cette notation est constituée de trois champs $\alpha|\beta|\gamma$.

Le premier champ α est constitué de deux éléments : $\alpha = \alpha_1\alpha_2$. Il présente la structure du problème décrit par l'environnement des machines utilisées ou :

- Le paramètre $\alpha_1 \in \{\emptyset, P, Q, R, F, J, O\}$ décrit le type des machines utilisées. Ces paramètres correspondent respectivement aux problèmes d'ordonnancement à une seule machine, à machines parallèles identiques, à machines parallèles uniformes, à machines

parallèles indépendantes, de type flow shop, de type job shop et de type open shop.

- Le paramètre α_2 indique le nombre de machines utilisées. Lorsqu'il n'est pas précisé, le nombre de machines est quelconque.
- D'autres paramètres peuvent être considérés dans le champ α . Dans un problème d'ordonnancement, h_{rk} représente les k périodes d'indisponibilités sur une machine M_r .

Le deuxième champ β décrit les caractéristiques des tâches et des machines. Il indique si des éléments spécifiques sont à prendre en compte. Nous citons par exemple :

- r_i : chaque tâche J_i possède une date de disponibilité.
- d_i : chaque tâche J_i possède une date échue.
- S_{ij} : le temps de préparation dépendant de la séquence entre les tâches J_i et J_j .
- S_{ri} : le temps de préparation de la machine M_r pour la tâche J_i .
- M_{rk} : la machine M_r possède k périodes d'indisponibilité.
- $prmu$: l'ordre (ou permutation) selon lequel les tâches passent sur la première machine est maintenu à travers le système. cette machine d'effectuer une autre tâche.
- nwt : no-wait implique que les tâches ne peuvent attendre entre deux machines successives.

Le dernier champ γ indique le critère d'optimisation, il peut être une combinaison de plusieurs critères.

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

- $\sum_i w_i \cdot T_i$: somme pondérée des retards.
- $E_{max} = \max\{E_i, i = 1, \dots, n\}$: maximum des avances.
- $\sum_i T_i$: nombre de tâches en retard. $\sum_i w_i \cdot T_i$: nombre pondéré des tâches en retard.

1.2.3 Les ateliers

Une classification des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines et leur ordre d'utilisation pour fabriquer un produit.

1.2.3.1 Les ateliers à une machine

Le problème d'atelier à une machine consiste à ordonnancer, sur une seule machine, des tâches constituées d'une seule opération. Pour la minimisation du temps d'exécution totale, l'ordonnancement des tâches toutes disponibles à l'instant 0, toute séquence est une solution optimale. En revanche, d'autres critères et la considération de nouvelles contraintes rendent le problème difficile à résoudre.

1.2.3.2 Les ateliers à machines parallèles

Dans ces ateliers, les machines sont regroupées en parallèle. Ces derniers peuvent effectuer la même opération. Ce type d'atelier peut être divisé en trois sous-catégories selon la vitesse d'exécution des machines :

- les ateliers à machines identiques : toute tâche peut s'exécuter sur n'importe quelle machine avec une même durée opératoire,
- les ateliers à machines uniformes : chaque machine possède sa propre vitesse, indépendamment de la durée de la tâche à exécuter,
- les ateliers à machines indépendantes : la vitesse de chaque machine dépend de l'opération à effectuer.

1.2.3.3 Les ateliers de type flow shop

Les ateliers de type flow shop ou à cheminement unique sont les ateliers constitués d'une ligne de production avec plusieurs machines en série. Dans ce type d'ateliers toutes les tâches

sont traitées par les machines dans le même ordre. Dans les ateliers de type flow shop hybride, des machines peuvent exister en plusieurs exemplaires identiques fonctionnant en parallèle.

La résolution du problème d'ordonnancement de type flow shop à deux machines est polynomiale par l'algorithme de Johnson (1954) suivant :

- Diviser l'ensemble des tâches en deux sous-ensembles disjoints, Sub_1 et Sub_2 ; Avec $Sub_1 = \{J_i, p_{i1} \leq p_{i2}\}$ et $Sub_2 = \{J_i, p_{i1} > p_{i2}\}$.
- Ordonner les tâches appartenant à Sub_1 dans un ordre croissant des p_{i1} et celles de Sub_2 dans un ordre décroissant des p_{i2} .
- Séquencer les tâches de Sub_1 puis de Sub_2 .

1.2.3.4 Les ateliers de type job shop

Appelés également ateliers à cheminement multiple, ce sont des ateliers où les opérations constituant une tâche sont réalisées selon un ordre déterminé, variant selon le mode opératoire de la tâche à exécuter. Le job shop flexible est une extension du modèle job shop classique ; dans lequel une opération peut être traitée sur plusieurs machines alternatives.

La résolution du problème d'ordonnancement de type job shop à deux machines est polynomiale par l'algorithme de Jackson (1956).

L'algorithme commence par une partition des N tâches à ordonner sur deux machines (M_1, M_2) en quatre sous-ensembles $Set_1, Set_2, Set_{12}, Set_{21}$. $Set_{kk'}$ est l'ensemble des tâches traitées d'abord sur la machine M_k ensuite sur la machine $M_{k'}$ et Set_k est l'ensemble des tâches traitées uniquement sur la machine M_k avec $k \neq k' \in \{1, 2\}$.

- Séquencer les tâches appartenant à Set_{12} et Set_{21} selon la règle de Johnson. L'ordre des tâches appartenant à Set_1 et Set_2 n'a aucun effet sur la date de fin du traitement sur la machine M_1 et M_2 respectivement, donc tout ordre est accepté.
- Séquencer les sous ensembles des tâches selon l'ordre suivant :
 - On machine M_1 : Séquencer $Set_{12} - Set_1 - Set_{21}$.
 - On machine M_2 : Séquencer $Set_{21} - Set_2 - Set_{12}$.

1.2.3.5 Les ateliers de type open shop

Ce type d'atelier est moins contraint que celui de type flow shop ou de type job shop. Ainsi, l'ordre des opérations n'est pas fixé a priori ; le problème d'ordonnancement consiste,

d'une part, à déterminer le cheminement de chaque produit et, d'autre part, à ordonnancer les produits en tenant compte les gammes obtenues, ces deux problèmes pouvant être résolus simultanément.

1.2.4 La complexité

La théorie de la complexité (Cook (1971), Karp (1972), Garey & Johnson (1979)) permet d'analyser les coûts de résolution des problèmes d'optimisation combinatoire, notamment en terme de temps de calcul.

Un problème de décision est un énoncé auquel la réponse peut être uniquement oui ou non. Chaque problème d'optimisation possède un problème de décision correspondant.

Un problème de décision P_1 est dit réductible à un autre problème de décision P_2 (on note $P_1 \alpha P_2$) s'il existe une fonction polynomiale f qui transforme chaque énoncé de P_1 en un autre énoncé de P_2 de telle manière que la réponse pour P_1 est oui si, et seulement, si la réponse pour P_2 est oui.

Un problème de décision est dit *NP* s'il existe un algorithme polynomial qui permet de le résoudre. Autrement, la classe des problèmes *NP* est la classe des problèmes de décision pouvant être résolus par un algorithme polynomial. Parmi la classe des problèmes *NP*, on distingue la classe des problèmes polynômiaux (la classe *P*) et la classe des problèmes *NP-Complet*.

Un algorithme est dit polynomial (de classe *P*) si sa complexité temporelle est bornée par un $O(p(x))$ où p est un polynôme et x est la longueur d'une instance du problème. Un problème de décision est dit polynomial s'il existe un algorithme polynomial qui permet de le résoudre. Il est dit pseudo-polynomial si sa complexité est bornée par un polynôme en fonction de la taille de la plus grande instance du problème.

Un problème *NP* est dit *NP-Complet* si tout problème *NP* est réductible polynomialement en ce problème. De surcroît, un problème Q est *NP-Complet* s'il est de la classe *NP* et qu'il existe un problème R connu pour être *NP-Complet* tel que $R \alpha Q$.

La classe *NP-Complet* est aussi la classe des problèmes qui ne sont pas solvables par un algorithme polynomial, sous l'hypothèse $P = NP$. Cependant, si un problème de *NP-Complet* est résolu en un temps polynomial, on obtiendrait automatiquement des algorithmes polynômiaux pour tous les problèmes *NP-Complet*.

Un problème est dit *NP-Complet* au sens faible s'il est résolu par un algorithme

pseudo-polynomial. Dans le cas contraire, il est dit *NP-Comple*t au sens fort.

Un problème d'optimisation est dit *NP-Difficile* si le problème de décision qui lui correspond est *NP-Comple*t. Un problème d'optimisation est effectivement plus difficile que le problème de décision correspondant.

1.2.5 Les méthodes de résolution

Les méthodes de résolution des problèmes d'ordonnancement sont des techniques de l'optimisation combinatoire (programmation mathématique, programmation dynamique, procédure par séparation et évaluation, théorie des graphes ou autres).

Ces méthodes peuvent soit garantir l'optimalité de la solution fournie par des méthodes dites exactes soit par l'utilisation des méthodes approchées (heuristiques, méta-heuristiques), efficaces pour les problèmes d'optimisation *NP-difficiles*.

1.2.5.1 Les méthodes exactes

Généralement, il n'est pas possible de construire un algorithme polynomial résolvant un problème d'optimisation *NP-difficile*. Toutefois, on peut développer des méthodes efficaces pour la résolution. Le but des méthodes exactes est de résoudre, en un temps de calcul le plus court possible, d'une manière optimale le problème à optimiser. Nous citons les méthodes d'optimisation exacte les plus utilisées en ordonnancement.

La programmation linéaire consiste à optimiser un critère ou plusieurs modélisés par une fonction objectif linéaire des variables sous contraintes d'égalité ou d'inégalité elles aussi linéaires. Un modèle de programmation est dit en nombre entiers lorsque les variables de décisions sont entières. Dans le cas où ces dernières sont entières et réelles, le modèle est classé en programmation linéaire mixte en nombre entiers (Mixed integer linear programming). La résolution d'un modèle de programmation linéaire se fait par des solveurs dédiés comme Cplex, Xpress, GAMS, LINGO et autres.

La procédure par séparation et évaluation est utilisée pour résoudre d'une façon exacte des problèmes d'optimisation combinatoire et introduite par Dantzig *et al.* (1954) pour la résolution de problème de voyageur de commerce. Cette méthode consiste à construire une arborescence dont la racine correspond à l'espace des solutions du problème initial.

Dans le cas d'un problème de minimisation, une borne supérieure (BS) de la valeur de la fonction objectif pour une solution réalisable est avant tout calculé en utilisant une méthode approchée. La méthode se base sur deux étapes conjointes : le branchement ou séparation consiste à diviser un ensemble de solutions en sous-ensembles et l'évaluation qui la solution partiel par les bornes inférieures obtenue du problème.

Le branchement consiste à décomposer un noeud de l'arborescence représentant un ensemble de solutions d'un problème en une partition de sous-ensemble. L'utilisation seule du branchement revient à effectuer une énumération complète de l'espace des solutions. Aussi l'évaluation des noeuds permet d'éliminer les branches qui s'avère qu'elles ne contiennent pas de solutions optimales. Plus précisément, pour chaque noeud une borne inférieure (toujours pour un problème de minimisation) est calculée. Si cette valeur est supérieure ou égale à la solution obtenue par le noeud, ce dernier est éliminée. La borne supérieure est mise à jour à chaque fois qu'une solution réalisable donnant une meilleure valeur pour la fonction objectif est trouvée. Par ailleurs, l'exploration se termine quand tous les noeuds possibles sont visités ou si une solution optimale est trouvée.

Les performances de la méthode dépendent évidemment de la qualité des bornes inférieures et supérieures mais également du schéma d'exploration et de type de branchement. En outre, plusieurs stratégies pour la sélection des sommets à séparer sont utilisées à savoir, l'exploration en profondeur ou en largeur.

La programmation dynamique est basée sur le principe d'optimalité de Bellman (1954) : Si C appartient à un chemin minimal (maximal) allant de A à B , alors les sous-chemins de ce chemin allant de A à C et de C à B sont minimaux (maximaux). Ce principe permet de calculer les plus courts (longs) chemins d'un graphe de manière récurrente en connaissant les sous-chemins optimaux des étapes précédentes pour calculer les sous-chemins optimaux de l'étape en cours. Cette technique peut s'étendre à des séquences de tâches dans les problèmes d'ordonnancement. Pour les problèmes *NP-difficiles* au sens faible, il est souvent possible de construire un algorithme de programmation dynamique pseudo-polynomial, pouvant être utilisé pour des problèmes de dimension raisonnable.

La relaxation lagrangienne est une méthodologie générale qui permet d'obtenir des bornes inférieures de bonne qualité pour certains problèmes d'optimisation combinatoire. L'idée de la technique consiste à supprimer (relaxer) une partie des contraintes (en principe celles qui rendent le problème difficile) en les introduisant dans la fonction objectif sous forme d'une pénalité (combinaison linéaire des contraintes relaxées) si les contraintes relaxées

ne sont pas respectées (Fisher (1976)). Les coefficients de cette combinaison sont appelés multiplicateurs de Lagrange, et la nouvelle fonction objectif est appelée le Lagrangien du problème. La résolution du nouveau problème se fait généralement en utilisant la méthode du sous-gradient ou bien la méthode de génération de colonnes.

La génération de colonnes : Dans la pratique, de nombreux problèmes d'optimisation combinatoire sont complexes et de grande taille. Ils possèdent un grand nombre de variables (colonnes). Ceci empêche donc leur résolution par les logiciels, disponibles aujourd'hui, de résolution des programmes linéaires. Pour pouvoir les traiter, la méthode de génération de colonnes est utilisée. L'idée principale consiste à décomposer le modèle (Dantzig & Wolfe (1960)) d'origine en un ou plusieurs sous-problèmes plus faciles à résoudre, l'ensemble étant généralement coordonné par un programme linéaire appelé maître. Le principe de l'algorithme de génération de colonnes consiste à résoudre un problème original (P) avec un sous-ensemble de colonnes (variables) de taille réduite, puis à l'alimenter itérativement avec de nouvelles colonnes susceptibles d'améliorer la solution courante, jusqu'à atteindre l'optimalité.

1.2.5.2 Les méthodes approchées

Même-ci la technologie informatique est en perpétuelle évolution, des problèmes *NP-difficile* reste couteux en terme de temps de calcul. Par ailleurs, dans plusieurs situations pratiques, la résolution d'un problème d'optimisation combinatoire de taille importante, se heurte à des temps de calcul et des tailles mémoire importants. Compte tenu de ces difficultés, l'optimisation combinatoire s'oriente vers le développement des heuristiques et méta-heuristiques. L'objectif de ces méthodes est d'obtenir une solution proche de l'optimum ou de "bonne qualité" en un temps raisonnable. La performance de telles méthodes est généralement calculée par le rapport entre la valeur de la solution fournie et la valeur de la solution optimale, ceci pour le pire des cas ou dans le cadre d'une moyenne de plusieurs instances. Si la solution optimale est non calculable, il est également possible d'étudier expérimentalement le comportement de la méthode approchée en comparant ses performances soit à celles d'autres méthodes, soit à des bornes inférieures de la solution optimale.

Les méta-heuristiques sont des méthodes de recherche génériques pouvant s'appliquer à des problèmes de nature complètement différentes, dédiées aux problèmes d'optimisation

difficile. Une méta-heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème considéré, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible.

Elles sont soit basées sur la recherche locale, à titre d'exemple, la recherche tabou, la recherche locale itérée (ILS) ou à base de population telles que les algorithmes génétiques et les algorithmes de colonies de fourmis.

La recherche locale est une procédure de recherche itérative qui, à partir d'une solution initiale, l'améliore progressivement en appliquant une série de modifications (ou mouvements) locales. On définit une structure de voisinage qui consiste à spécifier un voisinage pour chaque solution. En effet, à chaque itération, la recherche obtient une nouvelle solution réalisable légèrement différente de la solution courante qui la remplace si elle est meilleure. La recherche se termine si un optimum local est rencontré. L'inconvénient de cette méthode est que cet optimum local n'est pas toujours de bonne qualité. Pour faire face à cette limitation, des méthodes basées sur recherche locale, plus sophistiquées sont développées afin d'échapper aux minima locaux et améliorer la solution efficacement. Les méthodes les plus connues sont le recuit simulé (Kirkpatrick *et al.* (1983)), la recherche tabou (Glover (1989), Glover (1990)).

Contrairement aux méthodes de recherche locale qui font intervenir une solution unique, les méthodes à base de population, connues aussi sous l'appellation d'algorithmes évolutionnistes travaillent sur un groupe de solutions admissibles (population) à chacune des étapes du processus de recherche dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche. Après avoir généré une population initiale de solutions, une méthode évolutive tente d'améliorer la qualité moyenne de la population courante selon une fonction évaluation dite *fitness* en ayant recours à des principes d'évolution naturelle; la sélection des meilleurs candidats selon un critère préétabli. Les mutations/croisements seront également réalisés pour générer de nouveaux candidats. Les algorithmes évolutionnistes dont les algorithmes génétiques sont des algorithmes fortement paramétrables : taille de la population, nombre de générations limite, taux de croisement et de mutation, type d'opérateur de sélection, de croisement, de mutation et de remplacement. La méthode la plus connue est l'algorithme génétique (Holland (1975)). Une autre méthode à base de population est l'algorithme de colonies de fourmi introduit initialement par Dorigo & Gambardella (1996) pour la résolution du problème du voyageur de commerce, puis repris pour plusieurs autres domaines d'application.

Les heuristiques sont des méthodes empiriques basées sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de ces méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum, tout en essayant de l'obtenir en un temps de calcul raisonnable.

Ce sont des méthodes souvent utilisées pour avoir rapidement des solutions admissibles (réalisables) de qualité relativement bonne, bien que, en générale leur performance ne puisse être garantie.

- **Heuristiques de construction** : Ces heuristiques construisent des solutions selon des critères d'optimisation locaux à partir des données initiales. C'est-à-dire Elle déterminent une solution selon une règle de construction donnée. La spécificité de ce genre d'heuristiques est que le choix effectué à chaque itératinos n'est pas remis en cause. Également, un bon choix améliorant une solution partielle ne conduit pas nécessairement à une amélioration au niveau global. Une des méthodes constructives développée pour la recherche d'une séquence de durée minimale sur un atelier de type flow shop à plusieurs machines est l'algorithme NEH (Nawaz *et al.* (1983)).
- **Les règles de priorité** : Une méthode basée sur une règle de priorité permet d'obtenir de bons ordonnancements en un temps raisonnable. Nous citons entre autres,
 - La règle(*SPT*) (Shortest Processing Time) : Séquencer les opérations par ordre croissant des p_i .
 - La règle (*WSPT*) (Weighted Shortest Processing Time) : Séquencer les opérations par ordre croissant du ratio $\frac{p_i}{w_i}$.
 - La règle (*LPT*) (Longest Processing Time) : Séquencer les opérations par ordre décroissant des p_i .
 - La règle (*EDD*) (Earliest Due Date) : Séquencer les opérations par ordre croissant des d_i .

Deux critères caractérisent une heuristique ou un algorithme d'approximation : le temps d'exécution (en temps polynomial), la qualité de la solution obtenue par rapport à la solution optimale. En d'autres termes, même pour la pire instance, une solution de qualité est une solution proche de l'optimum. D'autre part, un algorithme a une garantie de performance s'il peut garantir a priori, indépendamment de la difficulté de l'instance d'entrée, que la solution est de bonne qualité par rapport à l'optimum.

Soit H un algorithme d'approximation pour un problème (nous prenons le cas de minimisation) P , une instance I du problème P . Soit la valeur objective de la solution

générée par l'algorithme d'approximation H est $H(I)$ et la valeur optimale est $OPT(I)$. Pour définir le ratio au pire cas de H en supposant que $OPT(I) > 0$, nous définissons le ratio mesurant la qualité de la solution approximative de l'instance I : $R(I) = \frac{H(I)}{OPT(I)}$.

On dit que l'algorithme H a une garantie de performance α si pour toute instance I , $R(I) \leq \alpha$. c'est-à-dire que pour toute instance I de P la valeur objective calculée par H est au plus α fois la valeur objective optimale ($H(I) \leq \alpha \cdot OPT(I)$). Autrement, H est appelé α -approximation s'il a une garantie de performance de α dans un temps polynomial, pour toute instance I de P .

Étant donné un algorithme H , on cherche à trouver le α le plus petit qui satisfait cette condition. Ceci donne lieu à la définition du ration au pire cas (RPC) pour l'algorithme H ;

$$RPC = \inf\{\alpha \geq 1 | H(I) \leq \alpha \cdot OPT(I) \quad \forall I \in P\}.$$

Autrement ;

$$RPC = \sup_{I \in P} \frac{H(I)}{OPT(I)}.$$

L'erreur relative ϵ de H pour une instance I du problème P (toujours de minimisation) est définit comme $\epsilon = \frac{H(I) - OPT(I)}{OPT(I)}$

L'erreur au pire cas (EPC) de H est donnée par :

$$EPC = \sup_{I \in P} \frac{H(I) - OPT(I)}{OPT(I)}.$$

1.3 Conclusion

Nous avons présenté dans ce chapitre quelques notions globales concernant les problèmes d'ordonnancement d'ateliers de production. Nous présentons ensuite les notations utilisées permettant de caractériser un problème d'ordonnancement ainsi que le différents types d'ateliers d'ordonnancement. Nous avons également rappelé les principes de la théorie de complexité. Finalement, une brève description des méthodes de résolution exactes et approchées les plus utilisées dans la littérature est présentée.

Le prochain chapitre focalise en particulier sur les problèmes d'ordonnancement avec

contraintes de disponibilité. Nous organisons l'état de l'art selon le type d'atelier et également une classification est faite selon le comportement de la période d'indisponibilité par rapport à l'exécution des tâches.

Chapitre 2

Problèmes d'ordonnancement avec contraintes de disponibilité : État de l'art

Résumé :

Nous dressons dans ce chapitre une étude bibliographique, portant sur les différents travaux dédiés au problème d'ordonnancement avec contraintes de disponibilité dans le contexte déterministe. Des références bibliographiques sont introduites pour décrire la problématique et les classifications et variantes proposées dans la littérature. Nous classifions les travaux selon le type d'atelier d'ordonnancement et également selon le type de contrainte de disponibilité.

2.1 Introduction

Le problème d'ordonnancement classique suppose que les machines sont disponibles pendant l'horizon de planification. Cette hypothèse peut être justifiée dans certaines situations, mais elle ne s'applique pas si des exigences de maintenance, des pannes de machines ou d'autres contraintes de disponibilité sont éventuellement prises en compte pendant des périodes de temps.

Ainsi, deux types de contraintes d'indisponibilité sont possibles. L'une concerne les indisponibilités de ressources aussi bien matérielles qu'humaines, planifiées ou connues à l'avance (contraintes déterministes), pour diverses raisons, à titre d'exemple, des activités de maintenance préventive, de nettoyage ou également absentéisme de personnel. Dans ce cas, les dates et les durées des périodes d'indisponibilité sont connues à l'avance. La machine peut être indisponible pour d'autres raisons imprévisibles telles que une panne de machine, une rupture de stock et bien d'autres raisons (contraintes non-déterministe).

Ce chapitre est consacré à la présentation des principaux résultats de la littérature sur les problèmes d'ordonnancement prenant en compte les contraintes de disponibilité des machines dans le cas déterministe.

2.2 Définitions

Le problème d'ordonnancement du job shop avec les contraintes de disponibilité a été étudié dans différents cas :

Lee (1999) a introduit trois types d'indisponibilité. Une opération *Sécable* est lorsque son traitement peut être interrompu par une période d'indisponibilité et reprise lorsque la machine est de nouveau disponible. Une opération *Non-sécable* définit le cas où son traitement doit être repris complètement depuis le début, après l'interruption par la période d'indisponibilité. Enfin, le cas d'une opération *semi-sécable* ne se terminant pas avant la période d'indisponibilité d'une machine, elle devrait être reprise partiellement aussitôt que la machine qui l'exécute est de nouveau disponible.

Aggoune (2002) définit le problème d'ordonnancement avec contraintes de disponibilité comme étant strictement *non-preemptif*. Dans ce cas, le traitement d'une opération ne peut être interrompu ni par une autre opération ni par une période d'indisponibilité. Ce cas est différent du cas *Non-sécable* du fait que lorsque l'opération ne peut être effectuée avant la

période d'indisponibilité, elle doit commencer et se terminer après.

Une autre classification est proposée par Mauguière *et al.* (2005). Une période d'indisponibilité qui permet d'interrompre une opération et de la reprendre après une période donnée est appelée période *franchissable* (crossable). Une période d'indisponibilité qui empêche l'interruption de toute opération, même si les opérations sont sécables, est appelée période *non-franchissable* (non-crossable).

Ma *et al.* (2010) ont déclaré qu'il existe deux cas d'ordonnancements préemptifs. Le premier cas est lorsqu'une opération interrompue est reprise par une seule machine. Ce cas est connu sous le nom de *préemption d'opération*. Le deuxième cas est quand l'opération interrompue est déplacée vers une autre machine ou reprise plus tard, ceci est connu comme *préemption arbitraire*. S'il n'y a qu'une seule machine dédiée à chaque opération, les *préemptions d'opération* et les *préemptions arbitraires* deviennent équivalentes. donc, pour les problèmes d'ordonnancement à une machine, flow shop et job shop, cette différence n'est pas requise.

Une autre classification est basée sur les données existant. On distingue donc deux types de problèmes; Si des informations complètes sur les données de la tâche avant le commencement de l'ordonnancement sont connues à l'avance, le problème est dit statique (Off-Line). Un problème d'ordonnancement dynamique (On-Line) nécessite l'ordonnancement des tâches en temps réel sans aucune connaissance des données dans le futur.

Nous dresserons un bref état de l'art sur le problème d'ordonnancement avec contraintes de disponibilité et classerons les articles cités selon les cas mentionnés ci-dessus.

2.3 Problèmes d'ordonnancement avec contraintes de disponibilité

2.3.1 Problème d'ordonnancement à une seule machine

2.3.1.1 Cas non-préemptif

Souissi (2005) a étudié le problème $1||\sum w_i C_i$ avec des périodes d'indisponibilités *non-préemptives*. Des bornes inférieures et des propriétés mathématiques du problème ont été démontrées. Une modélisation linéaire en nombres entiers, une procédure par séparation

et évaluation et une méthode de programmation dynamique ont également été proposées. Les tests ont montré que la méthode de programmation dynamique est plus efficace que la procédure par séparation et évaluation qui reste meilleure que le programme linéaire en nombres entiers.

Sadfi *et al.* (2005) ont proposé un algorithme d'approximation, avec une erreur au pire cas de $\frac{3}{17}$, pour résoudre le problème $1||C_{max}$ avec une seule période de maintenance.

Kacem *et al.* (2008) ont étudié le même problème que Souissi (2005) avec une période d'indisponibilité. Les auteurs ont conçu trois méthodes exactes pour résoudre ce problème : une procédure par séparation et évaluation basée sur des propriétés et bornes inférieures, un modèle de programmation linéaire mixte et une méthode de programmation dynamique. Ces approches résolvent des problèmes de 3000 tâches dans un temps de calcul raisonnable. Les résultats numériques ont montré, en outre, la complémentarité de la méthode de programmation dynamique et de la procédure par séparation et évaluation.

Pour le même problème, Kacem & Chu (2008a) ont proposé une autre procédure par séparation et évaluation, basée sur un ensemble des bornes inférieures et d'heuristiques améliorées. Des tests effectués ont démontré que l'algorithme capable de résoudre des instances de 6000 tâches dans un temps de calcul raisonnable.

Lee & Kim (2012) s'intéressent au problème d'ordonnancement d'une seule machine nécessitant une maintenance périodique avec objectif de minimiser le nombre de tâches en retard. Ils ont présenté une heuristique en deux phases ; une solution initiale est obtenue d'abord par une modification de l'algorithme de Moore (Moore (1968)) pour le problème sans maintenance, puis améliorée dans une seconde phase.

Vahedi-Nouri *et al.* (2013) ont considéré le problème avec l'effet d'apprentissage et plusieurs contraintes de disponibilité minimisant le temps total d'exécution. Une modélisation mathématique et une procédure par séparation et évaluation ont été proposées. En plus de résoudre le problème de manière optimale, comme le problème est *NP-difficile* au sens fort, deux méta-heuristiques basés sur l'algorithme génétique et le recuit simulé ont été développés afin obtenir des solutions de bonne qualité pour les problèmes de grande taille.

Benmansour *et al.* (2014) ont traité le problème d'ordonnancement d'un ensemble de tâches sur une machine en fonction d'une date d'échéance commune et restrictive. En particulier, ils s'intéressent au problème de la minimisation de la somme pondérée de l'avance et du retard maximal. Ce type de fonction objectif est lié à l'optimisation juste-à-temps où les pénalités, telles que les coûts de stockage et les frais supplémentaires pour les retards de livraison, doivent être évitées. Ils présentent un modèle linéaire en nombres entiers mixtes

pour résoudre le problème à l'optimalité. Bien que ce dernier problème puisse être résolu à l'optimalité pour les petites instances, les auteurs ont montré que le problème se réduit au problème de sac à dos unidimensionnel.

Yin *et al.* (2016) ont considéré le problème d'ordonnancement de plusieurs tâches indépendantes et simultanément disponibles sur une seule machine ayant une activité de maintenance fixe. L'objectif est de trouver la séquence de la tâche optimale pour minimiser le montant total des tâches en retard. Ils ont développé deux algorithmes de programmation dynamique pseudo-polynomiaux et un schéma d'approximation polynomial pour le problème.

Low *et al.* (2016) ont abordé le problème dans le but de minimiser le total du retard et d'avance. Deux méthodes exactes ont été proposées pour résoudre le problème : un modèle linéaire en nombres entiers mixtes et une méthode basée sur la programmation dynamique. Les expériences numériques ont montré que la méthode de programmation dynamique est bien meilleure que la programmation linéaire en nombres entiers mixtes et que son avantage augmente avec l'augmentation de la taille des problèmes.

Wan & Yuan (2018) ont étudié l'ordonnancement *non-préemptif* sur une seule machine avec une période d'indisponibilité de l'opérateur afin de minimiser le temps total d'achèvement pondéré. Les auteurs ont présenté un algorithme de temps pseudo-polynomial et un schéma d'approximation de temps entièrement polynomiale (FPTAS). Shabtay & Zofi (2018) ont étudié le même problème avec les temps de traitement des tâches contrôlables et une période d'indisponibilité de machine fixe pour la minimisation du makespan. Une autre hypothèse concerne le temps de traitement de la tâche qui est une fonction décroissante convexe de la quantité de ressource allouée à son opération de traitement. Ils ont supposé en outre qu'il y avait une restriction budgétaire sur le coût total d'allocation des ressources. Ils ont prouvé que le problème est *NP-difficile* et ont développé un algorithme d'approximation à facteur constant et un schéma d'approximation de temps entièrement polynomiale (FPTAS) pour le résoudre. Le FPTAS est obtenu en dépit du fait qu'ils ne pouvaient pas concevoir un algorithme en temps pseudo-polynomial pour trouver la solution optimale.

2.3.1.2 Cas sécable

Lee (1996) a étudié le problème d'une seule machine pour différents critères et identifié certains problèmes polynomiaux. En particulier, il a montré que le makespan (C_{max}) pour un problème d'une seule machine avec une période de disponibilité sécable $1|rs|C_{max}$, minimisé par toute séquence arbitraire est optimal. De plus, l'auteur a montré que le séquençement des opérations sous la règle *SPT* permettent de résoudre de manière optimale le problème

$1|rs|\sum C_i$. De même, les $1|rs|L_{max}$ est résolu à l'optimal par l'algorithme *EDD* (Earliest Due Date), où les tâches sont séquencées dans l'ordre décroissant de leurs dates d'échéance. Le problème $1|rs|\sum U_i$ est aussi résolu de façon optimale dans un temps $O(n\log n)$ par une modification de la règle de *Moore – Hodgson* avec n le nombre de tâches considérées.

Le problème $1||\sum w_i C_i$ (sans contraintes de disponibilité des machines) résolu par la règle (*WSPT*), devient *NP-difficile* au sens faible même si $w_i = p_i$ pour tout $i = 1, \dots, n$, quand des contraintes de disponibilité sont introduites. En outre, Lee (1996) a fourni un algorithme de programmation dynamique, et plusieurs heuristiques avec l'analyse au pire cas pour résoudre le problème d'une manière approchée.

Kacem & Chu (2008b) ont étudié les $1|rs|\sum w_i C_i$ avec une période d'indisponibilité sur la machine. De nouvelles propriétés concernant la performance au pire cas de l'heuristique *WSPT* et une approximation plus serrée (tight) de l'erreur au pire cas ont été proposées. La borne au pire cas est égale à 2 dans certaines conditions. Les résultats obtenus complètent ceux de Lee (1996).

Kacem *et al.* (2015) ont prouvé que le problème de maximisation du nombre pondéré de des tâches en avance sur une seule machine avec des contraintes de non-disponibilité ne peut pas admettre un FPTAS même si toutes les dates d'échéance sont égales et qu'une seule période de non-disponibilité est considérée. Néanmoins, ils ont montré dans ce cas que le problème admet un PTAS pour un nombre constant de périodes d'indisponibilité avec dates d'échéance arbitraires.

Luo *et al.* (2015) ont étudié le problème d'ordonnancement de l'activité de maintenance variable et de tâches sur une machine unique, où l'activité de maintenance doit débuter avant un délai donné, et sa durée est une fonction positive et non décroissante de sa date de départ. C'est à dire, plus tard l'activité de maintenance commence, plus sa durée est longue. Ils ont montré que les problèmes pour minimiser le makespan, la somme des temps d'achèvement, le retard maximum, et le nombre de tâches tardives sont tous solvables en temps polynomial.

Cui & Lu (2017) ont abordé le problème avec des périodes de maintenance préventive périodiques flexibles, où les dates de disponibilité des tâches sont également prises en compte. Il ont prouvé que le problème peut être résolu en temps polynomial avec la règle "*Earliest Release Date*".

2.3.1.3 Cas non-sécable

Adiri *et al.* (1989) ont traité le problème de machine unique avec une contrainte de

disponibilité non-sécable dans laquelle la machine est sujette à des pannes, avec objectif de minimiser le temps séjour en espérance mathématique. Les contextes déterministe et stochastique sur l'indisponibilité ont été considérés. Dans le cas déterministe, ils ont prouvé que le problème est *NP-difficile* au sens faible. Si la fonction de répartition sur le temps est concave, alors la méthode *SPT* minimise asymptotiquement le temps de séjour en espérance mathématique. Dans le cas de pannes multiples, la méthode *SPT* minimise le critère lorsque les dates de panne sont exponentiellement distribuées.

Lee (1996) a montré que le problème $1|nrs|C_{max}$ est *NP-difficile* au sens faible quand une période d'indisponibilité est considérée. L'auteur a démontré que l'algorithme *LPT* a une erreur relative serrée égale à $1/3$. L'auteur a déjà montré que l'algorithme *SPT* a une erreur relative au pire cas $2/7$ qui est serrée (Lee & Liman (1992)).

Lee (1996) a montré aussi que $1|nrs|L_{max}$, $1|nrs|\sum U_i$ et $1|nrs|\sum_i w_i C_i$ sont *NP-difficile* au sens faible. Le premier problème est résolu par un algorithme *EDD* avec une erreur relative égale à p_{max} (temps de traitement de la plus longue tâche). Le second problème est résolu par la règle de *Moore – Hodgson* avec une erreur relative égale à 1. Pour le troisième problème, l'auteur a montré que le ratio de performance de l'algorithme *WSPT* peut être arbitrairement élevé, même si $w_i = p_i$ pour tout $i = 1, \dots, n$.

Sadfi (2002) s'est intéressé au même problème que Lee & Liman (1992). Il a développé l'heuristique *MSPT* (*SPT* modifiée) avec une garantie de performance de $\frac{19}{17}$. La règle *MSPT* consiste à améliorer le résultat donné par *SPT* en inversant une tâche ordonnée avant la période d'indisponibilité avec une autre tâche séquencée après. L'auteur a également développé un algorithme de programmation dynamique de complexité pseudo-polynomiale conduisant à la solution optimale du problème.

Wang & Cheng (2007a) ont fourni un algorithme polynomial pour minimiser l'heure d'arrivée du dernier lot de livraison au centre de distribution en production par lots sur une seule machine. Ils ont d'abord montré que le problème est *NP-difficile*. Puis une heuristique avec une erreur au pire cas de $1/2$ est proposée et ils ont montré qu'elle est serrée.

Gawiejnowicz (2007) a développé un algorithme pour minimiser C_{max} résolvant le problème d'une seule machine avec n tâches détériorées et k périodes d'indisponibilité ($1 \leq k < n$). L'auteur a prouvé que le problème est *NP-difficile* s'il n'y a qu'une seule période d'indisponibilité; sinon, il est *NP-difficile* au sens fort.

Chen (2007) a considéré un problème d'ordonnancement de maintenance périodique sur une machine dans une entreprise textile. L'auteur a développé une heuristique quasi-optimale

et une procédure par séparation et évaluation pour minimiser T_{max} . Chen (2009) a étudié le même problème d'ordonnancement qui minimise le nombre de tâches en retard. Une heuristique basée sur l'algorithme de *Moore* a été développée pour fournir un ordonnancement approché quasi-optimal. La performance de l'heuristique a été évaluée en comparant sa solution avec la solution optimale obtenue par la procédure par séparation et évaluation. Des propriétés associées au problème ont été implémentés dans l'algorithme.

Yang *et al.* (2011) ont étudié le problème d'ordonnancement à une machine avec une période d'indisponibilité dédiée à la maintenance pour objectif de minimiser la somme des dates de fin. Des propriétés d'optimalité ont été démontrées. Ils ont montré que l'algorithme (*SPT*) proposé est optimal dans le cas non-sécable, sous certaines conditions.

Kacem & Paschos (2013) ont traité la minimisation de la somme pondérée des dates de fin sur une seule machine avec un intervalle de d'indisponibilité fixe. L'analyse montre que la règle de temps de traitement pondéré le plus court (*WSPT*) ne peut pas offrir une approximation différentielle pour le problème considéré dans le cas général. Néanmoins, une légère modification de cette règle fournit une approximation avec un ratio différentiel de $\frac{3 - \sqrt{5}}{2}$.

Le problème d'ordonnancement d'une seule machine avec un intervalle de non-disponibilité imprévisible, minimisant le retard maximal est traité par Kacem *et al.* (2014). Deux modèles possibles ont été proposés. Dans les deux cas, l'heure de début et la durée de l'intervalle de non-disponibilité sont inconnues. Ils ont montré que la règle de Jackson peut donner une 2-approximation serrée pour les deux modèles.

Kacem *et al.* (2015) ont considéré la maximisation du nombre pondéré de jobs en avance sur une machine avec des contraintes d'indisponibilité. Ils ont démontré que le problème ne peut pas admettre un FPTAS même lorsque toutes les dates d'échéance sont égales avec une seule période d'indisponibilité. Néanmoins, ils ont montré dans ce cas qu'il admet un schéma d'approximation temporelle polynomiale (PTAS) pour un nombre constant de période d'indisponibilité et de dates d'échéance arbitraires.

Hfaiedh *et al.* (2015) ont proposé un algorithme exact pour résoudre le problème de planification d'une seule machine sous une contrainte d'indisponibilité avec des dates de latence et de disponibilité pour la minimisation du délai de livraison maximum. Le problème étant *NP-difficile* au sens fort, une procédure par séparation et évaluation a été proposée utilisant l'algorithme de Jackson préemptif avec des contraintes de précedence pour calculer la borne inférieure.

Liu *et al.* (2016) ont étudié le même problème avec des périodes de maintenance périodique pour la minimisation du nombre de tâche en retard. Le problème a été démontré *NP-difficile* et une procédure par séparation et évaluation employant des bornes inférieures développées et plusieurs propriétés de dominance.

Zhao & Hsu (2017) ont exploré les problèmes d'ordonnancement avec des tâches en détérioration et des contraintes fixes de disponibilité de la machine. Il a été supposé que les tâches ont des temps de traitement différents avec le même taux de détérioration. L'objectif est de minimiser le temps total d'achèvement. Ils ont présenté un algorithme de programmation dynamique pseudo-polynomial et un schéma d'approximation entièrement polynomiale.

Cui & Lu (2017) ont abordé le problème avec des maintenances préventives périodiques (PM) flexibles, où les dates de disponibilité des tâches sont également prises en compte. Pour le cas non-sécable, les auteurs ont prouvé qu'il est *NP-difficile* au sens fort. Un modèle mathématique de programmation linéaire en nombres entiers mixtes (mixed integer programming) a été fourni. Ensuite, une heuristique *ERD-LPT* a été proposée. Une procédure par séparation et évaluation utilisant des règles de dominance ont été également opposée pour la résolution optimale des problèmes de petite à moyenne taille.

2.3.1.4 Cas sécable/non-sécable

Pour résoudre le problème $1, cr|rs/nrs| \max C_i + q_i$, Mauguière *et al.* (2003b) ont proposé une procédure par séparation et évaluation. La plupart des instances allant jusqu'à 100 opérations ont été résolues, bien que certaines instances de plus petites tailles sont non résolues.

Mauguière *et al.* (2003a) ont aussi développé une procédure par séparation et évaluation pour résoudre le problème $1, cr/ncr|r_i, rs/nrs, d_i| \max(C_i + q_i)$. Une autre méthode de résolution a été proposée par Mauguière *et al.* (2005). Le problème $1|pmtn, r_i, d_i, q_i| \max(C_i + l_i)$ *NP-difficile* au sens fort a été résolu par un algorithme d'approximation. Comme les problèmes avec $rs, cr/nc|rs, nrs, cr|rs/nrs$ et $pmtn$ sont des cas particuliers du problème avec $cr/nc|rs/nrs$, le $1, cr/ncr|r_i, rs/nrs, d_i| \max(C_i + q_i)$ sont aussi *NP-difficile* et résolus approximativement dans un temps raisonnable.

2.3.1.5 Cas semi-sécable

Detienne (2012) a étudié le problème de la minimisation du nombre pondéré de tâches en retard sur une machine soumise à des contraintes de disponibilité. L'auteur a considéré le cas des tâches semi-sécables et a montré que le problème est équivalent à un problème similaire sans contraintes de disponibilité, où les temps de traitement des tâches sont en fonction de leurs dates de début. Un modèle de programmation linéaire mixte est proposé pour la résolution optimale du problème.

2.3.2 Problème d'ordonnancement à machines parallèles

2.3.2.1 Cas non-préemptif

Xu *et al.* (2008) ont considéré un problème d'ordonnancement de deux machines parallèles où une machine est périodiquement indisponible avec l'objectif de minimiser le makespan. Il a été montré que le ratio au pire cas de la règle *LPT* classique et de l'algorithme de liste sont de $3/2$ et 2 respectivement, pour la version hors ligne et en ligne du problème.

Mellouli *et al.* (2009) ont considéré le problème $P||\sum C_i$ avec une période de maintenance planifiée sur chaque machine. Trois méthodes exactes (méthode de programmation linéaire en nombres entiers mixtes, une méthode de la programmation dynamique et procédure par séparation et évaluation) et des heuristiques constructives ont été proposées. Les auteurs ont fourni en outre des propriétés de dominance, une borne inférieure et deux schémas de branchement pour la procédure par séparation et évaluation.

Tan *et al.* (2011) ont traité le même problème avec des périodes indisponibles sur chacune des k premières machines, où $1 \leq k \leq 6$. Les auteurs ont montré que l'algorithme classique *SPT* a un ratio au pire cas égal à $1 + \frac{m-1}{m-k}$ quand $k < m$. De plus, ils ont prouvé que s'il y a exactement une période indisponible sur chacune des k premières machines, et que les périodes indisponibles ne se chevauchent pas, le pire cas de *SPT* est au plus de ratio $2 + \frac{k-1}{m-1}$ et aucun algorithme d'approximation de temps polynomial avec un ratio au pire des cas inférieur à $\frac{m}{m-1}$ ne peut exister quand $k = m$ sauf si $P = NP$.

Dans le cas de deux machines parallèles, Xu & Yang (2013) ont proposé un modèle de programmation mathématique où une machine est indisponible dans un intervalle de temps pour la minimisation du makespan. Les analyses de performance de l'algorithme (*LPT*) et de l'ordonnancement de liste (*LS*) sont présentées. Les résultats numériques montrent que l'algorithme *LPT* est plus efficace que l'algorithme *LS* et que l'erreur moyenne de

l’algorithme *LPT* est inférieure à 2% lorsque le nombre d’emplois est supérieur à vingt.

Wang *et al.* (2014) ont montré que le problème de la minimisation du temps total d’achèvement avec la détérioration des activités de maintenance sur des machines parallèles indépendantes pourrait être résolu de façon optimale par un algorithme de complexité $O(nm + 3)$. L’application d’une méthode d’analyse similaire au problème de minimisation de la charge totale permet également d’obtenir les mêmes résultats.

Wang & Cheng (2015) ont développé une heuristique en intégrant la stratégies d’ajustement vers l’arrière et la stratégie *two-step look-ahead* dans certaines heuristiques existantes pour des problèmes similaires sans la contrainte de disponibilité de la machine. Ils ont montré que l’heuristique proposée a un ratio au pire cas $4/3$ et que la borne est serrée.

Dans Costa *et al.* (2016), le problème d’ordonnancement de machines parallèles identiques avec des changements d’outils périodiques dus à l’usure est traité pour minimiser le temps d’achèvement total. Un modèle de programmation linéaire en nombres entiers mixtes (MILP) a été développé pour la résolution optimale pour les cas de test de petite taille (le problème est *NP-difficile* au sens fort). En outre, une méta-heuristique hybride basée sur des algorithmes génétiques a été spécifiquement conçue pour résoudre les instances de tailles importantes.

2.3.2.2 Cas sécable

Schmidt (1984) a étudié le problème $P_m|prmp,rs|C_{max}$. L’auteur a proposé un algorithme de complexité $O(n + m\log(n))$ pour un ordonnancement sécable réalisable dans le cas où toutes les machines sont disponibles pendant un nombre arbitraire de périodes. Dans Schmidt (1988), les dates de disponibilité et de fin souhaitée sont prises en compte. Il a été prouvé que le problème est solvable en $O(n\log(nm))$ étapes. Quand aucune date de disponibilité n’est imposée, la minimisation du plus grand délai peut être obtenue dans un temps proportionnel à $O(nm\log(n))$.

Lee (1991) s’est intéressé à la minimisation du makespan dans le problème des machines parallèles identiques lorsque ces machines ne sont pas toutes disponibles à l’instant zéro. L’auteur a proposé une heuristique basée sur la règle *LPT* avec une erreur relative de $1/2$, puis une amélioration de cet algorithme avec une erreur égale à $1/3$.

Lee (1996) a déclaré que le problème $P_m|rs|C_{max}$ est une extension du problème *NP-difficile* $P_m||C_{max}$. Deux méthodes d’approximation basées sur la règle *LPT* ont été développées. L’auteur a également développé un algorithme de programmation dynamique

pour résoudre de façon optimale le problème $P_2|rs|\sum_i w_i C_i$ considéré par Kaspi & Montreuil (1988). Selon ces derniers, l'ordonnancement des tâches avec la règle *SPT* constituent un ordre optimal pour ce problème si les indisponibilités commencent à l'instant 0.

Lin *et al.* (1998) ont étudié la maximisation de la plus petite date fin des tâches dans un environnement de m machines parallèles indisponibles à l'instant zéro. Les auteurs ont montré que la règle *LPT* a une erreur au pire cas égale à $\frac{2m-1}{3m-2}$.

Sheen & Liao (2007) ont envisagé de minimiser le retard maximal pour le problème d'ordonnancement de n tâches sur m machines parallèles identiques sous contraintes de disponibilité de machine et d'éligibilité. Il a été prouvé que la complexité de l'algorithme est $O((n + (2n + 2K))^3 \log(UB - LB))$, où K est le nombre total de périodes de disponibilité sur toutes les machines, et des bornes inférieures et supérieures ont été fournies pour la résolution optimale du problème.

Wang & Cheng (2007a) ont proposé une heuristique pour minimiser la date d'arrivée du dernier lot de livraison au centre de distribution, pour le problème de production par lots sur deux machines parallèles, dans le cas où une seule machine subit une période d'indisponibilité. Cette heuristique a une erreur au pire des cas de $2/3$.

Le problème de la minimisation de makespan pour l'ordonnancement des machines parallèles avec plusieurs périodes d'indisponibilité planifiées est considéré dans Hashemian *et al.* (2014). Le problème est d'abord formulé en tant que modèle de programmation linéaire en nombres entiers mixtes et résolu de façon optimale pour des problèmes de taille petite à modérément grande avec des contraintes de disponibilité multiples sur toutes les machines.

Liu & Lu (2016) ont étudié le problème d'ordonnancement qui considère à la fois la production et la livraison avec contraintes de disponibilité de la machine. Deux machines parallèles sont considérées, où une machine n'est pas disponible pendant une période donnée. Un seul véhicule est disponible pour livrer les tâches dans un délai de transport fixe à un centre de distribution. Le véhicule peut charger la plupart des tâches en tant que lot de livraison par trajet en raison de la contrainte de capacité du véhicule. L'objectif est de minimiser le temps de livraison de tous les tâches. Les auteurs ont considéré les cas sécables et non sécables. Pour chaque cas, ils proposent un algorithme d'approximation avec un ratio au pire cas borné à $3/2$.

2.3.2.3 Cas non-sécable

Lee & Liman (1993) ont traité le problème à deux machines parallèles pour minimiser

la somme des dates de fin des tâches, en supposant que l'une de ces machines n'est pas disponible en permanence. Les auteurs ont montré que le problème étudié est *NP-difficile* au sens faible et ont proposé un algorithme de programmation dynamique pseudo-polynomial, et une heuristique basée sur *SPT* ayant une erreur au pire cas égale à $1/2$.

Mosheiov (1994) a étudié le même problème en supposant que chaque machine n'est disponible que pour une période donnée. L'auteur a développé une heuristique et une borne inférieure basées sur *SPT*. Les deux heuristiques sont asymptotiquement optimaux à mesure que le nombre d'emplois augmente.

Lee (1996) a prouvé que le problème $P_m|nrs|C_{max}$ est *NP-difficile*. Les performances de deux règles heuristiques *SPT* et l'ordonnancement de liste *SL* sont également analysées. Les algorithmes *SPT* et *SL* ont respectivement des ratios au pire cas égaux à $\frac{m+1}{2}$ et m respectivement. L'auteur a également montré que $P_2|nrs|\sum_i w_i C_i$ est *NP-difficile*. Un algorithme de programmation dynamique a été développé pour résoudre efficacement le problème lorsque $w_i = 1$ avec la première machine disponible en permanence.

Zhao *et al.* (2011) ont considéré deux problèmes d'ordonnancement de machines parallèles lorsqu'une machine n'est pas disponible dans une période donnée fixe et connue à l'avance pour minimiser le temps total d'achèvement pondéré ($P_2|nrs|\sum_i w_i C_i$). Comme le problème est connu pour être *NP-difficile*, ils ont fourni un schéma d'approximation polynomial (FPTAS). L'étude a été généralisée pour le cas de m machines parallèles et un schéma d'approximation polynomial-temps est présenté. L'algorithme peut être étendu à un modèle de machines uniformes.

Tan *et al.* (2013) ont considéré le même problème que celui de Zhao *et al.* (2011). Ils ont prouvé que *SPT* a un ratio serré de $3/2$, s'il y a une période de non-disponibilité sur l'une des deux machines. Il a été prouvé également que *SPT* a un ratio de 2, s'il y a une période indisponible sur chaque machine, et que les périodes indisponibles sur deux machines ne se chevauchent pas, le petit ratio de l'algorithme à temps polynomial sauf si $P = NP$.

Zhao & Tang (2014) ont considéré le problème d'ordonnancement de machines parallèles avec le temps de traitement de la tâche est une fonction proportionnelle à son heure de début (tâches en détérioration) et chaque machine n'est pas disponible dans une période de temps spécifiée pour minimiser la somme pondérée des temps d'achèvement. Ils ont montré que le cas général du problème n'est pas approximable sauf si $P = NP$ et présenté un algorithme de programmation dynamique pseudo-polynomial. Ils ont présenté également un schéma d'approximation entièrement polynomial lorsque une seule machine est indisponible dans une période de temps spécifiée.

Beaton *et al.* (2016) ont traité le problème de la minimisation du makespan pour l'ordonnancement de machines parallèles avec des périodes de non disponibilité multiples pour les tâches non sécables, semi-sécables et sécables. Le problème a été formulé mathématiquement sous forme d'un modèle de programmation linéaire en nombres entiers mixtes et résolu de façon optimale de petites instances. Quatre heuristiques différentes ont été appliquées au problème pour la résolution approchée.

Zhao & Hsu (2017) ont examiné la version non-reprise du problème d'ordonnancement des tâches détériorées et des contraintes de disponibilité pour la minimisation le temps total d'achèvement. Ils ont présenté des algorithmes de programmation dynamique pseudo-polynomiaux pour les problèmes à une machine et à machines parallèles. Ils ont prouvé qu'il n'existe pas d'algorithme d'approximation en temps polynomial avec une borne constante au pire cas pour la version de machine parallèle, sauf si $P = NP$.

2.3.3 Problème d'ordonnancement de type Flow shop

2.3.3.1 Cas non-préemptif

Aggoune (2002) a proposé deux algorithmes pour résoudre le problème d'ordonnancement de type flow shop *non préemptif* avec contraintes de disponibilité (*FSPAC*), minimisant le critère du makespan. Le premier est basé sur un algorithme de liste, le second utilise l'approche géométrique (deux-jobs). Ces approches sont couplées à des méta-heuristiques pour améliorer leurs performances. Les résultats des tests ont montré que l'approche géométrique est meilleure que l'algorithme glouton. Le problème est résolu aussi par une procédure par séparation et évaluation basée sur des graphes disjonctifs avec borne inférieure utilisant l'approche géométrique à deux jobs avec des contraintes d'indisponibilité.

Cheng & Liu (2003) ont étudié le problème à deux machines sans attente (no-wait). Trois algorithmes d'approximation de borne $3/2$ sont proposés pour les cas où une période d'indisponibilité est imposée sur une seule machine et lorsque les périodes d'indisponibilité sur les deux machines se chevauchent.

Aggoune (2004a) a présenté deux variantes de *FSPAC*. Dans la première variante, les dates de début des tâches d'indisponibilité sont fixes tandis que dans la seconde sont flexibles sur des fenêtres de temps données. Une approche heuristique basée sur un algorithme génétique et une recherche taboue ont été proposées pour résoudre approximativement le problème de minimisation du makespan.

Aggoune & Portmann (2006) ont traité le même problème. Comme la minimisation du makespan est fortement *NP-difficile*, ils ont proposé une approche heuristique qui est une extension de l'approche géométrique développée pour le problème de l'ordonnancement de deux ateliers. Le but de cette heuristique est de résoudre approximativement le problème qui consiste à planifier les tâches deux par deux en fonction d'une séquence d'entrée, et en utilisant un algorithme polynomial. Les résultats numériques ont montré l'efficacité de l'approche proposée.

Yang *et al.* (2008) ont considéré un problème à deux machines avec une contrainte de disponibilité minimisant le makespan. Une période de temps constante est nécessaire pour entretenir la machine après avoir effectué au maximum un nombre fixe de tâches. Les auteurs ont proposé une heuristique pour résoudre ce problème. Les résultats ont montré que le pourcentage moyen des erreurs diminue à mesure que le nombre de tâches n augmente. D'où la capacité à résoudre des problèmes de grande taille.

Vahedi-Nouri *et al.* (2014) ont traité le problème de non-permutation, l'effet d'apprentissage sous contraintes de disponibilité. Les temps de disponibilité des tâches sont considérés. Il est supposé que le temps de traitement réel de chaque tâche sur une machine dépend de la position de cette tâche dans la séquence, et après le traitement d'un nombre spécifié de tâches sur chaque machine, une période d'indisponibilité se produit pour des raisons de maintenance. Selon ces hypothèses, un nouveau modèle de programmation linéaire mixte (MILP) a été proposé pour formuler le problème. Une méthode heuristique et un algorithme de recuit simulé ont été présentés pour la résolution approchée des problèmes de moyenne et de grande taille en raison de la complexité du problème.

Dans Gara-Ali & Espinouse (2015), les auteurs ont étudié le problème avec une période de maintenance avec une durée qui augmente avec le retard de son exécution. Ils ont montré que le problème étudié est *NP-difficile* et ont établi certaines conditions sur l'ordonnancement optimal. Une procédure par séparation et évaluation a été développée pour résoudre le problème de manière optimale.

Wang & Liu (2016) ont étudié un problème de l'ordonnancement intégré de la production et de la maintenance préventive (PM) avec délai de défaillance de chaque machine soumise à une distribution de probabilité Weibull. L'objectif est de trouver la séquence optimale des tâches et les décisions de maintenance optimales de sorte que le makespan soit minimisé. Quatre heuristiques basées sur l'algorithme génétique (GA) sont proposées. Les résultats numériques sur des grandes tailles de problèmes et différentes configurations indiquent les avantages potentiels de la solution d'ordonnancement intégrée et montrent également que

les heuristiques basées sur GA sont efficaces pour le problème étudié.

Lee & Kim (2017) ont traité un problème de type flow shop à deux machines dans lequel la machine nécessite des activités de maintenance préventive à démarrer dans une limite de temps de la tâche cumulée, après la période de maintenance qui la précède. Pour le problème de minimisation de retard total, ils ont développé des propriétés de dominance et des bornes inférieures ainsi qu'une heuristique et suggèrent une procédure par séparation et évaluation dans lequel ces propriétés, des bornes inférieures et l'heuristique sont employées.

Le problème d'ordonnancement des flux de permutation proportionnelle (*proportionate permutation flowshop scheduling problem*) de n tâches sur m machines, dans lequel les temps de traitement de chaque tâche sur toutes les machines sont identiques a été traité par Cheng *et al.* (2018). Ils ont considéré les activités de maintenance de durées égales et différentes. L'optimisation vise à minimiser le temps d'achèvement total, l'avance maximum, et leur retard maximum. Deux algorithmes d'optimisation $O(n^2m)$ pour résoudre ces problèmes ont été développés.

2.3.3.2 Cas sécable

Lee (1997) a étudié le problème $F2|rs|C_{max}$. Il a montré que le problème est *NP-difficile* au sens faible. Il a proposé des algorithmes pseudo-polynomiaux basés sur la programmation dynamique. De plus, l'auteur a développé une heuristique avec une erreur au pire cas de $1/2$ (respectivement $1/3$) lorsque la période d'indisponibilité se produit sur la première machine (respectivement la seconde). Il a démontré également que l'erreur relative obtenue en appliquant l'algorithme de Johnson est égale à 1 (respectivement $1/2$). Ce dernier résout le problème lorsqu'une ou les deux machines sont indisponibles à l'instant 0 comme démontré dans Lee (1999). Il a été démontré également que l'algorithme de Johnson est optimal lorsque les deux périodes d'indisponibilité ayant la même durée commencent au même temps. En outre, le problème est prouvé *NP-difficile* lorsque les machines ne sont pas disponibles à des dates différentes même si la durée des périodes d'indisponibilité est la même.

Pour le problème avec plusieurs indisponibilités sur chacune des deux machines, Błażewicz *et al.* (2001) ont analysé deux heuristiques constructives, l'algorithme de Johnson et l'heuristique *look-ahead*, et une heuristique basée sur le recuit simulé (SA). Ils ont conclu que l'heuristique basée sur ce dernier est plus efficace.

Kubiak *et al.* (2002) ont prouvé que les ordonnancements de permutation sont dominants pour le cas d'un atelier de type flow shop à deux machines *sécable*, avec plusieurs contraintes

de disponibilité sur les deux machines. Ils ont prouvé qu'aucune heuristique polynomiale avec une erreur au pire cas bornée ne peut exister quand au moins deux indisponibilités par machine sont considérées. Ils ont proposé des propriétés pour le problème, prouvé qu'il est *NP-difficile* au sens fort et une procédure par séparation et évaluation basée sur ces propriétés pour élaguer l'arbre de recherche. Les expérimentations numériques ont montré que les performances de l'algorithme de branchement et de liaison sont influencées par le nombre de périodes d'indisponibilité (définies comme des trous) considérées dans une instance.

Breit (2004) a étudié $F_2|rs|C_{max}$ avec une période d'indisponibilité sur la seconde machine. Un ratio au pire cas de $5/4$ est proposé améliorant ainsi le résultat de $4/3$ donné par Lee (1997).

Breit (2006) a développé un schéma d'approximation en temps polynomial pour résoudre le problème avec une période d'indisponibilité sur la première machine pour minimiser le critère du makespan. Cette approximation a été étendue pour résoudre le problème où la période d'indisponibilité est sur la deuxième machine.

Wang & Cheng (2007b) ont proposé deux heuristiques et ont montré que leurs bornes d'erreur au pire cas ne dépassaient pas $2/3$ le problème avec les temps de setup anticipés et une contrainte de disponibilité imposée à une seule machine.

Kubzin *et al.* (2009) ont proposé un algorithme d'approximation de performance garantie égale à $3/2$, pour résoudre le problème avec plusieurs intervalles de non-disponibilité sur la première machine.

2.3.3.3 Cas non-sécable

Cheng & Wang (1999) ont étudié le problème du flow shop à deux machines avec une contrainte de disponibilité imposée à chaque machine avec deux contraintes de disponibilité consécutives. Les auteurs ont fourni une heuristique pour le problème et ont montré qu'il y avait une erreur dans le cas le plus défavorable de $2/3$.

Lee (1999) a démontré que la règle de Johnson a une erreur relative égale à 1 lorsque la période d'indisponibilité est imposée sur la première machine ou la deuxième. Une autre heuristique garantie une erreur relative bornée à $1/2$ lorsque l'indisponibilité est considérée sur la seconde machine.

Espinouse *et al.* (1999) ont étudié le problème à deux machines sans attente (*no-wait*)

avec contraintes de disponibilité pour la minimisation du makespan. Les auteurs ont prouvé que le problème est *NP-difficile* même lorsque une seule période d'indisponibilité se produit sur l'une des machines, et *NP-difficile* au sens fort pour un nombre arbitraire de périodes d'indisponibilité. Ils ont également proposé des heuristiques basées sur l'algorithme de Gilmore and Gomory pour résoudre le problème avec une seule indisponibilité sur la première et la deuxième machine avec garantie de performance égale à 2. Espinouse *et al.* (2001) ont démontré que dans le cas de plusieurs indisponibilités, le problème devient fortement *NP-difficile*; et il n'y a pas d'heuristique de garantie de performance à moins que $P = NP$.

Kubzin & Strusevich (2005) ont étudié le même problème avec machines ayant une activité de maintenance d'une longueur définie par une fonction non décroissante, qui dépend de l'heure de début de cette activité (période d'indisponibilité). Un schéma d'approximation polynomiale est proposé pour résoudre le problème.

Kubzin & Strusevich (2006) ont considéré le $F_2||C_{max}$ dans lequel chaque machine a une activité de maintenance, et dont la durée dépend de son heure de début. Les auteurs ont prouvé que le problème est *NP-difficile* et qu'il est pseudo-polynomial résoluble par programmation dynamique. Ils présentent également un schéma d'approximation entièrement polynomial et un algorithme d'approximation de garantie $3/2$.

Allaoui *et al.* (2006) ont développé un modèle de programmation dynamique amélioré par rapport à celui proposé par Lee *et al.* (1997) et se sont focalisé sur la performance de la règle de Johnson comme heuristique pour les cas sécable et non-sécable. Ils ont établi certaines conditions dans lesquelles l'optimum est garanti et ont démontré que dans d'autres cas, sa performance est limitée à 2. Rapine (2013) a montré que ces conditions sont erronées, dans le cas sécable et non-sécable. Il a également mentionné que deux preuves et la complexité temporelle de leur approche dynamique ne sont pas correctes.

Allaoui *et al.* (2008) ont considéré le cas où l'une des deux machines doit être maintenue une seule fois pendant les premières périodes T de l'ordonnancement. Ils ont présenté quelques propriétés sur l'optimalité et puis ils ont montré que le problème est *NP-difficile*. Finalement, ils ont cherché les solutions optimales basées sur la règle de Johnson dans certaines conditions. Gara-Ali & Espinouse (2014) ont montré dans un Erratum que le problème de deux ordonnancements de flux machine avec une seule maintenance sur la seconde machine prouvée par Allaoui *et al.* (2008) (théorème 7) n'est pas *NP-difficile* et solvable par un algorithme en temps polynomial.

Hadda (2010) a proposé deux algorithmes d'approximation avec un ratio d'erreur au pire cas égale à 2 et $3/2$, lorsque le problème a un intervalle d'indisponibilité sur la première

machine.

Ben Chihaoui *et al.* (2011) considèrent le problème de type flow shop à deux machines avec contraintes de disponibilité de non attente (no-wait) et que les tâches ont des dates de disponibilité différentes pour la minimisation du makespan. Les intervalles de non-disponibilité des machines se chevauchent et sont connus à l'avance. Ils proposent des bornes inférieures et supérieures. Ces bornes sont utilisées dans une procédure par séparation et évaluation. De plus, la procédure peut être convertie en une heuristique de recherche gloutonne. Cette heuristique est capable de donner une solution quasi-optimale (écart de 1 %).

Hnaïen *et al.* (2015) ont traité le problème de type flow shop à deux machines avec une contrainte de disponibilité sur la première machine. Ils ont établi deux modèles de programmation linéaire en nombres mixtes (*MIP*) et une procédure par séparation et évaluation employant sur un ensemble de bornes inférieures et supérieures et des propriété d'optimalité de l'algorithme de Johnson. Il a été démontré qu'il y a un impact de la date de début la période d'indisponibilité sur la difficulté de résolution. L'impact de la durée de la période d'indisponibilité sur la performance de la procédure a été également pris en compte.

Cui *et al.* (2016) ont traité le problème de type flow shop de non permutation. Deux types de contraintes d'indisponibilité sont étudiées. Dans le premier cas, les intervalles de non-disponibilité sont périodiquement fixés et connus à l'avance tandis que le second cas concerne les intervalles flexibles et le temps de la tâche continu des machines ne peut pas dépasser un temps maximum autorisé. Deux modèles de programmation en nombres entiers binaires mixtes sont fournis. Les problèmes de minimisation du makespan pour ce type de problèmes est prouvée *NP-difficile* au sens fort. Donc, un algorithme génétique hybride incrémenté (HIGA) a été proposé pour résoudre efficacement les problèmes de grande taille.

Les propriétés du problème d'ordonnancement d'un flux sans attente de deux machines, avec un intervalle d'indisponibilité non-sécable, sont étudiées par Li *et al.* (2017). Cet article traite le problème avec période d'indisponibilité sur la première et deuxième machines respectivement. La complexité des problèmes est prouvée *NP-difficile*. L'algorithme de Gilmore et Gomory (GGA) est appliqué pour minimiser le makespan, et les conditions que cet algorithme assure les solutions optimales sont présentées. Les ratios de performance au pire cas de GGA ne dépasse pas 2.

Labidi *et al.* (2018) ont considéré le problème d'ordonnancement des tâches soumises à des dates de disponibilité inégales dans un atelier de type flow shop à deux machines avec contraintes de non-attente (no-wait) et de disponibilité sont prises en compte. En premier

lieu, ils ont proposé une nouvelle formulation mathématique pour le problème et en ont déduit des inégalités valides. Deuxièmement, ils ont proposé de nouvelles bornes inférieures basées sur des relaxations à une ou deux machines. Ces bornes inférieures sont intégrées dans une procédure par séparation et évaluation améliorée par des règles d'élimination et de dominance.

2.3.3.4 Cas semi-sécable

Lee (1999) a généralisé les résultats de complexité présentés dans Lee (1997) dans le cas où les opérations sont semi-sécables. L'auteur a montré que la minimisation du makespan est *NP-difficile* au sens faible lorsque la période d'indisponibilité est sur une des deux machines. Un algorithme de programmation dynamique a été développé lorsque la période d'indisponibilité est imposée sur la première machine, et la preuve que l'algorithme de Johnson a une erreur relative égale à 1 est donnée. Lorsque la période d'indisponibilité se produit sur la deuxième machine, l'algorithme de Johnson a une erreur relative égale à $\max\{1/2, \alpha\}$, où α est la partie à retraiter de l'opération semi-réutilisable, interrompu par la période d'indisponibilité. Dans le même article, l'auteur a montré que lorsque les deux machines ont une période d'indisponibilité (qui ne démarre pas à l'instant zéro), le problème est *NP-difficile* au sens faible, même si les dates de début et de fin des périodes d'indisponibilité ont les mêmes caractéristiques sur les deux machines. Dans ce cas, l'algorithme de Johnson a une erreur relative égale à α . Enfin, si les périodes d'indisponibilité des deux machines commencent à zéro, l'algorithme de Johnson permet de résoudre de façon optimale la minimisation du makespan.

2.3.4 Problème d'ordonnancement de type Job shop

2.3.4.1 Cas non-préemptif

Aggoune (2002) a proposé une procédure par séparation et évaluation pour résoudre le problème de job shop sous contraintes de disponibilité pour la minimisation de makespan. L'approche peut être généralisée à tout critère régulier. Le modèle de graphe disjonctif est utilisé pour représenter les noeuds des arbres. L'auteur a introduit une autre manière pour prendre en compte les périodes de disponibilité des machines, en introduisant des tâches fictives représentant les périodes d'indisponibilité fixes et flexibles. Le calcul des bornes inférieures est basé sur la résolution de sous-problèmes à deux jobs, en tenant compte des contraintes de précédence et de disponibilité ainsi que des dates de latence des opérations.

La complexité des problèmes d'ordonnancement prenant en compte deux tâches à planifier et les contraintes de disponibilité imposées aux machines ont également été traitées (voir aussi Aggoune (2004b)). Un algorithme polynomial appelé "*approche géométrique temporisée*" est d'abord proposé pour la minimisation du makespan, sous la contrainte de non-préemption. Ensuite, une généralisation au cas préemptif est développée. Ces algorithmes sont des extensions de l'approche géométrique proposée par Akers Jr & Friedman (1955) qui transforme le problème initial en une recherche du plus court chemin, ce qui permet de résoudre le problème classique d'ordonnancement à deux tâches de façon polynomial. Dans Aggoune (2010), un nouvel algorithme polynomial pour la planification de job shops avec deux jobs et des contraintes de disponibilité a été présenté. Ils ont supposé qu'il pouvait y avoir un nombre arbitraire de périodes de non-disponibilité sur chacune des m machines (avec $m > 2$). L'auteur a présenté une nouvelle représentation qui permet de traiter des temps de traitement modifiés et donc de prendre en compte des contraintes d'indisponibilité de plusieurs sortes.

Azem *et al.* (2007) ont traité le même problème que Aggoune (2002). Ils ont introduit une flexibilité sur les périodes d'indisponibilité de la machine sur des fenêtres de temps. Deux modèles mathématiques ont été présentés et comparés. Le premier est basé sur le graphe disjonctif et le second est indexé dans le temps. Les résultats ont montré que la méthode disjonctive est meilleure. Une approche de génération de colonnes a été également présentée dans Azem (2010) pour résoudre le problème d'ordonnancement du job shop avec et sans périodes d'indisponibilité fixes ou flexibles. Cette approche n'a pas fourni des résultats meilleurs que ceux de la formulation disjonctive du problème mais elle a fourni de meilleures relaxations linéaires dans un temps de calcul relativement petit.

Zribi *et al.* (2008) ont traité le problème de type Job shop avec machines de multi-usage (*Multi-Purpose*) sous contraintes de disponibilité. Une heuristique, basée sur des règles de priorité pour résoudre le problème d'affectation, est proposée et un algorithme de recherche locale est ensuite introduit pour améliorer cette solution d'affectation. Ils ont développé également un algorithme génétique pour résoudre le problème de séquençement. Par ailleurs, une borne inférieure est développée afin d'évaluer la qualité des solutions approchées.

Naderi *et al.* (2009) ont étudié l'ordonnancement de job shop avec des temps setup dépendant de la séquence et des politiques de maintenance préventive pour minimiser le makespan. Quatre méta-heuristiques basées sur le recuit simulé et les algorithmes génétiques ainsi que des adaptations de deux méta-heuristiques dans la littérature ont été utilisées pour résoudre le problème. La comparaison entre les performances des algorithmes a révélé l'efficacité de l'algorithme génétique hybride avec *SPT* par rapport aux autres algorithmes.

Mati (2010) traite les périodes d'indisponibilité fixes et non-préemptives dans un atelier de job shop avec n tâches et m machines pour la minimisation du makespan. Une heuristique *taboo thresholding* utilisant une nouvelle fonction de voisinage approximativement pour résoudre le problème. Fnaiech *et al.* (2012) ont proposé une méthode *Hopfield Neural Network (HNN)* utilisée pour résoudre le problème du job shop. L'objectif est de minimiser le makespan.

Guyon *et al.* (2014) ont proposé deux méthodes exactes pour résoudre un problème d'emploi du temps et d'ordonnancement de type job shop. Le problème est de trouver un emploi du temps à moindre coût, où les opérateurs ont des compétences différentes et travaillent des quarts de travail (shift), de sorte que la production, qui correspond à un atelier de type job shop avec contraintes de disponibilité des ressources. Ils ont introduit deux nouvelles procédures exactes : (1) une approche de décomposition et de génération de coupe et (2) une hybridation d'un processus de génération de coupe avec une stratégie par séparation et évaluation. Ils ont également proposé des coupes initiales améliorant ces méthodes ainsi qu'une approche de programmation en nombres entiers mixtes.

Fnaiech *et al.* (2015) ont présenté une méthode heuristique *NHGA* pour résoudre le problème. L'heuristique proposée comprend deux techniques; algorithme génétique modifié (*MGA*), inspiré de l'algorithme génétique standard. Le second, appelé déplacement heuristique des gènes, permet de réformer d'avantage les meilleurs chromosomes obtenus par la première technique *MGA*.

2.3.4.2 Cas sécable, sécable/non sécable

Mauguière *et al.* (2003b) ont proposé une procédure par séparation et évaluation pour le problème $J|rs|C_{max}$. Les résultats numériques montrent que résoudre le problème $J|rs|C_{max}$ est plus difficile que le problème sans périodes d'indisponibilité.

Mauguière *et al.* (2003a) ont proposé une procédure par séparation et évaluation pour le problème $J, cr|rs/nr(M_i^k)|C_{max}$. L'étude a été étendue à $J, cr/nr|r_j, rs/nr(M_i^k)|C_{max}$ par Mauguière *et al.* (2005).

2.3.5 Problème d'ordonnancement de type Open shop

2.3.5.1 Cas non-préemptif

Breit *et al.* (2003) ont étudié un problème d'ordonnancement de type open shop à deux

machines, dans lequel les machines ne sont pas continuellement disponibles pour l'exécution des tâches. L'objectif est de minimiser le makespan sous la condition de non-préemption des opérations. Ils ont présenté un algorithme d'approximation avec un ratio au pire cas de $4/3$ pour le problème avec une seule indisponibilité sur une seule machine. Le problème avec deux indisponibilités sur une des deux machines ne peut pas être approximé en temps polynomial sauf si $P = NP$.

2.3.5.2 Cas sécable

Lorigeon *et al.* (2002) ont étudié un problème d'ordonnancement d'un open shop avec opérations sécables sur deux machines avec une contrainte de disponibilité pour la minimisation du makespan comme critère. Comme le problème est *NP-difficile*, ils ont développé un algorithme de programmation dynamique pseudo-polynomiale pour résoudre le problème de façon optimale quand la machine n'est pas disponible à l'instant 0 et une formulation de programmation linéaire mixte, permettant de résoudre des instances allant jusqu'à 500 tâches de manière optimale. Enfin, ils ont prouvé que tout heuristique à une erreur relative au pire cas égale à 1.

Breit *et al.* (2001) ont étudié le même problème. Ils ont montré que le problème est *NP-difficile* et ont proposé un algorithme d'approximation avec un ratio au pire cas de $4/3$.

Kubzin *et al.* (2006) ont présenté deux schémas d'approximation en temps polynomial pour le même problème, l'une qui traite le problème avec une indisponibilité sur chaque machine et l'autre pour le problème avec plusieurs indisponibilités sur l'une des machines. Les problèmes avec une structure plus générale des intervalles de non-disponibilité ne sont pas approximables en temps polynomial dans un facteur constant, à moins que $P = NP$.

2.3.5.3 Cas non-sécable

Kubzin & Strusevich (2006) ont pris en compte les problèmes d'ordonnancement open shop à deux machines dans lesquels chaque machine doit être maintenue exactement une fois pendant l'horizon d'ordonnancement, et la durée de chacun de ces période d'indisponibilité dépend de son heure de début. L'objectif est de minimiser le makespan. Le problème étudié est résolu de manière polynomiale compte tenu des fonctions générales définissant la longueur des intervalles de maintenance.

2.4 Conclusion

Un état de l'art couvrant les problèmes d'ordonnancement de la production sous contraintes de disponibilité de ressources, a été présenté dans ce chapitre.

Bien que les efforts de recherche aient été déployés pour résoudre les problèmes d'ordonnancement des machines en intégrant les contraintes de disponibilité des ressources, ils sont essentiellement concentrés plus sur les problèmes d'une seule machine, de type machines parallèles et flow shop. En revanche, les chercheurs se sont moins focalisés sur les problèmes d'ordonnancement de type job shop et open shop. Nous retenons également de cette étude bibliographique que tous les tâches précédentes couvrant l'ordonnancement de type job shop sous contraintes de disponibilité ne se sont pas focalisé sur les problèmes de job shop à deux machines, pourtant étudiés largement pour la cas du flow shop.

Ainsi, notre étude vise à contribuer à combler ce manque en proposant de traiter dans les chapitres suivants les problèmes du job shop à deux machines sous contraintes de disponibilité.

Chapitre 3

Le problème d'ordonnancement de type Job shop à deux machines avec contraintes de disponibilité sur une machine

Résumé :

Dans ce chapitre, nous traitons le problème d'ordonnancement de type job shop à deux machines avec des contraintes de disponibilité sur une seule machine. Tout d'abord, nous modélisons le problème comme étant un job shop avec contraintes de disponibilité. Ensuite, nous établissons des propriétés concernant le séquençement optimal des tâches avant et après chaque période d'indisponibilité. Nous développons des heuristiques et nous étudions leur comportement au pire cas. Par la suite, nous proposons des bornes supérieures et inférieures ainsi qu'une procédure de séparation et évaluation pour la résolution exacte du problème. Les propriétés théoriques ainsi que les performances des méthodes développées sont discutées.

Publications :

- Banttaleb, Mourad, Faicel Hnaien, and Farouk Yalaoui. 2018. Two-machine job shop problem under availability constraints on one machine : Makespan minimization. *Computers & Industrial Engineering* 117 : 138 - 151.

- Banttaleb, Mourad, Faicel Hnaien, and Farouk Yalaoui. Two-machine job shop problem for makespan minimization under availability constraint. *IFAC-AMEST Biarritz, France 2016. IFAC-PapersOnLine* 49 (28) : 132 - 137.

3.1 Introduction

Nous étudions dans ce chapitre le problème d'ordonnancement de type job shop à deux machines avec des contraintes de disponibilité sur une seule machine.

Nous considérons le cas des opérations strictement non-préemptives avec l'objectif de minimiser le makespan. Nous présentons des méthodes permettant la résolution exacte et approchée du problème qui exploitent la règle de Jackson.

La deuxième section de ce chapitre est consacrée à la description du problème. En particulier, nous décrivons la complexité du problème ainsi que sa spécificité par rapport à un job shop à deux machines sans contraintes de disponibilité.

La troisième section est dédiée aux méthodes de résolution. Premièrement, des propriétés étudiant l'optimalité du problème sont proposées. Une règle d'ordonnancement des tâches pour le cas étudié est également présentée. Nous proposons aussi deux modèles linéaires mixtes et finalement une méthode de résolution exacte, à savoir une procédure par séparation et évaluation. Les propriétés démontrées sont employées pour le calcul des bornes inférieures et supérieures et pour la résolution du problème. Finalement, les résultats expérimentaux montrent l'efficacité de l'approche proposée et des méthodes présentées.

3.2 Description du problème

Cette section est consacrée à la description du problème d'ordonnancement de type job shop à deux machines avec plusieurs périodes d'indisponibilité sur une seule machine. Nous mettons l'accent sur la spécificité de ce problème ainsi que sur les hypothèses considérées.

Avant de décrire le problème de manière détaillée, nous introduisons les notations que nous adoptons dans la suite du manuscrit (voir tableau 3.1).

Le problème d'ordonnancement de type job shop à deux machines avec une période d'indisponibilité sur une seule machine, se définit de la manière suivante :

- Un ensemble de N tâches à réaliser sur deux machines M_1 et M_2 .
- La tâche J_i nécessite une seule machine à la fois durant p_{ij} .
- Chaque machine ne peut réaliser qu'une opération à la fois.
- La machine M_1 est indisponible durant U périodes, dont la date et la durée sont fixes et connues à l'avance.

Tableau 3.1 – Notations

Notation	Signification
N	Nombre de tâches à réaliser
U	Nombre de périodes d'indisponibilité
$J = \{J_1, J_2, \dots, J_N\}$	Ensemble des tâches à réaliser
$M = \{M_1, M_2\}$	Ensemble des machines
$I = \{I_1, I_2, \dots, I_{U+1}\}$	Ensemble d'intervalles entre U périodes d'indisponibilité
p_{ik}	Temps d'exécution de la tâche J_i sur la machine M_k
(σ_1^i, σ_2^i)	Ordre d'exécution de la tâche J_i sur les deux machines
s_u	Date de début de la $u^{\text{ème}}$ période d'indisponibilité sur la machine M_k
t_u	Date de fin de la $u^{\text{ème}}$ période d'indisponibilité sur la machine M_k
g_u	Durée de la $u^{\text{ème}}$ période d'indisponibilité sur la machine M_k
C_k^{AJK}	Date de fin sur la machine M_k obtenue par la méthode de Jackson
C_{max}^{JK}	Makespan trouvé par la méthode de Jackson
$Idle_k^{JK}$	Somme des durées d'inactivité de la solution donnée par l'algorithme de Jackson sur la machine M_k
$Idle_u$	Période d'inactivité générée par la $u^{\text{ème}}$ période d'indisponibilité
$Set_{kk'}$	Ensemble des tâches passant d'abord sur la machine M_k ensuite sur la machine $M_{k'}$
Set_k	Ensemble des tâches traitées uniquement sur la machine M_k
$Set_{kk'}^{I_u}$	Ensemble des tâches appartenant à I_u , traitées sur M_k puis sur $M_{k'}$
$Set_k^{I_u}$	Ensemble des tâches appartenant à I_u , traitées uniquement sur la machine M_k
$J_2 C_{max}$	Job shop à deux machines minimisant le Makespan
$J_2, h_{kU} a C_{max}$	Job shop à deux machines avec U périodes d'indisponibilité sur chaque machine M_k , minimisant le Makespan
$J_2, h_{1U} a C_{max}$	Job shop à deux machines avec U périodes d'indisponibilité sur la machine M_1 , minimisant le Makespan
$J_2, h_{k1} a C_{max}$	Job shop à deux machines minimisant le Makespan, avec contrainte de disponibilité sur chaque machine M_k
$J_2, h_{k1} a, s_k = 0 C_{max}$	Job shop à deux machines minimisant le Makespan, avec contrainte de disponibilité sur chaque machine M_k commençant au début de l'horizon de l'ordonnancement
$J_2, h_{11} a, s_1 = 0 C_{max}$	Job shop à deux machines minimisant le Makespan, avec contrainte de disponibilité sur la machine M_1 commençant au début de l'horizon de l'ordonnancement
$F_2, h_{11} a C_{max}$	Flow shop à deux machines minimisant le Makespan, avec contrainte de disponibilité sur la machine M_1

- Les opérations sont supposées strictement non-préemptives. C’est à dire, le traitement d’une opération n’est interrompue ni par celui d’une autre, ni par l’indisponibilité.
- L’objectif est de trouver le séquençement des opérations sur les machines qui minimise le makespan.

Sans perte de généralité, nous considérons que la machine M_1 est soumise à U périodes d’indisponibilité tout au long de l’horizon d’ordonnancement. En fait, le $J_2||C_{max}$ est symétrique, le choix de l’indisponibilité sur la machine M_1 ou M_2 est similaire.

Le problème d’ordonnancement de type job shop à deux machines avec contraintes de disponibilité est *NP-difficile*. En effet, Lee (1997) a démontré qu’un problème d’ordonnancement de type flow shop avec opérations sécables, sous contrainte de disponibilité sur l’une des deux machines est *NP-difficile* pour la minimisation du makespan. Par conséquent, l’algorithme de Jackson ne garantit pas l’optimalité du problème. La contribution dans ce chapitre est l’exploitation de l’algorithme de Jackson pour résoudre à l’optimal des instances de grandes taille dans un temps raisonnable.

3.3 Modélisation mathématique

Nous sommes amenés dans cette section à modéliser mathématiquement le problème d’ordonnancement de type job shop à deux machines avec contraintes de disponibilité sur une machine $(J_2, h_{1U}|a|C_{max})$. Nous avons choisi d’entamer cette approche afin d’avoir, d’une part, une idée claire sur la capacité de résolution de certaines tailles du problème et d’autre part, pour avoir un moyen qui permettra de prouver la pertinence de l’approche développée. Dans ce qui suit, nous formulons deux modèles linéaires en nombres entiers mixtes pour le problème de type $J_2, h_{1U}|a|C_{max}$. La formulation du premier modèle (MILP1) est dérivée de celui proposé par Applegate & Cook (1991). Le second modèle (MILP2) est adapté de celui proposé par Azem (2010) qui est une formulation mathématique du graphe disjonctif présenté par Roy & Sussman (1964), résolvant un problème de type job shop avec multiples indisponibilités sur chaque machine.

3.3.1 Formulation 1 : MILP1

L’objectif est de déterminer la date d’achèvement c_{ik} de chaque tâche J_i sur la machine M_k afin de minimiser le makespan. Chaque tâche J_i est traitée pendant p_{ik} unité de temps et l’ordre de traitement sur les deux machines est (σ_1^i, σ_2^i) .

Les variables de décision sont définies comme suit :

$$x_{ijk} = \begin{cases} 1 & \text{Si la tâche } J_i \text{ est séquencée avant } J_j \text{ sur la machine } M_k \\ 0 & \text{Sinon} \end{cases}$$

$$y_{iu} = \begin{cases} 1 & \text{Si la tâche } J_i \text{ est traitée avant la } u^{\text{ème}} \text{ période d'indisponibilité} \\ & \text{sur la machine } M_1 \\ 0 & \text{Sinon} \end{cases}$$

c_{ik} : Temps d'achèvement de la tâche J_i sur la machine M_k .

C_{max} : Makespan.

Fonction objectif :

$$\text{Min } C_{max}$$

Contraintes :

$$c_{ik} \geq p_{ik} \quad \forall i = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.1)$$

$$c_{i\sigma_2^i} - p_{i\sigma_2^i} \geq c_{i\sigma_1^i} \quad \forall i = 1, \dots, N \quad (3.2)$$

$$c_{ik} - c_{jk} + B \cdot x_{ijk} \geq p_{ik} \quad \forall i \neq j = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.3)$$

$$c_{jk} - c_{ik} + B \cdot (1 - x_{ijk}) \geq p_{jk} \quad \forall i \neq j = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.4)$$

$$c_{i1} - B \cdot (1 - y_{iu}) \leq s_u \quad \forall i = 1, \dots, N, \quad \forall u = 1, \dots, U \quad (3.5)$$

$$c_{i1} - t_u \cdot (1 - y_{iu}) \geq p_{i1} \quad \forall i = 1, \dots, N, \quad \forall u = 1, \dots, U \quad (3.6)$$

$$C_{max} \geq c_{ik} \quad \forall i = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.7)$$

$$C_{max}, c_{ik} \in \mathbb{N} \quad \forall i = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.8)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j = 1, \dots, N, \quad \forall k = 1, 2 \quad (3.9)$$

$$y_{iu} \in \{0, 1\} \quad \forall i = 1, \dots, N, \quad \forall u = 1, \dots, U \quad (3.10)$$

Les contraintes (3.1) assurent que la date de fin dépasse la durée du traitement de la tâche J_i sur la machine M_k . Les contraintes (3.2) indiquent que la date de début de la tâche J_i sur $M_{\sigma_2^i}$ n'est pas antérieure à son date de fin sur $M_{\sigma_1^i}$. J_i devrait être exécuté dans l'ordre $\{M_{\sigma_1^i}, M_{\sigma_2^i}\}$. Les contraintes (3.3) et (3.4) garantissent le non chevauchement des opérations des tâches J_i et J_j exécutées sur la même machine M_k , avec B un grand nombre. Les contraintes (3.5) garantissent que la date de fin de l'exécution des tâches avant

la $u^{\text{ème}}$ période d'indisponibilité, sur la machine M_1 , ne dépasse jamais s_u . Les contraintes (3.6) assurent que le date de fin de l'exécution de la tâche J_i sur la machine M_1 est supérieur ou égal à son temps d'exécution plus tu s'il est prévu après la $u^{\text{ème}}$ période d'indisponibilité. Les contraintes (3.7) assurent que le makespan est plus grand ou égale à la date de fin de tout les tâches. Les contraintes (3.8) indiquent que le makespan et la date de fin de chaque tâche sont des nombres entiers. Les contraintes (3.9) et (3.10) fournissent des restrictions binaires pour y_{iu} et x_{ijk} .

3.3.2 Formulation 2 : MILP2

Le modèle MILP2 est une adaptation de celui présenté par Azem (2010). Il considère que chaque tâche J_i est composée au plus de deux opérations O_{i1}, O_{i2} . Chaque opération O_{ij} est traitée sur la machine m_{ij} pendant p_{ij} . L'objectif du modèle suivant est de minimiser le makespan en déterminant la date de début s_{ij} de chaque opération O_{ij} .

$$x_{iji'j'} = \begin{cases} 1 & \text{if } O_{ij} \text{ est séquencée avant } O_{i'j'} \\ 0 & \text{Sinon} \end{cases}$$

$$y_{iju} = \begin{cases} 1 & \text{Si } O_{ij} \text{ est traitée avant la } u^{\text{ème}} \text{ période d'indisponibilité sur la machine } M_1 \\ 0 & \text{Sinon} \end{cases}$$

s_{ij} : Date de début de l'opération O_{ij} .

C_{max} : Makespan.

Fonction objectif

$$\text{Min } C_{max}$$

Contraintes :

$$s_{i(j+1)} \geq p_{ij} + s_{ij} \quad \forall i = 1, \dots, N, \quad j = 1 \quad (3.11)$$

$$s_{i'j'} - s_{ij} + B.(1 - x_{iji'j'}) \geq p_{ij} \quad \forall O_{ij} \neq O_{i'j'}, \quad \forall m_{ij} = m_{i'j'} \quad (3.12)$$

$$s_{ij} - s_{i'j'} + B.x_{iji'j'} \geq p_{i'j'} \quad \forall O_{ij} \neq O_{i'j'}, \quad \forall m_{ij} = m_{i'j'} \quad (3.13)$$

$$s_{ij} + B.y_{iju} \geq t_u \quad \forall O_{ij}, \quad m_{ij} = 1, \quad \forall u = 1, \dots, U \quad (3.14)$$

$$s_u - s_{ij} + B.(1 - y_{iju}) \geq p_{ij} \quad \forall O_{ij}, m_{ij} = 1, \forall u = 1, \dots, U \quad (3.15)$$

$$C_{max} \geq s_{ij} + p_{ij} \quad \forall i = 1, \dots, N \quad (3.16)$$

$$C_{max} \geq 0 \quad (3.17)$$

$$s_{ij} \geq 0 \quad \forall i = 1, \dots, N, \forall j = 1, 2 \quad (3.18)$$

$$x_{ijj'j'} \in \{0, 1\} \quad \forall O_{ij} \neq O_{i'j'}, \forall m_{ij} = m_{i'j'} \quad (3.19)$$

$$y_{iju} \in \{0, 1\} \quad \forall O_{ij}, m_{ij} = 1, \forall u = 1, \dots, U \quad (3.20)$$

Les contraintes (3.11) assurent que l'opération O_{ij+1} qui suit O_{ij} ne peut être exécutée qu'après la fin de l'exécution de O_{ij} . Les contraintes (3.12) et (3.13) garantissent le non chevauchement des opérations O_{ij} et $O_{i'j'}$ exécutées sur la même machine. Les contraintes (3.14) et (3.15) assurent que l'opération O_{ij} est traitée soit avant soit après la $u^{\text{ème}}$ période d'indisponibilité sur la machine M_1 . Les contraintes (3.16) indiquent que l'ordonnancement ne se termine qu'après l'exécution de la dernière opération de chaque tâche. Les contraintes de (3.17) à (3.20) fournissent des bornes pour les variables du modèle.

3.4 Propriétés

Comme indiqué précédemment, le problème étudié est *NP-difficile* et l'algorithme de Jackson ne permet pas de le résoudre de façon optimale. Ces résultats qui soulignent la difficulté de notre problème nous amènent à élaborer des méthodes de résolution garantissant l'optimalité. Pour ce faire, nous allons démontrer des propriétés concernant notamment les cas particuliers d'optimalité de l'algorithme de Jackson. Nous fournissons par ailleurs un ordre permettant de séquencer les tâches sur les deux machines de manière optimale.

Propriété 1. *Pour $J_2, h_{11}|a, s_1 = 0|C_{max}$, la solution obtenue par l'algorithme de Jackson est optimale.*

Démonstration. Nous supposons un ensemble de tâches ordonnancées sur deux machines suivant la règle de Jackson. Nous ajoutons à l'ordonnancement supposé une période d'indisponibilité de durée g_1 , commençant à $s_1 = 0$ au niveau de la machine M_1 . Dans ce cas, l'ensemble Set_{12} sur la machine M_1 commencera à $t_1 = g_1$. Ce décalage peut soit pousser la date de fin soit absorber une durée d'inactivité sur la machine M_1 . Il peut également engendrer une période d'inactivité au niveau de la machine M_2 au cas où l'exécution d'une tâche décalée sur la machine M_1 , empêche son traitement sur la machine M_2 déjà disponible.

Nous allons démontrer que l'algorithme de Jackson reste optimal sur les deux machines quelque soit le cas engendré.

— Sur la machine M_1 :

Cas (i) : Le séquençement optimal, obtenu par l'algorithme de Jackson contient des durées d'inactivité sur la machine M_1 , alors :

$$C_1^{JK} = \sum_{i \in N} p_{i1} + Idle_1^{JK} \quad (3.21)$$

(ia) : $g_1 \leq Idle_1^{JK}$ alors C_1^{AJK} reste optimale. Par conséquent, l'algorithme de Jackson assure une date de fin optimale sur M_1 .

(ib) : $g_1 > Idle_1^{JK}$ alors C_1^{AJK} est optimale (égale à la borne inférieure (eq. 3.22)) parce que la somme des durées d'inactivité est nulle (Figure 3.1).

$$C_1^{AJK} = \sum_{i \in N} p_{i1} + g_1 \quad (3.22)$$

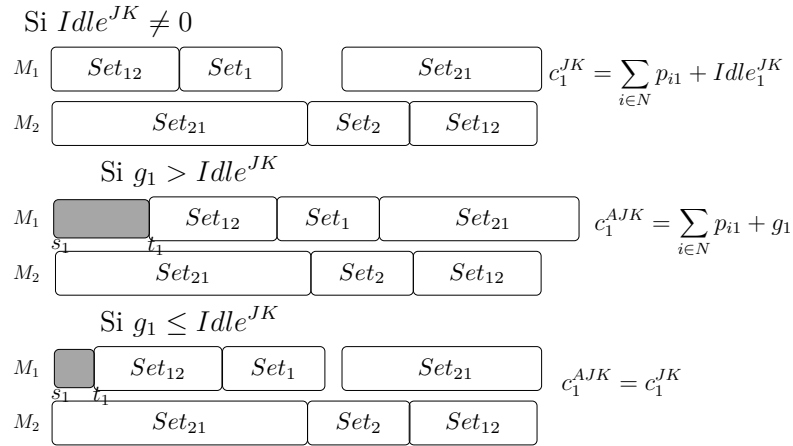


FIGURE 3.1 – Impact de la période d'indisponibilité à $s_1 = 0$, sur M_1 si $Idle_1^{JK} \neq 0$

Case (ii) : L'ordre selon la règle de Jackson ne contient aucune durée d'inactivité (voir figure 3.2), alors :

$$C_1^{JK} = \sum_{i \in N} p_{i1} \quad (3.23)$$

Alors,

$$C_1^{AJK} = \sum_{i \in N} p_{i1} + g_1 \quad (3.24)$$

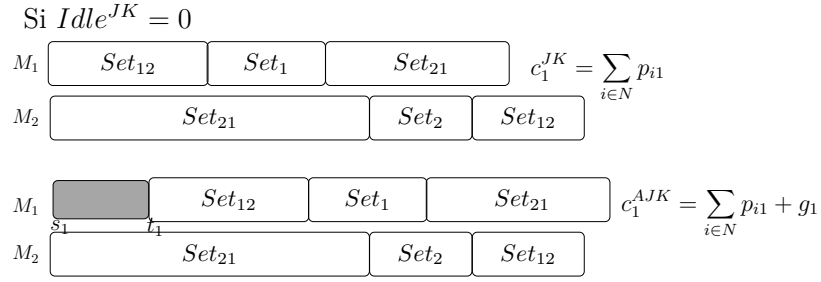


FIGURE 3.2 – Impact de la période d'indisponibilité à $s_1 = 0$, sur M_1 , si $Idle_1^{JK} = 0$

C_1^{AJK} est optimal car l'ordre des tâches sur la machine M_1 ne contient aucune durée d'inactivité.

Nous concluons que l'ordre de Jackson reste optimal sur la machine M_1 si une indisponibilité commence à $s_1 = 0$.

— Sur la machine M_2 (voir figure 3.3) :

Case (i) : Le décalage de Set_{12} ne génère aucun temps d'inactivité sur M_2 , alors C_2^{AJK} reste optimal.

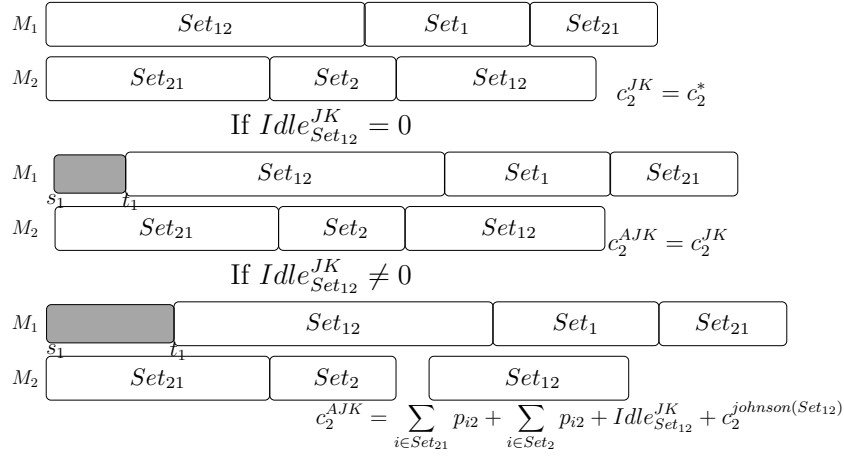
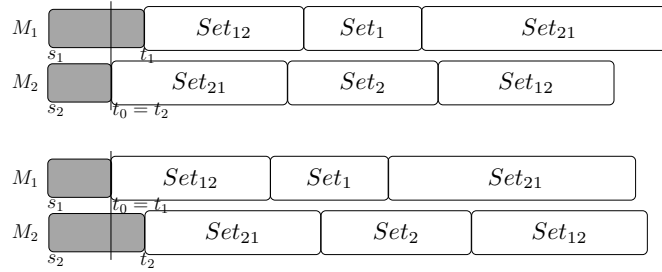
Case (ii) : La nouvelle date du début d'exécution de l'ensemble Set_{12} engendre une durée d'inactivité ($Idle_{Set_{12}}^{JK}$) au niveau de la machine M_2 , alors C_2^{AJK} est régie par le séquençement des tâches appartenant à cet ensemble. Étant donné que l'ordonnancement du Set_{12} est un $F_2, h_{11}|a, s_1 = 0|C_{max}$ et d'après Lee (1997), le séquençement optimal du Set_{12} suit la règle de Johnson. Comme le séquençement des tâches appartenant aux sets Set_{21} et Set_2 est optimal (le séquençement du Set_{21} suit l'ordre de Johnson tandis que celui du Set_2 est arbitraire), alors, l'ordre suivant la règle de Jackson sur la machine M_2 est optimal.

□

Propriété 2. Si la période d'indisponibilité au niveau de la machine M_2 est située au début de l'horizon d'ordonnancement ($J_2, h_{21}|a, s_1 = 0|C_{max}$), alors la solution obtenue par l'algorithme de Jackson est optimale.

Démonstration. Le problème d'ordonnancement de type job shop à deux machines est symétrique (atelier à deux cheminements inversés). Donc, la démonstration de cette propriété est la même que la précédente. Il suffit d'échanger M_1 et M_2 .

□

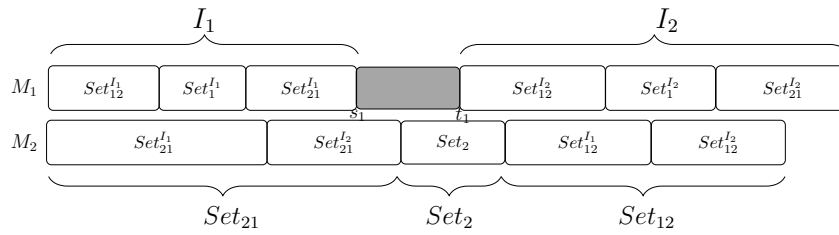

 FIGURE 3.3 – Impact de la période d'indisponibilité à $s_1 = 0$, sur M_2

 FIGURE 3.4 – Résolution du problème avec période d'indisponibilité sur chaque machine $s_k = 0$

Propriété 3. Pour $J_2, h_{k1} | a, s_k = 0 | C_{max}$, l'algorithme de Jackson fournit une solution optimale.

Démonstration. Il suffit de supposer que la date de début de l'horizon d'ordonnancement est à $\min(t_1, t_2)$ et dans ce cas, la taille de l'indisponibilité sera supposée égale à $\max(t_1 - t_2, t_2 - t_1)$ (voir figure 3.4). Ce décalage permet de rapporter le cas en l'occurrence à un des deux cas précédents (une période d'indisponibilité sur une seule machine).

□

Propriété 4. Pour le problème d'ordonnancement à deux machines avec une période d'indisponibilité sur une machine $J_2, h_{11} | a | C_{max}$, il existe un ordre optimal garantissant que les tâches avant et après la période d'indisponibilité sont séquencées suivant la règle de Jackson (figure 3.5).

FIGURE 3.5 – Ordonnancement des tâches l'ordre optimal " S^* " pour $J_2, h_{11}|a|C_{max}$

Démonstration. Nous démontrons d'abord que l'ordre proposé garantit que les ensembles de tâches à traiter avant et après la période d'indisponibilité sont ordonnés suivant la règle de Jackson.

Le $J_2, h_{11}|a|C_{max}$ est décomposé en deux sous-problèmes d'ordonnancement. La détermination de l'ensemble des tâches pour chaque sous-problème est fondée sur la disponibilité des tâches avant et après la période d'indisponibilité. Nous déterminons ensuite l'ordre de Jackson pour chacun des deux sous-ensembles.

- Le premier sous-problème ($Pb(1)$) comprends les tâches disponibles avant le début de la période d'indisponibilité s_1 . Sur la machine M_1 , les tâches disponibles sont les tâches achevables avant cette date, à savoir, $Set_{12}^{I_1}$, $Set_{21}^{I_1}$ et $Set_1^{I_1}$. Sur la machine M_2 , le sous-ensemble contient les tâches susceptibles d'être exécutés avant la date s_1 , en l'occurrence, les tâches appartenant aux sets $Set_{21} = \{Set_{21}^{I_2} \cup Set_{21}^{I_1}\}$, les tâches exécutables uniquement sur la machine M_2 (Set_2) et également l'ensemble $Set_{12}^{I_1}$ disponible pour être réalisé sur M_2 avant la date s_1 .

Ce sous-problème (Pb_1) est de type job shop à deux machines avec une indisponibilité à la fin de l'horizon de l'ordonnancement sur la machine M_1 . Donc, la règle de Jackson assure une solution optimale pour (Pb_1) (voir figure 3.6) vu que l'indisponibilité est située après la fin d'exécution de la dernière tâche sur la machine M_1 .

L'ordre des tâches appartenant aux sets $Set_{21}^{I_2}$ et Set_2 est arbitraire car ces deux sets ne sont traités que sur la machine M_2 .

- Le deuxième sous-problème (Pb_2) est également un job shop à deux machines avec une indisponibilité au début de chaque machine. Les tâches à ordonnancer dans ce cas, sont les tâches exécutables après t_1 . Nous citons les tâches appartenant aux sets $Set_{12}^{I_2}$, $Set_{21}^{I_2}$ et $Set_1^{I_2}$ sur la machine M_1 . Les tâches faisant parti des ensembles $Set_{21}^{I_2}$, Set_2 , $Set_{12}^{I_1}$ et $Set_{12}^{I_2}$ sont les ensembles à réaliser sur la machine M_2 parce qu'ils ne sont pas nécessairement exécutés avant s_1 . Ce sous-problème, définie un job

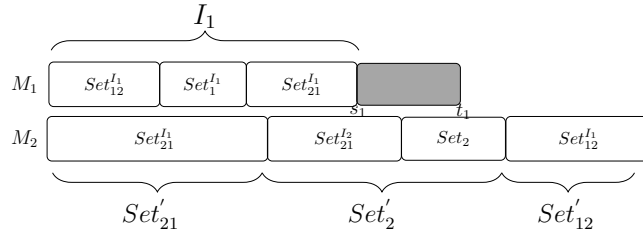


FIGURE 3.6 – Ordonnancement optimal avant la période d'indisponibilité

shop avec indisponibilité située au début de l'horizon d'ordonnancement sur chaque machine. D'après la propriété 3, l'algorithme de Jackson garantit l'optimalité et l'ordonnancement de (Pb_2) et est celui donné par la figure 3.7.

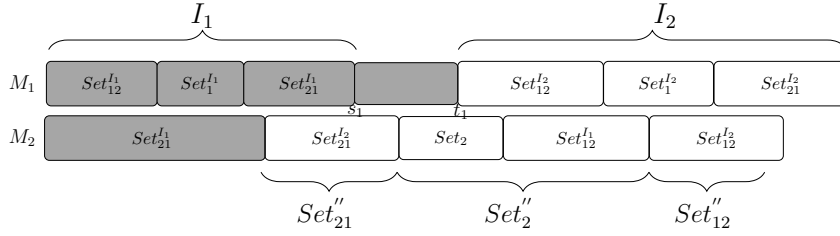


FIGURE 3.7 – Ordonnancement optimal après la période d'indisponibilité

L'optimalité de Jackson avant et après l'indisponibilité est garantie par l'ordre proposé.

Il reste à démontrer qu'il n'existe aucune solution meilleure que celle donnée par cet ordre (figure 3.5).

Nous vérifions que quelque soit l'ordre obtenu par la permutation de deux tâches distinctes, il ne peut que détériorer le makespan. En effet, soient J_i et J_j deux tâches exécutées dans le même sous-ensemble (tâches disponibles avant ou après la période d'indisponibilité). Soit C^{JK} makespan du sous-ensemble obtenu par la règle de Jackson et C^P est celui généré par la permutation de ces deux tâches sur M_1 ou M_2 . Si l'ordre avec permutation fournit un plus petit makespan alors $C^P \leq C^{JK}$, ce qui est impossible parce que comme démontré, la règle de Jackson est optimale. Donc toute permutation de ces deux tâches sur M_1 ou M_2 ne peut qu'accroître le makespan, car la règle de Jackson ne serait plus valide.

Si les deux tâches n'appartiennent pas au même sous-ensemble. L'une est traitée complètement avant l'indisponibilité (tâche appartenant à $Set_{21}^{I_1}$) et l'autre commence après sa fin (tâche appartenant à $Set_{12}^{I_2}$). Toute permutation de ces deux tâches sur la machine

M_2 génère une solution irréalisable car la contrainte de précédence entre deux opérations du même tâche n'est plus respectée. Ainsi, tout ordre autre que celui proposé ne peut qu'détériorer la solution. Ainsi, l'ordre des ensembles est l'optimal.

□

Nous généralisons la propriété 4 pour le cas de plusieurs indisponibilités sur une machine. Nous démontrons par récurrence que le propriété reste valide pour le cas de U indisponibilités sur une machine.

Propriété 5. *Pour un problème de type job shop à deux machines avec deux périodes d'indisponibilité sur une seule machine $(J_2, h_{12} | a | C_{max})$, il existe un ordre optimal garantissant que les tâches exécutées avant, après et entre les deux périodes d'indisponibilité sont ordonnancés suivant la règle de Jackson.*

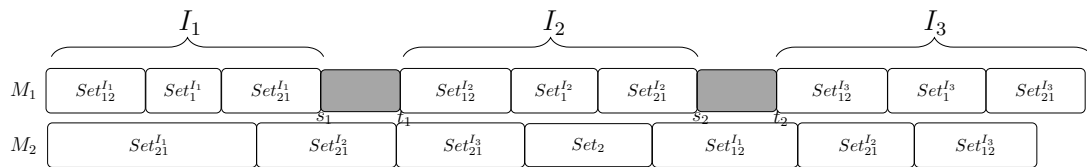


FIGURE 3.8 – Ordonnancement des tâches selon l'ordre optimal " S^* " pour $J_2, h_{12} | a | C_{max}$

Démonstration. Soit s_2 et t_2 la date de début et de fin respectivement de la deuxième période d'indisponibilité et $g_2 = t_2 - s_2$ sa durée. La propriété 4 affirme que l'ordre des deux premiers intervalles I_1 et I_2 est optimal. Si on considère une période d'indisponibilité au début de chaque machine, l'ordre des tâches appartenant à l'intervalle I_3 est optimal si et seulement s'il suit la règle de Jackson. D'autre part, quelle que soit la permutation entre deux tâches, l'ordre généré ne peut que détériorer la solution.

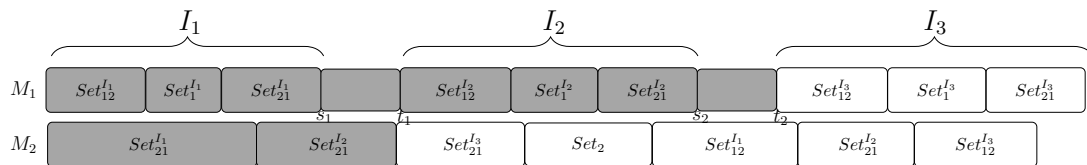


FIGURE 3.9 – Ordonnancement optimal des tâches après la deuxième période d'indisponibilité

□

Propriété 6. *Pour un problème de type job shop à deux machines avec $U > 2$ périodes d'indisponibilité sur la machine M_1 ($J_2, h_{1U}|a|C_{max}$), il existe un ordre optimal S^* garantissant que les tâches avant, après et entre chaque deux périodes d'indisponibilité consécutives, sont séquencées suivant la règle de Jackson.*

Démonstration. Nous démontrons par récurrence que la propriété valide pour le cas de $U = 1$ et $U = 2$ est vrai pour tout $U > 2$.

Sachant que l'ordre est optimal pour le cas de $U = 1$ et $U = 2$. Nous supposons qu'il l'est également pour le problème avec U périodes d'indisponibilité. L'ajout d'une $(U + 1)^{\text{ème}}$ indisponibilité génère un ensemble de opérations à exécuter après la fin de la dernière période d'indisponibilité. L'ordre de cet ensemble est optimal si et seulement s'il suit la règle de Jackson. Compte tenu de l'optimalité des intervalles générés par les U premières périodes d'indisponibilité, l'ordonnancement des N tâches sur deux machines avec $U + 1$ indisponibilités sur la machine M_1 est optimal.

□

3.5 Bornes supérieures et inférieures

3.5.1 Bornes supérieures

La valeur de cette borne est obtenue sur la base des heuristiques. La première consiste à ordonnancer les tâches selon la règle de Jackson, en tenant compte le positionnement des périodes d'indisponibilité et également la *non-préemption* des opérations. La deuxième heuristique est une amélioration de l'algorithme de Jackson pour le problème étudié.

Heuristique 1 : H1

Nous appliquons d'abord l'algorithme de Jackson pour ordonnancer les N tâches. Nous intégrons les U périodes d'indisponibilité en tant qu'opérations fictives et fixes commençant à s_u et se terminant à t_u . Soit J_i la tâche commençant juste avant s_u et se termine après cette date. Le traitement de cette tâche sur la machine M_1 commence à la date t_u (voir figure 3.10). Nous procédons de la même manière pour les autres indisponibilités, compte tenu des nouveaux dates de fin générées par la période d'indisponibilité précédente (Voir algorithme 1).

Algorithme 1 Heuristique 1 : H1

```

1: Entrée :
2:  $N$  tâches sur deux machines
3:  $U$  périodes d'indisponibilité sur la machine  $M_1$ 
4: Ordonnancer les tâches selon la règle de Jackson
5:  $u = 1$ 
6: Pour  $i = 1$  To  $N$  Faire
7:   Tant que  $u < U + 1$  Faire
8:     Si  $s_{i1} < s_u$  and  $c_{i1} > s_u$  Alors
9:        $s_{i1} = t_u$ 
10:      Actualiser les dates d'achèvement des tâches de  $I_{u+1}$ 
11:       $u = u + 1$ 
12:     Si  $c_{i1} \leq s_u$  Alors
13:       Actualiser les dates d'achèvement des tâches de  $I_u$ 
14:     break
15:   Fin Si
16: Fin Si
17: Fin Tant que
18: Fin pour
19: return  $C_{max}(H1)$ 

```

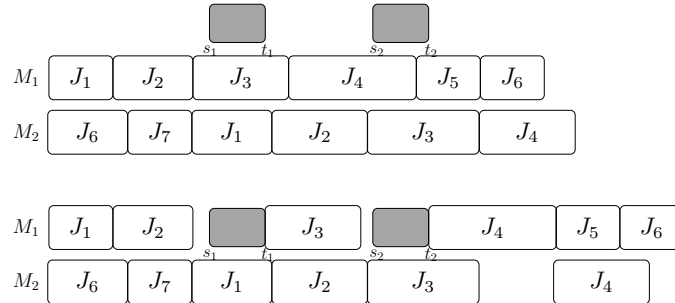


FIGURE 3.10 – Ordonnancement des tâches par l'heuristique H1

Lemme 1.

$$C_{max}(H1) \leq C_{max}^{JK} + \sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u \quad (3.25)$$

Démonstration. Soit G^{JK} le graphe disjonctif de la séquence obtenue selon la règle de Jackson. La valeur du plus long chemin est C_{max}^{JK} , le makespan de la séquence. Soit J_i la première tâche exécutée après la fin de la $u^{\text{ème}}$ indisponibilité. Nous considérons que le temps d'exécution de J_A sur la machine M_1 est $p_{i1} + g_u + Idle_u$. Le plus long chemin égal à $C_{max}(H1)$,

ne traverse pas nécessairement toutes les tâches comme J_i . Ainsi, dans tout les cas, la durée du chemin est au pire égale à $\sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u + C_{max}^{JK}$. $\square$

Propriété 7. Pour $J_2, h_{1U}|a|C_{max}$:

$$\frac{C_{max}(H1)}{C^*} \leq 1 + \frac{\sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u}{\sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u} \quad (3.26)$$

Démonstration. C_{max}^{JK} est le makespan du problème étudié sans contrainte de disponibilité, obtenu par l'algorithme de Jackson. Donc, il est évident que le makespan optimal de $J_2, h_{1U}|a|C_{max}$ est supérieur à C_{max}^{JK} .

$$C_{max}^{JK} \leq C^* \quad (3.27)$$

D'après le lemme 1,

$$\frac{C_{max}(H1) - C^*}{C^*} \leq \frac{C_{max}^{JK} + \sum_{u \in U} g_u + \sum_{u \in U} Idle_u - C^*}{C^*} \quad (3.28)$$

$$\leq \frac{C_{max}^{JK} + \sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u - C_{max}^{JK}}{C^*} \quad (3.29)$$

$$\leq \frac{C_{max}^{JK} + \sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u - C_{max}^{JK}}{\sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u} \quad (3.30)$$

Par conséquent,

$$\frac{C_{max}(H1)}{C^*} \leq 1 + \frac{\sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u}{\sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u} \quad (3.31)$$

$\square$

Theorem 3.1. Pour le problème $J_2, h_{1U}|a|C_{max}$, le ratio d'approximation par la règle de Jackson est égal à 2 dans le pire cas.

Démonstration. Nous considérons A et B tels que $A = \frac{\sum_{u=1}^U Idle_u}{\sum_{u=1}^U g_u}$ e $B = \frac{\sum_{i=1}^N p_{i1}}{\sum_{u=1}^U g_u}$.

Nous avons,

$$\frac{C_{max}(H1) - C^*}{C^*} \leq \frac{1 + A}{1 + B} \quad (3.32)$$

D'autre part, $\sum_{u=1}^U Idle_u \leq \sum_{i=1}^N p_{i1}$, donc $A \leq B$. Alors,

$$\frac{C_{max}(H1) - C^*}{C^*} \leq \frac{1 + A}{1 + B} \leq 1 \quad (3.33)$$

Par conséquent, le ratio d'approximation par la règle de Jackson ($\frac{C_{max}(H1)}{C^*}$) au pire cas est 2. $\square$

Propriété 8. Si $C_{max}(H1) = \sum_{i=1}^N p_{i2}$, alors le makespan obtenu par H1 est optimal.

Démonstration. $C_{max}(H1) = \sum_{i=1}^N p_{i2}$, donc, $\sum_{i=1}^N p_{i2} > C_1(H1)$. Alors,

$$\sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u \leq C_1(H1) \leq \sum_{i=1}^N p_{i2}$$

La borne inférieure de $J_2, h_{1U}|a|C_{max}$ est :

$$BI = \max\left(\sum_{i=1}^N p_{i2}, \sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u\right)$$

donc, $BI = \sum_{i=1}^N p_{i2}$. Par conséquent le makespan fourni par H1 est optimal. $\square$

Heuristique 2 : Jackson modifié (H2)

Nous introduisons quelques modifications à l'algorithme classique de Jackson afin d'améliorer sa précision par rapport aux périodes d'indisponibilité fixées. Nous regroupons les tâches dans des ensembles Set_{12} , Set_{21} , Set_1 et Set_2 .

- Les tâches appartenant aux ensembles Set_1 , Set_2 seront séquencées selon la règle *LPT*. Lee (1996) a montré que la règle LPT a une erreur dans le pire cas de $\frac{C_{LPT} - C^*}{C^*} \leq \frac{1}{3}$ pour le problème d'ordonnancement à une machine avec une seule contrainte de disponibilité et opérations non-sécables. Alors que pour des périodes d'indisponibilité multiples, Ji *et al.* (2007) ont prouvé que le ratio au pire cas de la règle *LPT* est égal $\frac{C_{LPT}}{C^*} \leq 2$ et ont montré qu'il n'y a pas d'algorithme d'approximation polynomiale au pire cas moins de 2 sauf si $P = NP$.

- Pour les tâches appartenant aux ensembles Set_{12} , Set_{21} :
 - Séquencer les tâches de $Sub_1 = \{J_i, p_{i1} \leq p_{i2}\}$ dans l'ordre croissant de p_{i1} . En cas d'égalité de p_{i1} , séquencer dans l'ordre décroissant de p_{i2} . En cas de toute autre égalité, séquencer arbitrairement.
 - Séquencer les tâches de $Sub_2 = \{J_i, p_{i1} > p_{i2}\}$ dans l'ordre décroissant de p_{i2} . En cas d'égalité de p_{i2} , séquencer dans l'ordre décroissant de p_{i1} . En cas de toute autre égalité, séquencer arbitrairement.

Nous introduisons les périodes d'indisponibilité en tenant en compte la non-préemption des opérations, comme décrit pour l'heuristique $H1$ (voir l'algorithme 2).

Algorithme 2 Heuristique 2 : Jackson modifié $H2$

```

1: Entrée :
2:  $N$  tâches sur deux machines
3:  $U$  périodes d'indisponibilité sur la machine  $M_1$ 
4: Ordonnancer les tâches selon la règle de Jackson modifiée
5:  $u = 1$ 
6: Pour  $i = 1$  To  $N$  Faire
7:   Tant que  $u < U + 1$  Faire
8:     Si  $s_{i1} < s_u$  and  $c_{i1} > s_u$  Alors
9:        $s_{i1} = t_u$ 
10:      Actualiser les dates d'achèvement des tâches de  $I_{u+1}$ 
11:       $u = u + 1$ 
12:     Si  $c_{i1} \leq s_u$  Alors
13:       Actualiser les dates d'achèvement des tâches de  $I_u$ 
14:     break
15:   Fin Si
16: Fin Si
17: Fin Tant que
18: Fin pour
19: return  $C_{max}(H1)$ 

```

La complexité des algorithmes heuristiques présentés est de $O(n \log n)$.

Soient $C_{max}(H1)$ et $C_{max}(H2)$ le makespan obtenu par $H1$ et $H2$ respectivement.

$$BS = \min(C_{max}(H1), C_{max}(H2)) \quad (3.34)$$

3.5.2 Bornes inférieures

La première borne inférieure BI_1 représente le cas où toutes les opérations sont exécutées sans aucun temps d'inactivité. Dans ce cas, nous avons :

$$BI_1 = \sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u \quad (3.35)$$

S'il n'y a aucun temps d'inactivité sur la machine M_2 :

$$BI_2 = \sum_{i=1}^N p_{i2} \quad (3.36)$$

BI_3 est égale à la date d'achèvement de la dernière opération des tâches ordonnancées selon la règle de Jackson.

$$BI_3 = C_{max}^{JK} \quad (3.37)$$

Nous calculons quatre bornes inférieures en se basant sur l'ordre proposé par la propriété 6.

Theorem 3.2. *Soit les bornes inférieures suivantes :*

$$BI_4 = t_U + \sum_{i \in \text{Set}_{12}^{I_{U+1}} \cup \text{Set}_{21}^{I_{U+1}} \cup \text{Set}_1^{I_{U+1}}} p_{i1} \quad (3.38)$$

$$BI_5 = \sum_{i \in \cup_{j=1}^U \text{Set}_{21}^{I_j}} p_{i2} + \min_{i \in \text{Set}_{21}^{I_{U+1}}} p_{i2} + \sum_{i \in \text{Set}_{21}^{I_{U+1}}} p_{i1} \quad (3.39)$$

$$BI_6 = \max\left(\sum_{i \in \text{Set}_{21} \cup \text{Set}_2} p_{i2}, \min_{i \in \text{Set}_{12}^{I_1}} p_{i1}\right) + \sum_{i \in \cup_{j=1}^{U+1} \text{Set}_{12}^{I_j}} p_{i2} \quad (3.40)$$

$$BI_7 = \max_{u \in \{1,2,\dots,U\}} \left(\max\left(\sum_{i \in \text{Set}_{21} \cup \text{Set}_2} p_{i2} + \sum_{i \in \cup_{j=1}^u \text{Set}_{12}^{I_j}} p_{i2}, t_u + \min_{i \in \text{Set}_{12}^{I_{u+1}}} p_{i1}\right) + \sum_{i \in \cup_{j=u}^U \text{Set}_{12}^{I_{j+1}}} p_{i2} \right) \quad (3.41)$$

Démonstration.

- L'exécution de l'ensemble des tâches considérées après la période d'indisponibilité ne se termine pas plus tôt que la somme de tout leurs périodes d'exécution. Après t_U , la machine M_1 finit au plus tôt à BI_4 si la somme des temps d'inactivité est égale à zéro.
- L'ordonnancement de l'ensemble $Set_{21}^{I_{U+1}}$ est un problème de type flow shop $F_2||C_{max}$. Il finit au plus tôt à $\min_{i \in Set_{21}^{I_{U+1}}} p_{i2} + \sum_{i \in Set_{21}^{I_{U+1}}} p_{i1}$. Comme l'exécution de $\cup_{j=1}^U Set_{21}^{I_j}$ finit à $\sum_{i \in \cup_{j=u}^U Set_{21}^{I_j}} p_{i2}$, alors : $C_{max} \geq \sum_{i \in \cup_{j=u}^U Set_{21}^{I_j}} p_{i2} + \min_{i \in Set_{21}^{I_{U+1}}} p_{i2} + \sum_{i \in Set_{21}^{I_{U+1}}} p_{i1}$.
- L'ordonnancement de l'ensemble $Set_{12}^{I_1}$ est un problème de type flow shop $F_2||C_{max}$. Cet ensemble est exécuté sur la machine M_1 puis M_2 . Son exécution sur la machine M_2 commence à $\min_{i \in Set_{12}^{I_1}} p_{i1}$ si la machine n'est pas en activité, sinon, à $\sum_{i \in Set_{21} \cup Set_2} p_{i2}$. La borne inférieure dans ce cas est BI_6 .
- Le $(u+1)^{\text{ème}}$ ensemble commence nécessairement après t_u . Donc, son exécution sur la machine M_2 commence à $t_u + \min_{i \in Set_{12}^{I_{u+1}}} p_{i1}$ si M_2 n'est pas en activité. Si ce n'est pas le cas, Son exécution sera à $\sum_{i \in Set_{21} \cup Set_2} p_{i2} + \sum_{i \in \cup_{j=1}^u Set_{12}^{I_j}} p_{i2}$.

□

Par conséquent, la borne inférieure BI est égale à la valeur maximale des bornes inférieures présentées ci-dessus.

$$BI = \max(BI_1, BI_2, BI_3, BI_4, BI_5, BI_6, BI_7) \quad (3.42)$$

3.6 Procédure par séparation et évaluation

Nous avons proposé dans la section précédente des propriétés sur l'optimalité de l'ordonnancement de type $J_2, h_{1U}|a|C_{max}$. Nous avons démontré que l'ordonnancement des ensembles des tâches avant et après chaque période d'indisponibilité suivent la règle de Jackson. Nous avons démontré également qu'il existe un ordre optimal pour la résolution d'un $J_2, h_{1U}|a|C_{max}$ garantissant cette condition. Cet ordre constitue une règle d'ordonnancement des tâches et nous amène à trouver une approche permettant de résoudre le problème de manière exacte. Cela signifie que l'objectif de l'algorithme à proposer est de définir les ensembles de tâches à exécuter avant, après et entre chaque deux périodes d'indisponibilité, puis de séquencer les ensembles suivant l'ordre optimal présenté la propriété 6.

Nous avons également proposé des bornes supérieures et inférieures du problème que nous allons employer dans une procédure par séparation et évaluation. Finalement, nous présentons et analysons les résultats expérimentaux menés sur des instances connues dans la littérature (Taillard (1993)).

Dans cette section, nous décrivons de manière détaillée la procédure par séparation et évaluation (*PSE*). Elle consiste à construire une arborescence où chaque noeud r correspond à une solution partielle du problème $J_2, h_{1U}|a|C_{max}$. Cette solution suit l'ordre développé dans la propriété 6. Elle garantit également que les tâches avant, après et entre chaque deux périodes d'indisponibilité suivent l'ordre de Jackson.

Un noeud de l'arborescence est caractérisé par les éléments suivants :

- Un niveau k représentant le nombre de tâches ordonnancées
- Une solution partielle, contenant un ordonnancement partiel des tâches traitées.
- Borne inférieure BI .

Auparavant, Aggoune (2002) proposait une procédure de séparation et évaluation (*PSE_Agg*) traitant le problème de type job shop pour la minimisation du makespan avec plusieurs contraintes de disponibilité sur chaque machine. Le graphe disjonctif est utilisé pour modéliser la solution dans chaque noeud de l'arbre. Par ailleurs, l'auteur a introduit une autre façon de prendre en compte les périodes de disponibilité des machines, en considérant des opérations fictives représentant les périodes d'indisponibilité. Le calcul des bornes inférieures est basé sur la résolution de sous-problèmes avec deux tâches, en tenant compte des contraintes de précédence et de disponibilité ainsi que des dates de latence des opérations. La sélection immédiate proposée par Carlier & Pinson (1989) pour le job shop sans contrainte de disponibilité, oriente directement certains arcs disjonctifs. Des sélections immédiates liées à l'activité de maintenance sont également ajoutées pour orienter les arcs reliant les opérations aux opérations fictives. En effet, en plus des bornes inférieures employées dans l'évaluation, la performance de *PSE_Agg* est due à l'utilisation de la sélection immédiate impliquant des opérations fictives dues à l'indisponibilité.

La procédure du *PSE* que nous suggérons est détaillée ci-après.

3.6.1 Initialisation

L'initialisation de la procédure avec une bonne solution permet d'accélérer la recherche en éliminant tous les noeuds moins bons que la solution initiale. Au départ, aucune tâche

n'est encore traitée. A ce niveau, la procédure nécessite une bonne borne supérieure et un ordre selon lequel les tâches seront traitées un par un. En effet, Nous déterminons la borne supérieure BS avant le commencement du branchement. Par ailleurs, cette initialisation fournit un ordre initial (ordre de Jackson obtenu par les heuristiques H1 et H2) qui sera l'ordre d'affectation des tâches lors de la procédure de branchement.

3.6.2 Branchement et séparation des noeuds

Nous adoptons une règle de branchement dans laquelle un noeud r au niveau k de l'arbre de recherche, correspondant à un ordonnancement partielle des k premières tâches. Le schéma de branchement consiste à choisir une nouvelle tâche non ordonnancée J_{k+1} et à décider de l'affecter dans les intervalles I_u . De ce fait, le noeud r possède $U + 1$ successeurs ($U + 1$ nouveaux solutions partiels) qui correspondent à $U + 1$ noeuds de branchement possibles. Cette affectation doit être effectuée de sorte que les ensembles de tâches (Set) soient ordonnancés selon l'ordre " S^* " démontré optimal (propriété 6). Une séquence complète est obtenue si toutes les tâches sont affectées. La stratégie d'exploration employée dans l'algorithme est une exploration en profondeur.

L'algorithme conserve la valeur du meilleur makespan trouvé s'il est inférieur à celui de la solution courante. Autrement, elle sera supprimée. Finalement, le branchement se termine lorsque tout les noeuds ont été visités ou une séquence complète avec un makespan égal à une borne inférieure est trouvée.

3.6.3 Évaluation des noeuds

L'évaluation d'une solution partielle consiste à calculer la valeur de sa borne inférieure, en l'occurrence le maximum des bornes inférieures BI_5 , BI_6 , BI_7 et BI_8 . Pendant l'évaluation, l'algorithme maintient la valeur de la meilleure solution trouvée. Lorsque la borne inférieure d'une solution partielle évaluée a une valeur supérieure ou égale à celle de la solution encours, il serait inutile d'explorer le noeud correspondant et ainsi tout ses successeurs seront éliminés.

Une autre évaluation existe et concerne le respect de la date de début la disponibilité de la machine. Une solution est exclue si $C_1^{AJK}(u) > s_u$. Le temps d'achèvement des tâches avant la $u^{\text{ème}}$ période de disponibilité ne peut pas dépasser sa date de début s_u . Une solution qui transgresse cette condition au niveau de la machine M_1 n'est pas faisable.

Algorithme 3 Algorithme de séparation et évaluation

Données :

- N tâches à traiter sur deux machines
- U Contraintes de disponibilité sur la machine M_1

Initialisation :

- Déterminer la borne supérieure BS
- Calculer la borne inférieure générale BI
- Déterminer l'ordre d'affectation des tâches (Ordre de la solution initiale)

Évaluation :

- Calculer la borne inférieure BI
- Si $BI \geq Best_Solution$ alors arrêter exploration
- Si $C_1^{AJK}(u) \geq s_u$ alors arrêter exploration

Séparation :

- Sélectionner le noeud à séparer
Si tout les noeuds sont des feuilles alors
 - $C_{max}^* = Best_Solution$
 - $sequence^* = sequence$
- Sélectionner la tâche à affecter
Si tout les tâches sont affectées (un noeud est une feuille), alors
 - Si $C_{max} < Best_Solution$ alors $Best_Solution = C_{max}$
 - Si $Best_Solution = BI$ alors arrêter
 - $C_{max}^* = Best_Solution$
 - $sequence^* = sequence$
- Effectuer la séparation et aller à l'étape **Évaluation**

Résultat :

- Makespan optimal : C_{max}^*
 - Séquence optimale : $sequence^*$
-

Tableau 3.2 – Paramètres de l'exemple

Jobs	1	2	3	4
Machine M_1	3	1	3	1
Machine M_2	2	3	1	3

3.6.4 Exemple illustratif

Nous illustrons par un exemple la procédure PSE proposée. Nous considérons deux machines exécutant quatre tâches. Les temps d'exécution des tâches sont présentées dans le tableau 3.2. Ces quatre tâches sont décomposées en deux ensembles $Set_{12} = \{3, 4\}$ et $Set_{21} = \{1, 2\}$. Concernant les périodes d'indisponibilité, la date de début de la première période est supposée à $s_1 = 3$ et sa fin à $t_1 = 5$, tandis que la seconde commence à $s_2 = 9$ et se termine à $t_2 = 11$.

Les figures 3.11 et 3.12 présentent la solution du problème d'ordonnancement sans et avec période d'indisponibilité respectivement. Le makespan obtenu par l'algorithme de Jackson est égal à $C_{max} = 9$.

$$BI = \max(BI1, BI2, BI3) = \max(8 + 2 + 2, 9, 9) = \max(12, 9, 9) = 12$$

La borne supérieure est la meilleure solution obtenue par les deux heuristiques $H1$, $H2$ (voir figure 3.12).

$$BS = \min(C_{max}(H1), C_{max}(H2)) = \min(15, 15) = 15$$

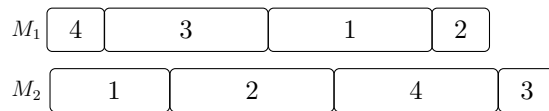
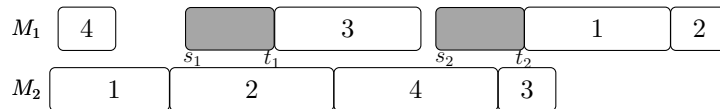


FIGURE 3.11 – Solution obtenue par l'algorithme de Jackson (sans période d'indisponibilité)


 FIGURE 3.12 – Solution trouvée par les heuristiques $H1$ et $H2$

Tout d'abord, nous obtenons la borne supérieure BS et l'ordre d'affectation des tâches pour le branchement. Dans cet exemple, l'ordre d'affectation des tâches est $[4, 3, 1, 2]$ et

BS est égal à 15. Dans chaque étape de branchement, trois noeuds sont possible à explorer, avant (B) ou après (A) ou entre les deux périodes d'indisponibilité (M), on calcule alors pour chaque noeud la borne inférieure correspondante, en considérant l'exploration en profondeur.

La figure 3.13 montre l'arborescence générée pour résoudre le problème, où (X) désigne un noeud éliminé. Le noeud correspondant à la solution partielle {4 Avant, 3 Avant} est exclu parce que la date d'achèvement de ces deux opérations est supérieur à la date de début de la première période d'indisponibilité. Le noeud suivant est la solution partielle {4 Avant, 3 Milieu }, etc. La date d'achèvement {4 Avant, 3 Après} est égale à la borne supérieure, il serait inutile de continuer l'exploration de ce noeud. La solution optimale est celle illustrée par la figure 3.14.

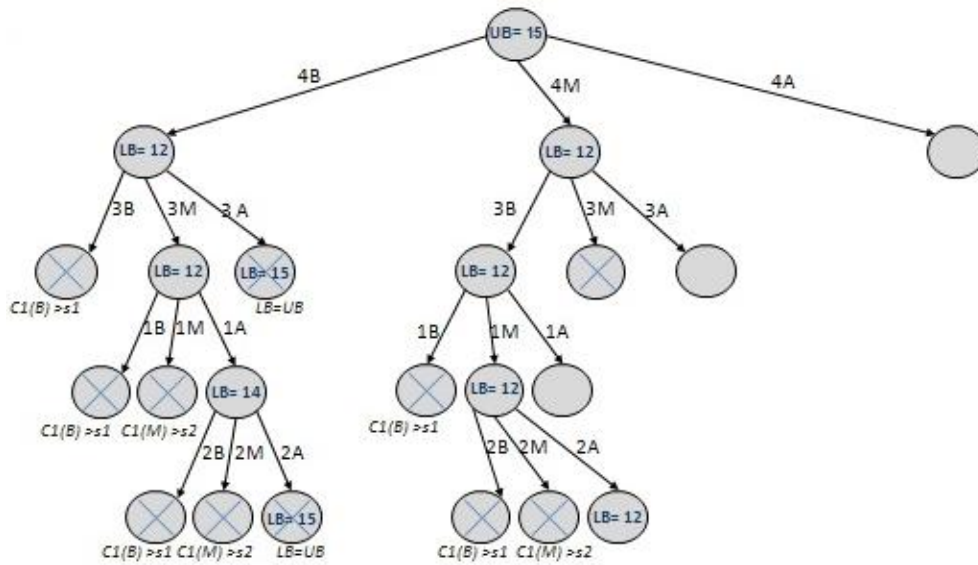


FIGURE 3.13 – L'arborescence du PSE pour l'exemple présenté

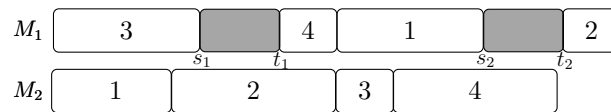


FIGURE 3.14 – La solution optimale obtenue par la PSE

3.7 Résultats expérimentaux

Nous fournissons des résultats numériques afin d'évaluer les performances des méthodes de résolution proposées. Les deux modèles MILP1 et MILP2 ont été résolus par le solveur IBM - Cplex interfacé avec C++ tandis que l'algorithme PSE proposé a été implémenté

sur $C++$. Les tests ont été exécutés sur un PC de fréquence 2.6 GHz Intel(R) Core (TM) i5 – 4210M CPU et 8.00 GB.

3.7.1 Description des instances

Afin d'évaluer l'efficacité des approches proposées de manière appropriée, nous effectuons nos tests sur certains benchmarks de la littérature. Nous comparons la PSE proposée à celle développée par Aggoune (2002).

Nous utilisons les instances (15 tâches / 15 machines, 20 tâches / 15 machines) Taillard (1993) pour le problème de type job shop. Pour chaque instance, nous avons pris seulement les deux premières valeurs de chaque tâche, correspondant à leurs durées sur les deux premières machines.

Nous ajoutons les paramètres concernant la période d'indisponibilité à fixer sur la machine M_1 . La durée de la $u^{\text{ème}}$ période d'indisponibilité est supposée égale à la moyenne des temps de traitement des opérations sur la machine M_1 : $g_u = t_u - s_u = \frac{\sum_{i=1}^N p_{i1}}{N}$.

Nous générons des instances prenant en compte l'emplacement de la $u^{\text{ème}}$ période d'indisponibilité au milieu de l'horizon d'ordonnancement disponible. La date de début de cette période est définie en fonction du nombre total de périodes d'indisponibilité prises en compte sur la machine M_1 : $g_u = t_u - s_u = \frac{\sum_{i=1}^N p_{i1}}{N}$.

Les positions des périodes d'indisponibilité sont choisies pour que la distance entre chaque deux périodes soit maximale. Le nombre de combinaisons des tâches possibles dans chaque sous-ensemble est donc maximal.

Les tests sont terminés lorsque l'optimum est trouvé ou lorsque le temps limite d'exécution de 3600 secondes a été atteint.

3.7.2 Analyse des résultats expérimentaux

Nous testons d'abord nos méthodes pour le problème avec deux périodes d'indisponibilité fixées sur la machine M_1 . D'après les résultats expérimentaux affichés dans le tableau 3.3, les modèles MILP1 et MILP2 ne peuvent résoudre de façon optimale aucune des instances lorsque la limite d'exécution est de 3600 secondes. Le temps requis pour trouver la meilleure solution est nettement plus important que le temps d'exécution limité. Cela est dû principalement à l'absence de borne inférieure de haute performance qui ferait

que la résolution converge rapidement vers la solution optimale. D'autant plus que les ordonnancements de permutation ne sont pas pris en compte dans la méthode de résolution.

Le tableau 3.4 illustre le pourcentage des instances résolues par la PSE proposée. Cette dernière est capable de résoudre de manière optimale 63 sur 80 instances.

Dans la suite, nous visons à déduire l'impact des paramètres N et U sur la performance de notre méthode en termes de complexité et de temps de calcul.

Le tableau 3.5 montre que les heuristiques fournissent une borne supérieure $BS = \min(C_{max}(H1), C_{max}(H2))$ de bonne qualité. Le gap égal à $\frac{BS - BI}{BI}$ permet d'illustrer ce résultat pour chaque cas traité. Comme indiqué dans la section 5, cet écart est dû à $\sum_{u=1}^U Idle_u$ par rapport à $\sum_{i=1}^N p_{i1}$ ainsi qu'à la taille de l'instance N . Ainsi, les instances présentant les écarts les plus importants sont celles considérant un nombre d'indisponibilités égal à $U = \{N/4, N/2\}$. Les heuristiques proposées se détériorent considérablement et c'est lorsque le nombre de temps d'inactivité générés augmente proportionnellement avec le nombre d'indisponibilités (voir les figures 3.15 et 3.16). Dans les autres cas, lorsque $U = 3N/4$, l'écart ($Gap = \frac{C_{max} - BI}{BI}$) diminue par rapport au cas de $U = \{N/4, N/2\}$ parce que lorsqu'un grand nombre de périodes d'indisponibilité est considéré, la durée des intervalles entre ces indisponibilités diminue. En conséquence, le problème devient un job shop de deux machines avec un petit nombre d'indisponibilités ayant de longues durées, d'où le petit écart de la solution initiale. De plus, ces instances sont plus faciles à résoudre, comme on le verra après. Un cas extrême doit être mentionné, quand $BS = \sum_{i \in N} p_{i2}$, donc BS est égal à BI (voir les valeurs BS en gras). Donc, la résolution des instances ayant $BS = BI$ (valeur en gras) par l'algorithme de Jackson est optimale.

Si la solution optimale ne peut pas être trouvée par la méthode exacte, la performance de la solution est indiquée par l'écart avec la borne inférieure $\frac{C_{max} - BI}{BI}$ (lorsqu'une solution optimale peut être trouvée, il n'est pas nécessaire de calculer cet écart et il est indiqué par le symbole $(-)$).

Pour le même problème et concernant la comparaison entre les procédures PSE et PSE Agg, les résultats présentés dans les tableaux 3.6, 3.7, 3.8 et 3.9 indiquent que ce dernier n'est capable de résoudre aucune instance de taille $N = \{15, 20\}$. D'autre part, dans certains cas, la valeur BS_{Agg} est meilleure que la valeur BS proposée, spécialement lorsque $U = \{N/2, N/4\}$. Nous avons donc choisi de lancer notre algorithme PSE avec la valeur $BS = \min(C_{max}(H1), C_{max}(H2), BS_{Agg})$.

Ainsi, la PSE résout optimalement toutes les instances de taille 15 tâches, quelque soit le nombre d'indisponibilités considérées. Pour les instances de 20 tâches, la PSE résout de façon optimale toutes les instances considérant $U = N/2, 3N/4$ périodes d'indisponibilité et 30% d'instances avec $U = N/2$. Pour $U = N/4$, les solutions obtenues pour 8 sur les 10 instances ont un écart inférieur à 1% (voir tableau 3.7). Nous remarquons également que les instances considérant $U = 3N/4$ sont plus faciles à résoudre que celle avec $U = N/4, N/2$. Intuitivement, plus la taille du problème (N, U) est grande, plus le temps de calcul est long vu le temps nécessaire pour tester tout les cas possibles. Néanmoins, le temps de calcul est moins important pour les problèmes considérant $U = 3N/4$. Ceci s'explique par le fait que plus le nombre de U est grand, plus le nombre d'intervalles avec de petites longueurs est grand, ce qui engendre moins de combinaisons. D'où la résolution facile du problème. Les tableaux 3.6 et 3.7 montrent que les solutions optimales sont égales à BI . Cela signifie que tout les temps d'inactivité générés par les périodes d'indisponibilité sont comblés ou que la date de fin de toutes les tâches est celle donnée par la machine M_2 . La limite de PSE_Agg est due principalement au nombre de tâches et de périodes d'indisponibilité sur une seule machine. En effet, en plus des limites inférieures employées pour l'évaluation, la performance de PSE_Agg est due à l'utilisation de la sélection immédiate impliquant des opérations de maintenance. Aggoune (2002) indique que PSE_Agg sont dédiés aux problèmes d'ordonnancement avec plusieurs indisponibilités sur chaque machine. Par ailleurs, les ordonnancement de permutation ne sont pas pris en compte par Aggoune (2002) dans la résolution. Donc, le nombre de combinaisons augmentera ainsi que les branches qui ne peuvent être rejetées ni par des bornes inférieures à cause du nombre de tâches à exécuter ni par la sélection immédiate. D'où le temps de calcul considérable.

Tableau 3.3 – Comparaison des modèles MILP1 et MILP2 ($N = \{15, 20\}$, $U = 2$)

N	$MILP1$			$MILP2$		
	C_{max}	$CPU(s)$	$Gap_Cplex(\%)$	C_{max}	$CPU(s)$	$Gap_Cplex(\%)$
15	902	3600	39.14	902	3600	28.64
	774	3600	9.56	774	3600	15.23
	884	3600	11.65	884	3600	18.39
	852	3600	12.31	852	3600	24.77
	918	3600	14.49	918	3600	20.54
	929	3600	12.59	929	3600	25.64
	1024	3600	13.96	1024	3600	16.84
	914	3600	10.83	914	3600	14.99
	1048	3600	18.22	1048	3600	27.64
	756	3600	6.75	756	3600	19.35
20	1118	3600	40.55	1118	3600	39.74
	1037	3600	38.94	1037	3600	48.25
	1115	3600	41.21	1115	3600	37.46
	1140	3600	42.74	1140	3600	53.21
	953	3600	43.23	953	3600	44.85
	1173	3600	30.34	1173	3600	36.98
	1222	3600	47.38	1222	3600	51.05
	1167	3600	39.07	1167	3600	42.84
	1138	3600	45.69	1138	3600	47.67
	1262	3600	53.80	1262	3600	51.42

Tableau 3.4 – Pourcentage des problèmes résolus optimalement par la PSE (%)

N	U			
	2	$N/4$	$N/2$	$3N/4$
15	100	100	100	100
20	100	0	30	100

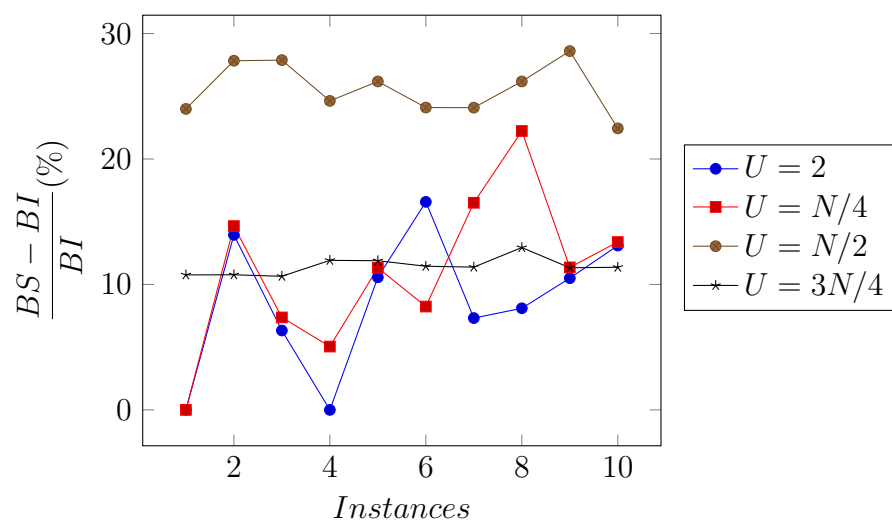


FIGURE 3.15 – $\frac{BS - BI}{BI}$ (%) pour chaque instance ($N = 15$, $U \in \{2, N/4, N/2, 3N/4\}$)

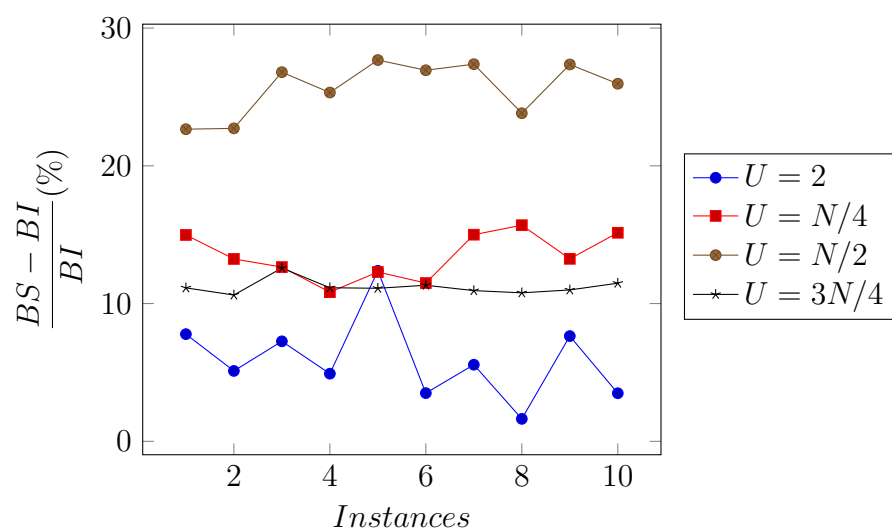


FIGURE 3.16 – $\frac{BS - BI}{BI}$ (%) pour chaque instance ($N = 20$, $U \in \{2, N/4, N/2, 3N/4\}$)

Tableau 3.6 – Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = 2$)

N	PSE_{Agg}						PSE					
	BI_{Agg}	BS_{Agg}	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$	BI	BS	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$
15	902	955	935	3.66	1490285	3600	902	902	902	–	–	–
	774	802	802	3.62	1475382	3600	774	802	774	–	2352	1.13
	884	986	934	5.66	1447908	3600	884	940	884	–	1625	0.95
	852	967	967	13.5	1695574	3600	852	852	852	–	–	–
	918	1010	1010	10.02	1684713	3600	918	1010	918	–	7063	2.15
	929	979	951	2.37	1721047	3600	929	979	929	–	5880	2.04
	1024	1122	1122	9.57	1718590	3600	1024	1099	1024	–	278	0.95
	914	950	946	3.50	1728932	3600	914	950	914	–	4146	1.61
	1048	1132	1113	6.20	1776276	3600	1048	1132	1048	–	5932	2.01
	756	832	813	7.54	1772278	3600	756	832	756	–	3552	1.21
20	1118	1201	1201	7.42	848969	3600	1118	1201	1118	–	34459	13.47
	1037	1115	1086	4.73	838937	3600	1037	1090	1037	–	20222	8.33
	1115	1162	1162	4.22	855360	3600	1115	1162	1115	–	4961	2.34
	1140	1305	1267	11.14	839235	3600	1140	1196	1140	–	111559	42.14
	953	1014	1014	6.40	850898	3600	953	1014	953	–	14488	5.14
	1173	1289	1254	6.91	844626	3600	1173	1214	1173	–	14119	5.38
	1222	1270	1270	3.93	838334	3600	1222	1270	1222	–	5380	2.06
	1167	1233	1233	5.66	837349	3600	1167	1186	1167	–	31460	12.43
	1138	1261	1228	7.91	847499	3600	1138	1225	1138	–	1592	1.14
	1262	1374	1314	4.12	850684	3600	1262	1306	1262	–	122370	46.14

Tableau 3.7 – Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = N/4$)

N	PSE_{Agg}						PSE					
	BI_{Agg}	BS_{Agg}	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$	BI	BS	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$
15	902	962	962	6.65	1715545	3600	902	902	902	–	–	–
	819	878	841	2.69	1765973	3600	819	878	819	–	18759	7.91
	936	1080	1047	11.86	1741528	3600	936	1005	937	–	945882	417.75
	852	1025	1025	20.31	1753516	3600	852	895	852	–	10106	4.47
	972	1048	1039	6.89	1755977	3600	972	1048	972	–	107632	44.42
	983	1046	1006	2.34	1754528	3600	983	1046	983	–	27270	10.03
	1084	1198	1192	9.96	1751277	3600	1084	1198	1085	–	45084	18.62
	967	1065	1065	11.00	1759000	3600	967	1065	967	–	169657	69.12
	1109	1209	1209	9.02	1754450	3600	1109	1209	1109	–	327186	149.80
	800	888	888	11.00	1756304	3600	800	888	800	–	7799	4.18
20	1268	1408	1408	11.04	856807	3600	1268	1408	1270	0.16	3939217	3600
	1178	1284	1284	9.00	853722	3600	1178	1284	1180	0.17	4007921	3600
	1265	1373	1336	5.61	836605	3600	1265	1373	1272	0.55	3982506	3600
	1293	1416	1416	9.51	838350	3600	1293	1416	1355	4.80	3723160	3600
	1082	1236	1186	9.61	846524	3600	1082	1215	1083	0.09	3987455	3600
	1332	1527	1527	14.64	824444	3600	1332	1485	1335	0.23	3987289	3600
	1387	1472	1472	6.13	795888	3600	1387	1472	1392	0.36	3973789	3600
	1326	1426	1391	4.90	839852	3600	1326	1426	1340	1.06	3646643	3600
	1291	1379	1379	6.82	842527	3600	1291	1379	1292	0.08	3992631	3600
	1433	1532	1532	6.91	818634	3600	1433	1532	1434	0.07	3643213	3600

Tableau 3.8 – Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = N/2$)

N	PSE_{Agg}						PSE					
	BI_{Agg}	BS_{Agg}	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$	BI	BS	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$
15	938	1131	1131	20.58	1658289	3600	938	1131	1029	–	523760	554.17
	999	1131	1128	12.91	1693178	3600	999	1131	1095	–	213057	249.23
	1144	1376	1376	20.28	1703227	3600	1144	1376	1285	–	198807	213.11
	1003	1212	1171	16.75	1560010	3600	1003	1212	1079	–	308714	319.78
	1188	1471	1471	23.82	1632221	3600	1188	1471	1311	–	3275	4.89
	1199	1424	1424	18.77	1649979	3600	1199	1424	1363	–	167103	183.04
	1324	1564	1564	18.13	1655118	3600	1324	1564	1545	–	1771	2.98
	1179	1400	1386	17.56	1656087	3600	1179	1400	1372	–	3427	3.71
	1353	1593	1593	17.74	1658559	3600	1353	1593	1591	–	15631	15.93
	976	1125	1125	15.27	1673164	3600	976	1125	1074	–	173175	183.36
20	1518	1779	1779	17.19	822250	3600	1518	1779	1695	11.66	1719625	3600
	1413	1585	1585	12.17	794376	3600	1413	1585	1574	11.39	1709996	3600
	1515	1777	1777	17.29	817816	3600	1515	1777	1711	–	1116639	2426.86
	1548	1771	1766	14.08	816427	3600	1548	1771	1676	–	1462950	3140.95
	1297	1461	1461	12.64	809811	3600	1297	1461	1456	12.26	1499409	3600
	1597	1898	1898	18.85	805067	3600	1597	1898	1811	13.40	1597563	3600
	1662	1901	1901	14.38	802743	3600	1662	1901	1828	9.98	1553095	3600
	1591	1890	1890	18.79	799090	3600	1591	1890	1844	15.90	1617751	3600
	1546	1853	1853	19.86	800066	3600	1546	1853	1776	14.88	1657225	3600
	1718	2052	2052	19.44	803729	3600	1718	2052	2003	–	40305	83.66

Tableau 3.9 – Comparaison entre PSE and PSE_Agg ($N = \{15, 20\}$, $U = 3N/4$)

N	PSE_{Agg}						PSE					
	BI_{Agg}	BS_{Agg}	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$	BI	BS	C_{max}	$Gap(\%)$	$Nodes$	$CPU(s)$
15	1106	1257	1257	13.65	1508379	3600	1106	1225	1199	–	7823	13.50
	1179	1295	1295	9.48	1608920	3600	1179	1295	1284	–	335	1.36
	1352	1572	1572	16.27	1612978	3600	1352	1496	1484	–	517	1.77
	1183	1524	1524	28.83	1548505	3600	1183	1324	1300	–	1910	3.57
	1404	1602	1602	14.10	1668449	3600	1404	1571	1524	–	290	2.17
	1415	1577	1577	11.45	1678737	3600	1415	1577	1555	–	656	3.22
	1564	1731	1731	10.68	1677040	3600	1564	1731	1727	–	37	1.14
	1391	1581	1540	10.71	1678920	3600	1391	1571	1526	–	290	1.75
	1597	1780	1779	11.40	1671451	3600	1597	1778	1778	–	325	2.05
	1152	1346	1334	15.80	1680518	3600	1152	1283	1249	–	2467	4.90
20	1768	1962	1948	10.18	819499	3600	1768	1962	1940	–	1082	3.57
	1648	1888	1865	13.17	824657	3600	1648	1823	1792	–	937	3.01
	1765	1976	1976	11.95	819651	3600	1765	1976	1939	–	4580	12.32
	1803	2104	2104	16.69	814038	3600	1803	2004	2003	–	70	1.05
	1512	1664	1664	10.05	816550	3600	1512	1664	1663	–	9221	28.74
	1862	2133	2133	14.55	820162	3600	1862	2073	2042	–	283	1.12
	1937	2136	2136	10.27	815972	3600	1937	2136	2129	–	984	3.58
	1856	2042	2042	10.02	818665	3600	1856	2042	2032	–	22216	72.13
	1801	2009	2009	11.55	815864	3600	1801	1999	1954	–	3058	11.16
	2003	2244	2244	12.03	816250	3600	2003	2233	2209	–	67	1.37

3.8 Conclusion

Nous avons présenté dans ce chapitre deux modèles mathématiques en nombre entiers mixtes et une procédure par séparation et évaluation (*PSE*) permettant de résoudre de manière exacte le problème d'ordonnancement du job shop à deux machines sous contraintes de disponibilité sur une machine.

Nous avons proposé de tenir en compte les propriétés d'optimalité de l'algorithme de Jackson dans la résolution par la procédure *PSE*. Également, une nouvelle règle d'ordonnancement a été présentée et son optimalité est également démontrée. Pour cela, des propriétés concernant principalement la prise en compte de l'indisponibilité dans l'application de l'algorithme de Jackson ont été prouvées. Cela nous a amené également à proposer des bornes inférieures et supérieures, basées sur cet algorithme.

Les résultats expérimentaux menés sur des instances adaptées de la littérature (instances de Taillard (1993)) prouvent l'efficacité de la procédure *PSE* mise en oeuvre. Cependant, la méthode exacte s'avère limitée pour la résolution des instances de grande taille et contenant un grand nombre de périodes d'indisponibilité. La résolution approchée est envisageable dans ce cas, pour l'obtention des solutions de bonne qualité dans un temps raisonnable.

L'indisponibilité des deux machines durant la même horizon d'ordonnancement est aussi un cas industriel réel. Le chapitre suivant sera consacré au problème d'ordonnancement de type job shop à deux machines avec une contrainte de disponibilité sur chaque machine.

Chapitre 4

Le problème d'ordonnancement de type Job shop à deux machines avec une contrainte de disponibilité sur chaque machine

Résumé :

Ce chapitre est consacré à l'étude du problème d'ordonnancement de type job shop à deux machines avec une contrainte de disponibilité sur chaque machine. Après une modélisation mathématique classique du problème, nous démontrons des propriétés théoriques sur la dominance des ordonnancements de permutation pour le cas étudié et aussi des propriétés sur l'optimalité de l'ordre de Jackson. Nous développons des bornes supérieures par des heuristiques. Une étude au pire cas est faite pour l'heuristique basée sur la règle de Jackson. Nous présentons par la suite deux procédures par séparation et évaluation ainsi qu'un modèle mathématique tenant en compte les propriétés établies. Une étude comparative basée sur des instances issues de la littérature et d'autres aléatoires est présentée et discutée.

Publications :

- Banttaleb, Mourad, Faicel Hnaien, and Farouk Yalaoui. Minimizing the makespan in the two-machine job shop problem under availability constraints. *International Journal*

of Production Research. Taylor & Francis, 2018.

4.1 Introduction

Nous avons étudié dans le chapitre précédent le problème d’ordonnancement de type job shop à deux machines avec la prise en compte des contraintes de disponibilité sur une machine. Des propriétés de dominance concernant l’optimalité de l’algorithme de Jackson ont été démontrées. Une procédure par séparation et d’évaluation a été présentée pour la minimisation du makespan dans le cas d’opération strictement *non-préemptive*.

Ce chapitre est consacré à l’étude du problème d’ordonnancement de type job shop à deux machines avec une contrainte de disponibilité sur chaque machine. Pour plusieurs raisons industrielles telles la maintenance préventive, chaque machine peut avoir besoin d’un arrêt ou plusieurs planifiés au cours de l’horizon de production.

La date de début et la durée de chacune des deux indisponibilités sont connues à l’avance et les opérations à exécuter sur chaque machine sont strictement *non-préemptives*. Les applications industrielles du problème sont nombreuses entre autres, l’industrie de fabrication/usinage et le transport.

Nous décrivons dans la section suivante le problème considéré. La troisième section est dédiée à la modélisation mathématique du problème ; un modèle de programmation linéaire en nombres mixtes est présenté. Dans la section 4, des propriétés étudiant la dominance des ordonnancements de permutation et l’optimalité du problème sont proposées. Nous développons dans les sections 5-6-7 respectivement des bornes supérieures et inférieures, deux procédures par séparation et évaluation et finalement une nouvelle formulation mathématique du problème basée sur la règle de Jackson. Les résultats expérimentaux sont finalement présentés pour évaluer l’approche proposée et les méthodes développées.

4.2 Description du problème

Le problème d’ordonnancement de type job shop à deux machines avec une période d’indisponibilité sur chaque machine $(J_2, h_{k1}|a|C_{max})$ se définit de la manière suivante.

- Un ensemble de N tâches à réaliser sur deux machines M_1 et M_2 .
- Une tâche J_i nécessite une seule machine à la fois durant p_{ij} .
- Chaque tâche J_i suit un ordre de traitement (σ_1^i, σ_2^i) à travers les deux machines.
- Chaque machine ne peut réaliser qu’une opération à la fois.

- Une période d'indisponibilité est considérée sur chaque machine. Leurs débuts et durées sont fixes et connus à l'avance.
- Les opérations sont supposées strictement non-préemptives. C'est à dire, le traitement d'une opération n'est interrompue ni par celui d'une autre, ni par l'indisponibilité.
- L'objectif est de trouver le séquençement des tâches sur les machines qui minimise le makespan.

4.3 Modélisation mathématique : MILP3

Nous utilisons la modélisation proposée par Applegate & Cook (1991). Les contraintes de disponibilité sont formulées et ajoutées au problème.

$$x_{ijk} = \begin{cases} 1 & \text{Si } J_i \text{ est séquencée avant } J_j \text{ sur } M_k \\ 0 & \text{Sinon} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{Si } J_i \text{ commence avant la période d'indisponibilité sur } M_k \\ 0 & \text{Sinon} \end{cases}$$

c_{ik} : Date de fin de la tâche J_i sur la machine M_k .

C_{max} : Makespan.

Fonction objectif

$$\text{Min } C_{max}$$

Contraintes

$$c_{ik} \geq p_{ik} \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.1)$$

$$c_{i\sigma_2^i} - p_{i\sigma_2^i} \geq c_{i\sigma_1^i} \quad \forall i \in \{1, \dots, N\} \quad (4.2)$$

$$c_{ik} - c_{jk} + B \cdot x_{ijk} \geq p_{ik} \quad \forall i \neq j \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.3)$$

$$c_{jk} - c_{ik} + B \cdot (1 - x_{ijk}) \geq p_{jk} \quad \forall i \neq j \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.4)$$

$$c_{ik} - B \cdot (1 - y_{ik}) \leq s_k \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.5)$$

$$c_{ik} - t_k \cdot (1 - y_{ik}) \geq p_{ik} \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.6)$$

$$C_{max} \geq c_{ik} \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.7)$$

$$C_{max}, c_{ik} \text{ entier} \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.8)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.9)$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.10)$$

Les contraintes (4.1) assurent que la date de fin de la tâche J_i n'est pas antérieure à son temps de traitement sur la machine M_k . Les contraintes (4.2) indiquent que la date de début de la tâche J_i sur $M_{\sigma_2^i}$ n'est pas antérieure à son date de fin sur $M_{\sigma_1^i}$. La tâche J_i devrait être exécutée dans l'ordre $(M_{\sigma_1^i}, M_{\sigma_2^i})$. Les contraintes (4.3) et (4.4) garantissent le non chevauchement des opérations des tâches J_i et J_j exécutées sur la même machine M_k , avec B un grand nombre. Les contraintes (4.5) garantissent que la date de fin de l'exécution des tâches avant la période d'indisponibilité sur la machine M_k ne dépasse jamais s_k . Les contraintes (4.6) assurent que le date de fin d'exécution de la tâche J_i sur la machine M_k est supérieur ou égal à son temps d'exécution plus t_u s'il est prévu après la période d'indisponibilité. Les contraintes (4.7) confirment que le makespan est supérieur ou égal à l'heure d'achèvement de chaque tâche. Les contraintes (4.8) indiquent que le makespan et la date de fin de chaque tâche sont des nombres entiers. Les contraintes (4.9) et (4.10) fournissent des restrictions binaires pour y_{ik} et x_{ijk} .

4.4 Propriétés

Propriété 9. *Les ordonnancements de permutation ne sont pas dominants pour le problème $J_2, h_{kU}/a/C_{max}$.*

Démonstration. Lee (1999) a prouvé que l'ordonnancement de permutation n'est pas nécessairement dominant pour le problème de type flow shop à des deux avec contraintes de disponibilité sur les deux machines. Il propose un contre-exemple et considère une instance avec 3 tâches J_i (p_{i1}, p_{i2}), à savoir, J_1 ($1, 2k+1$), J_2 ($k, 2$) et J_3 ($k+1, 1$) et deux périodes d'indisponibilité commençant à $s_1 = k+1$, $s_2 = 2k+4$ et se terminant à $t_1 = k+2$, $t_2 = 2k+5$ respectivement (Tableau 4.1). La seule solution optimale est de séquencer $\{J_1, J_2, J_3\}$ sur M_1 et $\{J_1, J_3, J_2\}$ sur M_2 (voir figure 4.1), avec makespan égal à $2k+7$. Toute autre séquence respectant les ordonnancements de permutation fournit un makespan supérieur à $2k+7$.

Tableau 4.1 – Temps d'exécution Lee (1999)

Jobs	1	2	3
Machine M_1	1	k	k+1
Machine M_2	2k+1	2	1

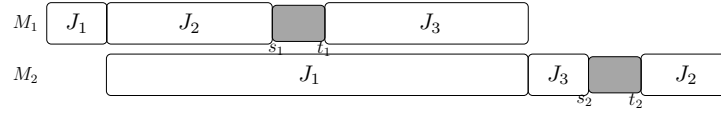


FIGURE 4.1 – Illustration du contre-exemple

Comme le problème de type flow shop est un cas particulier du problème de type job shop, les ordonnancements de permutation ne sont pas dominants pour $J_2, h_{kU}/a/C_{max}$.

□

Par la propriété suivante, nous démontrons que les ordonnancements de permutation restent dominants pour $J_2, h_{k1}|a|C_{max}$ sous une certaine condition.

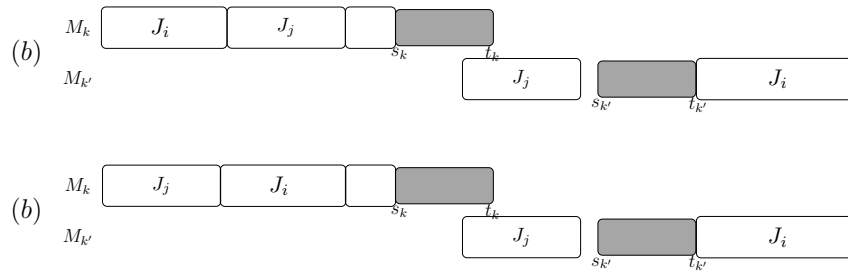
Propriété 10. *Pour $J_2, h_{k1}|a|C_{max}$, si $\min_{J_i \in J}(p_{i1} + p_{i2}) > \max(s_1 - t_2, s_2 - t_1)$, les ordonnancements de permutation sont dominants.*

Démonstration.

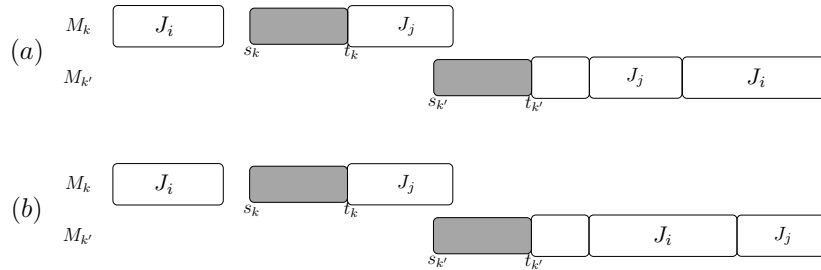
Nous supposons qu'il existe une séquence telle que J_i précède immédiatement J_j sur la machine M_k et la suit sur la machine $M_{k'}$. Soient $J_{A^k}, J_{A^{k'}} \in Set_{kk'}$ les premières tâches démarrant après la fin de la période d'indisponibilité sur la machine $M_k, M_{k'}$ respectivement.

Nous discutons trois cas possibles, y compris le cas où un séquençement de deux tâches consécutives (J_i, J_j) est interrompu par une période d'indisponibilité.

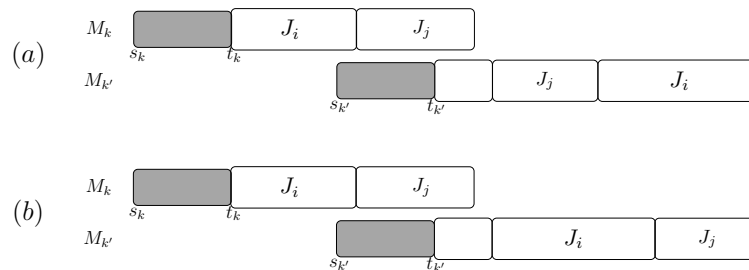
Cas (1) : Le traitement continu des deux tâches $\{J_i, J_j\}$ est interrompu par la période d'indisponibilité sur la machine $M_{k'}$. Dans ce cas, $J_i = J_{A^{k'}}$ et J_j est traitée avant $s_{k'}$ sur la machine $M_{k'}$. Tenant compte la condition $\min_{J_i \in J}(p_{i1} + p_{i2}) > \max(s_1 - t_2, s_2 - t_1)$, l'exécution des tâches J_i et J_j se termine avant s_k parce que J_i précède J_j sur M_k . Donc, traiter J_j avant J_i sur la machine M_k n'augmentera pas le makespan (voir figure 4.2).


 FIGURE 4.2 – Illustration du cas $J_i = J_{A^{k'}}$

Cas (2) : La période d'indisponibilité sur la machine M_k désamorce le traitement continu des deux tâches $\{J_i, J_j\}$. Cela engendre le cas $J_j = J_{A^k}$. J_i et J_j sont traitées après $t_{k'}$. Donc, traiter J_i avant J_j sur la machine $M_{k'}$ n'impactera pas le makespan (voir figure 4.3) étant considéré la condition $\min_{J_i \in J} (p_{i1} + p_{i2}) > \max(s_1 - t_2, s_2 - t_1)$.


 FIGURE 4.3 – Illustration du cas $J_j = J_{A^k}$

Cas (3) : Les deux tâches $\{J_i, J_j\}$ sont traitées sur les deux machines sans interruption. ($\{J_{A^k}, J_{A^{k'}}\} \neq \{J_i, J_j\}$). Les tâches J_i et J_j sont tout les deux exécutées avant ou après la période d'indisponibilité sur chaque machine. Donc traiter J_i juste avant J_j sur les deux machines n'impactera pas le makespan (voir figure 4.4 pour l'illustration).


 FIGURE 4.4 – Illustration du cas $J_i = J_{A^k}$

En conclusion, quel que soit le positionnement des deux tâches J_i et J_j , les ordonnancements de permutation sont dominants pour $J_2, h_{k1}|a|C_{max}$, si $\min_{J_i \in J}(p_{i1} + p_{i2}) > \max(s_1 - t_2, s_2 - t_1)$.

□

Nous utiliserons la notation $J_2, h_{k1}|a, d_{t_k, s_{\bar{k}}}|C_{max}$ pour désigner le problème souligné par la propriété précédente. Cette propriété permet de prendre en compte les ordonnancements de permutation. Ainsi, le temps de résolution serait moins important en raison de la simplicité combinatoire relative à l'ordonnement considérant le même ordre des tâches sur les deux machines. De plus, le cas considéré est également important dans l'industrie et essentiel pour optimiser des cas industriels. Par exemple, certaines activités d'intervention nécessitant l'arrêt des deux machines (Exemple : la maintenance préventive) peuvent être effectuées simultanément sur les deux machines ou consécutivement ou nécessitent une durée minimale entre les deux arrêts pour de nombreuses raisons telles que le déplacement ou le transport des ressources d'intervention. Ainsi, une durée maximale entre les deux indisponibilités peut être tolérée tout en assurant que la condition reste valide qui garantit que les ordonnancements de permutation sont dominants. Les cas où les deux périodes d'indisponibilité commencent au même temps ou lorsqu'elles se suivent immédiatement sont également inclus.

Corollaire 1. *Pour $J_2, h_{k1}|a|C_{max}$, les ordonnancements de permutation sont dominants lorsque les deux périodes d'indisponibilité commencent au même moment ou lorsqu'elles se succèdent immédiatement.*

Comme mentionné par Jackson (1956) et démontré pour $F2||C_{max}$ par Johnson (1954), les ordonnancements de permutation sont dominants pour la résolution optimale du problème de type job shop à deux machines sans contraintes de disponibilité ($J_2||C_{max}$). Nous avons démontré par la propriété 10 qu'il existe une condition sous laquelle les ordonnancements de permutation restent dominants pour le problème étudié. Cela nous permettra d'utiliser les propriétés suivantes.

Propriété 11. *Pour $J_2, h_{k1}|a, d_{t_k, s_{\bar{k}}}|C_{max}$, il existe un ordre optimal " S^* " suivant la règle de Jackson avant et après la période d'indisponibilité sur chaque machine (figure 4.5).*

Démonstration. Le raisonnement développé concernera la machine M_1 mais reste valide pour celui sur la machine M_2 car le problème est symétrique.

Nous définissons le problème (Pb1) qui considère l'ensemble des tâches à traiter avant la période de disponibilité sur la machine M_1 (voir figure 4.6). Le problème Pb1 est de

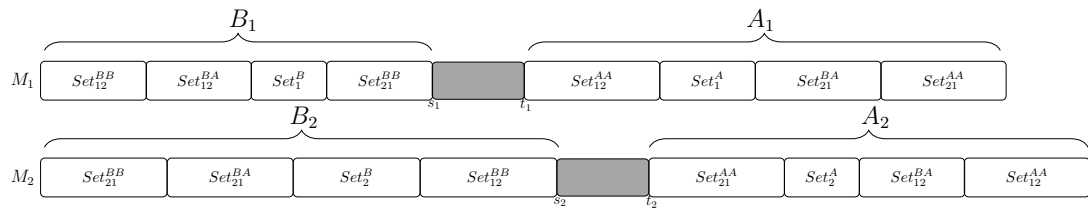


FIGURE 4.5 – Ordonnancement des tâches selon l'ordre optimal " S^* " pour $J_2, h_{k1}|a, d_{t_k, s_k}|C_{max}$

type $J_2, h_{11}|a|C_{max}$, donc l'ordre optimal est celui fourni par la propriété 2 garantissant par ailleurs que les tâches avant et après l'indisponibilité commencent à s_2 et finissant à t_2 sur la machine M_2 sont séquencées selon la règle de Jackson.

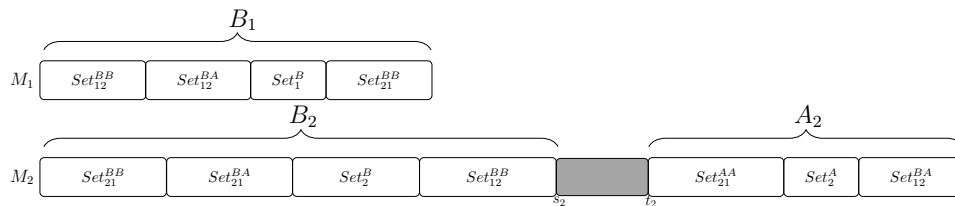


FIGURE 4.6 – Ordonnancement des tâches selon l'ordre optimal " S^* " avant la période d'indisponibilité sur la machine M_1

Concernant les tâches à exécuter après la période d'indisponibilité sur la machine M_1 (voir figure 4.7). Le problème d'ordonnancement de ces tâches (Pb2) est de type $J_2, h_{21}|a, s_1 = 0|C_{max}$. La propriété 2 affirme que les tâches dans chaque intervalle B_2 et A_2 sont ordonnés selon la règle de Jackson. L'ajout d'une période d'indisponibilité sur la machine M_1 à $t = 0$ et finissant à $t = t_1$ n'affectera pas la séquence optimale des ensembles (voir la propriété 1). D'autre part, si nous supposons que l'ordre proposé n'est pas optimal, il existerait donc un ordonnancement optimal assurant un makespan inférieur à celui donné par " S^* ". Néanmoins, l'échange successif de la position des tâches conduira de n'importe quel ordonnancement donné à un au moins aussi bon que celui donné par l'algorithme de Jackson.

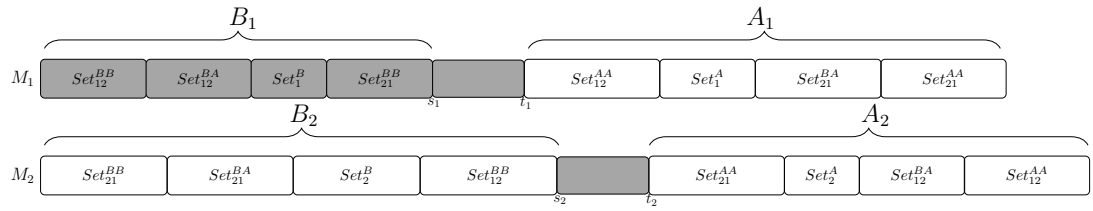


FIGURE 4.7 – Ordonnancement des tâches selon l'ordre optimal "S*" après la période d'indisponibilité sur la machine M_1

□

4.5 Bornes supérieures et inférieures

4.5.1 Bornes supérieurs

La valeur de la borne supérieure est obtenue par l'une des deux heuristiques suivante. La première consiste à séquencer les tâches selon la règle de Jackson, en tenant compte les périodes d'indisponibilité et de la *non-préemption* des opérations. La deuxième heuristique est une modification de l'algorithme de Jackson.

Heuristique 1 : H3

Nous appliquons la méthode de Jackson pour ordonnancer les N tâches, puis nous intégrons les périodes d'indisponibilité comme une opération fictive et fixe sur chaque machine M_k commençant à s_k et se terminant à t_k . Le traitement de la tâche J_i commençant juste avant s_k et se terminant après cette date sur la machine M_k commencera à la date t_k (voir figure 4.8).

Propriété 12. Pour $J_2, h_{k1}|a, d_{t_k, s_k}|C_{max}$,
$$\frac{C_{max}(H3) - C^*}{C^*} \leq \frac{\max_{i \in \{1,2\}}(g_i + Idle_i)}{C_{max}^{JK}}.$$

Démonstration. Soit C_k^{JK} la date de fin de traitement de la dernière tâche sur la machine M_k obtenue par l'algorithme de Jackson pour $J_2||C_{max}$ et $C_k(H3)$ celle obtenue par l'heuristique H3 sur la machine M_k . Soit J_{Ak} la première tâche traitée après la fin de la période d'indisponibilité sur la machine M_k . Nous considérons que le temps de traitement de J_{Ak} sur la machine M_k est $p_{ik} + g_k + Idle_k$. La date de fin maximale sur chaque machine M_k se termine au plus tard à $C_k^{JK} + g_k + Idle_k$ si le traitement des tâches selon la règle de Jackson,

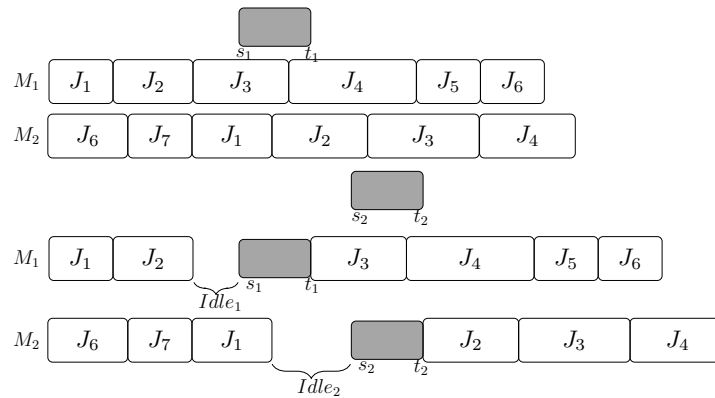


FIGURE 4.8 – Ordonnancement des tâches par l'heuristique H3

sans tenir compte des contraintes de disponibilité, se fait sans aucun temps d'inactivité. Alors,

$$C_k(H3) \leq C_k^{JK} + g_k + Idle_k \quad \forall k \in \{1, 2\} \quad (4.11)$$

$$\leq C_k^{JK} + \max_{i \in \{1, 2\}} (g_i + Idle_i) \quad \forall k \in \{1, 2\} \quad (4.12)$$

Le makespan obtenu par l'algorithme H3,

$$C_{max}(H3) \leq C_{max}^{JK} + \max_{i \in \{1, 2\}} (g_i + Idle_i) \quad (4.13)$$

C_{max}^{JK} est le makespan de l'ordonnancement selon la règle de Jackson sans contraintes de disponibilités. Donc, le makespan optimal (C^*) pour $J_2, h_{k1}|a, d_{t_k, s_{\bar{k}}}|C_{max}$ est supérieure à C_{max}^{JK} . Donc,

$$C_{max}^{JK} \leq C^* \quad (4.14)$$

Par conséquent,

$$C_{max}(H3) \leq C^* + \max_{i \in \{1, 2\}} (g_i + Idle_i) \quad (4.15)$$

$$\frac{C_{max}(H3) - C^*}{C^*} \leq \frac{\max_{i \in \{1, 2\}} (g_i + Idle_i)}{C^*} \quad (4.16)$$

$$\frac{C_{max}(H3) - C^*}{C^*} \leq \frac{\max_{i \in \{1,2\}}(g_i + Idle_i)}{C_{max}^{JK}} \quad (4.17)$$

□

Heuristique 2 : Jackson modifié (H4)

Nous appliquons les mêmes modifications de l'algorithme de Jackson, décrites au chapitre précédent (section 1.5.1 Heuristique 2)

Soient $C_{max}(H3)$ et $C_{max}(H4)$ les makespan obtenus par $H3$ et $H4$ respectivement.

$$BS = \min(C_{max}(H3), C_{max}(H4)) \quad (4.18)$$

4.5.2 Bornes inférieures

Nous présentons les bornes inférieures pour le problème $J_2, h_{k1} | a, d_{t_k, s_k} | C_{max}$.

BI_1 représente le cas où toutes les opérations sont traitées consécutivement sans aucun temps d'inactivité.

$$BI_1 = \max\left(\sum_{i=1}^N p_{i1} + g_1, \sum_{i=1}^N p_{i2} + g_2\right) \quad (4.19)$$

BI_2 est le makespan des tâches séquencées selon la règle de Jackson sans contraintes de disponibilité.

$$BI_2 = C_{max}^{JK} \quad (4.20)$$

4.6 Procédure par séparation et évaluation

Nous développons dans cette section des procédure par séparation et évaluation pour la résolution exacte du problème. Nous présentons deux schémas de branchement, leurs noeuds correspondant ainsi que la stratégie d'exploration adoptée par chaque procédure.

Le choix du schéma branchement est primordial pour la performance d'une PSE. Cette décision devrait être basée sur un nombre limité d'informations sur le noeud qui pourrait conduire à une solution optimale. De plus, le nombre de noeuds imposés dans chaque niveau

ne donne pas nécessairement une information précise sur la qualité des bornes calculées. Donc, la stratégie d'exploration devrait fournir précisément une valeur significative de bornes dans chaque niveau, de sorte que la convergence vers l'optimalité soit rapide. Ces choix ont donc une influence majeure sur l'efficacité de la procédure.

Les deux schémas de branchement garantissent que les tâches exécutées avant et après chaque période d'indisponibilité sont séquencées selon la règle de Jackson. Ainsi, chaque solution trouvée assure que l'ordre des tâches avant et après la période d'indisponibilité est optimal. Donc, l'objectif de ces schémas est finalement de définir les ensembles de tâches à exécuter avant et après la période d'indisponibilité sur chaque machine, en respectant l'ordre démontré optimal par la propriété 11.

4.6.1 Le premier schéma de branchement

Nous avons prouvé, par la propriété 11, qu'il existe un ordre optimal pour le problème d'ordonnancement $J_2, h_{k1}|a, d_{t_k, s_{\bar{k}}}|C_{max}$. Tous les ensembles définis dans cet ordre pourraient être classés en trois catégories de tâches comme suit,

- $Cat_1 = \{Set_{12}^{BB}, Set_1^{BB}, Set_2^{BB}, Set_{21}^{BB}\}$ contient les tâches traitées entièrement avant le début de toute période d'indisponibilité sur les deux machines.
- $Cat_2 = \{Set_{12}^{BA}, Set_{21}^{BA}\}$ inclut les tâches traitées sur l'une des deux machines avant la période d'indisponibilité correspondante et terminés après celle de la machine suivante.
- $Cat_3 = \{Set_{12}^{AA}, Set_1^{AA}, Set_2^{AA}, Set_{21}^{AA}\}$: concerne des tâches dont le traitement sur chaque machine ne commence qu'après la fin de la période d'indisponibilité.

L'objectif de cette procédure est de définir les ensembles de tâches appartenant à chaque catégorie, planifiés ainsi suivant l'ordre démontré par la propriété 11, respectant la règle de Jackson dans chaque catégorie.

Theorem 4.1. BI_3, BI_4, BI_5 et BI_6 sont quatre bornes inférieures pour PSE1 :

$$BI_3 = t_1 + \sum_{i \in Set_{12}^{AA} \cup Set_1^A \cup Set_{21}^{BA} \cup Set_{21}^{AA}} p_{i1} \quad (4.21)$$

$$BI_4 = t_2 + \sum_{i \in Set_{21}^{AA} \cup Set_2^A \cup Set_{12}^{BA} \cup Set_{12}^{AA}} p_{i2} \quad (4.22)$$

$$BI_5 = t_1 + \min_{i \in Set_{12}^{AA}} p_{i1} + \sum_{i \in Set_{12}^{AA}} p_{i2} \quad (4.23)$$

$$BI_6 = t_2 + \min_{i \in Set_{21}^{AA}} p_{i2} + \sum_{i \in Set_{21}^{AA}} p_{i1} \quad (4.24)$$

Démonstration.

- Le traitement de l'ensemble des tâches après la période d'indisponibilité se termine au plus tôt à la somme de tout leurs temps d'exécution. Après t_k , la machine M_k finit au plus tôt à BI_3 , BI_4 pour $k = 1$, $k = 2$ respectivement, si la somme des temps d'inactivité est égale à zéro.
- L'ensemble Set_{12}^{AA} est un problème d'ordonnancement de type flow shop $(F_2||C_{max})$. Il se termine au plus tôt à $\min_{i \in Set_{12}^{AA}} p_{i1} + \sum_{i \in Set_{12}^{AA}} p_{i2}$. Comme le traitement de Set_{12}^{AA} commence à t_1 , alors : $BI_5 = t_1 + \min_{i \in Set_{12}^{AA}} p_{i1} + \sum_{i \in Set_{12}^{AA}} p_{i2}$.
- De même, Set_{21}^{AA} est un flow shop $F_2||C_{max}$ commençant après t_2 . Donc, $BI_6 = t_2 + \min_{i \in Set_{21}^{AA}} p_{i2} + \sum_{i \in Set_{21}^{AA}} p_{i1}$.

□

La borne inférieure du problème BI_{PSE1} est égale à la valeur maximale des bornes inférieures présentées ci-dessus.

$$BI_{PSE1} = \max(BI_1, BI_2, BI_3, BI_4, BI_5, BI_6) \quad (4.25)$$

Un noeud de l'arbre est caractérisé par les éléments suivants :

- Un niveau $k \leq N$ représentant le nombre de tâches ordonnancées.
- Une solution partielle correspondant à un ordonnancement partiel des tâches réalisées.
- Une borne inférieure, BI_{PSE1} .

Un noeud r au $k^{\text{ème}}$ niveau de l'arborescence correspond à un ordonnancement partiel des k premières tâches candidates. La liste de ces tâches est triée selon la règle de Jackson sur la machine M_1 . Le schéma de branchement consiste à choisir une nouvelle tâche non ordonnancée J_{k+1} et à décider de l'affecter à l'une des trois catégories. Par conséquent, le

noeud r a trois noeuds successeurs possibles. Une séquence complète est obtenue si toutes les tâches sont affectées. La stratégie de branchement en profondeur est adoptée. Il consiste à choisir le premier noeud candidat descendant du noeud courant dans l'arbre.

L'évaluation d'une solution partielle consiste à calculer la valeur de sa borne inférieure. Lorsque la borne inférieure d'une valeur de solution partielle est supérieure ou égale à la valeur de la meilleure solution en cours maintenue, il sera inutile de continuer l'exploration du noeud correspondant. Une autre évaluation concerne le respect de la date de début de la période disponibilité de la machine et la *non-préemption* des opérations. En effet, une solution est exclue si la date de fin des tâches sur chaque machine avant la période d'indisponibilité est supérieure à sa date de début. La procédure se termine lorsque tous les noeuds ont été visités ou une séquence complète avec un makespan égal à la borne inférieure est trouvée.

4.6.2 Le second schéma de branchement

Nous proposons un autre schéma de branchement basé principalement sur la stratégie adoptée pour l'ordonnancement lorsque l'occurrence d'une période d'indisponibilité est limitée à une machine. Au début, l'algorithme de Jackson définit les séquences de tâches sur chaque machine M_1 et M_2 . Ces séquences sont les listes des candidats sur chaque machine. Le but de cette stratégie est de décider pour chaque tâche candidate sa date de traitement avant ou après la période d'indisponibilité sur chaque machine. Ainsi, le schéma de branchement affecte les tâches avant ou après la période d'indisponibilité sur chaque machine suivant l'ordre donné par la liste des candidats.

A chaque noeud r au $k^{\text{ème}}$ niveau de l'arbre de recherche est associée un ordonnancement partiel composé des $k \leq 2N$ premières opérations, réalisées sur les deux machines. Le schéma de branchement consiste à choisir la machine sur laquelle la tâche est exécutée et de décider par la suite son affectation avant ou après l'indisponibilité. En effet, la machine est choisie si le nombre de tâches restant à exécuter est le plus petit. Par conséquent, le noeud r a deux successeurs (deux nouvelles solutions partielles). Cette affectation doit être effectuée de sorte que les tâches soient séquencées suivant l'ordre donné par la propriété 11. Une séquence complète est obtenue si tous les tâches sont affectées. La stratégie d'exploration de l'algorithme est en profondeur comme utilisée par le schéma (*PSE1*).

Pendant l'évaluation, l'algorithme maintient la valeur de la meilleure solution trouvée. La faisabilité de chacune des solutions partielles est vérifiée et sa borne inférieure est calculée et comparée à la meilleure solution actuelle. S'il peut être établi que la solution partielle

ne peut pas amener à une solution optimale, la branche entière est rejetée. Une solution n'est pas réalisable si la date de fin des tâches avant la période d'indisponibilité sur une des deux machines est supérieure à sa date de début ou si la contrainte de précédence entre les opérations d'une même tâche n'est pas respectée. La procédure se termine lorsque tous les noeuds ont été visités ou que le makespan d'une séquence complète est égal à la borne inférieure du problème.

Theorem 4.2. *BI_7 et BI_8 sont deux bornes inférieures pour l'évaluation des noeuds générés par la procédure PSE2.*

$$BI_7 = t_1 + \sum_{i \in A1} p_{i1} \quad (4.26)$$

$$BI_8 = t_2 + \sum_{i \in A2} p_{i2} \quad (4.27)$$

Démonstration.

Le traitement de l'ensemble des tâches attribuées après la période d'indisponibilité se termine au plus tôt à la somme de tous leurs temps de traitement. Après t_k , la machine M_k finit au plus tôt à BI_7 , BI_8 pour $k = 1$, $k = 2$ respectivement, si la somme des temps d'inactivité est égale à zéro.

□

La borne inférieure BI_{PSE2} est égale à,

$$BI_{PSE2} = \max(BI_1, BI_2, BI_7, BI_8) \quad (4.28)$$

Un noeud de l'arborescence est caractérisé par les éléments suivants :

- Un niveau $k \leq 2N$ représentant le nombre de tâches séquencées.
- Une solution partielle, d'un ordonnancement partiel des opérations traitées.
- Une borne inférieure BI_{PSE2} .

4.6.3 Exemple illustratif

Nous prenons un exemple pour illustrer nos procédures PSE1 et PSE2. Nous considérons que deux machines sont nécessaires pour traiter quatre tâches. Ces tâches appartiennent à

Tableau 4.2 – Paramètres de l'exemple considéré

Jobs	1	2	3	4
Machine M_1	3	1	3	1
Machine M_2	2	3	2	2

deux ensembles $Set_{12} = \{3, 4\}$ et $Set_{21} = \{1, 2\}$ (voir Tableau 4.2). La date de début de la première période d'indisponibilité est supposée à $s_1 = 3$ et sa fin à $t_1 = 6$, alors que la seconde commence à $s_2 = 8$ et se termine à $t_2 = 10$.

Le makespan obtenu par l'algorithme de Jackson pour le problème sans contraintes de disponibilité est égal à $C_{max} = 9$. Nous déterminons la valeur de la borne supérieure comme la meilleure solution donnée par les deux heuristiques $H3$, $H4$ (voir Figure 4.9).

$$BS = \min(H3, H4) = \min(13, 13) = 13 \quad (4.29)$$

La borne inférieure du problème est :

$$BI = \max(BI_1, BI_2) = \max(11, 9) = 11 \quad (4.30)$$

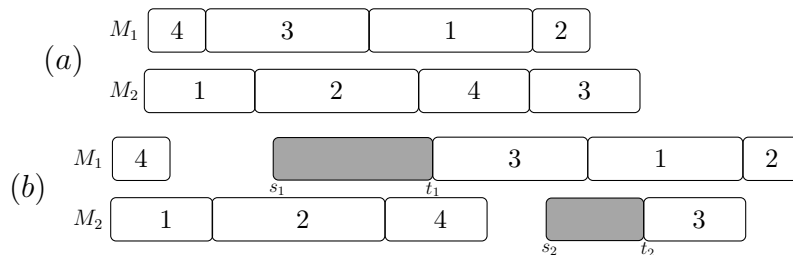
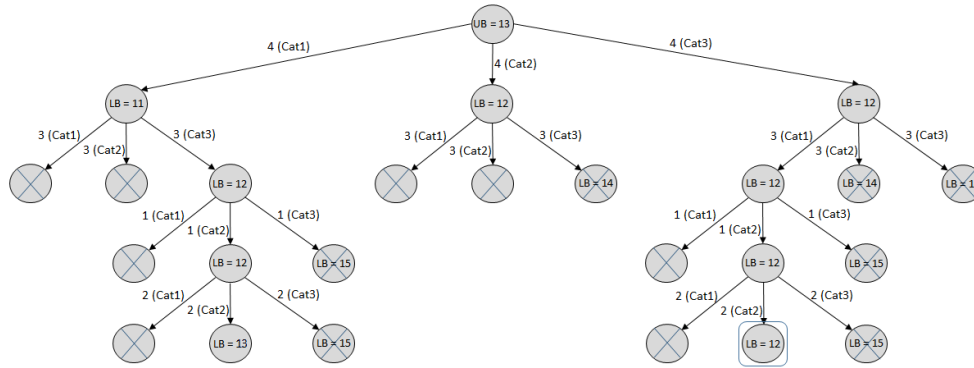


FIGURE 4.9 – (a) Solution obtenu par l'algorithme de Jackson (sans contraintes de disponibilité), (b) Solution trouvée par les heuristiques $H3$ et $H4$

Nous obtenons la valeur BS et l'ordre d'affectation des tâches pour le branchement. Dans cet exemple, l'ordre d'affectation des opérations sur les machines M_1 et M_2 est respectivement $[4, 3, 1, 2]$, $[1, 2, 4, 3]$ et BS est égale à 13.

Pour l'algorithme $PSE1$, chaque branchement génère trois noeuds, appartenant à une des trois catégories; catégorie 1 ($Cat1$) ou catégorie 2 ($Cat2$) ou catégorie 3 ($Cat3$). Nous calculons ensuite pour chaque noeud la borne inférieure correspondante, en considérant l'exploration en profondeur. La figure 4.10 montre l'arbre générée pour résoudre le problème, où (X) désigne un noeud éliminé. Le noeud qui correspond à la solution partielle $\{4(Cat1),$

$3(Cat1)$ } est éliminée car la date de fin de ces deux tâches est supérieure à la date de début de la période d'indisponibilité sur la machine M_1 . Le noeud suivant est la solution partielle $\{4(Cat1), 3(Cat2)\}$, etc. La date de fin de la solution partielle $\{4(Cat1), 3(Cat2), 1(Cat3)\}$ est supérieur à la solution en cours, donc continuer l'exploration de ce noeud n'est plus nécessaire. La solution optimale avec $C_{max} = 12$ est obtenue par la séquence $4(Cat3), 3(Cat1), 1(Cat2), 2(Cat2))$ }.

FIGURE 4.10 – L'arborescence obtenue par *PSE1*

En ce qui concerne *PSE2*, deux noeuds possibles sont obtenus, avant la période d'indisponibilité (Bk) ou après (Ak) sur chaque machine M_k , nous calculons pour chaque noeud la borne inférieure correspondante, en considérant l'exploration en profondeur. Dans la figure 4.11, nous illustrons l'arbre générée par *PSE2*. L'exclusion du noeud correspondant à la solution partielle $\{4(B1), 1(B2), 3(B1)\}$ est du à la date de fin de ces deux tâches, supérieur à la date début de l'indisponibilité période sur la machine M_1 . Le temps d'achèvement de la solution partielle $\{4(B1), 1(A2), 3(A1), 2(A2)\}$ est supérieur à la solution actuelle, donc il n'est pas nécessaire de continuer l'exploration de ce noeud. La solution optimale avec $C_{max} = 12$ est donnée par la séquence $\{4(A1), 1(B2), 3(B1), 2(B2), 1(A1), 4(A2), 2(A1), 3(B2)\}$.

La séquence obtenue par *PSE2* est la même que celle trouvée par *PSE1* (voir figure 4.12) et son makespan, optimal est $C_{max} = 12$.

Pour illustrer d'avantage l'efficacité des propriétés démontrées, nous appliquons et évaluons la même approche utilisée par un nouveau modèle de programmation linéaire à nombre entier mixte (MILP4).

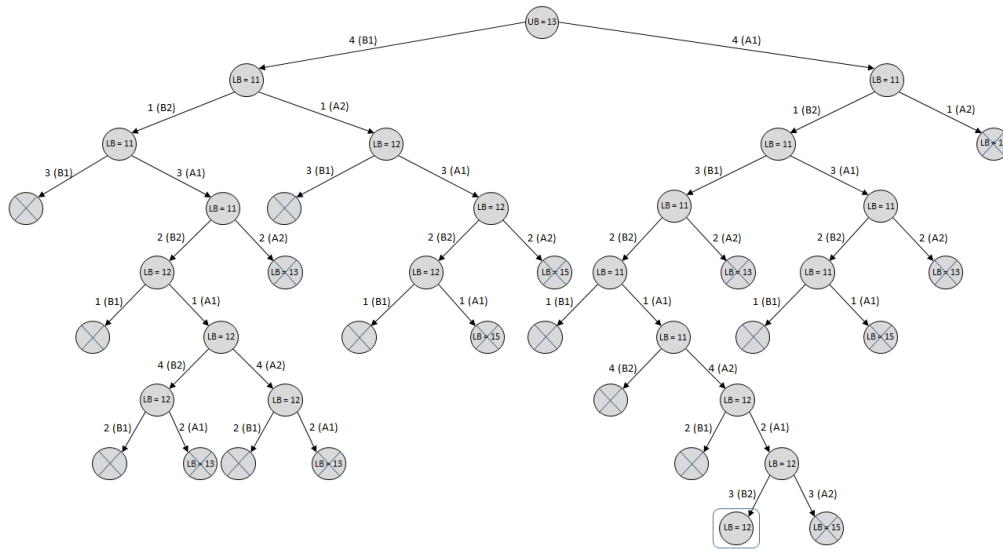
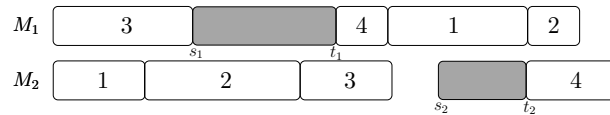

 FIGURE 4.11 – L'arborescence obtenue par *PSE2*


FIGURE 4.12 – La solution optimale obtenue

4.7 Modèle de programmation linéaire en nombres entiers mixtes (MILP4)

Nous appliquons la même approche par un nouveau modèle de programmation linéaire en nombres entiers mixtes (MILP4). L'objectif est de trouver l'ensemble des tâches à exécuter avant et après chaque période d'indisponibilité sachant la séquence de chaque ensemble de tâches qui suit la règle de Jackson.

La différence majeure entre MILP4 et MILP3 est que dans MILP4, l'ordonnancement des tâches est obtenu par l'algorithme de Jackson qui résout de façon optimale la séquence en temps polynomial. Contrairement à MILP3 qui est une modélisation du problème d'ordonnancement de type job shop avec contraintes de disponibilité où les ordonnancements de permutation ne sont pas pris en comptes. L'approche suivie dans MILP4 contient deux phases : D'abord, l'ordre de Jackson est déterminé. Ensuite, le modèle décide le placement de chaque tâche avant ou après la période d'indisponibilité sur chaque machine, en s'assurant que les tâches sont organisées en fonction de l'ordre démontré par la propriété 11.

$$\gamma_{[i][j][k]} = \begin{cases} 1 & \text{Si } J_{[i]} \text{ et } J_{[j]} \text{ appartiennent au même ensemble } (A_k \text{ ou } B_k) \text{ sur} \\ & \text{la machine } M_k \\ 0 & \text{Sinon} \end{cases}$$

$$y_{[i][k]} = \begin{cases} 1 & \text{Si } J_{[i]} \text{ commence avant la périodes d'indisponibilité sur la machine } M_k \\ 0 & \text{Sinon} \end{cases}$$

$c_{[i]k}$: la date de fin de la tâche $J_{[i]}$ sur la machine M_k .

C_{max} : Makespan.

Fonction objectif

$$\text{Min } C_{max}$$

Contraintes

$$c_{[i]\sigma_2^{[i]}} - p_{[i]\sigma_2^{[i]}} \geq c_{[i]\sigma_1^{[i]}} \quad \forall [i] \in \{1, \dots, N\} \quad (4.31)$$

$$c_{[i]k} - B \cdot (1 - y_{[i]k}) \leq s_k \quad \forall [i] \in \{1, \dots, N\}, \quad \forall k \in \{1, 2\} \quad (4.32)$$

$$c_{[i]k} - t_k \cdot (1 - y_{[i]k}) \geq p_{ik} \quad \forall [i] \in \{1, \dots, N\}, \quad \forall k \in \{1, 2\} \quad (4.33)$$

$$c_{[i]k} \geq p_{[i]k} \quad \forall [i] \in \{1, \dots, N\}, \quad \forall k \in \{1, 2\} \quad (4.34)$$

$$C_{max} \geq c_{[i]k} \quad \forall [i] \in \{1, \dots, N\}, \quad \forall k \in \{1, 2\} \quad (4.35)$$

$$c_{[j]k} - c_{[i]k} + B \cdot (1 - \gamma_{[i][j]k}) \geq p_{[j]k} \quad \forall [i], [j], [j] > [i], \quad \forall k \in \{1, 2\} \quad (4.36)$$

$$c_{[j]k'} - c_{[i]k'} + B \cdot (1 - y_{[i]k} + y_{[j]k}) \geq p_{[j]k'} \quad \forall [i], [j] \in \text{Set}_{12}, \forall k, k' \in \{1, 2\} \quad (4.37)$$

$$c_{[j]k'} - c_{[i]k'} + B \cdot (1 - y_{[i]k} + y_{[j]k}) \geq p_{[j]k'} \quad \forall [i], [j] \in \text{Set}_{21}, \quad \forall k, k' \in \{1, 2\} \quad (4.38)$$

$$1 - y_{[i]k} + y_{[j]k} \geq \gamma_{[i][j]k} \quad \forall [i], [j], [j] > [i], \quad \forall k \in \{1, 2\} \quad (4.39)$$

$$1 - y_{[j]k} + y_{[i]k} \geq \gamma_{[i][j]k} \quad \forall [i], [j], [j] > [i], \quad \forall k \in \{1, 2\} \quad (4.40)$$

$$1 - (y_{[i]k} + y_{[j]k}) \leq \gamma_{[i][j]k} \quad \forall [i], [j], [j] > [i], \quad \forall k \in \{1, 2\} \quad (4.41)$$

$$y_{[i]k} + y_{[j]k} - 1 \leq \gamma_{[i][j]k} \quad \forall [i], [j], [j] > [i], \quad \forall k \in \{1, 2\} \quad (4.42)$$

$$C_{max} \geq t_k + \sum_{i \in N} (p_{[i]k} \cdot (1 - y_{[i]k})) \quad \forall k \in \{1, 2\} \quad (4.43)$$

$$lb1 + B \cdot (1 - y_{[i]1}) \leq p_{[i]1} \quad \forall [i] \in \text{Set}_{12} \quad (4.44)$$

$$C_{max} \geq t_{11} + lb1 + \sum_{i \in \text{set}_{12}} (p_{[i]2} \cdot (1 - y_{[i]2})) \quad (4.45)$$

$$lb2 + B \cdot (1 - y_{[i]2}) \leq p_{[i]2} \quad \forall [i] \in \text{Set}_{21} \quad (4.46)$$

$$C_{max} \geq t_{21} + lb_2 + \sum_{i \in set_{21}} (p_{[i]1} \cdot (1 - y_{[i]1})) \quad (4.47)$$

$$C_{max} \geq BI_1 \quad (4.48)$$

$$C_{max} \geq BI_2 \quad (4.49)$$

$$C_{max}, c_{[i]k} \text{ entier} \quad \forall [i] \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.50)$$

$$\gamma_{[i][j]k} \in \{0, 1\}, \quad y_{[i]k} \in \{0, 1\} \quad \forall [i], [j] \in \{1, \dots, N\}, \forall k \in \{1, 2\} \quad (4.51)$$

La description des contraintes (4.31 – 4.35) est détaillée pour la formulation de MILP3. Les contraintes (4.36) assurent que les tâches ($J_{[i]}$ et $J_{[j]}$) sont dans le même sous-ensemble (avant ou après la période d'indisponibilité) sur une machine suivent l'ordre de Jackson. Les contraintes (4.37 – 4.38) assurent que l'ordre des deux tâches appartenant à Set_{12} ou Set_{21} est le même dans chacune des deux machines. Les contraintes (4.39 – 4.42) garantissent que (4.36) ne fonctionne qu'avec les tâches du même sous-ensemble. Les contraintes (4.43 - 4.47) représentent respectivement les bornes inférieures BI_3 , BI_4 , BI_5 et BI_6 , proposées dans la section précédente. Les contraintes (4.48 - 4.49) assurent que C_{max} est supérieur à la borne inférieure BI_1 et BI_2 . Dans les équations (4.44) - (4.47)), deux variables entières artificielles lb_1 et lb_2 sont utilisées pour linéariser ($\min_{i \in Set_{12}^{AA}} p_{i1}$) et ($\min_{i \in Set_{21}^{AA}} p_{i2}$) respectivement. Les contraintes (4.50) indiquent que le makespan et la date de fin d'une tâche J_i sont des nombres entiers. Les contraintes (4.51) sont des restrictions binaires pour $y_{[i]k}$ et $\gamma_{[i][j]}$.

4.8 Résultats expérimentaux

Dans cette section nous présentons les résultats numériques afin d'évaluer les performances des méthodes de résolution proposées. Les deux modèles MILP3 et MILP4 ont été résolus par le solveur IBM - Cplex interfacé avec C++ et l'algorithme PSE proposé a été implémenté sur C++. Les tests ont été exécuté sur un PC de fréquence 2.6GHz Intel(R) Core (TM) i5-4210M CPU et 8.00 GB.

4.8.1 Description des paramètres

Les paramètres concernant la période d'indisponibilité à fixer sur chaque machine sont créés et ajoutés aux données de chaque instance. La durée des périodes d'indisponibilité et leurs dates de début prennent en compte les cinq variantes du problème présentées dans le tableau 4.3.

Tableau 4.3 – Les configurations considérées

P	s_1	g_1	s_2	g_2
$P1$	$\frac{1}{2} \sum_{i \in N} p_{i1}$	$\frac{\sum_{i=1}^N p_{i1}}{N}$	s_1	g_1
$P2$	$\frac{1}{2} \sum_{i \in N} p_{i1}$	$\frac{\sum_{i=1}^N p_{i1}}{N}$	$s_1 + g_1$	g_1
$P3$	$\frac{1}{2} \sum_{i \in N} p_{i1}$	$\frac{\sum_{i=1}^N p_{i1}}{N}$	$s_1 + g_1 + \min_{J_i \in J} (p_{i1} + p_{i2}) - 1$	$\frac{\sum_{i=1}^N p_{i1}}{N}$
$P4$	$\frac{1}{2} \sum_{i \in N} p_{i1}$	$4 * \frac{\sum_{i=1}^N p_{i1}}{N}$	$s_1 + g_1$	$\frac{\sum_{i=1}^N p_{i1}}{N}$
$P5$	$\frac{1}{2} \sum_{i \in N} p_{i1}$	$\frac{\sum_{i=1}^N p_{i1}}{N}$	$s_1 + g_1$	$4 * \frac{\sum_{i=1}^N p_{i1}}{N}$

- $P1$ considère que les deux périodes d'indisponibilité commencent et finissent en même temps.
- $P2$ traite des périodes d'indisponibilité ayant la même durée et planifiées de manière séquentielle sur les deux machines. Sans perte de généralité, nous considérons que la période de disponibilité sur la machine M_2 commence juste après la fin de celle sur M_1 étant donné que le job shop à deux machines est symétrique.
- $P3$ concerne le cas où les deux périodes d'indisponibilité ne sont traitées ni simultanément ni séquentiellement. En revanche, le temps entre les deux périodes d'indisponibilité est inférieur à $\min_{J_i \in J} (p_{i1} + p_{i2})$.
- $P4$ concerne le cas de périodes d'indisponibilité traitées séquentiellement à travers les deux machines et ayant des durées différentes ($g_1 = 4 * g_2$).
- $P5$ étudie le cas des périodes d'indisponibilité traitées séquentiellement à travers les deux machines et ayant des durées différentes ($g_2 = 4 * g_1$).

Afin d'évaluer d'avantage les performances des méthodes proposées, des expériences ont été réalisées sur des instances générées aléatoirement. Ce choix est établi car les instances étudiés de la littérature peuvent ne pas évaluer parfaitement la performance des méthodes proposées. Nous générons donc 10 instances au hasard pour chaque valeur de $N = \{15, 20, 30\}$. Le temps d'exécution de chaque opération est sélectionné aléatoirement dans l'ensemble $\{10; 20; \dots; 90; 100\}$.

Les tests sont terminés lorsque l'optimum est trouvé ou lorsque la limite du temps de calcul de 3600 secondes est atteinte.

4.8.2 Analyse des résultats expérimentaux

Le tableau 4.5 illustre la performance du modèle du modèle *MILP3* appliqué sur les instances de Taillard de taille $N = \{15, 20, 30\}$. Les temps CPU nécessaires pour les résoudre ainsi que les *Gap_Cplex* sont présentés pour évaluer le modèle. Le tableau présente les résultats pour les trois premières configurations *P1*, *P2* et *P3*. Comme indiqué, *MILP3* n'est capable de résoudre de manière optimale aucune des instances lorsque le temps de calcul est de 3600 secondes. Ceci est dû au nombre de solutions potentielles qui ne sont pas nécessairement des ordonnancements de permutation. En effet, le modèle ne prend pas en compte uniquement les ordonnancements de permutation dans la résolution et ne contient également aucune borne susceptible d'accélérer la résolution. D'où le temps de calcul important pour trouver ou justifier une solution optimale.

Les tableaux 4.6 et 4.7 mettent en exergue les performances de l'algorithme *PSE1* résolvant les cinq configurations considérées. La borne inférieure *BI*, supérieure *BS*, le makespan C_{max} , le temps de calcul $CPU(s)$, les noeuds atteints en branchement et l'écart $Gap = \frac{C_{max} - BI}{BI}(\%)$ sont considérés pour illustrer la performance de l'algorithme. Lorsqu'une solution optimale est trouvée, il n'est pas nécessaire de calculer l'écart mentionné (*Gap*). Il est donc indiqué par le symbole (-). Nous remarquons que *PSE1* est capable de résoudre toutes les instances étudiées, alors que *PSE2* n'est pas capable de résoudre 9 des 10 instances de taille 30 comme illustré dans les tableaux 4.8 et 4.9. Pour les instances non résolues par *PSE2*, l'écart moyen dans les deux cas est égal à 1,60 %. La différence entre *PSE1* et *PSE2* est due à la stratégie de séparation et d'exploration suivie dans chaque schéma de branchement. Les résultats montrent également que les deux algorithmes ont la même performance quel que soit le problème étudié. Pour le reste de l'étude nous limiterons nos expérimentations aux trois premières configurations *P1*, *P2* et *P3*.

Les tableaux 4.10, 4.11 et 4.12 comparent les performances de *PSE1*, *PSE2* et *MILP4* résolvant les configurations *P1*, *P2* et *P3*. *PSE1* et *MILP4* résolvent toutes les instances alors que *PSE2* n'est pas capable de résoudre 9 de 10 instances de taille de 30 tâches. La bonne performance de *MILP4* comparé à *MILP3* est due à l'application de la propriété démontrée 11 impliquant l'optimalité de la règle de Jackson avant et après chaque période d'indisponibilité et en raison des bornes inférieures déduites de la même propriété. Cela a permis à l'ordonnancement des opérations d'être effectué par l'algorithme de Jackson en temps polynomiale, contrairement au modèle *MILP3* qui contient des contraintes d'ordonnancement et d'indisponibilité rendant ainsi la résolution plus difficile. D'un autre côté, le modèle *MILP4* fonctionne de la même manière que la procédure *PSE1*. En

effet, comme l'ordre optimal des opérations est déterminé à l'avance (selon la règle de Jackson), la décision consiste à faire le choix optimal des opérations à traiter avant ou après chaque période d'indisponibilité. Le tableau 4.4 affiche le temps de calcul moyen, minimal et maximal pour chaque instance des trois méthodes *PSE1*, *PSE2* et *MILP4* pour les trois configurations *P1*, *P2* et *P3*. Nous remarquons que *PSE1* est significativement mieux que *PSE2* et légèrement meilleur que *MILP4* pour les instances de Taillard de taille $N = \{15, 20, 30\}$.

Les résultats montrent que les bornes inférieures sont toujours atteintes et cela est dû aux temps d'exécutions des tâches proposées par les instances de Taillard et capables d'occuper les temps d'inactivité générés par les périodes d'indisponibilité. Les instances de Taillard ne permettent donc pas une évaluation parfaite de la performance des méthodes proposées. D'où la génération de nouvelles instances afin d'éviter les cas où le makespan optimal est égal à la borne inférieure. Pour ce faire, nous générons aléatoirement des instances dont les temps d'exécutions des tâches sur les deux machines appartient à l'ensemble $\{10, 20, 30, \dots, 100\}$.

Le tableau 4.13 illustre la performance des méthodes *PSE1*, *PSE2* et *MILP4* appliquées sur les instances générées aléatoirement pour la configuration *P3*. *PSE1* est plus rapide que *PSE2* en ce qui concerne la résolution des instances de 15 tâches. Pour les instances de taille 20 et 30, *PSE1* trouve les mêmes résultats pour les instances résolus à l'optimum par *MILP4*. Nous notons que les instances résolues optimalement avec un petit nombre de noeuds, correspondent aux instances dont la solution optimale égale à la borne inférieure (*BI*). Pour les instances générées, le *MILP4* fonctionne mieux que les procédures *PSE1* et *PSE2*. Par ailleurs, l'écart est calculé comme $Gap = \frac{C_{max} - C_{max}(MILP4)}{C_{max}(MILP4)}(\%)$ si $C_{max}(MILP4)$ est optimal, sinon il est calculé en fonction de la limite inférieure ($Gap = \frac{C_{max} - BI}{BI}(\%)$). En plus de l'ordre optimal prouvé dans la propriété 11 et les bornes inférieures insérées, la bonne performance du modèle *MILP4*, comparé à *PSE1* est due aux coupes d'addition (Cliques cuts, Cover cuts, Implied bound cuts, Mixed integer rounding cuts,...) intégrées dans le logiciel commercial Cplex. Néanmoins, l'utilisation des solveurs commerciaux n'est pas toujours autorisée par les décideurs d'une entreprise ou d'un atelier de production. D'où l'importance de la procédure par séparation et évaluation qui dépasse également le modèle *MILP4* en performance sur les instances de Taillard, plus réaliste (comme mentionné dans Taillard (1993)) que celles générées aléatoirement.

Pour le même problème et concernant la comparaison entre *PSE1*, *PSE2* et *PSE_{Agg}*, les résultats présentés dans les tableaux 4.14, 4.15 et 4.16 correspondent à la résolution

des instances de Taillard alors que 4.17 représente les résultats obtenus pour les instances générées aléatoirement. Tous les résultats indiquent que $PSE1$ et $PSE2$ sont meilleurs que PSE_{Agg} . Ce dernier n'est capable de résoudre aucune instance de taille $N = \{15, 20, 30\}$. La limite de PSE_{Agg} est principalement due au nombre de tâches et de périodes d'indisponibilité par machine (plus de combinaisons). En effet, en plus des bornes inférieures utilisées dans l'évaluation ne pouvant pas éliminer les branches en raison du grand nombre d'opérations, la performance de PSE_{Agg} est également due à l'utilisation de la sélection immédiate impliquant des tâches de maintenance. Aggoune (2002) indique que PSE_{Agg} fonctionne bien avec plusieurs indisponibilités sur chaque machine. Ainsi, plus le nombre d'indisponibilités sur chaque machine est grand, plus PSE_{Agg} élimine les branches dans l'arbre. Une autre raison importante est que l'ordonnancement de permutation n'est pas pris en compte dans la résolution par Aggoune (2002). Par conséquent le nombre de combinaisons augmentera aussi bien que les branches qui ne peuvent être rejetées ni par des bornes inférieures ni par la sélection immédiate.

Tableau 4.4 – Temps d'exécution moyen (en secondes) of PSE1, PSE2 et MILP4 (Instances de Taillard)

P	N	$PSE1$			$PSE2$			$MILP4$		
		Min	$Mean$	Max	Min	$Mean$	Max	Min	$Mean$	Max
$P1$	15	0.05	0.41	2.66	1.34	48.88	290.92	0.59	0.97	2.12
	20	0.02	0.54	3.13	39.25	660.99	3600	1.15	1.76	3.37
	30	0.03	1.21	8.36	927.69	3332.77	3600	1.06	4.75	7.43
$P2$	15	0.05	0.48	2.58	2.03	55.22	33.62	0.66	0.93	1.62
	20	0.02	0.54	0.95	35.72	5853.97	3600	0.69	1.57	2.63
	30	0.03	2.45	14.04	1298.42	3369.84	3600	1.99	5.09	9.37
$P3$	15	0.05	0.42	1.58	0.78	33.07	63.71	0.59	0.82	1.06
	20	0.02	0.24	0.50	19.11	439.07	3600	1.42	1.80	2.39
	30	0.03	7.36	39.49	1021.01	439.16	3600	3.32	5.21	10.01

Tableau 4.5 – Résultats obtenus par MILP3 (instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$)

N	BI	$P1$			$P2$			$P3$		
		C_{max}	$CPU(s)$	$GAP_Cplex(\%)$	C_{max}	$CPU(s)$	$GAP_Cplex(\%)$	C_{max}	$CPU(s)$	$GAP_Cplex(\%)$
15	944	944	3600	22.56	944	3600	20.02	944	3600	24.64
	729	729	3600	14.27	729	3600	14.27	729	3600	18.51
	832	832	3600	29.09	832	3600	23.19	832	3600	17.55
	897	897	3600	17.95	897	3600	16.61	897	3600	14.81
	956	956	3600	25.41	956	3600	26.36	956	3600	27.34
	875	875	3600	18.92	875	3600	15.77	875	3600	17.37
	964	964	3600	25.52	964	3600	16.28	964	3600	18.98
	869	869	3600	18.99	869	3600	19.56	869	3600	24.64
	987	987	3600	23.01	987	3600	26.34	987	3600	28.80
	777	777	3600	12.87	777	3600	20.34	777	3600	15.95
20	1132	1132	3600	43.46	1132	3600	44.61	1132	3600	46.85
	990	990	3600	32.42	990	3600	35.05	990	3600	36.27
	1065	1065	3600	33.33	1065	3600	35.12	1065	3600	44.41
	1089	1089	3600	52.67	1089	3600	45.50	1089	3600	37.37
	977	977	3600	34.19	977	3600	41.15	977	3600	31.56
	1170	1170	3600	42.22	1170	3600	37.78	1170	3600	39.25
	1167	1167	3600	40.61	1167	3600	54.76	1167	3600	50.57
	1114	1114	3600	38.90	1114	3600	41.21	1114	3600	36.26
	1103	1103	3600	38.53	1103	3600	40.07	1103	3600	32.33
	1243	1243	3600	35.80	1243	3600	38.94	1243	3600	37.59
30	1723	1723	3600	76.66	1723	3600	74.17	1723	3600	76.83
	1699	1699	3600	75.98	1699	3600	75.81	1699	3600	75.84
	1740	1740	3600	74.08	1740	3600	76.04	1740	3600	75.14
	1787	1787	3600	78.51	1787	3600	76.35	1787	3600	76.41
	1658	1658	3600	74.72	1658	3600	74.76	1658	3600	72.61
	1780	1780	3600	78.99	1780	3600	78.43	1780	3600	76.18
	1830	1830	3600	78.10	1830	3600	74.60	1830	3600	74.98
	1486	1486	3600	70.45	1486	3600	71.40	1486	3600	73.38
	1642	1642	3600	77.34	1642	3600	76.82	1642	3600	72.58
	1584	1584	3600	74.14	1584	3600	70.99	1584	3600	74.98

Tableau 4.6 – Résultats obtenus par PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$)

N	P1						P2						P3					
	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	1021	944	0.06	223	-	944	970	944	0.18	627	-	944	1003	944	0.05	166	-
	729	768	729	0.05	215	-	729	768	729	0.06	221	-	729	768	729	0.05	221	-
	832	867	832	0.22	895	-	832	867	832	0.23	883	-	832	867	832	0.22	857	-
	897	969	897	0.19	701	-	897	920	897	0.61	2162	-	897	902	897	0.36	1297	-
	956	990	956	0.06	241	-	956	976	956	0.24	816	-	956	984	956	0.15	514	-
	875	946	875	0.32	1330	-	875	946	875	0.05	193	-	875	946	875	0.09	364	-
	964	1037	964	0.07	296	-	964	1037	964	0.09	334	-	964	1037	964	0.08	328	-
	869	905	869	0.17	656	-	869	905	869	0.06	237	-	869	905	869	0.90	3448	-
	987	1031	987	2.66	10541	-	987	1004	987	2.58	9033	-	987	1004	987	1.59	6442	-
	777	780	777	0.26	950	-	777	780	777	0.75	1971	-	777	798	777	0.77	2713	-
20	1132	1172	1132	3.13	8594	-	1132	1222	1132	0.64	1400	-	1132	1154	1132	0.55	1480	-
	990	1020	990	0.18	563	-	990	1020	990	0.10	266	-	990	1020	990	0.02	64	-
	1065	1087	1065	0.02	64	-	1065	1087	1065	0.03	68	-	1065	1087	1065	0.09	214	-
	1089	1135	1089	0.34	1210	-	1089	1135	1089	0.12	408	-	1089	1135	1089	0.04	154	-
	977	1001	977	0.19	502	-	977	1044	977	0.23	627	-	977	979	977	0.05	130	-
	1170	1235	1170	0.36	1112	-	1170	1181	1170	0.92	2006	-	1170	1215	1170	0.981	2408	-
	1167	1225	1167	0.16	467	-	1167	1225	1167	0.23	487	-	1167	1225	1167	0.14	442	-
	1114	1180	1114	0.10	320	-	1114	1180	1114	0.06	151	-	1114	1180	1114	0.05	167	-
	1103	1173	1103	0.58	1817	-	1103	1117	1103	0.72	1952	-	1103	1148	1103	0.20	652	-
	1243	1284	1243	0.34	1129	-	1243	1268	1243	0.95	2881	-	1243	1268	1243	0.41	1303	-
30	1723	1742	1723	0.06	153	-	1723	1742	1723	0.06	153	-	1723	1742	1723	0.06	153	-
	1699	1719	1699	1.01	2418	-	1699	1725	1699	0.16	388	-	1699	1747	1699	39.50	87759	-
	1740	1788	1740	0.02	56	-	1740	1778	1740	2.04	4143	-	1740	1742	1740	0.22	473	-
	1787	1790	1787	0.90	1856	-	1787	1839	1787	0.54	1169	-	1787	1814	1787	0.55	1184	-
	1658	1674	1658	0.24	635	-	1658	1727	1658	6.06	15126	-	1658	1670	1658	0.03	76	-
	1780	1816	1780	0.18	426	-	1780	1806	1780	0.31	704	-	1780	1791	1780	12.30	27050	-
	1830	1859	1830	1.18	3042	-	1830	1859	1830	1.17	3041	-	1830	1859	1830	1.18	3034	-
	1486	1507	1486	0.10	266	-	1486	1510	1486	0.10	283	-	1486	1535	1486	0.14	385	-
	1642	1734	1642	8.36	18600	-	1642	1675	1642	14.04	30074	-	1642	1691	1642	18.58	42145	-
	1584	1608	1584	0.07	172	-	1584	1608	1584	0.05	125	-	1584	1608	1584	0.05	129	-

Tableau 4.7 – Résultats obtenus par PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$, $P4$ et $P5$)

N	P2					P4					P5							
	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	970	944	0.18	627	-	944	947	944	0.02	68	-	1070	1096	1070	0.18	627	-
	729	768	729	0.06	221	-	864	903	864	0.05	221	-	781	827	781	0.05	181	-
	832	867	832	0.23	883	-	988	1023	988	0.23	883	-	840	909	840	1.44	5763	-
	897	920	897	0.61	2162	-	897	986	897	0.43	1680	-	1032	1055	1032	0.66	2259	-
	956	976	956	0.24	816	-	1026	1042	1026	0.01	36	-	1118	1138	1118	0.45	1542	-
	875	946	875	0.05	193	-	1037	1108	1037	0.05	193	-	875	946	875	0.15	633	-
	964	1037	964	0.09	334	-	1144	1217	1144	0.09	361	-	1074	1123	1074	2.29	7431	-
	869	905	869	0.06	237	-	1020	1064	1020	0.01	36	-	1028	1032	1028	0.21	717	-
	987	1004	987	2.58	9033	-	1170	1187	1170	1.62	6482	-	1169	1183	1169	2.10	6515	-
	777	780	777	0.75	1971	-	844	876	844	0.02	62	-	909	912	909	0.99	3207	-
20	1132	1222	1132	0.64	1400	-	1218	1269	1218	23.45	74035	-	1282	1372	1282	0.36	902	-
	990	1020	990	0.10	266	-	1131	1161	1131	0.08	266	-	1110	1138	1110	0.77	2179	-
	1065	1087	1065	0.03	68	-	1215	1237	1215	0.02	68	-	1136	1144	1136	0.01	20	-
	1089	1135	1089	0.12	408	-	1242	1288	1242	0.12	408	-	1139	1141	1139	1.30	4046	-
	977	1044	977	0.23	627	-	1039	1055	1039	5.50	18301	-	1106	1173	1106	1.29	3231	-
	1170	1181	1170	0.92	2006	-	1279	1288	1279	0.01	42	-	1329	1340	1329	2.60	7204	-
	1167	1225	1167	0.23	487	-	1332	1390	1332	0.15	487	-	1173	1225	1173	0.07	246	-
	1114	1180	1114	0.06	151	-	1273	1339	1273	0.05	151	-	1177	1180	1177	9.42	27471	-
	1103	1117	1103	0.72	1952	-	1240	1270	1240	0.13	454	-	1256	1267	1256	0.67	1804	-
	1243	1268	1243	0.95	2881	-	1376	1439	1376	0.06	201	-	1414	1419	1414	2.09	5996	-
30	1723	1742	1723	0.06	153	-	1888	1907	1888	0.06	153	-	1723	1742	1723	0.05	141	-
	1699	1725	1699	0.16	388	-	1805	1813	1805	0.05	148	-	1858	1884	1858	5.69	12288	-
	1740	1778	1740	2.04	4143	-	1789	1812	1789	23.99	59461	-	1896	1934	1896	22.56	42727	-
	1787	1839	1787	0.54	1169	-	1787	1798	1787	3.72	8429	-	1934	1986	1934	15.70	29728	-
	1658	1727	1658	6.06	15126	-	1808	1812	1808	0.06	70	-	1817	1886	1817	37.96	82670	-
	1780	1806	1780	0.31	704	-	1816	1888	1816	0.50	1328	-	1939	1965	1939	0.25	509	-
	1830	1859	1830	1.17	3041	-	2007	2036	2007	1.18	3041	-	1830	1859	1830	0.55	1439	-
	1486	1510	1486	0.10	283	-	1627	1648	1627	0.10	267	-	1621	1651	1621	28.31	61929	-
	1642	1675	1642	14.04	30074	-	1642	1692	1642	6.43	16150	-	1786	1819	1786	38.74	80478	-
	1584	1608	1584	0.05	125	-	1737	1761	1737	0.05	125	-	1681	1761	1681	0.48	1074	-

Tableau 4.8 – Résultats obtenus par PSE2 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$, $P2$ et $P3$)

N	P1						P2						P3					
	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	1021	944	4.80	24662	-	944	970	944	29.61	107827	-	944	1003	944	12.81	67821	-
	729	768	729	5.69	38095	-	729	768	729	6.83	38563	-	729	768	729	11.11	71640	-
	832	867	832	54.58	348565	-	832	867	832	53.89	259173	-	832	867	832	32.54	203917	-
	897	969	897	13.46	73590	-	897	920	897	14.28	51244	-	897	902	897	53.69	293441	-
	956	990	956	49.40	298180	-	956	976	956	16.08	62971	-	956	984	956	17.02	103338	-
	875	946	875	1.34	9166	-	875	946	875	2.06	8890	-	875	946	875	0.78	5214	-
	964	1037	964	40.93	246925	-	964	1037	964	39.52	189779	-	964	1037	964	32.48	199808	-
	869	905	869	18.08	114246	-	869	905	869	20.28	90447	-	869	905	869	29.21	190625	-
	987	1031	987	280.92	1819804	-	987	1004	987	334.63	1456064	-	987	1004	987	57.13	353362	-
	777	780	777	9.53	57410	-	777	780	777	34.99	141109	-	777	798	777	63.72	366018	-
20	1132	1172	1137	3600	16910076	-	1132	1222	1151	3600	12547549	-	1132	1154	1132	2743.33	13292983	-
	990	1020	990	66.55	342070	-	990	1020	990	54.20	187049	-	990	1020	990	19.11	99058	-
	1065	1087	1065	39.25	210082	-	1065	1087	1065	35.72	133328	-	1065	1087	1065	44.51	231910	-
	1089	1135	1089	47.58	250936	-	1089	1135	1089	63.91	251737	-	1089	1135	1089	33.13	171990	-
	977	1001	977	2217.06	10483912	-	977	1044	979	3600	12445986	-	977	979	977	740.75	3539581	-
	1170	1235	1170	201.21	997986	-	1170	1181	1170	468.48	1685732	-	1170	1215	1170	104.18	527612	-
	1167	1225	1167	124.04	614995	-	1167	1225	1167	111.17	404393	-	1167	1225	1167	94.88	464300	-
	1114	1180	1114	56.91	280488	-	1114	1180	1114	52.72	194088	-	1114	1180	1114	40.97	202672	-
	1103	1173	1103	163.99	784314	-	1103	1117	1103	384.79	1460018	-	1103	1148	1103	431.28	2126110	-
	1243	1284	1243	93.37	467493	-	1243	1268	1243	168.75	645298	-	1243	1268	1243	139.47	702759	-
30	1723	1742	1724	3600	13311509	0.06	1723	1742	1742	3600	9765414	1.10	1723	1742	1742	3600	12386792	1.10
	1699	1719	1719	3600	13154796	1.18	1699	1725	1725	3600	9888178	1.53	1699	1747	1747	3600	13008288	2.83
	1740	1788	1788	3600	12497596	2.76	1740	1778	1778	3600	9273171	2.18	1740	1742	1742	3600	12343797	0.11
	1787	1790	1790	3600	12236847	0.17	1787	1839	1839	3600	9415562	2.91	1787	1814	1814	3600	9775274	1.51
	1658	1674	1660	3600	13189616	0.12	1658	1727	1718	3600	10121782	3.62	1658	1670	1670	3600	9812781	0.72
	1780	1816	1816	3600	13095295	2.02	1780	1806	1806	3600	9868754	1.46	1780	1791	1791	3600	10075292	0.62
	1830	1859	1859	3600	13171671	1.58	1830	1859	1859	3600	9880187	1.58	1830	1859	1859	3600	9768468	1.58
	1486	1507	1500	3600	14043145	0.94	1486	1510	1510	3600	10703254	1.62	1486	1535	1535	3600	10400988	3.30
	1642	1734	1734	3600	13158205	5.60	1642	1675	1675	3600	9780592	2.01	1642	1691	1691	3600	12940763	2.98
	1584	1608	1584	927.69	3630638	-	1584	1608	1584	1298.42	3762814	-	1584	1608	1584	1021.01	3945625	-

Tableau 4.9 – Résultats obtenus par PSE1 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$, $P4$ et $P5$)

N	P2						P4						P5					
	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BI	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	970	944	29.61	107827	-	944	947	944	77.46	386651	-	1070	1096	1070	68.32	282545	-
	729	768	729	6.83	38563	-	864	903	864	8.05	39405	-	781	827	781	18.78	96208	-
	832	867	832	53.89	259173	-	988	1023	988	6104	270106	-	840	909	480	1.61	9874	-
	897	920	897	14.28	51244	-	897	986	897	27.48	104856	-	1032	1055	1032	25.35	104724	-
	956	976	956	16.08	62971	-	1026	1042	1026	5.39	27065	-	1118	1138	1118	29.87	130295	-
	875	946	875	2.06	8890	-	1037	1108	1037	2.12	8890	-	875	946	875	4.09	17665	-
	964	1037	964	39.51	189779	-	1144	1217	1144	43.12	187626	-	1074	1123	1074	10.65	55463	-
	869	905	869	20.28	90447	-	1020	1064	1020	3.68	21137	-	1028	1032	1028	9.64	46021	-
	987	1004	987	334.63	1456064	-	1170	1187	1170	91.88	416325	-	1169	1183	1169	122.76	545507	-
	777	780	777	34.99	141109	-	844	876	844	4.50	24552	-	909	912	909	99.80	443879	-
20	1132	1222	1151	3600	12547549	-	1218	1269	1218	1379.77	5041366	-	1282	1372	1289	3600	12955637	-
	990	1020	990	54.20	187049	-	1131	1161	1131	44.18	185983	-	1110	1138	1110	190.57	769456	-
	1065	1087	1065	35.72	133328	-	1215	1237	1215	31.74	133328	-	1136	1144	1136	119.04	447699	-
	1089	1135	1089	63.91	251737	-	1242	1288	1242	62.71	251737	-	1139	1141	1139	32.75	123309	-
	977	1044	979	3600	12445986	-	1039	1055	1039	513.62	1911198	-	1106	1173	1108	3600	12565745	-
	1170	1181	1170	468.48	1685732	-	1279	1288	1279	59.08	235875	-	1329	1340	1329	579.87	2063925	-
	1167	1225	1167	111.17	404393	-	1332	1390	1332	100.57	404393	-	1173	1225	1173	55.47	192010	-
	1114	1180	1114	52.72	194088	-	1273	1339	1273	54.68	194088	-	1177	1180	1177	51.90	196174	-
	1103	1117	1103	384.79	1460018	-	1240	1270	1240	139.06	541695	-	1256	1267	1256	1667.79	6364330	-
	1243	1268	1243	168.75	645298	-	1376	1439	1376	113.48	474990	-	1414	1419	1414	250.78	949984	-
30	1723	1742	1742	3600	9765414	1.10	1888	1907	1907	3600	10087266	1.01	1723	1742	1734	3600	9924365	0.64
	1699	1725	1725	3600	9888178	1.53	1805	1813	1813	3600	9919711	0.44	1858	1884	1884	3600	10165734	1.40
	1740	1778	1778	3600	9273171	2.18	1789	1812	1812	3600	9417979	1.29	1896	1934	1899	3600	9314539	0.16
	1787	1839	1839	3600	9415562	2.91	1787	1798	1798	3600	9674579	0.62	1934	1986	1969	3600	9416619	1.81
	1658	1727	1718	3600	10121782	3.62	1808	1812	1812	3600	9758787	0.22	1817	1886	1842	3600	9651687	1.38
	1780	1806	1806	3600	9868754	1.46	1816	1888	1888	3600	9922909	3.96	1939	1965	1954	3600	10406697	0.77
	1830	1859	1859	3600	9880187	1.58	2007	2036	2036	3600	9343585	1.44	1830	1859	1834	3600	10059202	0.22
	1486	1510	1510	3600	10703254	1.62	1627	1648	1648	3600	10621584	1.29	1621	1651	1644	3600	13740250	1.42
	1642	1675	1675	3600	9780592	2.01	1642	1692	1692	3600	10054685	3.05	1786	1819	1791	3600	13118894	0.28
	1584	1608	1584	1298.42	3762814	-	1737	1761	1737	1332.33	3762853	-	1681	1761	1681	1332.33	5993745	-

Tableau 4.10 – Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P1$)

N	BI	BS	$PSE1$				$PSE2$				$MILP4$			
			C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	Gap_Cplex	
15	944	1021	944	0.06	223	–	944	4.80	24662	–	944	0.47	0	
	729	768	729	0.05	215	–	729	5.69	38095	–	729	0.63	0	
	832	867	832	0.22	895	–	832	54.58	348565	–	832	0.45	0	
	897	969	897	0.19	701	–	897	13.46	73590	–	897	0.35	0	
	956	990	956	0.06	241	–	956	49.40	298180	–	956	0.87	0	
	875	946	875	0.32	1330	–	875	1.34	9166	–	875	0.40	0	
	964	1037	964	0.07	296	–	964	40.93	246925	–	964	0.49	0	
	869	905	869	0.17	656	–	869	18.08	114246	–	869	0.61	0	
	987	1031	987	2.66	10541	–	987	280.92	1819804	–	987	1.03	0	
	777	780	777	0.26	950	–	777	9.53	57410	–	777	0.48	0	
20	1132	1172	1132	3.13	8594	–	1137	3600	16910076	–	1132	2.08	0	
	990	1020	990	0.18	563	–	990	66.55	342070	–	990	1.09	0	
	1065	1087	1065	0.02	64	–	1065	39.25	210082	–	1065	1.40	0	
	1089	1135	1089	0.34	1210	–	1089	47.58	250936	–	1089	1.01	0	
	977	1001	977	0.19	502	–	977	2217.06	10483912	–	977	0.95	0	
	1170	1235	1170	0.36	1112	–	1170	201.21	997986	–	1170	2.32	0	
	1167	1225	1167	0.16	467	–	1167	124.04	614995	–	1167	2.44	0	
	1114	1180	1114	0.10	320	–	1114	56.91	280488	–	1114	2.17	0	
	1103	1173	1103	0.58	1817	–	1103	163.99	784314	–	1103	1.81	0	
	1243	1284	1243	0.34	1129	–	1243	93.37	467493	–	1243	1.95	0	
30	1723	1742	1723	0.06	153	–	1724	3600	13311509	0.06	1723	7.70	0	
	1699	1719	1699	1.01	2418	–	1719	3600	13154796	1.18	1699	9.46	0	
	1740	1788	1740	0.02	56	–	1788	3600	12497596	2.76	1739	6.82	0	
	1787	1790	1787	0.90	1856	–	1790	3600	12236847	0.17	1787	3.31	0	
	1658	1674	1658	0.24	635	–	1660	3600	13189616	0.12	1658	7.79	0	
	1780	1816	1780	0.18	426	–	1816	3600	13095295	2.02	1780	3.04	0	
	1830	1859	1830	1.18	3042	–	1859	3600	13171671	1.58	1830	5.62	0	
	1486	1507	1486	0.10	266	–	1500	3600	14043145	0.94	1486	6.63	0	
	1642	1734	1642	8.36	18600	–	1734	3600	13158205	5.60	1642	3.36	0	
	1584	1608	1584	0.07	172	–	1584	927.69	3630638	–	1584	5.90	0	

Tableau 4.11 – Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P2$)

N	BI	BS	$PSE1$				$PSE2$				$MILP4$			
			C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	Gap	$Cplex$
15	944	970	944	0.18	627	–	944	29.61	107827	–	944	0.37	0	0
	729	768	729	0.06	221	–	729	6.83	38563	–	729	0.59	0	0
	832	867	832	0.23	883	–	832	53.89	259173	–	832	0.51	0	0
	897	920	897	0.61	2162	–	897	14.28	51244	–	897	0.24	0	0
	956	976	956	0.24	816	–	956	16.08	62971	–	956	0.80	0	0
	875	946	875	0.05	193	–	875	2.06	8890	–	875	0.76	0	0
	964	1037	964	0.09	334	–	964	39.52	189779	–	964	0.39	0	0
	869	905	869	0.06	237	–	869	20.28	90447	–	869	0.53	0	0
	987	1004	987	2.58	9033	–	987	334.63	1456064	–	987	3.64	0	0
	777	780	777	0.75	1971	–	777	34.99	141109	–	777	0.77	0	0
20	1132	1222	1132	0.64	1400	–	1151	3600	12547549	–	1132	0.82	0	0
	990	1020	990	0.10	266	–	990	54.20	187049	–	990	1.01	0	0
	1065	1087	1065	0.03	68	–	1065	35.72	133328	–	1065	1.41	0	0
	1089	1135	1089	0.12	408	–	1089	63.91	251737	–	1089	0.78	0	0
	977	1044	977	0.23	627	–	979	3600	12445986	–	977	1.52	0	0
	1170	1181	1170	0.92	2006	–	1170	468.48	1685732	–	1170	1.01	0	0
	1167	1225	1167	0.23	487	–	1167	111.17	404393	–	1167	1.48	0	0
	1114	1180	1114	0.06	151	–	1114	52.72	194088	–	1114	0.71	0	0
	1103	1117	1103	0.72	1952	–	1103	384.79	1460018	–	1103	0.75	0	0
	1243	1268	1243	0.95	2881	–	1243	168.75	645298	–	1243	1.14	0	0
30	1723	1742	1723	0.06	153	–	1742	3600	9765414	1.10	1723	8.83	0	0
	1699	1725	1699	0.16	388	–	1725	3600	9888178	1.53	1699	5.25	0	0
	1740	1778	1740	2.04	4143	–	1778	3600	9273171	2.18	1740	8.47	0	0
	1787	1839	1787	0.54	1169	–	1839	3600	9415562	2.91	1787	4.41	0	0
	1658	1727	1658	6.06	15126	–	1718	3600	10121782	3.62	1658	5.21	0	0
	1780	1806	1780	0.31	704	–	1806	3600	9868754	1.46	1780	12.01	0	0
	1830	1859	1830	1.17	3041	–	1859	3600	9880187	1.58	1830	6.90	0	0
	1486	1510	1486	0.10	283	–	1510	3600	10703254	1.62	1486	7.84	0	0
	1642	1675	1642	14.04	30074	–	1675	3600	9780592	2.01	1642	6.03	0	0
	1584	1608	1584	0.05	125	–	1584	1298.42	3762814	–	1584	3.29	0	0

Tableau 4.12 – Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances de Taillard $N = \{15, 20, 30\}$, $P3$)

N	BI	BS	$PSE1$				$PSE2$				$MILP4$			
			C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	$Nodes$	$Gap(\%)$	C_{max}	$CPU(s)$	Gap	$Cplex$
15	944	1003	944	0.05	166	–	944	12.81	67821	–	944	0.59	0	0
	729	768	729	0.05	221	–	729	11.11	71640	–	729	0.66	0	0
	832	867	832	0.22	857	–	832	32.54	203917	–	832	0.76	0	0
	897	902	897	0.36	1297	–	897	53.69	293441	–	897	0.66	0	0
	956	984	956	0.15	514	–	956	17.02	103338	–	956	0.75	0	0
	875	946	875	0.09	364	–	875	0.78	5214	–	875	1.05	0	0
	964	1037	964	0.08	328	–	964	32.48	199808	–	964	0.92	0	0
	869	905	869	0.90	3448	–	869	29.21	190625	–	869	1.01	0	0
	987	1004	987	1.59	6442	–	987	57.13	353362	–	987	0.96	0	0
	777	798	777	0.77	2713	–	777	63.72	366018	–	777	0.85	0	0
20	1132	1154	1132	0.55	1480	–	1132	2743.33	13292983	–	1132	1.61	0	0
	990	1020	990	0.02	64	–	990	19.11	99058	–	990	1.56	0	0
	1065	1087	1065	0.09	214	–	1065	44.51	231910	–	1065	1.73	0	0
	1089	1135	1089	0.04	154	–	1089	33.13	171990	–	1089	2.11	0	0
	977	979	977	0.05	130	–	977	740.75	3539581	–	977	1.41	0	0
	1170	1215	1170	0.981	2408	–	1170	104.18	527612	–	1170	1.47	0	0
	1167	1225	1167	0.14	442	–	1167	94.88	464300	–	1167	1.49	0	0
	1114	1180	1114	0.05	167	–	1114	40.97	202672	–	1114	1.94	0	0
	1103	1148	1103	0.20	652	–	1103	431.28	2126110	–	1103	2.40	0	0
	1243	1268	1243	0.41	1303	–	1243	139.47	702759	–	1243	2.29	0	0
30	1723	1742	1723	0.06	153	–	1742	3600	12386792	1.10	1723	3.52	0	0
	1699	1747	1699	39.50	87759	–	1747	3600	13008288	2.83	1699	3.32	0	0
	1740	1742	1740	0.22	473	–	1742	3600	12343797	0.11	1740	5.57	0	0
	1787	1814	1787	0.55	1184	–	1814	3600	9775274	1.51	1787	6.47	0	0
	1658	1670	1658	0.03	76	–	1670	3600	9812781	0.72	1658	3.40	0	0
	1780	1791	1780	12.30	27050	–	1791	3600	10075292	0.62	1780	5.69	0	0
	1830	1859	1830	1.18	3034	–	1859	3600	9708468	1.58	1830	4.39	0	0
	1486	1535	1486	0.14	385	–	1535	3600	10400988	3.30	1486	10.01	0	0
	1642	1691	1642	18.58	42145	–	1691	3600	12940763	2.98	1642	4.17	0	0
	1584	1608	1584	0.05	129	–	1584	1021.01	3945625	–	1584	5.60	0	0

Tableau 4.13 – Comparaison des résultats obtenus par PSE1, PSE2 et MILP4 (Instances aléatoires, $P3$)

N	BI	PSE1					PSE2					MILP4				
		BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)	C _{max}	CPU(s)	Gap_Cplex		
15	1045	1083	1053	136.42	580784	–	1083	1053	1891.35	8989991	–	1053	1.36	0		
	853	907	857	102.93	441332	–	907	857	2088.08	10108864	–	857	0.68	0		
	821	871	821	0.11	459	–	871	821	7.46	31182	–	821	0.22	0		
	992	1056	996	134.23	564962	–	1056	996	875.63	4040355	–	996	1.38	0		
	906	958	913	65.17	275737	–	958	913	1770.71	8437680	–	913	1.39	0		
	874	920	877	239.76	762757	–	920	877	2022.17	9613372	–	877	1.84	0		
	1120	1210	1120	0.02	47	–	1210	1120	5.01	26871	–	1120	0.32	0		
	970	1005	975	193.92	554631	–	1005	975	1985.19	9474141	–	975	1.19	0		
	960	970	960	0.01	25	–	970	960	0.48	2006	–	960	0.21	0		
	821	831	821	0.01	15	–	831	821	6.65	27255	–	821	0.47	0		
20	1291	1306	1296	3600	7699181	0.00	1306	1296	3600	13823632	0.00	1296	15.24	0		
	1081	1091	1081	0.05	99	–	1091	1081	29.69	122301	–	1081	0.46	0		
	1218	1279	1219	3600	7767143	0.00	1279	1219	3600	13785815	0.00	1219	20.05	0		
	1113	1117	1117	3600	7884961	0.00	1117	1117	3600	18132674	0.00	1117	28.85	0		
	1312	1366	1316	3600	8327689	0.00	1366	1316	3600	18187703	0.00	1316	9.18	0		
	1249	1305	1255	3600	8103980	0.00	1305	1255	3600	19019267	0.00	1255	61.88	0		
	1302	1351	1311	3600	8096330	0.00	1351	1311	3600	18951258	0.00	1311	25.74	0		
	1207	1214	1214	3600	8410093	0.00	1214	1214	3600	18953215	0.00	1214	24.77	0		
	1302	1311	1311	3600	8425768	0.00	1311	1311	3600	18679289	0.00	1311	21.05	0		
	1354	1407	1357	3600	8388219	0.00	1407	1357	3600	19097015	0.00	1357	32.84	0		
30	1705	1733	1713	3600	6058946	0.47	1733	1733	3600	13733696	1.64	1713	3600	0.47		
	1808	1814	1814	3600	6074555	0.33	1814	1814	3600	13853968	0.33	1814	3600	0.33		
	1922	1971	1931	3600	6211670	0.47	1971	1931	3600	14658592	0.47	1931	3600	0.47		
	2025	2093	2033	3600	6240605	0.39	2093	2033	3600	15083229	0.40	2033	3600	0.39		
	1922	1941	1931	3600	6435738	0.47	1941	1931	3600	14335396	0.47	1931	3600	0.47		
	1405	1463	1413	3600	6692669	0.00	1463	1463	3600	14038068	4.13	1413	1705.08	0.00		
	1880	1950	1880	0.05	83	–	1950	1880	1942.12	7583942	–	1880	2.27	0.00		
	1777	1829	1779	3600	6120226	0.11	1829	1829	3600	13859207	2.93	1779	3600	0.11		
	1488	1534	1494	3600	6394448	0.40	1534	1534	3600	13946120	3.09	1494	3600	0.40		
	1756	1808	1758	3600	6079854	0.11	1808	1808	3600	14366484	2.96	1758	3600	0.11		

Tableau 4.14 – Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P1$)

N	BI	PSE1				PSE2				PSE_agg						
		BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	1021	944	0.06	223	-	1021	944	4.80	24662	-	1023	1023	3600	1305157	7.72
	729	768	729	0.06	215	-	768	729	5.69	38095	-	775	770	3600	1410466	5.32
	832	867	832	0.24	895	-	867	832	54.58	348565	-	912	912	3600	1246227	8.77
	897	969	897	0.20	701	-	969	897	13.46	73590	-	964	934	3600	1269283	3.96
	956	990	956	0.07	241	-	990	956	49.40	298180	-	1035	1014	3600	1321983	5.72
	875	946	875	0.33	1330	-	946	875	1.34	9166	-	945	945	3600	1445192	7.41
	964	1037	964	0.09	296	-	1037	964	40.93	246925	-	1062	1045	3600	1369165	7.75
	869	905	869	0.19	656	-	905	869	18.08	114246	-	918	910	3600	1305545	4.51
	987	1031	987	2.70	10541	-	1031	987	280.92	1819804	-	1073	1073	3600	1353701	8.01
	777	780	777	0.26	950	-	780	777	9.53	57410	-	802	792	3600	1226909	1.89
20	1132	1172	1132	3.06	8594	-	1172	1137	3600	16910076	-	1173	1173	3600	594673	3.50
	990	1020	990	0.17	563	-	1020	990	66.55	342070	-	1105	1083	3600	604210	10.41
	1065	1087	1065	0.01	64	-	1087	1065	39.25	210082	-	1167	1137	3600	608469	6.33
	1089	1135	1089	0.34	1210	-	1135	1089	47.58	250936	-	1204	1191	3600	607991	8.56
	977	1001	977	0.18	502	-	1001	977	2217.06	10483912	-	1092	1066	3600	616376	10.04
	1170	1235	1170	0.36	1112	-	1235	1170	201.21	997986	-	1296	1271	3600	619156	7.95
	1167	1225	1167	0.15	467	-	1225	1167	124.04	614995	-	1283	1283	3600	758062	9.04
	1114	1180	1114	0.11	320	-	1180	1114	56.91	280488	-	1195	1175	3600	786672	5.19
	1103	1173	1103	0.56	1817	-	1173	1103	163.99	784314	-	1195	1187	3600	781348	7.08
	1243	1284	1243	0.34	1129	-	1284	1243	93.37	467493	-	1322	1311	3600	785559	5.19
30	1723	1742	1723	0.06	153	-	1742	1724	3600	13311509	0.06	1784	1784	3600	272075	3.42
	1699	1719	1699	1.01	2418	-	1719	1719	3600	13154796	1.18	1787	1787	3600	267262	9.96
	1740	1788	1740	0.03	56	-	1788	1788	3600	12497596	2.76	1871	1871	3600	270400	7.00
	1787	1790	1787	0.87	1856	-	1790	1790	3600	12236847	0.17	1888	1843	3600	266930	3.04
	1658	1674	1658	0.24	635	-	1674	1660	3600	13189616	0.12	1795	1795	3600	264872	7.63
	1780	1816	1780	0.18	426	-	1816	1816	3600	13095295	2.02	1854	1854	3600	245693	3.99
	1830	1859	1830	1.15	3042	-	1859	1859	3600	13171671	1.58	1938	1935	3600	245462	5.43
	1486	1507	1486	0.11	266	-	1507	1500	3600	14043145	0.94	1635	1604	3600	262740	10.70
	1642	1734	1642	9.01	18600	-	1734	1734	3600	13158205	5.60	1696	1696	3600	255424	3.18
	1584	1608	1584	0.07	172	-	1608	1584	927.69	3630638	-	1717	1705	3600	253986	12.24

Tableau 4.15 – Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P2$)

N	BI	PSE1				PSE2				PSE_agg						
		BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	970	944	0.18	627	–	970	944	29.61	107827	–	1017	1008	3600	1504888	6.78
	729	768	729	0.06	221	–	768	729	6.83	38563	–	767	767	3600,01	1491927	5.21
	832	867	832	0.23	883	–	867	832	53.89	259173	–	906	898	3600	1632812	7.93
	897	920	897	0.61	2162	–	920	897	14.28	51244	–	951	951	3600	1709429	6.02
	956	976	956	0.24	816	–	976	956	16.08	62971	–	1020	1007	3600	1772557	5.33
	875	946	875	0.05	193	–	946	875	2.06	8890	–	951	951	3600	1750153	8.69
	964	1037	964	0.09	334	–	1037	964	39.52	189779	–	1064	1046	3600	1767688	8.51
	869	905	869	0.06	237	–	905	869	20.28	90447	–	913	913	3600	1758543	5.06
	987	1004	987	2.58	9033	–	1004	987	334.63	1456064	–	1076	1055	3600	1762596	6.89
	777	780	777	0.75	1971	–	780	777	34.99	141109	–	788	788	3600	1767490	1.42
20	1132	1222	1132	0.64	1400	–	1222	1151	3600	12547549	–	1165	1153	3600,01	859363	1.86
	990	1020	990	0.10	266	–	1020	990	54.20	187049	–	1094	1070	3600,01	868567	8.08
	1065	1087	1065	0.03	68	–	1087	1065	35.72	133328	–	1159	1136	3600,01	863441	6.67
	1089	1135	1089	0.12	408	–	1135	1089	63.91	251737	–	1337	1322	3600,01	859503	21.40
	977	1044	977	0.23	627	–	1044	979	3600	12445986	–	1071	1040	3600,01	864878	6.45
	1170	1181	1170	0.92	2006	–	1181	1170	468.48	1685732	–	1266	1266	3600,01	864455	8.21
	1167	1225	1167	0.23	487	–	1225	1167	111.17	404393	–	1290	1255	3600,01	862529	7.54
	1114	1180	1114	0.06	151	–	1180	1114	52.72	194088	–	1190	1186	3600,01	865579	6.46
	1103	1117	1103	0.72	1952	–	1117	1103	384.79	1460018	–	1188	1179	3600,01	862951	6.89
	1243	1268	1243	0.95	2881	–	1268	1243	168.75	645298	–	1321	1321	3600,01	864487	6.28
30	1723	1742	1723	0.06	153	–	1742	1742	3600	9765414	1.10	1791	1791	3600,04	297986	3.95
	1699	1725	1699	0.16	388	–	1725	1725	3600	9888178	1.53	1882	1863	3600,04	295188	9.65
	1740	1778	1740	2.04	4143	–	1778	1778	3600	9273171	2.18	1875	1875	3600,04	298895	7.76
	1787	1839	1787	0.54	1169	–	1839	1839	3600	9415562	2.91	1834	1834	3600,04	295175	2.63
	1658	1727	1658	6.06	15126	–	1727	1718	3600	10121782	3.62	1800	1788	3600,04	297021	7.84
	1780	1806	1780	0.31	704	–	1806	1806	3600	9868754	1.46	1849	1833	3600,03	295147	2.98
	1830	1859	1830	1.17	3041	–	1859	1859	3600	9880187	1.58	1936	1936	3600,03	296880	5.79
	1486	1510	1486	0.10	283	–	1510	1510	3600	10703254	1.62	1653	1618	3600,05	280344	8.88
	1642	1675	1642	14.04	30074	–	1675	1675	3600	9780592	2.01	1686	1684	3600,03	279464	2.56
	1584	1608	1584	0.05	125	–	1608	1584	1298.42	3762814	–	1798	1772	3600,05	277401	11.87

Tableau 4.16 – Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances de Taillard $N = \{15, 20, 30\}$, $P3$)

N	BI	PSE1				PSE2				PSE_agg						
		BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	944	1003	944	0.05	166	–	1003	944	12.81	67821	–	1073	1030	3600	1244398	9.11
	729	768	729	0.05	221	–	768	729	11.11	71640	–	798	762	3600	1276335	4.53
	832	867	832	0.22	857	–	867	832	32.54	203917	–	941	903	3600	1261593	8.53
	897	902	897	0.36	1297	–	902	897	53.69	293441	–	952	922	3600	1267116	2.79
	956	984	956	0.15	514	–	984	956	17.02	103338	–	1042	1012	3600	1274066	5.86
	875	946	875	0.09	364	–	946	875	0.78	5214	–	995	953	3600	1264789	9.91
	964	1037	964	0.08	328	–	1037	964	32.48	199808	–	1150	1085	3600	1247856	12.55
	869	905	869	0.90	3448	–	905	869	29.21	190625	–	956	926	3600	1272148	6.56
	987	1004	987	1.59	6442	–	1004	987	57.13	353362	–	1121	1074	3600	1261489	8.81
	777	798	777	0.77	2713	–	798	777	63.72	366018	–	863	821	3600	1249863	5.66
20	1132	1154	1132	0.55	1480	–	1154	1132	2743.33	13292983	–	1209	1174	3600	793523	3.71
	990	1020	990	0.02	64	–	1020	990	19.11	99058	–	1085	1042	3600	805176	5.25
	1065	1087	1065	0.09	214	–	1087	1065	44.51	231910	–	1196	1147	3600	801685	7.70
	1089	1135	1089	0.04	154	–	1135	1089	33.13	171990	–	1386	1281	3600	836212	17.63
	977	979	977	0.05	130	–	979	977	740.75	3539581	–	1104	1074	3600	838822	9.93
	1170	1215	1170	0.981	2408	–	1215	1170	104.18	527612	–	1333	1303	3600	802694	11.37
	1167	1225	1167	0.14	442	–	1225	1167	94.88	464300	–	1350	1294	3600	821475	10.88
	1114	1180	1114	0.05	167	–	1180	1114	40.97	202672	–	1246	1212	3600	836478	8.80
	1103	1148	1103	0.20	652	–	1148	1103	431.28	2126110	–	1249	1210	3600	805472	9.70
	1243	1268	1243	0.41	1303	–	1268	1243	139.47	702759	–	1370	1340	3600	823476	7.80
30	1723	1742	1723	0.06	153	–	1742	1742	3600	12386792	1.10	1823	1793	3600	289672	4.06
	1699	1747	1699	39.50	87759	–	1747	1747	3600	13008288	2.83	1937	1888	3600	286086	11.12
	1740	1742	1740	0.22	473	–	1742	1742	3600	12343797	0.11	1914	1884	3600	289505	8.28
	1787	1814	1787	0.55	1184	–	1814	1814	3600	9775274	1.51	1863	1833	3600	286439	2.57
	1658	1670	1658	0.03	76	–	1670	1670	3600	9812781	0.72	1842	1800	3600	287434	8.56
	1780	1791	1780	12.30	27050	–	1791	1791	3600	10075292	0.62	1924	1878	3600	287439	5.51
	1830	1859	1830	1.18	3034	–	1859	1859	3600	9768468	1.58	1973	1943	3600	280987	6.17
	1486	1535	1486	0.14	385	–	1535	1535	3600	10400988	3.30	1725	1654	3600	284763	11.31
	1642	1691	1642	18.58	42145	–	1691	1691	3600	12940763	2.98	1735	1733	3600	283084	5.54
	1584	1608	1584	0.05	129	–	1608	1584	1021.01	3945625	–	1859	1825	3600	284012	15.21

Tableau 4.17 – Comparaison des résultats obtenus par PSE1, PSE2 et PSE_agg (Instances aléatoires , $P3$)

N	BI	PSE1					PSE2					PSE_Agg				
		BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)	BS	C _{max}	CPU(s)	Nodes	Gap(%)
15	1045	1083	1053	136.42	580784	–	1083	1053	1891.35	8989991	–	1128	1128	1665227	3600	7.12
	853	907	857	102.93	441332	–	907	857	2088.08	10108864	–	1004	976	1714907	3600	13.89
	821	871	821	0.11	459	–	871	821	7.46	31182	–	902	872	1773167	3600	6.21
	992	1056	996	134.23	564962	–	1056	996	875.63	4040355	–	1068	1068	1767940	3600	7.23
	906	958	913	65.17	275737	–	958	913	1770.71	8437680	–	1009	999	1772985	3600	9.42
	874	920	877	239.76	762757	–	920	877	2022.17	9613372	–	951	941	1773913	3600	7.30
	1120	1210	1120	0.02	47	–	1210	1120	5.01	26871	–	1220	1130	1771612	3600	0.89
	970	1005	975	193.92	554631	–	1005	975	1985.19	9474141	–	1005	995	1768540	3600	2.05
	960	970	960	0.01	25	–	970	960	0.48	2006	–	1060	980	1768189	3600	2.08
	821	831	821	0.01	15	–	831	821	6.65	27255	–	921	841	1772746	3600	2.44
20	1291	1306	1296	3600	7699181	0.00	1306	1296	3600	13823632	0.00	1347	1347	858349	3600	3.94
	1081	1091	1081	0.05	99	–	1091	1081	29.69	122301	–	1113	1103	857624	3600	2.04
	1218	1279	1219	3600	7767143	0.00	1279	1219	3600	13785815	0.00	1311	1251	865260	3600	2.63
	1113	1117	1117	3600	7884961	0.00	1117	1117	3600	18132674	0.00	1224	1154	857186	3600	3.31
	1312	1366	1316	3600	8327689	0.00	1366	1316	3600	18187703	0.00	1378	1368	863431	3600	3.95
	1249	1305	1255	3600	8103980	0.00	1305	1255	3600	19019267	0.00	1384	1364	862306	3600	8.69
	1302	1351	1311	3600	8096330	0.00	1351	1311	3600	18951258	0.00	1593	1523	860272	3600	16.17
	1207	1214	1214	3600	8410093	0.00	1214	1214	3600	18953215	0.00	1291	1251	863349	3600	3.05
	1302	1311	1311	3600	8425768	0.00	1311	1311	3600	18679289	0.00	1373	1333	863638	3600	1.68
	1354	1407	1357	3600	8388219	0.00	1407	1357	3600	19097015	0.00	1421	1421	864939	3600	4.72
30	1705	1733	1713	3600	6058946	0.47	1733	1733	3600	13733696	1.64	1788	1788	296577	3600	4.87
	1808	1814	1814	3600	6074555	0.33	1814	1814	3600	13853968	0.33	1982	1842	296648	3600	1.88
	1922	1971	1931	3600	6211670	0.47	1971	1931	3600	14658592	0.47	2033	2023	291429	3600	5.25
	2025	2093	2033	3600	6240605	0.39	2093	2033	3600	15083229	0.40	2097	2097	297690	3600	3.56
	1922	1941	1931	3600	6435738	0.47	1941	1931	3600	14335396	0.47	2013	1973	296847	3600	2.65
	1405	1463	1413	3600	6692669	0.00	1463	1463	3600	14038068	4.13	1532	1502	289554	3600	6.90
	1880	1950	1880	0.05	83	–	1950	1880	1942.12	7583942	–	1954	1924	293074	3600	2.34
	1777	1829	1779	3600	6120226	0.11	1829	1829	3600	13859207	2.93	1845	1845	291440	3600	3.83
	1488	1534	1494	3600	6394448	0.40	1534	1534	3600	13946120	3.09	1572	1572	289651	3600	5.65
	1756	1808	1758	3600	6079854	0.11	1808	1808	3600	14366484	2.96	1873	1843	260209	3600	4.95

4.9 Conclusion

Nous avons étudié un problème d'ordonnancement de type job shop à deux machines avec une période d'indisponibilité sur chaque machine pour minimiser le makespan. Nous avons démontré que les ordonnancements de permutation sont dominants pour les cas étudiés. Ce résultat permet d'appliquer la règle de Jackson dans nos approches de résolution. Nous avons ensuite proposé des propriétés prouvant un ordre optimal des tâches basé sur la règle de Jackson et garantissant l'ordre optimal de chaque ensemble de tâches avant et après chaque période d'indisponibilité. Nous avons présenté deux modèles MILP : le premier est un modèle de job shop classique adapté à notre problème tandis que le second est une application de l'approche présentée. Nous avons également développé deux algorithmes PSE basés sur les propriétés décrites avec des schémas de branchement différents. Des expériences réalisées sur des instances générées aléatoirement et à partir de la littérature indiquent l'efficacité de l'approche proposée par rapport à celle de la littérature. De plus, nous en déduisons que les méthodes *PSE1* et *MILP4* sont plus efficaces que *PSE2*. Le chapitre suivant sera consacré aux méthodes de résolution approchées du problème notamment une heuristique de construction et une méta-heuristique.

Chapitre 5

Méthodes approchées pour le problème d'ordonnancement de type Job shop à deux machines avec contraintes de disponibilité sur une machine

Résumé :

Nous développons dans ce chapitre des méthodes pour la résolution approchée du problème d'ordonnancement de type job shop à deux machines avec des contraintes de disponibilité sur une seule machine. Nous avons démontré par l'étude au pire cas de l'heuristique basée sur la règle de Jackson, que plus le nombre d'indisponibilités et l'erreur relative sont proportionnels. Par conséquent, des méthodes approchées tenant en compte les temps d'inactivité générés par chaque période d'indisponibilité sont développées, à savoir, une heuristique de construction et un algorithme de recherche locale itérée avec une liste tabou. Ces deux algorithmes prennent en compte l'optimalité de l'ordre de Jackson avant et après chaque période d'indisponibilité. Les expérimentations basées sur des benchmarks et une comparaison avec des méthodes issues de la littérature et avec la procédure par séparation et évaluation présentée dans le chapitre 3, montrent la qualité des méthodes introduites.

Publications :

- Benttaleb, Mourad, Faicel Hnaïen, and Farouk Yalaoui. An iterated local search algorithm tabu based for the two machine scheduling problem under availability constraints. *Computers & operations research*, (En cours de soumission).
- Benttaleb, Mourad, Faicel Hnaïen, and Farouk Yalaoui. Heuristic algorithms for two-machine scheduling problem under availability constraints on one machine : makespan minimization. *16th IFAC Symposium on Information Control Problems in Manufacturing 11-13 June, Bergamo, Italy* 2018.
- Benttaleb, Mourad, Faicel Hnaïen, and Farouk Yalaoui. A constructive heuristic for two machine job shop scheduling problem under availability constraints on one machine. *7th International Conference on Metaheuristics and Nature Inspired Computing META'18 Marrakesh, Morocco*, 2018.

5.1 Introduction

Nous présentons dans ce chapitre des méthodes approchées pour le problème d'ordonnancement de type job shop à deux machines avec contraintes de disponibilité sur une seule machine.

Parmi les conclusion tirées du chapitre 4, nous rappelons que les méthodes exactes (y compris la procédure par séparation et évaluation) sont limitées concernant la résolution des instances de grandes tailles avec un nombre de périodes d'indisponibilité égal à $N/4$ et $N/2$. D'où le besoin de développer de nouveaux algorithmes efficaces pour la résolution approchée du problème.

Nous considérons le cas des opérations strictement *non-préemptives* avec l'objectif de minimiser le makespan. Après un rappel des propriétés démontrées dans le chapitre 4 et dont on aura besoin dans le développement des algorithmes dans les sections suivantes.

La troisième section de ce chapitre est consacrée à une méthode de construction basée sur la règle de Jackson. La quatrième section est dédiée à une méta-heuristique de recherche locale itérée (ILS) avec une liste tabou. Une étude de sensibilité est élaborée pour calibrer les paramètres par la méthode ANOVA.

Les résultats expérimentaux montrent l'efficacité de l'approche proposée et des méthodes présentées par rapport à celles de la littérature.

5.2 Propriétés

Nous rappelons la propriété suivante afin de démontrer les choix suivis dans le développement des méthodes approchées proposées dans ce chapitre.

Rappel propriété 6. *Pour un problème de type job shop à deux machines avec $U \geq 2$ périodes d'indisponibilité sur la machine M_1 ($J_2, h_{1U}|a|C_{max}$), il existe un ordre optimal garantissant que les tâches avant, après et entre chaque deux périodes d'indisponibilité consécutives, sont séquencées suivant la règle de Jackson.*

Nous rappelons également l'algorithme H1 qui est l'application de la règle de Jackson pour la résolution du problème $J_2, h_{1U}|a|C_{max}$ (voir l'algorithme 4).

Algorithme 4 Heuristique 1 : H1

```

1: Entrée :
2:  $N$  tâches sur deux machines
3:  $U$  périodes d'indisponibilité sur la machine  $M_1$ 
4: Ordonnancer les tâches selon la règle de Jackson
5:  $u = 1$ 
6: Pour  $i = 1$  To  $N$  Faire
7:   Tant que  $u < U + 1$  Faire
8:     Si  $s_{i1} < s_u$  et  $c_{i1} > s_u$  Alors
9:        $s_{i1} = t_u$ 
10:      Actualiser les dates d'achèvement des tâches de  $I_{u+1}$ 
11:       $u = u + 1$ 
12:     Si  $s_{i1} \leq s_u$  Alors
13:       Actualiser les dates d'achèvement des tâches de  $I_u$ 
14:     break
15:   Fin Si
16: Fin Si
17: Fin Tant que
18: Fin Pour
19: return  $C_{max}(H1)$ 

```

Rappel du Théorème 3.1. *Pour le problème $J_2, h_{1U}|a|C_{max}$, le ratio d'approximation par la règle de Jackson est égal à 2 dans le pire cas.*

Rappel propriété 7. *Pour $J_2, h_{1U}|a|C_{max}$:*

$$\frac{C_{max}(H1)}{C^*} \leq 1 + \frac{\sum_{u=1}^U g_u + \sum_{u=1}^U Idle_u}{\sum_{i=1}^N p_{i1} + \sum_{u=1}^U g_u} \quad (5.1)$$

Rappel propriété 8. *Si $C_{max}(H1) = \sum_{i=1}^N p_{i2}$, alors le makespan obtenu par H1 est optimal.*

Nous remarquons selon la propriété 7 que l'erreur augmente proportionnellement avec la somme des temps d'inactivité générés par la non-préemption des tâches avant chaque période d'indisponibilité. Donc, réduire le gap de la solution approchée par rapport à l'optimal équivaut à minimiser la somme des temps d'inactivité. Cela amène à développer des algorithmes pour décider de l'ensemble des tâches à traiter dans chaque intervalle.

Nous proposons dans un premier temps une heuristique de construction (H_const)

minimisant les temps d'inactivité avant chaque période d'indisponibilité et par la suite une recherche locale itérée avec une liste taboue (*ILS_LT*).

5.3 Heuristique Constructive (*H_const*)

La procédure de construction utilisée par l'algorithme *H_const* est également basée sur la règle de Jackson. Il s'agit d'ordonnancer une tâche après l'autre suivant la séquence obtenue par la règle de Jackson. Étant donnée une séquence représentant l'ordre de Jackson, *H_const* permet de construire l'ordonnancement associé à cette séquence tenant en compte les périodes d'indisponibilité et la *non-préemption* des opérations.

L'horizon d'ordonnancement contenant U périodes d'indisponibilité, est divisé à $U + 1$ intervalle $I_1, \dots, I_{u+1}$. *H_const* essaie d'ordonnancer les tâches au plus tôt dans ces $U + 1$ intervalles. Donc, étant donné une tâche J_i à ordonnancer, l'algorithme repart à partir du premier intervalle jusqu'à son affectation finale. Ainsi, il détermine l'intervalle dans lequel la tâche en question sera insérée en considérant les $(i - 1)$ tâches déjà insérées et en vérifiant la condition $C_1(I_u) \leq S_u \forall u \in 1, \dots, U + 1$ sachant que toute tâche déjà placée l'est de manière définitive. L'ordre globale des tâches ordonnancées est celui démontré par la propriété 6. La procédure se répète en insérant la tâches suivantes selon l'ordre de Jackson. L'algorithme se termine lorsque toutes les tâches sont affectées.

L'algorithme 5 permet de résoudre de manière approchée le problème étudié $J_2, h_{1U}|a|C_{max}$.

5.4 Recherche locale itérée (Iterated local search algorithm)

La recherche locale itérée (Iterated Local Search ou ILS) est une méta-heuristique qui implique l'application itérative d'une heuristique de recherche locale et l'utilisation d'une perturbation comme mécanisme de diversification.

A chaque itération, une nouvelle solution initiale est générée et utilisée par la recherche locale comme nouveau point de départ de la recherche. Cette solution initiale est générée en effectuant de manière aléatoire un certain nombre de modifications appropriées constituant la perturbation, à une solution localement optimale. En effet, au lieu de générer une

Algorithme 5 Heuristique H_const

```
1: Input :
2:  $N$  tâches sur deux machines
3:  $U$  périodes d'indisponibilité sur la machine  $M_1$ 
4: Ordonnancer les tâches selon la règle de Jackson
5: Pour  $i = 1$  To  $N$  Faire
6:    $u = 1$ 
7:   Tant que  $u < U + 1$  Faire
8:     Si  $s_{i1} < s_u$  et  $c_{i1} > s_u$  Alors
9:        $s_{i1} = t_u$ 
10:      Actualiser les dates d'achèvement des tâches de  $I_{u+1}$ 
11:       $u = u + 1$ 
12:     Si  $c_{i1} \leq s_u$  Alors
13:       Actualiser les dates d'achèvement des tâches de  $I_u$ 
14:     break
15:   Fin Si
16: Fin Si
17: Fin Tant que
18: Fin Pour
19: return  $C_{max}(H1)$ 
```

nouvelle solution initiale, le mécanisme de perturbation génère une solution prometteuse en conservant une partie de la structure de la solution optimale localement. Malgré sa simplicité, l'algorithme *ILS* s'est avéré être une méthode efficace pour résoudre les problèmes d'optimisation combinatoire surtout lorsque son choix est justifié et suffisant.

Comme mentionné, l'objectif est de minimiser les temps d'inactivité généré par la *non-préemption* des opérations par les contraintes de disponibilité. Des recherches locales peuvent être efficaces pour minimiser ces temps d'inactivité. Hors, dans ce genre de problème combinatoire l'optimum local est inévitable, d'où la nécessité de la perturbation. Ainsi, l'algorithme *ILS* est nécessaire et suffisant pour résoudre approximativement et efficacement le problème $J_2, h_{1U}|a|C_{max}$.

Nous décrivons en détail l'algorithme en question. L'objectif est de trouver l'ensemble approprié des tâches à exécuter dans chaque intervalle ordonnancées suivant la règle de Jackson afin de minimiser les temps d'inactivité, et ainsi le makespan.

L'algorithme *ILS_LT* comprend trois composantes principales : la construction de la solution initiale, la procédure de recherche locale et le mécanisme de perturbation.

Dans les sections suivantes, nous décrivons les principales composantes de l'algorithme. En outre, nous discutons des paramètres de l'algorithme dans le but de trouver la meilleure combinaison améliorant sa performance.

5.4.1 La solution initiale

L'algorithme *ILS_LT* nécessite une solution initiale pour lancer sa recherche. En général, la solution initiale peut être générée au moyen d'une procédure aléatoire ou d'une heuristique gloutonne. Dans cette tâche, la procédure de construction utilisée par l'algorithme *ILS_LT* est basée sur la règle de Jackson. Précisément, nous implémentons l'algorithme constructif de Jackson décrit dans la section précédente pour obtenir la solution initiale. L'heuristique utilisée fournit une solution initiale de qualité relativement élevée pour démarrer la méta-heuristique par rapport à l'algorithme de Jackson présenté dans le chapitre 4. Nous discutons cette performance dans la section dédiée aux résultats expérimentaux.

5.4.2 La recherche locale

La procédure de recherche locale effectue une descente de la solution initiale (qui peut être la solution initiale ou perturbée) jusqu'à ce qu'elle atteigne un optimum local. La puissance de chaque recherche locale réside dans le voisinage considéré et dans la façon dont il est exploré.

Cette recherche locale alterne entre deux façons de chercher une meilleure solution. Il est basé sur l'échange de deux tâches appartenant à deux intervalles différents (échange dit swap neighborhood) ou le déplacement d'une tâche d'un intervalle à un autre (insertion dit insert neighborhood). L'optimalité locale est atteinte si la solution ne peut plus être améliorée. Un paramètre gérant la décision sur le nombre d'itérations acceptées pour atteindre un optimal local est défini et intitulé MaxOpt. Toute solution irréalisable est écartée ; c'est le cas où la tâche insérée amène à $C_1^{I_u} > s_u$ pour un I_u quelconque.

5.4.3 La perturbation

La procédure de perturbation est choisie pour éviter le cycle entre un optimum local et son meilleur voisin. À ce stade, l'algorithme génère son mécanisme de perturbation, qui applique intensivement un des deux types de recherche locale sélectionnées, partant de l'optimum

local courant. Cette solution optimale locale perturbée devient un nouveau point de départ pour la prochaine itération de la recherche locale.

Un bon équilibre entre intensification et diversification est essentiel pour la performance de toute méta-heuristique. La procédure de perturbation joue un rôle important dans la recherche de cet équilibre. Toute solution irréalisable est gardée et rendue réalisable en essayant d'insérer la tâche en contrainte dans l'un des intervalles suivant au plus tôt jusqu'à son insertion finale.

Un paramètre gérant le nombre de fois la procédure de perturbation procède une recherche locale est défini et noté *MaxPrtb*.

Toute séquence déjà visitée est ajoutée à la liste taboue afin de diversifier la zone de recherche et d'accélérer l'obtention d'une meilleure solution. Cette méthode pourrait être utilisée comme perturbation étant donné que l'objectif de la recherche tabou classique est d'assurer une diversification dans chaque itération au moyen d'une liste de tabou.

5.4.4 Le critère d'acceptation

Le critère d'acceptation est également important pour trouver un bon équilibre entre la diversification et l'intensification de la recherche. En conséquence, l'algorithme est capable de dépenser suffisamment de temps de calcul pour encore améliorer la meilleure solution conduisant à plus d'intensification. Si le nombre d'itérations sans amélioration augmente, l'algorithme perd en coût de calcul et n'aboutit plus à une meilleure solution.

L'algorithme peut également passer plus de temps dans l'étape de la diversification afin d'améliorer la solution courante. La perturbation d'une solution optimale localement amène à une nouvelle solution non voisine. Une diversification intense peut être coûteuse en temps de calcul et pouvant faire perdre la qualité de la séquence courante. Donc, ses paramètres *Maxoptl* et *MaxPrtb* sont primordiales pour accepter une solution de bonne qualité en un temps le moins possible.

La meilleure solution parmi tous les optima locaux visités est la meilleure solution (Voir l'algorithme 6). Cette solution assure également que tout l'ordonnancement est de permutation et satisfait l'ordre des tâches dans chaque intervalle comme démontré dans la propriété 6.

5.4.5 Le critère d'arrêt

L'exécution de l'algorithme est arrêtée lorsqu'un nombre maximum d'itérations est atteint noté *MaxIter*, ou lorsque une borne inférieure est atteinte.

5.5 Résultats expérimentaux

Les algorithmes proposés sont codés en *C++*. Les tests ont été exécutés sur un PC de fréquence 2.6 GHz Intel(R) Core (TM) i5-4210M CPU et 8.00 GB.

5.5.1 Génération d'instances

Nous utilisons les instances de Taillard (1993) pour le problème de type job shop (15 tâches / 15 machines, 20 tâches / 15 machines). Pour chaque instance, nous avons pris seulement les deux premières valeurs de chaque tâche, correspondant à leurs durées sur les deux premières machines.

Nous ajoutons les paramètres concernant la période d'indisponibilité à fixer sur la machine M_1 . La durée de la $u^{\text{ème}}$ période d'indisponibilité est supposée égale à la moyenne des temps de traitement des opérations sur la machine M_1 : $g_u = t_u - s_u = \frac{\sum_{i=1}^N p_{i1}}{N}$.

Nous générons des instances prenant en compte l'emplacement de la $u^{\text{ème}}$ période d'indisponibilité au milieu de l'horizon d'ordonnancement disponible. La date de début de cette période est définie en fonction du nombre total de périodes d'indisponibilité prises en compte sur la machine M_1 : $g_u = t_u - s_u = \frac{\sum_{i=1}^N p_{i1}}{N}$.

5.5.2 Paramétrage et analyse de sensibilité

L'algorithme *ILS_LT* utilise trois paramètres :

- *MaxIter* : Ce paramètre définit les itérations maximales de l'algorithme.
- *MAxOptl* : Ce paramètre est utilisé pour trouver l'optimum local. Si le nombre d'itérations suivantes sans amélioration atteint *MAxOptl*, l'algorithme maintient la solution courante comme optimum local.
- *MAxPrtb* : Ce paramètre est utilisé pour limiter le nombre de recherches locales

Algorithmme 6 Algorithmme *ILS_LT*

```
1:  $s^* \leftarrow C_{max}(H1)$ 
2:  $Iter \leftarrow 0$ 
3:  $Tabu\ list \leftarrow \emptyset$ 
4:  $LocalOpt \leftarrow \emptyset$ 
5: Tant que  $Iter \leq MaxIter$  Faire
6:    $Tabu\ list \leftarrow s^*$ 
7:    $s \leftarrow Localsearch(s^*)$ 
8:    $Optl \leftarrow 0$ 
9:   Tant que  $Optl \leq MaxOptl$  Faire
10:    Si  $s \notin Tabu\ list$  Alors
11:      Si  $f(s) < f(s^*)$  Alors
12:         $s^* \leftarrow s$ 
13:         $f(s^*) \leftarrow f(s)$ 
14:         $Optl \leftarrow 0$ 
15:      Sinon
16:         $Optl \leftarrow Optl + 1$ 
17:        Si  $Optl = MaxOptl$  Alors
18:           $LocalOpt \leftarrow s^*$ 
19:          Fin Si
20:          Fin Si
21:           $Tabu\ list \leftarrow s$ 
22:          Fin Si
23:          Tant que
24:             $Prtb \leftarrow 0$ 
25:            Tant que  $Prtb \leq MaxPrtb$  Faire
26:               $s^* \leftarrow Localsearch(s^*)$ 
27:               $Prtb \leftarrow Prtb + 1$ 
28:            Fin Tant que
29:             $Iter \leftarrow Iter + 1$ 
30:          Fin Tant que
31:           $Tabu\ list \leftarrow \emptyset$ 
32:           $s^* \leftarrow min(LocalOpt)$ 
33:        return  $s^*$ 
```

Tableau 5.1 – Valeurs des paramètres

Paramètre	Valeurs considérées
MaxIter	100, 500, 1000
MaxOpt	2, 4, 6, 10
MaxPrtb	$N/10$, $N/4$, $N/3$, $N/2$

successives sur une solution dans chaque perturbation. Cela permet de gérer la diversification d'une solution.

Nous effectuons une expérience factorielle complète avec ces trois paramètres. Les niveaux suivants ont été testés : $MaxPrtb \in \{n/10, n/4, n/3, n/2\}$, $MaxOpt \in \{2, 4, 6, 10\}$, $MaxIter \in \{100, 500, 1000\}$. Le tableau 5.1 présente pour chaque paramètre les valeurs considérées. Toutes les combinaisons de valeurs des trois paramètres donnent 48 configurations de l'algorithme *ILS_LT* pour chaque instance. Pour chaque combinaison de paramètres, nous exécutons l'algorithme 10 fois. Nous déterminons l'écart moyen, le temps de calcul ($CPU(s)$), la meilleure solution trouvée ($Best$), la pire solution (Wst), la moyenne des 10 solutions (Avg) et la déviation calculée comme $Dev = \frac{Avg - Best}{Avg}$.

Afin d'évaluer la sensibilité des paramètres sélectionnés, nous avons résolu toutes les instances pour chaque combinaison de paramètres. Cette section présente le résultat des expériences réalisées sur les méthodes développées incluant les *ILS_LT* sous les différentes combinaisons de paramètres.

Trois séries d'expériences sont effectuées pour examiner l'effet de différents paramètres sur la performance de l'algorithme pour chaque cas de test (instance, combinaison de paramètres). Le premier ensemble d'expériences traite de la performance de l'algorithme en termes de qualité de la solution. Le makespan est celui donné par Avg ; tandis que la deuxième étudie la performance de chaque configuration d'algorithme mesurée par $Dev = \frac{Avg - Best}{Avg}$ puisque la solution de chaque instance du paramétrage est toujours non nulle, où $Best$ et Avg sont la meilleure solution trouvée et la moyenne de toutes les solutions des tests lancés respectivement. La troisième étudie la combinaison des paramètres impactant significativement le temps de calcul de l'algorithme.

Les tableaux 5.2, 5.3 et 5.4 montrent le résultat de l'ANOVA des expériences pour les instances de tailles 15, 20 et 30 tâches respectivement. Ils comprennent le F-ratio et P-value pour chaque réponse Avg , Dev et $CPU(s)$ ainsi que leurs interactions.

Tableau 5.2 – Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 15$

<i>Source</i>	<i>Avg</i>		<i>Dev</i>		<i>CPU(s)</i>	
	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>
<i>A : MaxIter</i>	0.09	0.913	3.23	0.040	2707.90	0.000
<i>B : MaxOpt</i>	0.00	1.000	0.14	0.935	0.08	0.971
<i>C : MacPrtb</i>	0.17	0.916	30.21	0.000	81.39	0.000
<i>A * B</i>	0.00	1.000	0.26	0.956	0.04	1.000
<i>A * C</i>	0.00	1.000	0.46	0.838	44.52	0.000
<i>B * C</i>	0.00	1.000	0.34	0.962	0.07	1.000
<i>A * B * C</i>	0.00	1.000	0.53	0.942	0.12	1.000

Selon les résultats de ces tableaux, nous remarquons que le paramètre *MaxOpt* n’a pas un impact par rapport aux paramètres *MAxPrtb* et *MAxIter*. Ce dernier est très significatif en terme de coût de temps de calcul. Nous remarquons également que ces deux paramètres impactent la déviation de l’algorithme.

Nous analyserons également les effets principaux des paramètres ainsi que leurs interactions sur *Avg*, *Dev* et *CPU(s)* pour les instances de tailles 20. L’objectif est de visualiser l’effet de chaque paramètre (figures 5.4, 5.5 et 5.6) ainsi que l’interaction (figures 5.1, 5.2 et 5.3).

En ce qui concerne la qualité de la solution obtenue par la valeur *Avg*, nous remarquons que plus le nombre d’itérations et la perturbation sont élevés, meilleure est la solution. Les figures 5.4, 5.5 et 5.6 montrent que le paramètre *MaxOpt* n’a pas un grand impact quand il agit seul, mais sa combinaison avec *MaxPrtb* conduit à une minimisation du makespan de manière significative comme illustré dans la figure 5.1. Nous concluons à partir de ces figures que les valeurs minimisant le makespan sont $\{500, 1000\}$ pour *MAxIter* et $\{N/3, N/2\}$ pour *MAxPrtb* respectivement. Cependant, en termes de minimisation de déviation des solutions *Dev*, le choix des paramètres *MAxOpt* = 6 et *MaxPrtb* = $N/3$ permet de contrôler l’algorithme par rapport à l’écart des solutions comme le montrent les figures 5.3 et 5.6.

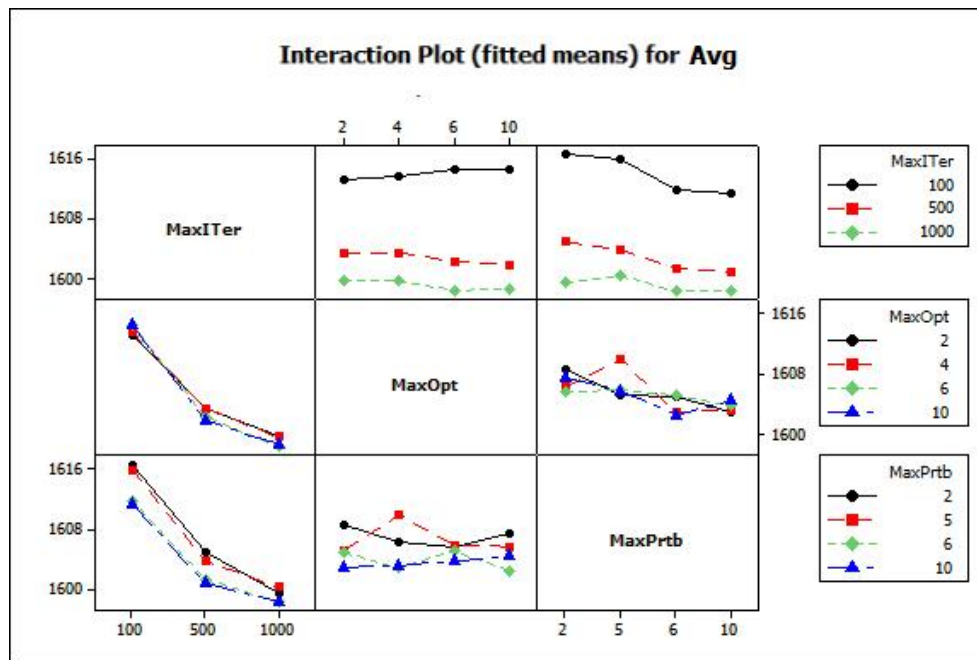
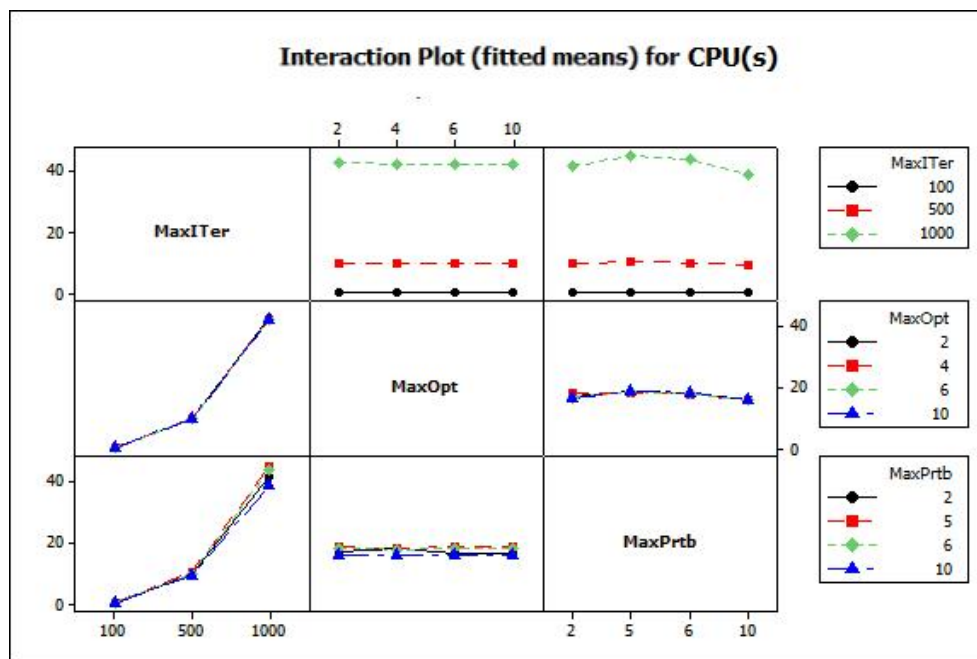
Il est clair à partir des figures 5.2 et 5.5 que *MaxIter* a l’impact le plus significatif sur le temps de calcul. Puisque la performance de l’algorithme selon la valeur *Avg* de makespan n’est pas significativement différente quand le choix de *MaxIter* est 500 ou 1000, alors le meilleur choix de *MaxIter* qui répond le mieux au compromis de la qualité de la solution, le temps de calcul et l’écart de la solution est de *MaxIter* = 500.

Tableau 5.3 – Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 20$

<i>Source</i>	<i>Avg</i>		<i>Dev</i>		<i>CPU(s)</i>	
	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>
<i>A : MaxIter</i>	0.11	0.896	10.97	0.000	12963.88	0.000
<i>B : MaxOpt</i>	0.00	1.000	0.10	0.958	0.38	0.768
<i>C : MacPrtb</i>	0.01	0.999	3.11	0.026	40.19	0.000
<i>A * B</i>	0.00	1.000	0.56	0.765	0.36	0.905
<i>A * C</i>	0.00	1.000	0.71	0.643	21.10	0.000
<i>B * C</i>	0.00	1.000	0.19	0.995	0.16	0.997
<i>A * B * C</i>	0.00	1.000	0.37	0.992	0.12	1.000

Tableau 5.4 – Résultats ANOVA pour Avg , Dev et $CPU(s)$, $N = 30$

<i>Source</i>	<i>Avg</i>		<i>Dev</i>		<i>CPU(s)</i>	
	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>	<i>F – ratio</i>	<i>P – ratio</i>
<i>A : MaxIter</i>	0.03	0.975	2.42	0.091	3417.29	0.000
<i>B : MaxOpt</i>	0.00	1.000	0.19	0.904	0.78	0.506
<i>C : MacPrtb</i>	0.00	1.000	0.48	0.695	4.38	0.005
<i>A * B</i>	0.00	1.000	0.45	0.845	0.91	0.487
<i>A * C</i>	0.00	1.000	0.55	0.767	2.45	0.024
<i>B * C</i>	0.00	1.000	0.15	0.998	0.09	1.000
<i>A * B * C</i>	0.00	1.000	0.30	0.998	0.09	1.000

FIGURE 5.1 – Diagramme des interactions pour *Avg*FIGURE 5.2 – Diagramme des interactions pour *CPU(s)*

5.5.3 Analyse des résultats expérimentaux

Nous comparons les trois algorithmes développés à ceux proposés par Aggoune (2002) pour la résolution approchée du problème de type job shop sous contraintes de disponibilité. Le premier proposé par l'auteur est un algorithme de liste avec une liste taboue nommé *AL* –

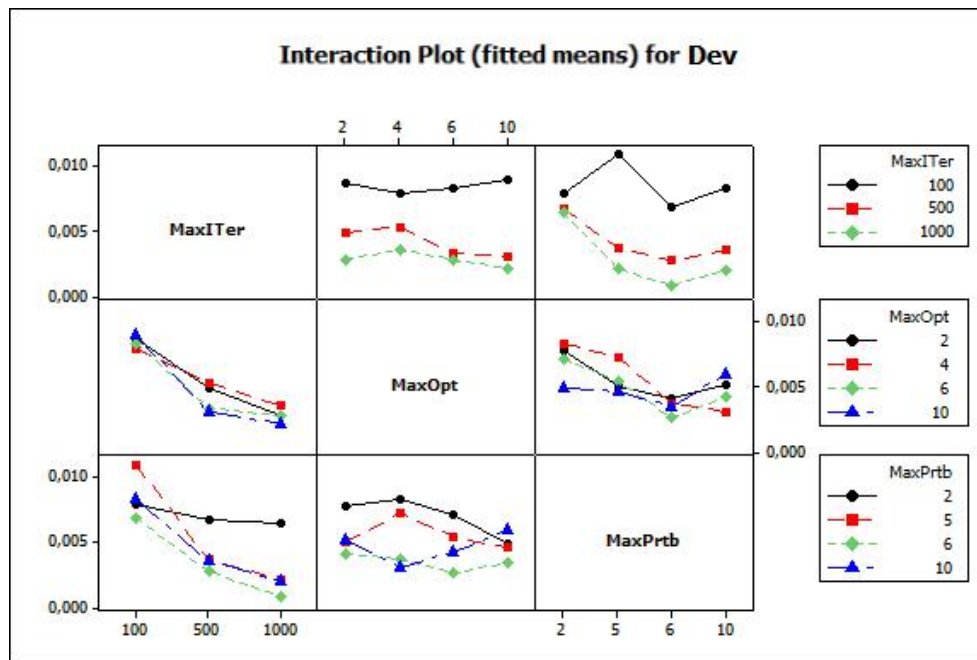


FIGURE 5.3 – Diagramme des interactions pour *Dev*

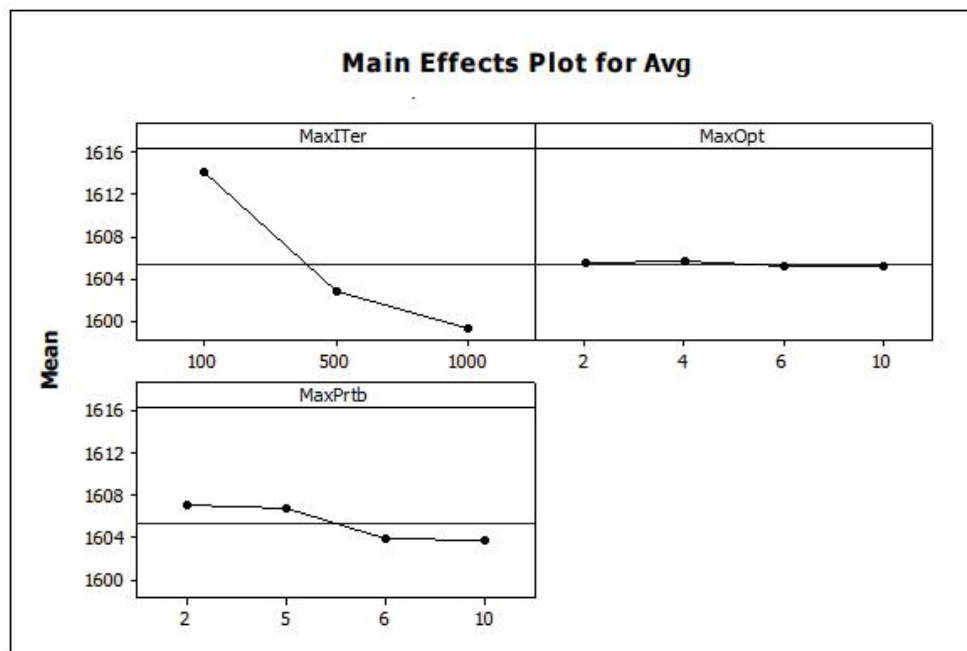
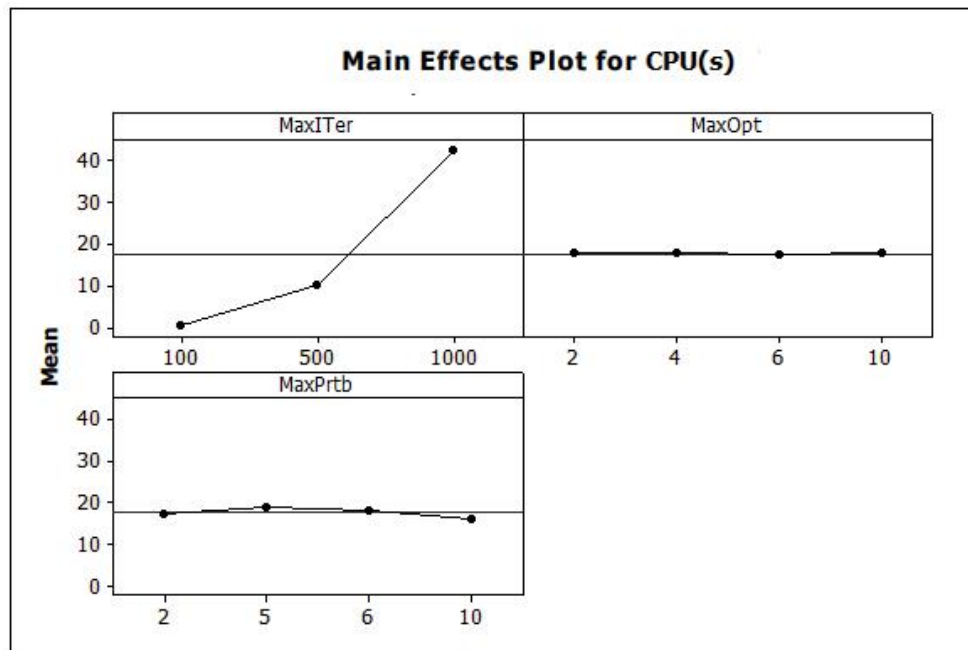
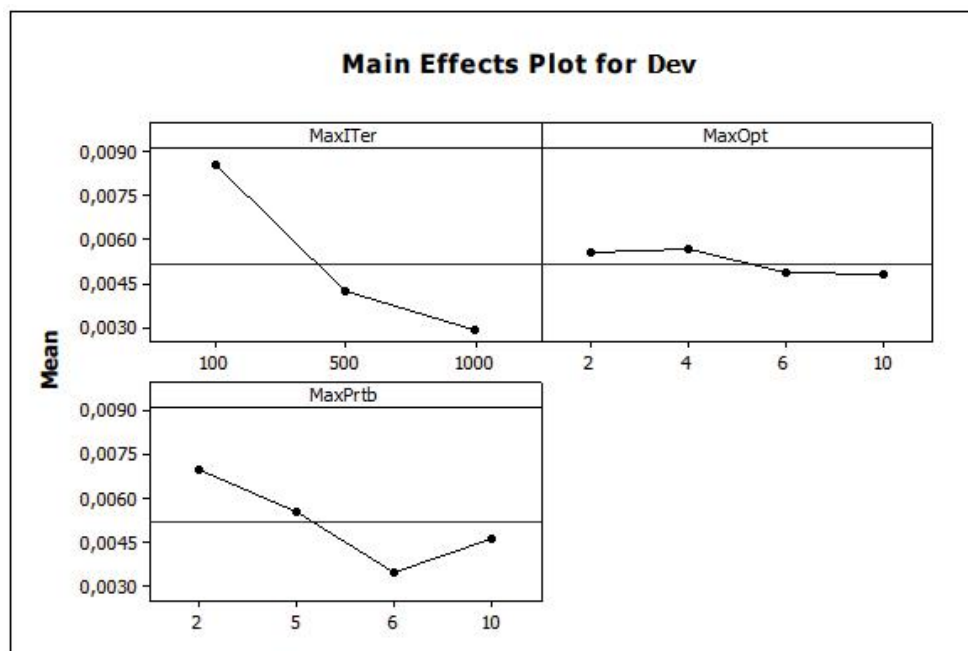


FIGURE 5.4 – Graphe des effets principaux pour *Avg*

RT. Il ordonnance les tâches l'un après l'autre selon une séquence de traitement. Le deuxième noté *H2 - RT* permet de résoudre l'ordonnancement au moyen de l'algorithme *2-jobs* appliqué aux tâches deux par deux. Nous déterminons le makespan des deux algorithmes *AL - RT* et *H2 - RT*. Le makespan gardé $C_{max}(H_Agg)$ est égale à $C_{max}(H_Agg) = \min(C_{max}(AL - RT), C_{max}(H2 - RT))$. Les tableaux 5.5 et 5.6 résument la performances

FIGURE 5.5 – Graphe des effets principaux pour $CPU(s)$ FIGURE 5.6 – Graphe des effets principaux pour Dev

des différents algorithmes y compris la performance de ceux proposés par Aggoune (2002). Ils affichent la borne inférieure de chaque instance (LB), la meilleure ($Best$), la pire (Wst) et la moyenne (Avg) et le temps de calcul moyen $CPU(s)$ des 10 exécutions de l'algorithme ILS_LT . Le makespan obtenu par la PSE limitée à 3600s (PSE_Stop) et le gap par rapport à la borne inférieure ($Gap(\%)$) si la solution obtenue n'est pas optimale. Le gap

entre la solution moyenne Avg et la solution obtenue par PSE_Stop est calculé et égal à $\frac{Avg - C_{max}}{Avg}$.

ILS_LT fonctionne mieux que H_Agg pour 29 des 30 instances lorsque $U = N/4$ et dans 25 des 30 instances pour le cas $U = N/4$. L'heuristique H_{cons} a également prouvé son efficacité par rapport à H_Agg dans 29 des 30 instances pour le cas $U = N/4$ et dans 23 des 30 instances pour le cas $U = N/4$.

Les tableaux 5.7 et 5.8 concernent la comparaison entre la méthode ILS_LT et la procédure par séparation et évaluation présenté dans le chapitre 3 pour les cas de $U = N/4$ et $U = N/2$ respectivement. Lorsque le nombre de périodes de disponibilité est égal à $U = N/4$, l'algorithme ILS_LT réduit l'écart avec la procédure arrêtée à 3600 seconds (PSE_Stop) à moins de 0.38% pour les instances de taille 15 tâches et trouve la solution optimale pour 4 instances. La première instance est résolue de façon optimale par l'heuristique H1 (règle de Jackson) car la solution obtenue est égale à la borne inférieure (cas où $C_{max} = \sum_{i=1}^N p_{i2}$, voir propriété 8). Concernant les instances de 20 tâches, l'algorithme améliore considérablement la qualité de deux instances, le temps de calcul moyen étant 5.55 secondes et dépasse de loin les résultats obtenus par PSE_Stop pour les instances de 30 tâches avec -11.68 % en moyenne par rapport à PSE_Stop (voir tableau 5.7).

Comme le montre le tableau 5.8 pour le cas de $U = N/2$, ILS_LT garantit des solutions assez proches de l'optimum pour des instances de 15 jobs . L'écart moyen avec le résultat donné par PSE_Stop est égal à 1.3% obtenu en 4.76 secondes et 3 des 10 instances sont résolues de façon optimale. L'écart moyen des solution de ILS_LT par rapport à PSE_Stop est égal à 1.9% obtenu en temps moyen égale 10.21 secondes pour les instances de taille 20 et -5.59% en 27.62 secondes en moyenne pour ceux de 30 tâches.

5.6 Conclusion

Nous avons présenté dans ce chapitre des méthodes approchées pour la résolution du problème d'ordonnancement de type job shop à deux machines avec plusieurs périodes d'indisponibilité sur une machine pour la minimisation du makespan $(J_2, h_{1U}|a|C_{max})$

Les méthodes approchées développées, à savoir une heuristique de construction H_{const} et une méta-heuristique de recherche locale itérée avec liste tabou ILS_LT ont été étudiées et comparées.

H_{const} est une heuristique gloutonne fondée sur l'utilisation de la règle de Jackson pour

Tableau 5.5 – Comparaison des différentes méthodes approchées $U = N/4$, $N = 15, 20, 30$)

N	LB	$H1$		H_const		ILS_LT					H_Agg	
		C_{max}	$CPU(s)$	C_{max}	$CPU(s)$	$Best$	Avg	Wst	Dev	$CPU(s)$	C_{max}	$CPU(s)$
15	902	902	<0.001	902	<0.001	902	902	902	0.00	<0.001	962	0.002
	819	939	<0.001	824	<0.001	820	822.2	824	0.27	3.07	878	0.001
	936	1005	<0.001	949	0.001	940	942.3	943	0.24	2.31	1080	0.002
	852	895	<0.001	855	<0.001	852	852	852	0.00	<0.001	1025	0.002
	972	1082	<0.001	988	<0.001	976	979.8	985	0.39	2.28	1048	0.002
	983	1064	<0.001	1024	<0.001	983	985.6	988	0.26	1.26	1046	0.002
	1084	1263	<0.001	1102	<0.001	1085	1086.9	1088	0.18	2.29	1198	0.002
	967	1182	<0.001	993	<0.001	972	974.2	979	0.23	2.47	1065	0.002
	1109	1235	<0.001	1205	<0.001	1114	1115.1	1117	0.10	2.39	1209	0.001
	800	911	<0.001	825	<0.001	801	802.5	804	0.19	2.10	888	0.002
20	1268	1458	<0.001	1358	0.001	1293	1309.6	1318	1.28	5.68	1408	0.003
	1178	1334	<0.001	1199	0.002	1187	1197.4	1199	0.88	5.28	1284	0.002
	1265	1425	<0.001	1347	0.002	1273	1287.1	1298	1.11	6.09	1373	0.003
	1293	1433	<0.001	1319	0.002	1301	1312.8	1319	0.91	6.83	1416	0.002
	1082	1215	<0.001	1108	0.002	1094	1098.5	1107	0.41	5.18	1236	0.002
	1332	1485	<0.001	1340	0.001	1340	1340	1340	0.00	5.16	1527	0.002
	1387	1595	<0.001	1452	0.002	1401	1411	1423	0.71	5.21	1472	0.002
	1326	1534	<0.001	1397	0.001	1340	1354.6	1362	1.09	5.06	1426	0.002
	1291	1462	<0.001	1368	0.001	1314	1325.7	1335	0.89	5.14	1379	0.003
	1433	1650	<0.001	1494	0.001	1443	1452.9	1465	0.69	5.87	1532	0.002
30	2053	2367	<0.001	2119	0.004	2089	2099	2106	0.48	12.99	2207	0.005
	1964	2355	<0.001	1986	0.004	1980	1985.4	1986	0.27	11.94	2064	0.005
	1945	2164	<0.001	1993	0.003	1989	1992.4	1993	0.17	12.01	1994	0.005
	1828	2234	<0.001	1849	0.003	1849	1849	1849	0.00	13.15	1890	0.005
	1967	2152	<0.001	2031	0.004	1996	2012.4	2021	0.82	12.82	2035	0.005
	1975	2211	<0.001	2015	0.003	2004	2013.1	2015	0.45	12.81	2065	0.005
	2184	2454	<0.001	2249	0.004	2233	2238.5	2244	0.25	14.47	2350	0.005
	1768	2020	<0.001	1815	0.005	1777	1809.1	1815	1.81	14.81	1795	0.005
	1784	2018	<0.001	1810	0.002	1810	1810	1810	0.00	15.11	1832	0.004
	1890	2183	<0.001	1944	0.003	1926	1936.5	1944	0.55	14.71	2055	0.004

Tableau 5.6 – Comparaison des différentes méthodes approchées $U = N/2$, $N = 15, 20, 30$)

N	LB	$H1$		H_const		ILS_LT					H_Agg	
		C_{max}	$CPU(s)$	C_{max}	$CPU(s)$	$Best$	Avg	Wst	Dev	$CPU(s)$	C_{max}	$CPU(s)$
15	938	1163	<0.001	1083	0.001	1054	1061.3	1076	0.69	4.57	1131	0.002
	999	1277	<0.001	1095	0.001	1095	1095	1095	0.00	4.96	1131	0.001
	1144	1463	<0.001	1348	0.002	1299	1299.6	1305	0.05	5.55	1376	0.001
	1003	1250	<0.001	1139	0.002	1092	1100.6	1105	0.79	5.16	1212	0.002
	1188	1499	<0.001	1383	0.002	1344	1359.5	1371	1.15	4.86	1471	0.002
	1199	1488	<0.001	1424	0.002	1363	1376.3	1381	0.98	4.80	1424	0.001
	1324	1643	<0.001	1564	0.001	1563	1563.7	1564	0.04	4.08	1564	0.002
	1179	1497	<0.001	1447	0.002	1386	1386	1386	0.00	4.20	1400	0.002
	1353	1740	<0.001	1665	0.002	1591	1591	1591	0.00	5.05	1591	0.001
	976	1195	<0.001	1125	0.001	1078	1088.6	1098	0.98	4.44	1125	0.001
20	1518	1862	<0.001	1739	0.005	1730	1736	1739	0.35	9.63	1779	0.003
	1413	1734	<0.001	1631	0.005	1597	1601.7	1613	0.29	10.78	1585	0.002
	1515	1921	<0.001	1777	0.004	1733	1736.4	1742	0.20	10.53	1777	0.003
	1548	1940	<0.001	1745	0.005	1745	1745	1745	0.00	11.97	1771	0.002
	1297	1656	<0.001	1505	0.004	1467	1472	1481	0.34	10.56	1461	0.003
	1597	2020	<0.001	1875	0.002	1875	1875	1875	0.00	9.31	1898	0.002
	1662	2117	<0.001	1901	0.002	1863	1874.6	1889	0.62	10.93	1901	0.002
	1591	1970	<0.001	1890	0.004	1846	1851.2	1859	0.28	9.08	1890	0.002
	1546	1969	<0.001	1834	0.003	1783	1786.3	1796	0.19	9.88	1853	0.003
	1718	2164	<0.001	2075	0.003	2075	2075	2075	0.00	9.46	2052	0.003
30	2493	3167	<0.001	2819	0.010	2819	2819	2819	0.00	26.10	2809	0.005
	2388	3050	<0.001	2799	0.012	2799	2799	2799	0.00	21.54	2910	0.006
	2361	2937	<0.001	2840	0.012	2803	2812.2	2820	0.33	17.97	2840	0.006
	2220	2817	<0.001	2586	0.009	2566	2578.7	2586	0.49	27.94	2638	0.006
	2391	2975	<0.001	2690	0.010	2690	2690	2690	0.00	34.26	2665	0.005
	2399	3060	<0.001	2759	0.012	2759	2759	2759	0.00	29.55	2810	0.006
	2656	3307	<0.001	2962	0.010	2962	2962	2962	0.00	33.42	3066	0.005
	2144	2622	<0.001	2420	0.009	2420	2420	2420	0.00	37.94	2422	0.006
	2168	2667	<0.001	2472	0.009	2464	2471.2	2472	0.29	27.95	2472	0.005
	2298	2932	<0.001	2754	0.010	2700	2723.9	2736	0.89	19.59	2816	0.005

Tableau 5.7 – Comparaison entre *ILS_LT* et *PSE_Stop* ($U = N/4$, $N = 15, 20, 30$)

N	LB	<i>ILS - LT</i>				<i>PSE_Stop</i>			$(Avg - C_{max})/Avg$ (%)
		<i>Best</i>	<i>Avg</i>	<i>Wst</i>	<i>CPU(s)</i>	C_{max}	<i>Gap</i> (%)	<i>CPU(s)</i>	
15	902	902	902	902	<0.001	902	–	–	0.00
	819	820	822.2	824	3.07	819	–	8.10	0.39
	936	940	942.3	943	2.31	937	–	419.22	0.56
	852	852	852	852	<0.001	852	–	4.77	0.00
	972	976	979.8	985	2.28	972	–	46.40	0.80
	983	983	985.6	988	1.26	983	–	12.50	0.26
	1084	1085	1086.9	1088	2.29	1085	–	18.62	0.17
	967	972	974.2	979	2.47	967	–	71.34	0.74
	1109	1114	1115.1	1117	2.39	1109	–	150.32	0.55
	800	801	802.5	804	2.10	800	–	3.70	0.31
20	1268	1293	1309.6	1318	5.68	1270	0.16	3600	3.02
	1178	1187	1197.4	1199	5.28	1180	0.17	3600	1.45
	1265	1273	1287.1	1298	6.09	1272	0.55	3600	1.17
	1293	1301	1312.8	1319	6.83	1355	4.58	3600	-3.21
	1082	1094	1098.5	1107	5.18	1083	0.09	3600	1.41
	1332	1340	1340	1340	5.16	1335	0.22	3600	0.37
	1387	1401	1411	1423	5.21	1392	0.36	3600	1.35
	1326	1340	1354.6	1362	5.06	1340	1.04	3600	1.08
	1291	1314	1325.7	1335	5.14	1292	0.08	3600	2.54
	1433	1443	1452.9	1465	5.87	1598	10.33	3600	-9.99
30	2053	2089	2099	2106	12.99	2365	13.19	3600	-12.67
	1964	1980	1985.4	1986	11.94	2354	16.57	3600	-18.57
	1945	1989	1992.4	1993	12.01	2109	7.78	3600	-5.85
	1828	1849	1849	1849	13.15	2191	16.57	3600	-18.50
	1967	1996	2012.4	2021	12.82	2152	8.60	3600	-6.94
	1975	2004	2013.1	2015	12.81	2207	10.51	3600	-9.63
	2184	2233	2238.5	2244	14.47	2435	10.31	3600	-8.78
	1768	1777	1809.1	1815	14.81	2020	12.48	3600	-11.66
	1784	1810	1810	1810	15.11	2018	11.60	3600	-11.49
	1890	1926	1936.5	1944	14.71	2183	13.42	3600	-12.73

Tableau 5.8 – Comparaison entre *ILS_LT* et *PSE_Stop* ($U = N/2$, $N = 15, 20, 30$)

N	LB	<i>ILS_LT</i>				<i>PSE</i>			$(Avg - C_{max})/Avg$ (%)
		<i>Best</i>	<i>Avg</i>	<i>Wst</i>	$CPU(s)$	C_{max}	$Gap(\%)$	$CPU(s)$	
15	938	1054	1061.3	1076	4.57	1029	–	552.989	3.04
	999	1095	1095	1095	4.96	1095	–	245.57	0.00
	1144	1299	1299.6	1305	5.55	1285	–	215.01	1.12
	1003	1092	1100.6	1105	5.16	1079	–	321.80	1.96
	1188	1344	1359.5	1371	4.86	1311	–	3.48	3.57
	1199	1363	1376.3	1381	4.80	1363	–	181.54	0.97
	1324	1563	1563.7	1564	4.08	1545	–	2.01	1.20
	1179	1386	1386	1386	4.20	1372	–	3.57	1.01
	1353	1591	1591	1591	5.05	1591	–	15.93	0.00
	976	1078	1088.6	1098	4.44	1074	–	181.27	0.34
20	1518	1730	1736	1739	9.63	1695	10.44	3600	2.36
	1413	1597	1601.7	1613	10.78	1585	10.85	3600	1.04
	1515	1733	1736.4	1742	10.53	1711	–	2385.38	1.46
	1548	1745	1745	1745	11.97	1676	–	3494.31	3.95
	1297	1467	1472	1481	10.56	1456	10.92	3600	1.09
	1597	1875	1875	1875	9.31	1832	12.83	3600	2.29
	1662	1863	1874.6	1889	10.93	1828	9.08	3600	2.49
	1591	1846	1851.2	1859	9.08	1844	13.72	3600	0.39
	1546	1783	1786.3	1796	9.88	1776	12.95	3600	0.58
	1718	2075	2075	2075	9.46	2003	–	89.42	3.47
30	2493	2819	2819	2819	26.10	3053	18.34	3600	-8.30
	2388	2799	2799	2799	21.54	2935	18.64	3600	-4.86
	2361	2803	2812.2	2820	17.97	2869	17.71	3600	-2.02
	2220	2566	2578.7	2586	27.94	2730	18.68	3600	-5.87
	2391	2690	2690	2690	34.26	2923	18.20	3600	-8.66
	2399	2759	2759	2759	29.55	2936	18.29	3600	-6.42
	2656	2962	2962	2962	33.42	3145	15.55	3600	-6.18
	2144	2420	2420	2420	37.94	2583	17.00	3600	-6.74
	2168	2464	2471.2	2472	27.95	2634	17.69	3600	-6.59
	2298	2700	2723.9	2736	19.59	2733	15.92	3600	-0.33

l'ordonnancement des tâches une par une au plus tôt dans les différents intervalles générés par les contraintes de disponibilité, garantissant ainsi que l'ordre des tâches dans chaque intervalle est selon la règle de Jackson. La méthode *ILS_LT* emploie également cette règle et se base sur une recherche locale d'insertion et d'échange entre deux tâches appartenant à deux intervalles différents. La perturbation étant une répétition successive de recherches locales est choisie comme étant un processus de diversification pour échapper tout optimum local. De plus, une liste taboue est considérée pour empêcher toute séquence déjà visitée. Nous avons déterminé les paramètres de la méta-heuristique *ILS_LT* qui assurent le compromis qualité de la solution, la stabilité de l'algorithme (avec moins d'écart entre la solution moyenne et la meilleure trouvée) et le temps de calcul dédié. Ensuite, les différentes méthodes ont été comparées à celles présentées dans la littérature. Une deuxième comparaison avec une méthode exacte stoppée à 3600 secondes est également considérée.

Les résultats expérimentaux illustrent l'efficacité de l'algorithme *ILS_LT*. Cette méthode est dédiée à la résolution approchée du problème que nous avons démontré non soluble à l'optimum dans un temps de calcul acceptable à savoir le problème avec un nombre de période d'indisponibilité égale à $N/2$ ou $N/4$ avec le nombre de tâches $N = \{15, 20, 30\}$.

Conclusion générale

Les travaux de recherches développés dans cette thèse portent sur l'ordonnancement des systèmes de production pour la maximisation de la productivité, face aux aléas et perturbations de ces systèmes.

Le problème traité est l'ordonnancement avec contraintes de disponibilité pour la minimisation de la date de fin du traitement de toutes les tâches (Makespan). Nous avons considéré, en particulier, le problème d'ordonnancement de type job shop à deux machines avec la prise en compte des contraintes de disponibilité dont les dates de début et de fin sont connues à l'avance. Nous avons considéré également tout au long de la thèse que les opérations sont strictement non-preemptives, ce qui signifie qu'une tâche en-cours de traitement ne peut être interrompue ni par l'exécution d'une autre tâche ni par le commencement d'une période d'indisponibilité. Des propriétés théoriques ont été établies et des méthodes exactes et approchées ont été développées ainsi que des études d'approximation au pire cas pour les heuristiques utilisant la règle de Jackson.

Nous avons mené tout d'abord une étude bibliographique portant sur les problèmes d'ordonnancement sous contraintes de disponibilité, après avoir présenté les notions sur l'ordonnancement, les types d'ateliers, la théorie de complexité ainsi que les principales méthodes de résolution de ce problème.

L'étude et la résolution du problème d'ordonnancement de type job shop à deux machines avec contraintes de disponibilité est dédiée d'abord au cas de plusieurs contraintes de disponibilité sur une machine. Nous avons démontré des propriétés théoriques concernant l'optimalité de l'algorithme de Jackson. Une procédure par séparation et évaluation a été développée par la suite pour la résolution exacte du problème. Cette approche tient en compte le fait que le séquençement des tâches avant et après chaque période d'indisponibilité est selon la règle de Jackson qui est optimal. Par conséquent, la procédure consiste à trouver l'ensemble de tâches à traiter dans chaque intervalle sachant le séquençement optimal de ces tâches. Des bornes supérieures issues des heuristiques ainsi que des inférieures ont été

élaborées. Les résultats obtenus montrent l'efficacité des propriétés démontrées et la méthode exacte développée. Pour les instances de grandes tailles, avec un nombre de tâches et de contraintes de disponibilité élevé, nous avons proposé des méthodes approchées, à savoir, une heuristique constructive et une recherche locale itérée pour obtenir des solutions de bonne qualité en un temps raisonnable pour les instances non résolues à l'optimum.

Le problème d'ordonnancement de type Job shop à deux machines avec contrainte de disponibilité sur chaque machine est également traité. Nous avons développé des propriétés traitant la dominance des ordonnancements de permutation. Un nouvel ordre de tâches en présence des contraintes de disponibilité a été démontré optimal. Il garantit que le séquençement des tâches selon l'ordre de Jackson avant et après la période d'indisponibilité sur chaque machine est optimal. Nous avons proposé de ce fait des procédures par séparation et évaluation et une nouvelle formulation mathématique du problème. Notre contribution dans ce sens est de proposer une méthode de résolution exacte employant au mieux les propriétés démontrées. Les résultats expérimentaux ont prouvé l'efficacité des méthodes présentées.

Il serait intéressant de pouvoir généraliser les approches développées et en particulier les propriétés théoriques pour la résolution non-déterministe et en ligne (on-line) du problème. En effet, il est envisageable de considérer l'ordonnancement robuste pour le cas non-déterministe qui permettra d'anticiper ou réviser l'ordonnancement en tenant en compte les contraintes de disponibilité ni estimées ni connues à l'avance, notamment à cause d'une panne de machine, un retard de livraison, une rupture de stock ou bien pour d'autres raisons. Il faudrait pour cela développer des méthodes d'optimisation basées sur le cas déterministe étudié dans cette thèse. Les propriétés théoriques que nous avons développées peuvent s'appliquer, à savoir l'ordre optimal des tâches avant et après chaque période d'indisponibilité.

Une deuxième perspective serait de tenir en compte la date et la durée de la période d'indisponibilité déterminées en temps réel. Il s'agira d'ordonnancer les tâches sur la base des informations retenues. Par exemple, les informations sur la périodes d'indisponibilité peuvent être obtenues selon une fonction de dégradation ou estimées en se basant sur un historique de données. Dans ce cas, l'ordonnancement d'une tâche immédiat et irrévocable devrait prendre en compte les contraintes de disponibilité sans aucune connaissance de l'ordonnancement des tâches futures. Il s'agira donc de penser à employer les propriétés développées et en particulier, l'ordre optimal des tâches démontré pour ordonnancer ou réordonnancer les tâches disponibles lorsque une indisponibilité se présente.

Bibliographie

- Adiri, Igal, Bruno, John, Frostig, Esther, & Kan, AHG Rinnooy. 1989. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, **26**(7), 679–696.
- Aggoune, Riad. 2002. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. Ph.D. thesis, Université de Metz.
- Aggoune, Riad. 2004a. Minimizing the makespan for the flow shop scheduling problem with availability constraints. *European Journal of Operational Research*, **153**(3), 534–543.
- Aggoune, Riad. 2004b. Two-Job Shop Scheduling Problems with Availability Constraints. *Pages 253–259 of : ICAPS*.
- Aggoune, Riad. 2010. An Improved Solution Algorithm for Two-Job Shop Scheduling Problems with Availability Constraints. *In : Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 3.
- Aggoune, Riad, & Portmann, Marie-Claude. 2006. Flow shop scheduling problem with limited machine availability : A heuristic approach. *International Journal of Production Economics*, **99**(1), 4–15.
- Akers Jr, Sheldon B, & Friedman, Joyce. 1955. A non-numerical approach to production scheduling problems. *Journal of the Operations Research Society of America*, **3**(4), 429–442.
- Allaoui, H, Artiba, A, Elmaghraby, SE, & Riane, F. 2006. Scheduling of a two-machine flowshop with availability constraints on the first machine. *International Journal of Production Economics*, **99**(1), 16–27.
- Allaoui, H, Lamouri, S, Artiba, A, & Aghezzaf, E. 2008. Simultaneously scheduling n jobs and the preventive maintenance on the two-machine flow shop to minimize the makespan. *International Journal of Production Economics*, **112**(1), 161–167.

- Applegate, David, & Cook, William. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, **3**(2), 149–156.
- Azem, Sadia. 2010. *Ordonnancement des systemes flexibles de production sous contraintes de disponibilite des ressources*. Ph.D. thesis, Saint-Etienne, EMSE.
- Azem, Sadia, Aggoune, Riad, & Dauzère-Pérès, Stéphane. 2007. Disjunctive and time-indexed formulations for non-preemptive job shop scheduling with resource availability constraints. *Pages 787–791 of : Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*. IEEE.
- Beaton, Clifford, Diallo, Claver, & Gunn, Eldon. 2016. Makespan minimization for parallel machine scheduling of semi-resumable and non-resumable jobs with multiple availability constraints. *INFOR : Information Systems and Operational Research*, 1–12.
- Bellman, Richard. 1954. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, **60**(6), 503–515.
- Ben Chihaoui, Faten, Kacem, Imed, Hadj-Alouane, Atidel B, Dridi, Najoua, & Rezg, Nidhal. 2011. No-wait scheduling of a two-machine flow-shop to minimise the makespan under non-availability constraints and different release dates. *International Journal of Production Research*, **49**(21), 6273–6286.
- Benmansour, Rachid, Allaoui, Hamid, Artiba, Abdelhakim, & Hanafi, Saïd. 2014. Minimizing the weighted sum of maximum earliness and maximum tardiness costs on a single machine with periodic preventive maintenance. *Computers & Operations Research*, **47**, 106–113.
- Blazewicz, Jacek, Lenstra, Jan Karel, & Kan, AHG Rinnooy. 1983. Scheduling subject to resource constraints : classification and complexity. *Discrete applied mathematics*, **5**(1), 11–24.
- Błażewicz, Jacek, Breit, Joachim, Formanowicz, Piotr, Kubiak, Wiesław, & Schmidt, Günter. 2001. Heuristic algorithms for the two-machine flowshop with limited machine availability. *Omega*, **29**(6), 599–608.
- Breit, Joachim. 2004. An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint. *Information Processing Letters*, **90**(6), 273–278.
- Breit, Joachim. 2006. A polynomial-time approximation scheme for the two-machine flow shop scheduling problem with an availability constraint. *Computers & Operations Research*, **33**(8), 2143–2153.

- Breit, Joachim, Schmidt, Günter, & Strusevich, Vitaly A. 2001. Two-machine open shop scheduling with an availability constraint. *Operations Research Letters*, **29**(2), 65–77.
- Breit, Joachim, Schmidt, Günter, & Strusevich, Vitaly A. 2003. Non-preemptive two-machine open shop scheduling with non-availability constraints. *Mathematical Methods of Operations Research*, **57**(2), 217–234.
- Carlier, Jacques, & Pinson, Éric. 1989. An algorithm for solving the job-shop problem. *Management science*, **35**(2), 164–176.
- Chen, Wen-Jinn. 2009. Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega*, **37**(3), 591–599.
- Chen, WJ. 2007. Scheduling of jobs and maintenance in a textile company. *The International Journal of Advanced Manufacturing Technology*, **31**(7-8), 737–742.
- Cheng, Chen-Yang, Ying, Kuo-Ching, Chen, Hsia-Hsiang, & Lin, Jia-Xian. 2018. Optimization algorithms for proportionate flowshop scheduling problems with variable maintenance activities. *Computers & Industrial Engineering*, **117**, 164 – 170.
- Cheng, TC Edwin, & Liu, Zhaohui. 2003. Approximability of two-machine no-wait flowshop scheduling with availability constraints. *Operations Research Letters*, **31**(4), 319–322.
- Cheng, TC Edwin, & Wang, Guoqing. 1999. Two-machine flowshop scheduling with consecutive availability constraints. *Information Processing Letters*, **71**(2), 49–54.
- Cook, Stephen A. 1971. The complexity of theorem-proving procedures. *Pages 151–158 of : Proceedings of the third annual ACM symposium on Theory of computing*. ACM.
- Costa, Antonio, Cappadonna, Fulvio Antonio, & Fichera, Sergio. 2016. Minimizing the total completion time on a parallel machine system with tool changes. *Computers & Industrial Engineering*, **91**, 290–301.
- Cui, Wei-Wei, & Lu, Zhiqiang. 2017. Minimizing the makespan on a single machine with flexible maintenances and jobs’ release dates. *Computers & Operations Research*, **80**, 11–22.
- Cui, Wei-Wei, Lu, Zhiqiang, Zhou, Binghai, Li, Chen, & Han, Xiaole. 2016. A hybrid genetic algorithm for non-permutation flow shop scheduling problems with unavailability constraints. *International Journal of Computer Integrated Manufacturing*, **29**(9), 944–961.

- Dantzig, George B, & Wolfe, Philip. 1960. Decomposition principle for linear programs. *Operations research*, **8**(1), 101–111.
- Dantzig, George Bernard, Fulkerson, Delbert R, & Johnson, Selmer M. 1954. On a linear-programming, combinatorial approach to the traveling-salesman problem. *Operations Research*, 393–410.
- Detienne, Boris. 2012. Minimizing the weighted number of late semi-resumable jobs with deterministic machine availability constraints. *IFAC Proceedings Volumes*, **45**(6), 111–116.
- Dorigo, Marco, & Gambardella, Luca Maria. 1996. A study of some properties of Ant-Q. *Pages 656–665 of : International Conference on Parallel Problem Solving from Nature*. Springer.
- Espinouse, Marie-Laure, Formanowicz, Piotr, & Penz, Bernard. 1999. Minimizing the makespan in the two-machine no-wait flow-shop with limited machine availability. *Computers & Industrial Engineering*, **37**(1), 497–500.
- Espinouse, Marie-Laure, Formanowicz, Piotr, & Penz, Bernard. 2001. Complexity results and approximation algorithms for the two machine no-wait flow-shop with limited machine availability. *Journal of the Operational Research Society*, **52**(1), 116–121.
- Fisher, Marshall L. 1976. A dual algorithm for the one-machine scheduling problem. *Mathematical programming*, **11**(1), 229–251.
- Fnaiech, N, Fitouri, C, Varnier, C, Fnaiech, F, & Zerhouni, N. 2015. A New Heuristic Method for Solving Joint Job Shop Scheduling of Production and Maintenance. *IFAC-PapersOnLine*, **48**(3), 1802–1808.
- Fnaiech, Nader, Hammami, Hayfa, Yahyaoui, Amel, Varnier, Christophe, Fnaiech, Farhat, & Zerhouni, Noureddine. 2012. New Hopfield Neural Network for joint Job Shop Scheduling of production and maintenance. *Pages 5535–5541 of : IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE.
- Gara-Ali, Ahmed, & Espinouse, Marie-Laure. 2014. Erratum to :“Simultaneously scheduling n jobs and the preventive maintenance on the two-machine flow shop to minimize the makespan”[Int. J. Prod. Econ. 112 (2008) 161–167]. *International Journal of Production Economics*, **153**, 361–363.
- Gara-Ali, Ahmed, & Espinouse, Marie-Laure. 2015. A two-machine flow-shop scheduling with a deteriorating maintenance activity on the second machine. *Pages 481–488 of :*

- Industrial Engineering and Systems Management (IESM), 2015 International Conference on.* IEEE.
- Garey, Michael R, & Johnson, David S. 1979. *Computers and intractability : A guide to the theory of NP-completeness*. W.H. Freeman and Company.
- Gawiejnowicz, Stanisław. 2007. Scheduling deteriorating jobs subject to job or machine availability constraints. *European Journal of Operational Research*, **180**(1), 472–478.
- Glover, Fred. 1989. Tabu search—part I. *ORSA Journal on computing*, **1**(3), 190–206.
- Glover, Fred. 1990. Tabu search—part II. *ORSA Journal on computing*, **2**(1), 4–32.
- Graham, Ronald L, Lawler, Eugene L, Lenstra, Jan Karel, & Kan, AHG Rinnooy. 1979. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Pages 287–326 of : Annals of discrete mathematics*, vol. 5. Elsevier.
- Guyon, Olivier, Lemaire, Pierre, Pinson, Eric, & Rivreau, David. 2014. Solving an integrated job-shop problem with human resource constraints. *Annals of Operations Research*, **213**(1), 147–171.
- Hadda, Hatem. 2010. An improved algorithm for the two machine flow shop problem with several availability constraints. *4OR*, **8**(3), 271–280.
- Hashemian, Navid, Diallo, Claver, & Vizvári, Béla. 2014. Makespan minimization for parallel machines scheduling with multiple availability constraints. *Annals of Operations Research*, **213**(1), 173–186.
- Hfaiedh, Walid, Sadfi, Chérif, Kacem, Imed, & Hadj-Alouane, Atidel. 2015. A branch-and-bound method for the single-machine scheduling problem under a non-availability constraint for maximum delivery time minimization. *Applied Mathematics and Computation*, **252**, 496–502.
- Hnaïen, Faïcel, Yalaoui, Farouk, & Mhadhbi, Ahmed. 2015. Makespan minimization on a two-machine flowshop with an availability constraint on the first machine. *International Journal of Production Economics*, **164**, 95–104.
- Holland, J. H. 1975. *Adaptation in natural and artificial systems*. Ph.D. thesis, Ann Arbor, MI : University of Michigan Press.
- Jackson, James R. 1956. An extension of Johnson’s results on job IDT scheduling. *Naval Research Logistics Quarterly*, **3**(3), 201–203.

- Ji, Min, He, Yong, & Cheng, TC Edwin. 2007. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & operations research*, **34**(6), 1764–1770.
- Johnson, Selmer Martin. 1954. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, **1**(1), 61–68.
- Kacem, Imed, & Chu, Chengbin. 2008a. Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *International Journal of Production Economics*, **112**(1), 138–150.
- Kacem, Imed, & Chu, Chengbin. 2008b. Minimizing the weighted flow time on a single machine with the resumable availability constraint : worst case of the WSPT heuristic. *International Journal of Computer Integrated Manufacturing*, **21**(4), 388–395.
- Kacem, Imed, & Paschos, Vangelis Th. 2013. Weighted completion time minimization on a single-machine with a fixed non-availability interval : Differential approximability. *Discrete Optimization*, **10**(1), 61–68.
- Kacem, Imed, Chu, Chengbin, & Souissi, Ahmed. 2008. Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers & operations research*, **35**(3), 827–844.
- Kacem, Imed, Nagih, Anass, & Seifaddini, Maryam. 2014. Maximum lateness minimization with positive tails on a single machine with an unexpected non-availability interval. *Pages 1–5 of : Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. IEEE.
- Kacem, Imed, Kellerer, Hans, & Lanuel, Yann. 2015. Approximation algorithms for maximizing the weighted number of early jobs on a single machine with non-availability intervals. *Journal of Combinatorial Optimization*, **30**(3), 403–412.
- Karp, Richard M. 1972. Reducibility among combinatorial problems. *Pages 85–103 of : Complexity of computer computations*. Springer.
- Kaspi, M, & Montreuil, B. 1988. On the scheduling of identical parallel processes with arbitrary initial processor available Times. 88–12.
- Kirkpatrick, Scott, Gelatt, C Daniel, & Vecchi, Mario P. 1983. Optimization by simulated annealing. *science*, **220**(4598), 671–680.

- Kubiak, Wiesław, Błażewicz, Jacek, Formanowicz, Piotr, Breit, Joachim, & Schmidt, Günter. 2002. Two-machine flow shops with limited machine availability. *European Journal of Operational Research*, **136**(3), 528–540.
- Kubzin, Mikhail A, & Strusevich, Vitaly A. 2005. Two-machine flow shop no-wait scheduling with machine maintenance. *JOR*, **3**(4), 303–313.
- Kubzin, Mikhail A, & Strusevich, Vitaly A. 2006. Planning machine maintenance in two-machine shop scheduling. *Operations Research*, **54**(4), 789–800.
- Kubzin, Mikhail A, Strusevich, Vitaly A, Breit, J, & Schmidt, G. 2006. Polynomial-time approximation schemes for two-machine open shop scheduling with nonavailability constraints. *Naval Research Logistics (NRL)*, **53**(1), 16–23.
- Kubzin, Mikhail A, Potts, Chris N, & Strusevich, Vitaly A. 2009. Approximation results for flow shop scheduling problems with machine availability constraints. *Computers & Operations Research*, **36**(2), 379–390.
- Labidi, M., Kooli, A., Ladhari, T., Gharbi, A., & Suryahatmaja, U. S. 2018. A Computational Study of the Two-Machine No-Wait Flow Shop Scheduling Problem Subject to Unequal Release Dates and Non-Availability Constraints. *IEEE Access*, **6**, 16294–16304.
- Lee, Chung-Yee. 1991. Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, **30**(1), 53–61.
- Lee, Chung-Yee. 1996. Machine scheduling with an availability constraint. *Journal of global optimization*, **9**(3-4), 395–416.
- Lee, Chung-Yee. 1997. Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. *Operations research letters*, **20**(3), 129–139.
- Lee, Chung-Yee. 1999. Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, **114**(2), 420–429.
- Lee, Chung-Yee, & Liman, Surya Danusaputro. 1992. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, **29**(4), 375–382.
- Lee, Chung-Yee, & Liman, Surya Danusaputro. 1993. Capacitated two-parallel machines scheduling to minimize sum of job completion times. *Discrete Applied Mathematics*, **41**(3), 211–222.

- Lee, Chung-Yee, Lei, Lei, & Pinedo, Michael. 1997. Current trends in deterministic scheduling. *Annals of Operations Research*, **70**, 1–41.
- Lee, Ju-Yong, & Kim, Yeong-Dae. 2012. Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. *Computers & Operations Research*, **39**(9), 2196–2205.
- Lee, Ju-Yong, & Kim, Yeong-Dae. 2017. Minimizing total tardiness in a two-machine flowshop scheduling problem with availability constraint on the first machine. *Computers & Industrial Engineering*, **114**, 22–30.
- Li, Debiao, Chen, Kejia, & Wang, Xiao. 2017. Properties of two-machine no-wait flow-shop scheduling with a non-resumable unavailable interval. *Journal of Industrial and Production Engineering*, **34**(3), 232–238.
- Lin, Guo-Hui, Yao, En-Yu, & He, Yong. 1998. Parallel machine scheduling to maximize the minimum load with nonsimultaneous machine available times. *Operations research letters*, **22**(2), 75–81.
- Liu, Ming, Wang, Shijin, Chu, Chengbin, & Chu, Feng. 2016. An improved exact algorithm for single-machine scheduling to minimise the number of tardy jobs with periodic maintenance. *International Journal of Production Research*, **54**(12), 3591–3602.
- Liu, Peihai, & Lu, Xiwen. 2016. Integrated production and job delivery scheduling with an availability constraint. *International Journal of Production Economics*, **176**, 1–6.
- Lorigeon, T, Billaut, JC, & Bouquard, JL. 2002. A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint. *Journal of the Operational Research Society*, **53**(11), 1239–1246.
- Low, Chinyao, Li, Rong-Kwei, & Wu, Guan-He. 2016. Minimizing Total Earliness and Tardiness for Common Due Date Single-Machine Scheduling with an Unavailability Interval. *Mathematical Problems in Engineering*, **2016**.
- Luo, Wenchang, Cheng, TC Edwin, & Ji, Min. 2015. Single-machine scheduling with a variable maintenance activity. *Computers & Industrial Engineering*, **79**, 168–174.
- Ma, Ying, Chu, Chengbin, & Zuo, Chunrong. 2010. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, **58**(2), 199–211.
- Mati, Yazid. 2010. Minimizing the makespan in the non-preemptive job-shop scheduling with limited machine availability. *Computers & Industrial Engineering*, **59**(4), 537–543.

- Mauguière, P, Bouquard, JL, & Billaut, JC. 2003a. A branch and bound algorithm for a job shop scheduling problem with availability constraints. *Pages 147–148 of : Proceedings of the sixth workshop on models and algorithms for planning and scheduling problems, MAPSP'2003*.
- Mauguière, P, Billaut, JC, & Bouquard, JL. 2003b. Scheduling resumable and non-resumable operations. *Pages 166–167 of : Proceedings of the joint international meeting EURO/INFORMS*.
- Mauguière, P, Billaut, J-C, & Bouquard, J-L. 2005. New single machine and job-shop scheduling problems with availability constraints. *Journal of scheduling*, **8**(3), 211–231.
- Mellouli, Racem, Sadfi, Cherif, Chu, Chengbin, & Kacem, Imed. 2009. Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. *European Journal of Operational Research*, **197**(3), 1150–1165.
- Moore, J Michael. 1968. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, **15**(1), 102–109.
- Mosheiov, G. 1994. Minimizing the sum of job completion times on capacitated parallel machines. *Mathematical and Computer Modelling*, **20**(6), 91–99.
- Naderi, B, Zandieh, M, & Ghomi, SMT Fatemi. 2009. Scheduling sequence-dependent setup time job shops with preventive maintenance. *The International Journal of Advanced Manufacturing Technology*, **43**(1-2), 170–181.
- Nawaz, Muhammad, Ensore Jr, E Emory, & Ham, Inyong. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, **11**(1), 91–95.
- Rapine, Christophe. 2013. Erratum to “Scheduling of a two-machine flowshop with availability constraints on the first machine”[International Journal of Production Economics 99 (2006) 16–27]. *International Journal of Production Economics*, **142**(1), 211–212.
- Roy, S, & Sussman, B. 1964. Les problèmes d’ordonnancement avec contraintes disjonctives.
- Sadfi, Chérif. 2002. *Problèmes d’ordonnancement avec minimisation des encours*. Ph.D. thesis, Grenoble, INPG.
- Sadfi, Cherif, Penz, Bernard, Rapine, Christophe, Błażewicz, Jacek, & Formanowicz, Piotr. 2005. An improved approximation algorithm for the single machine total completion time

- scheduling problem with availability constraints. *European journal of operational research*, **161**(1), 3–10.
- Schmidt, Günter. 1984. Scheduling on semi-identical processors. *Zeitschrift für Operations Research*, **28**(5), 153–162.
- Schmidt, Günter. 1988. Scheduling independent tasks with deadlines on semi-identical processors. *Journal of the Operational Research Society*, 271–277.
- Shabtay, Dvir, & Zofi, Moshe. 2018. Single machine scheduling with controllable processing times and an unavailability period to minimize the makespan. *International Journal of Production Economics*, **198**, 191 – 200.
- Sheen, Gwo-Ji, & Liao, Lu-Wen. 2007. Scheduling machine-dependent jobs to minimize lateness on machines with identical speed under availability constraints. *Computers & operations research*, **34**(8), 2266–2278.
- Souissi, Ahmed Sâadeddine. 2005. *Ordonnancement avec prise en compte des indisponibilités dépendantes et indépendantes*. Ph.D. thesis, Troyes.
- Taillard, Eric. 1993. Benchmarks for basic scheduling problems. *European journal of operational research*, **64**(2), 278–285.
- Tan, Zhiyi, Chen, Yong, & Zhang, An. 2011. Parallel machines scheduling with machine maintenance for minsum criteria. *European Journal of Operational Research*, **212**(2), 287–292.
- Tan, Zhiyi, Chen, Yong, & Zhang, An. 2013. On the exact bounds of SPT for scheduling on parallel machines with availability constraints. *International Journal of Production Economics*, **146**(1), 293–299.
- Vahedi-Nouri, Behdin, Fattahi, Parviz, Rohaninejad, Mohammad, & Tavakkoli-Moghaddam, Reza. 2013. Minimizing the total completion time on a single machine with the learning effect and multiple availability constraints. *Applied Mathematical Modelling*, **37**(5), 3126–3137.
- Vahedi-Nouri, Behdin, Fattahi, Parviz, Tavakkoli-Moghaddam, Reza, & Ramezani, Reza. 2014. A general flow shop scheduling problem with consideration of position-based learning effect and multiple availability constraints. *The International Journal of Advanced Manufacturing Technology*, **73**(5-8), 601–611.

- Wan, Long, & Yuan, Jinjiang. 2018. Single-machine scheduling with operator non-availability to minimize total weighted completion time. *Information Sciences*, **445-446**, 1 – 5.
- Wang, Li-Yan, Huang, Xue, Ji, Ping, & Feng, En-Min. 2014. Unrelated parallel-machine scheduling with deteriorating maintenance activities to minimize the total completion time. *Optimization letters*, **8**(1), 129–134.
- Wang, Shijin, & Liu, Ming. 2016. Two-machine flow shop scheduling integrated with preventive maintenance planning. *International Journal of Systems Science*, **47**(3), 672–690.
- Wang, Xiuli, & Cheng, TC. 2007a. Machine scheduling with an availability constraint and job delivery coordination. *Naval Research Logistics (NRL)*, **54**(1), 11–20.
- Wang, Xiuli, & Cheng, TC Edwin. 2007b. Heuristics for two-machine flowshop scheduling with setup times and an availability constraint. *Computers & operations research*, **34**(1), 152–162.
- Wang, Xiuli, & Cheng, TCE. 2015. A heuristic for scheduling jobs on two identical parallel machines with a machine availability constraint. *International Journal of Production Economics*, **161**, 74–82.
- Xu, Dehua, & Yang, Dar-Li. 2013. Makespan minimization for two parallel machines scheduling with a periodic availability constraint : mathematical programming model, average-case analysis, and anomalies. *Applied Mathematical Modelling*, **37**(14), 7561–7567.
- Xu, Dehua, Sun, Kaibiao, & Li, Hongxing. 2008. Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan. *Computers & operations research*, **35**(4), 1344–1349.
- Yang, Dar-Li, Hsu, Chou-Jung, & Kuo, Wen-Hung. 2008. A two-machine flowshop scheduling problem with a separated maintenance constraint. *Computers & Operations Research*, **35**(3), 876–883.
- Yang, Shan-lin, Ma, Ying, Xu, Dong-ling, & Yang, Jian-bo. 2011. Minimizing total completion time on a single machine with a flexible maintenance activity. *Computers & Operations Research*, **38**(4), 755–770.
- Yin, Yunqiang, Xu, Jianyou, Cheng, TCE, Wu, Chin-Chia, & Wang, Du-Juan. 2016. Approximation schemes for single-machine scheduling with a fixed maintenance activity to minimize the total amount of late work. *Naval Research Logistics (NRL)*, **63**(2), 172–183.

- Zhao, Chuan-Li, & Hsu, Chou-Jung. 2017. Scheduling deteriorating jobs with machine availability constraints to minimize the total completion time. *Journal of Industrial and Production Engineering*, **34**(5), 323–329.
- Zhao, Chuanli, & Tang, Hengyong. 2014. Parallel machines scheduling with deteriorating jobs and availability constraints. *Japan Journal of Industrial and Applied Mathematics*, **31**(3), 501–512.
- Zhao, Chuanli, Ji, Min, & Tang, Hengyong. 2011. Parallel-machine scheduling with an availability constraint. *Computers & Industrial Engineering*, **61**(3), 778–781.
- Zribi, Nozha, El Kamel, Abdelkader, & Borne, Pierre. 2008. Minimizing the makespan for the MPM job-shop with availability constraints. *International Journal of Production Economics*, **112**(1), 151–160.

Mourad BENTTALEB

Doctorat : Optimisation et Sûreté des Systèmes

Année 2018

Gestion de production sous incertitudes

Dans un contexte de production en perpétuelle évolution, l'entreprise est tenue à s'adapter en permanence aux fluctuations des marchés (demandes aléatoires) et aux perturbations internes de ses systèmes (pannes, absentéisme, ...). Face à ces incertitudes, la gestion de production permanente bénéficiant également de la puissance de la technologie informatique est indispensable. L'ordonnancement des ateliers de production est l'un des axes d'optimisation contribuant à l'amélioration de la productivité face aux aléas. Il s'agit dans ce contexte de définir les politiques d'ordonnancement avec la prise en compte de l'indisponibilité des ressources (gestion de la maintenance, de ressources humaines ...). Dans cette thèse, nous nous intéressons plus particulièrement aux problèmes d'ordonnancement d'atelier à chemins multiples (Job shop) à deux machines avec des ressources non disponibles en permanence. Le but de nos travaux est de proposer des méthodes de résolution exactes et approchées pour ce type de problèmes, exploitant autant que possible des propriétés théoriques pour la minimisation du makespan qui revient à maximiser la productivité.

Mots clés : recherche opérationnelle - optimisation combinatoire - ordonnancement (gestion) - programmation linéaire - algorithmes d'approximation - métaheuristiques.

Production Management Under Uncertainties

Production is always in perpetual evolution. Indeed, company needs to adapt its performance to market fluctuations (random requests) and internal perturbations within its systems (breakdowns, absenteeism, etc.). To cope with these uncertainties, permanent production management benefiting from the power of computer technology is fundamental. The scheduling of production is one of optimization's tools contributing to the improvement of productivity in front of hazards. In this context, the objective is to define scheduling policies, taking into account the unavailability of resources (management of maintenance, human resources, etc.). In this thesis, we are particularly interested in the study of the two-machine job shop scheduling problems with resources not always available. The aim of our work is to propose methods for exact and approximate resolution for this type of problem, exploiting as much as possible theoretical properties for the minimization of the total completion time (makespan) which is to maximize the productivity.

Keywords: operations research - combinatorial optimization - production scheduling - linear programming - approximation algorithms - metaheuristics.