N° d'ordre : 17/2021-C/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE
FACULTE DE MATHEMATIQUES

Thèse de Doctorat

Présentée pour l'obtention du grade de Docteur
EN : MATHEMATIQUES
Spécialité : Recherche Opérationnelle & Mathématiques Discrètes

Par
Ryma Zineb BADAOUI

Sujet

Ordonnancement préemptif avec délai de transport

Soutenue publiquement le 17/ 07/2021, devant le jury composé de :

M.BELBACHIR Hacène, Professeur, à l'U.S.T.H.B	Président
M.BOUDHAR Mourad, Professeur, à l'U.S.T.H.B	Directeur de Thèse
M.DAHANE Mohammed, Maitre de conférences Habilité, U. Lorraine, Metz	Co-Directeur de Thèse
M.AIT ZAI Abdelhakim, Professeur, USTHB,	Examinateur
M.BOUTICHE Mohamed Amine, Maitre de conférences A, USTHB,	Examinateur
M.BERRICHI Ali, Professeur, UMB Boumerdès,	Examinateur
Mme.HANED Amina, Maitre de conférences B, UISC. Alger 3,	Invitée

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse,
Pr. Mourad BOUDHAR de m'avoir guidé durant toutes ces années. Je le remercie pour sa patience et sa disponibilité. Je le remercie pour ses orientations et ses remarques pertinentes ayant permis l'aboutissement de ce travail.

M. Mohammed DAHANE, Maitre de Conférences HDR à l'université de Lorraine, par qui j'ai eu la chance d'être, très chaleureusement, accueillie . Je tiens à lui exprimer mes sincères remerciements et ma profonde gratitude. Je le remercie pour sa gentillesse, son incommensurables générosité, sa disponibilité sans faille, son écoute et son soutien, sans oublier tous les moyens techniques et scientifiques nécessaires au succés de mon passage à l'ENIM, qu'il a mis à ma disposition. Je le remercie également pour ses inlassables lectures et relectures, pour toutes ses corrections et orientations si précieuses, pour tous ses conseils lors de nos échanges électroniques. Une personne qui restera, pour moi, uns modèle pour la suite de ma carrière incha ALLAH.

Merci également à M. le professeur Hacène BELBACHIR d'avoir bien voulu présider le jury de cette thèse. Merci à M. les professeurs Abdelhakim AITZAI, Ali BERRICHI et Mohamed Amine BOUTICHE d'avoir accepté d'être examinateurs de ce modeste travail. Je remercie aussi Mme. Amina Haned d'avoir accepté de faire partie de ce jury.

Je remercie également, tous mes enseignants de la faculté de mathématiques de l'USTHB et particulièrement, M. Mohammed YAGOUNI auprès de qui j'ai énormément appris. Je le remercie pour toute sa bienveillance à mon égard, pour tout le temps qu'il m'a consacré, pour toute l'aide qu'il m'a apporté tant sur le plan scientifique que sur le plan humain. Merci d'avoir tant cru en moi durant tous mes moments difficiles, d'avoir toujours eu les paroles qui m'ont permis de surmonter pleins d'obstacles. Je le remercie pour tous ses conseils et ses encouragements qui m'ont accompagné depuis mon master. Qu'il trouve ici l'expression de mon plus profond respect.

Merci à M. Larbi BENAISSA ainsi qu'à M. Noureddine HANNOUN pour leurs précieux conseils concernant la langue anglaise.

Je voudrais aussi exprimer ma profonde gratitude, d'abord et avant tout, envers ma maman, puis envers toute ma famille, mes amies et tous ceux qui me sont chers pour leur soutien. Que toute personne ayant contribué de prés ou de loin à l'aboutissement de ce travail trouve ici l'expression de mes remerciements les plus sincères. Enfin, J'ai une pensée toute émue et toute chaleureuse pour mon papa, qui n'est plus parmi nous (quoi que...), et à qui je dédie mon travail. *Tu n'es plus là où tu étais, mais tu es partout là où je suis.*

Résumé

Ce travail de doctorat aborde le problème d'ordonnancement préemptif d'un ensemble de n tâches indépendantes sur m machines identiques en vue de minimiser le critère du makespan. La contrainte imposée est relative au temps nécessaire quant au déplacement des tâches préemptées d'une machine à une autre. Nous présentons un programme dynamique ainsi qu'un algorithme exact destinés à résoudre le problème sur trois machines en considérant des délais de transport identiques. Nous montrons par la suite, que ce même cas admet un schéma d'approximation complètement polynomial. Pour le problème général sur m machines en considérant des délais de transport variables, nous proposons d'abord une nouvelle formulation mathématique linéaire en variables mixtes, puis une stratégie de résolution approchée reposant sur plusieurs méthodes de résolution de type heuristique et métaheuristique. L'idée principale est de déterminer, par une recherche locale, une séquence de machines selon laquelle tous les éventuels transports de tâches préemptées vont être effectués. La seconde étape de la stratégie consiste à déterminer la meilleure affectation possible des tâches aux machines de telle sorte à minimiser le makespan. Cette étape est concrétisée par l'adaptation de deux métaheuristiques, à savoir le Recuit Simulé et la Recherche à Voisinage Variable.

Mots-clés : Ordonnancement ; machines identiques ; préemption ; délais de transport ; programme linéaire ; métaheuristique.

Abstract

This doctoral work adresses the preemptive scheduling problem of independent jobs on identical machines. The aim is to minimize the makespan under the imposed constraints, namely the ones that relate the transportation delays which are required to transport a preempted job from one machine to another. We present a dynamic program and an exact algorithm for solving the three machines problem when the transport parameter is fixed. Also, for the same problem we show that there is a fully polynomial time approximation scheme (FPTAS). For the general problem, the contribution is twofold. First, we propose a new linear programming formulation in real and binary decision variables. Then, we propose and implement a solution strategy, which consists of two stages. The goal of the first stage is to obtain the best machines order using a Local Search strategy. For the second stage, the objective is to determine the best possible sequence of jobs. To solve the preemptive scheduling problem with transportation delays, we propose a heuristic as well as adaptations of two metaheuristics (Simulated Annealing and Variable Neighborhood Search), each one with two modes of evaluation.

Keywords : Scheduling ; identical machines ; preemption ; transportation delays ; linear programming ; metaheuristic.

Table des matières

Table des matières **vii**

Liste des tableaux **ix**

Table des figures **xi**

Introduction générale **1**

1 Concepts et définitions de base **6**
1.1 Introduction 7
1.2 Généralités et définitions 7
1.2.1 Les tâches 9
1.2.2 Les ressources 11
1.2.3 Les contraintes 14
1.2.4 L'objectif 14
1.2.5 Notation des problèmes d'ordonnancement 16
1.2.6 Représentation d'un ordonnancement 18
1.3 Notions sur la complexité des algorithmes et des problèmes 21
1.3.1 L'algorithme 21
1.3.2 Complexité algorithmique 22
1.3.3 Complexité des problèmes 25
1.4 Méthodes de résolution 27
1.4.1 Les méthodes exactes 28

1.4.2 Les méthodes approchées 30
1.4.3 Les algorithmes d'approximation 37

2 Position du problème et état de l'art 40
2.1 Introduction 40
2.2 Description mathématique du problème 41
2.3 État de l'art 43
2.3.1 Problèmes sans délais de transport 43
2.3.2 Problèmes avec délais de transport 51

3 Le cas de trois machines 55
3.1 Introduction 55
3.2 Programme dynamique 57
3.3 Résolution pseudo-polynomiale 59
3.4 Schéma d'approximation complètement polynomial 64
3.5 Expérimentation numérique 69

4 Modélisation mathématique 75
4.1 Introduction 75
4.2 Le modèle mathématique 78
4.3 Détails du modèle mathématique 78
4.4 Expérimentation et résultats numériques 84

5 Méthodes Approchées pour le problème à m machines 87
5.1 Introduction 87
5.2 Recherche locale pour déterminer la séquence de machines . 89
5.3 Évaluation des ordonnancements 91
5.4 L'heuristique H-*conj* pour déterminer une séquence de tâches 93
5.5 Le Recuit Simulé (RS) pour déterminer une séquence de tâches 95
5.6 La Recherche à Voisinage Variable pour déterminer une séquence de tâches 99
5.7 Expérimentation numérique 103

Conclusion générale 111

Annexe 115

Bibliographie **137**

Liste des tableaux

1.1 Désignations des champs $\alpha_1\alpha_2$. 17

2.1 Durées opératoires 42
2.2 Délais de transport 42

3.1 Résultats des mesures rapportées pour les instances dont les $p_i \in [1, 50]$ 70
3.2 Résultats des mesures rapportées pour les instances dont les $p_i \in [10, 50]$ 70
3.3 Résultats des mesures rapportées pour les instances dont les $p_i \in [20, 50]$ 71

4.1 Contre-exemple proposé par Shams & Salmasi (2014) 76
4.2 Instance particulière dont le nombre de migrations est égal à $m - 1$. 82
4.3 Valeurs de C_{max} calculées pour différentes valeurs de r. . . . 83
4.4 Résultats de l'exécution du modèle par ILOG CPLEX 12.6 . 85

5.1 Temps de calcul moyen et nombre de solutions optimales pour les instances avec $p_j \in [1, 100]$ 106
5.2 Déviation moyenne et maximum pour les instances avec $p_j \in [1, 100]$. 106

5.3 Temps de calcul moyen et nombre de solutions optimales pour les instances avec $p_j \in [50, 100]$ 110
5.4 Déviation moyenne et maximum pour les instances avec $p_j \in [50, 100]$. 111
5 Durées opératoires des tâches. 116
6 Valeurs de la fonction F calculées pour $P_1 = 0$. 117
7 Valeurs de la fonction F calculées pour $P_1 = 1$. 117
8 Valeurs de la fonction F calculées pour $P_1 = 2$. 117
9 Valeurs de la fonction F calculées pour $P_1 = 3$. 118
10 Valeurs de la fonction F calculées pour $P_1 = 4$. 118
11 Valeurs de la fonction F calculées pour $P_1 = 5$. 118
12 Valeurs de la fonction F calculées pour $P_1 = 6$. 119
13 Valeurs de la fonction F calculées pour $P_1 = 7$. 119
14 Valeurs de la fonction F calculées pour $P_1 = 8$. 119
15 Valeurs de la fonction F calculées pour $P_1 = 9$. 120
16 Valeurs de la fonction F calculées pour $P_1 = 10$. 120
17 Valeurs de la fonction F calculées pour $P_1 = 11$. 120
18 Valeurs de la fonction F calculées pour $P_1 = 12$. 121
19 Détails d'ordonnancements obtenus. 121

Table des figures

1.1 Représentation des caractéristiques d'une tâche 10
1.2 Fonctionnement de quelques ateliers 13

2.1 Ordonnancement réalisable pour le $P3|pmtn|C_{max}$ 42
2.2 Ordonnancement réalisable pour le $P3|pmtn|C_{max}$ 43

3.1 Cas 1 : une tâche T_j est ignorée 60
3.2 Cas 2 : deux tâches T_j, T'_j sont ignorées 60
3.3 Représentation du principe des algorithmes d'approximation 65
3.4 Comparaison des temps de calcul pour les différentes exécutions d'algorithmes 72

4.1 Ordonnancement obtenu par le modèle de Boudhar & Haned (2009). 76
4.2 Ordonnancement optimal. 76
4.3 Ordonnancement optimal d'une instance particulière 82
4.4 Représentation du C_{max} en fonction de r. 83
4.5 Répartitions des valeurs de r dans les ordonnancements optimaux. 86

5.1 Variation des énergies d'un système par un Recuit Simulé. . 96
5.2 Représentations des temps d'exécution moyens (en secondes)108
5.3 Représentations des déviations moyennes (%). 109

4 Ordonnancement optimal de longueur 12 obtenu par DP pour l'instance de la table 5 123

Introduction générale

On dit que l'optimisation est le facteur essentiel permettant d'atteindre l'excellence tant recherchée par l'Homme instinctivement. À travers les âges, cet instinct ressenti par l'être humain s'est toujours traduit par la quête du meilleur, voire de l'excellent dans tous les domaines touchant à son évolution. Par la suite, l'être humain ne s'est plus contenté de trouver des solutions à ses problèmes primaires, mais s'est également tourné vers un autre objectif, celui d'améliorer ses acquis manifestant une volonté continue d'employer tous les moyens existants lui permettant de se mesurer aux prouesses de ses semblables, et de viser sans cesse cette supériorité et cette distinction qui mène vers un gain non négligeable, notamment dans le monde actuel, un monde de plus en plus compétitif.

A notre sens, ce principe traduit l'essence même et la motivation principale de la notion d'optimisation étudiée en Recherche Opérationnelle. Cette discipline s'est vu regrouper une panoplie de problèmes de différents domaines ainsi que l'ensemble des outils intervenant dans la résolution de ses problématiques.

La Recherche Opérationnelle est apparue vers la fin des années quarante au temps de la seconde guerre mondiale. Sa naissance revient à la collaboration entre scientifiques et militaires britanniques qui souffraient d'une sévère politique de désarmement infligée par l'ennemi allemand.

L'état britannique disposait de nouvelles inventions en matière de radars et d'antennes de signalisation dont il souhaitait en tirer tout le profit possible pour protéger son pays.

Bien qu'à travers ce petit historique, la Recherche Opérationnelle est considérée comme étant une science relativement récente, il n'en demeure pas moins qu'elle se soit avérée être à l'origine de l'étude de problèmes bien plus anciens, et s'est vue entreprendre par des savants d'époques lointaines à l'instar de Newton, Leibniz, Bernoulli ou encore Charles Babbage [1] considéré, par beaucoup, comme étant le père de la Recherche Opérationnelle.

Une notion très souvent évoquée en Recherche opérationnelle est celle de l'optimisation combinatoire. Ce volet regroupe l'ensemble des problèmes auxquels il est question de trouver, pour un critère donné, de bonnes solutions, voire, les meilleures solutions possibles. Actuellement, outre les problèmes purement mathématiques, l'optimisation combinatoire contribue de manière considérable dans différents domaines tels que l'industrie, la production, la gestion, le transport ou encore dans le domaine médical, d'où son inexorable utilisation pour la prise de décision et ce dans tous les domaines, qui de nos jours, sont aussi concurrentiels les uns que les autres.

Parmi les problèmes d'optimisation combinatoire, nous nous intéressons à un des problèmes d'ordonnancement de tâches. Ces derniers représentent une partie importante des sujets étudiés en Recherche Opérationnelle, ceci étant dû à leurs multiples champs d'application. En effet, les problèmes d'ordonnancement sont représentatifs d'un bon nombre de situations problématiques qui relèvent de différents secteurs notamment le secteur industriel à travers la gestion de la production et l'ensemble

1. Mathématicien anglais (1792- 1871) dont le penchant scientifique s'est illustré par l'application des mathématiques dans différentes problématiques rencontrées dans l'industrie, l'économie ou encore le transport. Il est considéré comme étant le premier informaticien de l'histoire grâce à ses idées novatrices comme celle d'avoir imaginé puis conçu ses deux fameuses machines de calcul. Il s'agit de la différentielle puis l'analytique qui est considérée, de nos jours, comme étant l'ancêtre de l'ordinateur en termes de concepts, de logique et de principes mathématiques. Babbage avait passé une bonne partie de sa vie à concevoir dans le moindre détail sa machine de calcul qui n'avait malheureusement pas fonctionné de son vivant.

des ressources matérielles et humaines nécessaires. Ils interviennent également en informatique lors du partage de la mémoire ou encore lors de la répartition des traitements par les processeurs. Ces problèmes d'ordonnancement sont aussi importants pour la gestion de tout type de projets afin de mener à terme l'ensemble des objectifs en assurant un profit maximum.

Nous pouvons donc définir un problème d'ordonnancement lorsque l'on dispose d'un ensemble de tâches à effectuer par des ressources de même ou de différentes natures. Il faut donc décider de quelle tâche sera exécutée par quelle ressource et quand cela se fera-t-il, en précisant la date de début de traitement de chacune des tâches. Cette affectation doit être la meilleure possible (ou la plus proche de l'être) pour un critère d'optimisation fixé dès le départ.

Ainsi, cette thèse s'inscrit dans le domaine de l'optimisation combinatoire. Nous nous intéressons particulièrement à un problème d'ordonnancement de tâches indépendantes sur des ressources de type machines parallèles identiques, avec comme objectif la minimisation de la durée totale de l'ordonnancement. D'autres contraintes sont imposées au problème, la plus importante et qui représente le sujet principal de notre thèse concerne l'existence de durées de transport à respecter par les tâches lorsque le traitement de ces dernières est interrompu puis repris ultérieurement sur une autre machine.

Dans ce même contexte, et contrairement à ce qui se passe réellement, il a été très souvent considéré que le traitement d'une tâche interrompue et évacuée vers une autre machine est repris à l'instant même de son interruption. Cette situation n'étant pas très représentative de la réalité, car ceci ne tient nullement compte des caractéristiques de l'environnement dans lequel sont exécutées ces tâches. En effet, il se peut que la distance entre les deux machines ne puisse permettre une telle possibilité. Par ailleurs, la taille de la tâche peut également nécessiter un opérateur ou un autre dispositif de chargement et/ou de transport et dont la disponibilité est variable. Ces quelques cas de figures montrent qu'il est dans beaucoup de cas indispensable de respecter un certain délai avant la reprise, sur une autre machine, d'une tâche interrompue. Ces délais sont imposés par une contrainte qui fait l'objet d'étude de notre thèse pour le cas de problèmes

d'ordonnancement sur machines parallèles.

L'objectif de cette thèse est de concevoir puis de mettre en oeuvre des algorithmes en vue d'apporter une contribution à la résolution d'un problème d'optimisation combinatoire. Il s'agit précisément d'un problème d'ordonnancement de tâches sur des machines parallèles en présence de contraintes relatives aux délais de transport. Les contraintes imposées, et énoncées précédemment, représentent des délais de transport entre les machines qui dépendent essentiellement de la distance entre celles-ci. Dans un premier temps, nous nous sommes intéressés à l'étude d'un cas particulier (celui où l'on considère trois machines) en s'inspirant de travaux précédents cités dans la littérature (travaux sur deux machines), et cela moyennant des méthodes de résolution exactes. Par la suite, nous nous sommes penchés sur la modélisation du problème et ce, suite à plusieurs modélisations proposées mais discutées auparavant. Après cela, nous avons achevé notre étude par la proposition de quelques méthodes de résolution pour tenter de pallier le manque que nous avons constaté dans la littérature, du moins à notre connaissance, en termes de méthodes de résolution.

Organisation de la thèse

Le manuscrit est organisé de la manière suivante :

Le premier chapitre est dédié à la présentation de généralités sur la théorie d'ordonnancement. On y retrouve en premier des concepts de base constituant l'environnement d'un problème d'ordonnancement, puis des notions générales sur la complexité et la classification des problèmes d'optimisation combinatoire. Par la suite, nous abordons les deux classes de méthodes de résolutions existantes, puis nous proposons une description des méthodes les plus utilisées de chaque classe.

Le second chapitre débute avec la description mathématique de l'objet de cette thèse, à savoir le problème d'ordonnancement préemptif sur machines parallèles identiques en présence de délais de transport. Un exemple illustratif et explicatif de la problématique est également présenté. Plus

loin, nous exposons, sous forme d'état de l'art, tous les travaux rencontrés lors de notre phase de recherche bibliographique qui présentent un rapport direct ou indirect avec le problème étudié.

Le troisième chapitre aborde l'étude d'un cas particulier, celui de trois machines. Ce chapitre représente une généralisation d'un cas précédemment étudié sur deux machines. Nous présentons le déroulement d'un programme dynamique qui sera sollicité par la suite lors de l'exécution d'un algorithme pseudo-polynomial conçu pour la résolution exacte du cas cité. Par la suite, nous montrons qu'il existe pour ce cas particulier comportant trois machines, un algorithme d'approximation complètement polynomial.

Dans le quatrième chapitre, nous proposons une nouvelle formulation mathématique du problème étudié sous la forme d'un modèle mathématique linéaire mixte. Nous présentons également, les motivations ayant mené à cette nouvelle formulation ainsi que le principe de son exécution. Les résultats des instances générées sont présentés et discutés à la fin de ce dernier.

Le dernier chapitre est dédié à la résolution par des méthodes approchées du problème d'ordonnancement considéré dans cette thèse. Nous présentons les méthodes utilisées en l'occurrence, une adaptation des métaheuristiques suivantes : le recuit simulé et la recherche à voisinage variable. Les résultats des multiples exécutions de ces adaptations sont rapportés en fin de chapitre pour montrer leurs performances.

Le manuscrit se termine par des conclusions et quelques perspectives.

1

Concepts et définitions de base

Sommaire

1.1 Introduction **7**
1.2 Généralités et définitions **7**
1.2.1 Les tâches 9
1.2.2 Les ressources 11
1.2.3 Les contraintes 14
1.2.4 L'objectif 14
1.2.5 Notation des problèmes d'ordonnancement . . . 16
1.2.6 Représentation d'un ordonnancement 18
1.3 Notions sur la complexité des algorithmes et des problèmes **21**
1.3.1 L'algorithme 21
1.3.2 Complexité algorithmique 22
1.3.3 Complexité des problèmes 25
1.4 Méthodes de résolution **27**
1.4.1 Les méthodes exactes 28
1.4.2 Les méthodes approchées 30
1.4.3 Les algorithmes d'approximation 37

1.1 Introduction

Ce chapitre constitue un rappel de quelques notions de base qui interviennent en théorie d'ordonnancement. Nous définissons en premier lieu les différentes entités qui permettent de constituer et de définir un problème d'ordonnancement. Nous présentons par la suite, le concept de complexité lié aux problèmes d'optimisation en général et aux problèmes d'ordonnancement. Ce premier chapitre s'achève par un aperçu de quelques approches de résolution développées et mises en place au service de la résolution de tels problèmes.

En raison du grand nombre de problèmes qu'ils modélisent, ainsi que de la diversité de leurs champs d'application, les problèmes d'ordonnancement demeurent largement étudiés, aussi bien par les chercheurs de différentes disciplines (mathématiques, informatique, génie industriel, ...), que par les professionnels d'entreprises activant dans les secteurs de l'industrie et de la manufacture de manière générale. À travers les années et jusqu'à l'heure actuelle, en dépit de la nature– très souvent– ardue de ces problèmes, les travaux scientifiques réalisés et dédiés à leur résolution demeurent d'actualité. Cette réflexion peut facilement être justifiée par le nombre impressionnant de références scientifiques ayant abordé des problématiques d'ordonnancement. À titre d'exemple, nous citons[1] les ouvrages suivants : (Baker, 1974; Rinnoy Kan, 1976; French, 1982; Carlier *et al.*, 1988; Tanaev *et al.*, 1994; Lenstra & Shmoys, 1995; Chu & Proth, 1996; Esquirol & Lopez, 1999; Pinedo, 2000; Lopez & Roubelat, 2000; Conway *et al.*, 2003; Leung, 2004; Tsoukiàs, 2006; Pinedo, 2012; Blazewicz *et al.*, 2013).

1.2 Généralités et définitions

Comme mentionné en introduction générale, un problème d'ordonnancement consiste donc à affecter des tâches à des machines pouvant les exécuter dans l'optique d'optimiser un critère fixé. En littérature, nous pouvons retrouver plusieurs citations qui permettent de définir un pro-

1. Bien évidemment, cette liste d'ouvrages est loin d'être exhaustive

blème d'ordonnancement. Nous choisissons de rapporter, ci-dessous, celle qui revient le plus souvent, à savoir la définition donnée par J. Carlier et P. Chrétienne dans Carlier *et al.* (1988).

> *Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.[...]. Enfin, il faut programmer les tâches de façon à optimiser un certain objectif...*
>
> Problèmes d'ordonnancement, *1988*

Plus formellement, un problème d'ordonnancement peut se définir comme suit :

Étant donné un ensemble de tâches à réaliser et un ensemble de ressources (humaines ou matérielles) permettant leurs réalisations, le problème consiste donc à organiser (ou ordonnancer) la réalisation[1] de ces tâches par les ressources précédentes en précisant dans le temps le début de traitement de chaque tâche de telle sorte à ce que toutes les tâches soient ordonnancées et que toutes les contraintes soient respectées, le tout en optimisant l'objectif considéré par le problème en question[2]. Cet objectif est exprimé sous la forme d'une fonction mathématique.

Ainsi, résoudre un problème d'ordonnancement se fait en déterminant l'affectation de chaque tâche (éventuellement, de chaque partie de tâche) à la machine (ou aux machines) pouvant la (ou les) réaliser. Par conséquent, il est possible de déterminer la date de début de chaque tâche sur chaque machine (si celle-ci y est exécutée).

1. À noter que la définition de la notion de *réalisation d'une tâche* dépend directement de l'environnement du problème d'ordonnancement en question. Par exemple, il peut s'agir d'un programme à exécuter par des processeurs dans le domaine informatique, ou encore d'une activité à accomplir lorsque nous nous situons dans le domaine de gestion de projet et tant d'autres exemples.
2. Il est tout à fait possible de considérer plusieurs objectifs (critères) à optimiser.

Ainsi, à partir de la définition précédente nous pouvons extraire les quatre éléments fondamentaux qui composent un problème d'ordonnancement, à savoir les tâches, les ressources, les contraintes imposées et l'objectif à optimiser. Dans la suite de cette section, nous présentons quelques définitions pour les notions citées.

1.2.1 Les tâches

La tâche est une entité soit élémentaire, soit formée à partir de plusieurs opérations représentant une activité ou un travail à réaliser. Notée T_j, une tâche est caractérisée par un temps de traitement, appelé aussi durée opératoire ou *processing time* en anglais. Cette durée que l'on note p_j, exprime le nombre d'unités de temps nécessaires à la réalisation de la tâche T_j.

De plus, une tâche peut également être caractérisée par une date de disponibilité (*release date* en anglais), notée r_j, avant laquelle le traitement de T_j ne peut commencer, et une date échue (*due date* en anglais) avant laquelle la tâche T_j doit finir son traitement. Cette date, que l'on note d_j, est également considérée comme étant une date de fin de traitement **au plus tard** ou encore **souhaité** et tout dépassement de cette date encourt une pénalité. Tout de même, le dépassement peut ne pas être toléré lorsqu'est définie la notion de date de fin de traitement **impérative** (*deadline* en anglais) notée $\tilde{d}_j$. Dans certains cas, une valeur w_j est aussi associée à la tâche T_j, qui indique un poids (*weight*) ou une mesure d'importance pour cette tâche par rapport aux autres tâches.

Suite à la résolution du problème d'ordonnancement, toute tâche T_j se verra attribuer de nouvelles valeurs à partir de l'ordonnancement[1] résultant. Il s'agit des paramètres t_j et c_j indiquant respectivement la date effective de début de traitement (*starting time*), et la date de fin de traitement (*completion time*) de la tâche T_j.

D'autres valeurs peuvent être définies pour chaque T_j à l'exemple de la valeur du retard algébrique $L_j = c_j - d_j$ et qui représente l'écart par rap-

1. le terme ordonnancement (ou *schedule* en anglais) est également attribué à la solution obtenue pour un problème d'ordonnancement.

port à la fin souhaité (*lateness*)[1]. Lorsque cette valeur est positive, la tâche T_j est considérée comme étant en retard d'une quantité notée D_j et peut être définie comme suit $D_j = \max\{L_j, 0\}$. Par ailleurs, si $L_j \leq 0$, la tâche peut être considérée comme étant en avance d'une valeur notée E_j, et peut être définie par $E_j = \max\{d_j - c_j, 0\}$. Toutes ces notions sont représentées dans la figure 1.1.

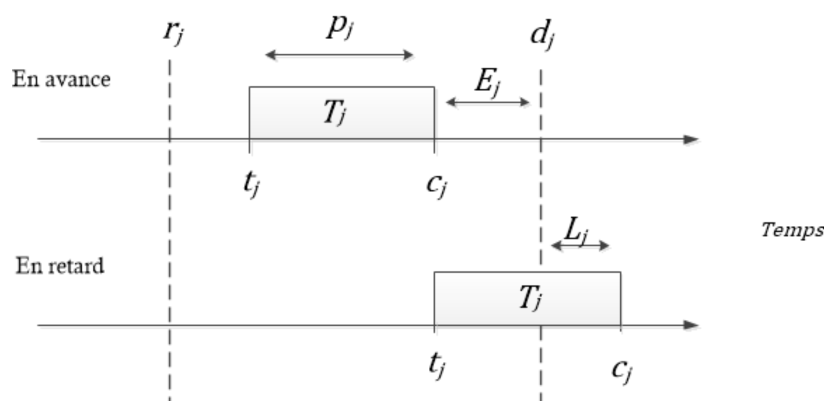


FIGURE 1.1 – Représentation des caractéristiques d'une tâche

D'autres caractéristiques d'un problème d'ordonnancement relatives aux tâches existent dans la réalité. Par exemple, lorsque le traitement d'une tâche doit être réalisé sans que celle-ci ne soit interrompue, la tâche est dite *non préemptive* et doit par conséquent être réalisée en une seule fois. Dans le cas opposé, c'est-à-dire lorsque le traitement d'une tâche peut s'interrompre à tout moment pour être repris ultérieurement, celle-ci est dite *préemptive* ou encore *morcelable*.

Aussi, lorsque les données d'un problème d'ordonnancement, à savoir les tâches et leurs durées opératoires, sont connues à priori et ne subissent aucun changement lors de leurs traitements, le problème d'ordonnancement est dit *déterministe*. En revanche, lorsque les données ne sont pas connues à l'avance ou que des changements surviennent durant la phase de traitement de ces tâches (une détérioration impliquant le changement

1. En réalité, bien que la date de fin de traitement au plus tard ait été dépassée, la tâche en question est autorisée à rester sur la machine.

des entités par exemple), le problème d'ordonnancement est dit *non déterministe*. De nos jours, Ce cas de figure est très fréquent en pratique, ce qui a conduit à l'émergence d'un nouvel axe de recherche comportant la prise en charge de l'aspect incertain de ces problèmes.

1.2.2 Les ressources

Une ressource est un moyen, matériel ou humain, destiné à réaliser une tâche. La capacité ainsi que la disponibilité de chaque ressource, qu'elles soient limitées ou non, sont connues à l'avance. Selon sa nature, une ressource est dite *renouvelable* lorsqu'elle redevient disponible après la réalisation de la tâche qui lui est allouée, elle peut donc être réutilisée aussitôt comme c'est le cas des ressources représentées par des machines ou du personnel. Dans ce cas, la ressource est dite *disjonctive* lorsqu'elle ne peut réaliser qu'une tâche à la fois. Autrement, elle est qualifiée de *cumulative* (comme par exemple, le cas d'un groupe d'ouvrier).

Par ailleurs, une ressource est dite *consommable* s'il arrive qu'elle soit épuisée après avoir terminé l'exécution d'une tâche. Ici, nous pouvons citer l'exemple des ressources qui sont en réalité des sommes d'argent ou de la matière première. Dans ce cas, une politique de réapprovisionnement efficace doit accompagner le travail d'ordonnancement.

La combinaison de ces deux types de ressources donne lieu à de nouvelles ressources, il s'agit de ressources *doublement limitées* rencontrées dans la thèse de Artigues (1997), et dans laquelle l'auteur met l'accent sur ce qu'il qualifie de préparation des ressources. Dans cette étude, l'auteur cite le cas d'une pompe à carburant qui est à la fois renouvelable (la pompe) et consommable (le carburant).

Un autre type de ressources existe pour les problèmes d'ordonnancement dits **multi-ressources** dans lesquels plusieurs ressources sont nécessaires simultanément à la réalisation d'une tâche. Autrement, ces problèmes sont appelés **mono-ressource** (Ben Hmida, 2009).

Dans le cadre de notre thèse, nous nous intéressons aux ressources du

premier type cité, c'est-à-dire des ressources renouvelables et disjonctives. Les ressources disponibles sont plus exactement des machines, et dans ce cas les problèmes d'ordonnancement moyennant de telles ressources sont également appelés *problèmes d'ordonnancement d'atelier*. Dans cette catégorie de problèmes, nous distinguons deux types de machines qui sont les suivantes :

- **Les machines parallèles** : qui sont considérées comme une généralisation des problèmes d'ordonnancement à une seule machine. Ces ateliers sont formés de machines caractérisées par le fait que toutes les tâches, constituées chacune d'une seule opération, peuvent être exécutées par n'importe quelle machine. Selon les caractéristiques de ces ressources, trois types de machines parallèles existent, il s'agit des :
  - **Machines identiques**, dans lesquelles les durées opératoires des tâches ne dépendent pas des machines qui les exécutent, ce qui signifie que la vitesse d'exécution d'une tâche est la même sur toutes les machines.
  - **Machines uniformes**, dans lesquelles chaque machine a une vitesse propre à elle et indépendante des tâches.
  - **Machines générales**, dans lesquelles chaque machine a une vitesse d'exécution pour chaque tâche.

- **Les machines spécialisées** : appelées également machines dédiées, dans lesquelles chaque tâche est formée d'un ensemble d'opérations pouvant être exécutées par un sous ensemble de machines et suivant un ordre particulier. Selon ces particularités, se distinguent trois types de machines, à savoir :
  - **Les machines à cheminement unique (*flow shop*)**, où l'ensemble des opérations de chaque tâche doit s'exécuter par toutes les machines en suivant le même ordre de passage sur les machines qui est fixé et connu à l'avance, comme c'est le cas par exemple des usines de montage de produits ou d'assemblage de voitures. Si de plus, l'ordre d'exécution des tâches est le même sur toutes les machines, le problème d'ordonnancement est celui du *flow shop de permutation*.

- **Les Machines à cheminements multiples (*job shop*)**, qui sont considérées comme une généralisation du cas précédent. Les opérations des tâches sont exécutées sur des sous ensemble de machines suivant des ordres de passage différents et propres à chaque tâche. Les ordres de passage sur les machines diffèrent d'une tâche à une autre, mais doivent être connus et fixés à l'avance.
- **Les machines à cheminements libres (*open shop*)**, où le passage des tâches sur les machines n'est pas imposé à priori, et les opérations sont exécutées dans n'importe quel ordre de machines. Cet ordre est généralement à déterminer en plus de l'ordonnancement de toutes les opérations.

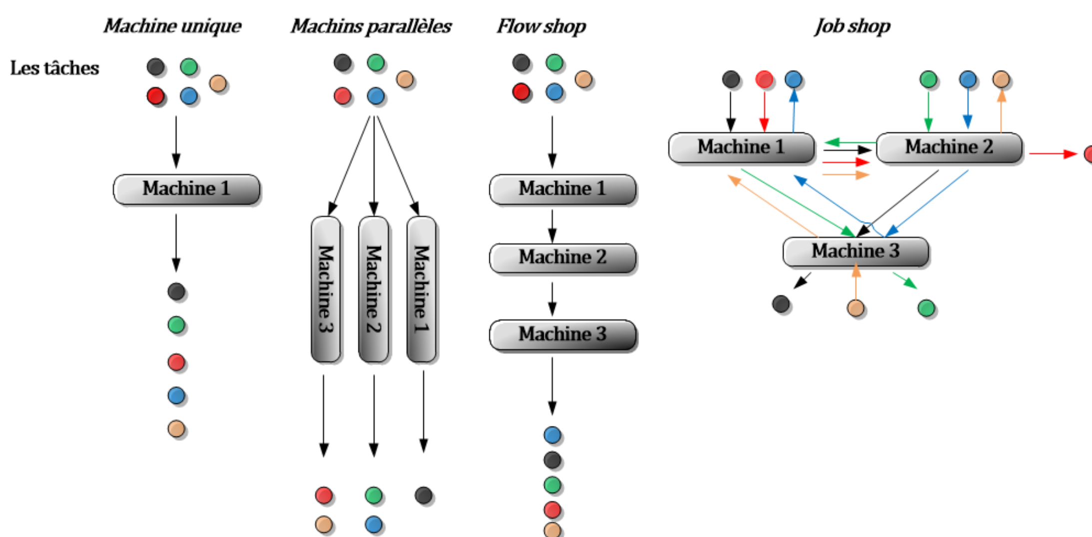


FIGURE 1.2 – Fonctionnement de quelques ateliers

Pour les problèmes d'ordonnancement d'ateliers, il existe une extension définie par l'introduction de la notion de *flexibilité de ressources*, qui suscite de plus en plus l'intérêt des chercheurs. Par exemple, il s'agit plutôt de *flexible flow shop* ou de *flexible job shop*, et qui sont des versions plus générales et par conséquent plus complexes que celles des problèmes initiaux (c'est-à-dire, sans flexibilité de ressources). Dans cette relaxation, chaque opération d'une tâche est susceptible d'être traitée par un sous ensemble de machines candidates. Ainsi, en plus des objectifs retenus dans les ver-

sions classiques, il est aussi question de déterminer par quelle machine doit être traitée chaque opération afin d'ordonnancer l'ensemble des opérations en un minimum de temps. Pour le lecteur intéressé, nous citons quelques travaux comportant des bibliographies riches concernant cette notion : Brucker & Schlie (1990); Botta-Genoulaz (1996); Dauzère-Pérès *et al.* (1998); Linn & Zhang (1999); Jansen *et al.* (2005).

1.2.3 Les contraintes

Les contraintes représentent les restrictions ou encore les conditions à respecter lors de la construction d'un ordonnancement. L'existence de contraintes rend le problème plus complexe, et ce en réduisant le nombre de solutions possibles. Les contraintes peuvent être relatives aux tâches et/ou aux ressources. Mathématiquement, une contrainte représente une restriction sur les valeurs numériques que peuvent prendre les variables de décision intervenant en modélisant le problème.

Il existe plusieurs classifications concernant les contraintes. De manière générale, elles peuvent être **endogènes** si elles sont liées directement aux composantes du problème et à leurs performances. Par exemple, ces contraintes peuvent représenter des dates de disponibilité, des limites sur les capacités des machines ou encore des relations entre les tâches telles que la précédence ou le conflit. Par ailleurs, elles peuvent être **exogènes** lorsqu'elles sont d'un niveau supérieur au problème donc imposées par des décisions externes. Nous pouvons citer l'exemple des dates de fin de traitement au plus tard imposées par les clients. D'autres classifications beaucoup plus utilisées sont regroupées et présentées dans les thèses suivantes : Artigues (1997); Toumi (2007). Nous citerons plusieurs exemples de contraintes un peu plus tard dans la partie 1.2.5.

1.2.4 L'objectif

Éxprimé à l'aide d'une fonction mathématique, l'objectif ou le critère d'optimisation est une évaluation numérique de la qualité d'un ordonnancement. Concrètement, l'objectif d'un problème d'ordonnancement permet de classer toutes les séquences de tâches possibles (solutions), et ce, en guidant la méthode de résolution utilisée vers la meilleure solution

possible. Ainsi, en plus de la notion d'ordonnancement **réalisable**, qui satisfait toutes les contraintes du problème, il est possible de définir un ordonnancement **optimal**, c'est-à-dire celui qui satisfait au mieux l'objectif considéré.

Ci-dessous, nous présentons une sélection des objectifs les plus étudiés par les chercheurs. Notez que les objectifs cités sont tous des fonctions à minimiser :

- **La durée totale (*makespan*) :** notée C_{max}, et représente la longueur d'un ordonnancement. Le makespan est égal à la date de fin de traitement de la dernière tâche exécutée, c'est-à-dire égal à $\max\limits_{j=1...n} \{C_j\}$.
- **La somme des dates de fin de traitement :** notée $\sum_{j=1}^{n} C_j$. Cet objectif qui peut aussi être considéré pour le cas pondéré est appelé par certains *flow time pondéré*. Il s'agira donc de minimiser la valeur $\sum_{j=1}^{n} w_j C_j$.
- **Le plus grand retard :** noté L_{max}, est égale au plus grand retard enregistré par les tâches lorsque le problème est soumis à des dates échues. La somme des retards est également utilisée et se définit par $\sum_{j=1}^{n} L_j$.
- **Le nombre de retards :** noté $\sum_{j=1}^{n} U_j$, dans lequel on ne s'intéresse pas à la durée des retards mais plutôt à leur nombre. Pour ce faire, une variable binaire U_j est définie pour chaque tâche, et vaut 1 lorsque la tâche est en retard et 0 sinon. Dans ce cas, il est aussi utile d'attribuer des poids aux tâches qui mesurent la pénalité engendrée suite aux retards. Il s'agira donc de minimiser $\sum_{j=1}^{n} w_j U_j$.
- **La variation des dates de fin :** notée TADC de l'anglais *Total absolute deviation of job completion times*, et exprimée par l'expression

$\sum_{i=1}^{n}\sum_{j=1}^{n}|C_j - C_i|$. Ce critère a été initié par Hsu *et al.* (2013).

Remarque 1 : Compte tenu de la nature réelle des problèmes d'ordonnancement, il est souvent difficile de se contenter d'un seul critère à optimiser. Afin de présenter la meilleure modélisation possible du problème, les chercheurs sont souvent amenés à optimiser plus d'un seul critère à la fois. Dans ce cas, le problème n'est plus un problème d'optimisation simple mais devient un problème d'optimisation multicritère.

Remarque 2 : Certaines études rencontrées lors de notre recherche suivent une méthodologie différente de la nôtre concernant la résolution des problèmes d'ordonnancement. Ces approches issues de Erschler (1976), transforment les objectifs en contraintes externes, et se donnent pour nouvel objectif la caractérisation des ordonnancements *admissibles*. Nous citons le travail de Artigues (1997), dans lequel l'auteur définit un ordonnancement *admissible* comme étant un ordonnancement vérifiant les contraintes internes et externes du problème.

1.2.5 Notation des problèmes d'ordonnancement

Dans la littérature, il existe différentes notations permettant de désigner un problème d'ordonnancement. Nous citons par exemple, celles de Conway *et al.* (1967); Maccarthy & Liu (1993), qui comportent quatre champs de la forme A | B | C | D représentant respectivement, les machines, les tâches, les contraintes et les objectifs. Il existe également la notation de Graham *et al.* (1979) comportant trois champs de la forme $\alpha|\beta|\gamma$, qui est la plus utilisée de toutes les notations existantes, c'est d'ailleurs celle que nous adoptons dans notre manuscrit. Les champs en question sont détaillés dans ce qui suit :

- **Le champ α :** qui comporte deux sous champs $\alpha_1\alpha_2$, et qui décrit les caractéristiques des ressources utilisées. Dans le contexte d'ordonnancement d'ateliers, α_1 représente le type de machines utilisées et α_2 indique le nombre de machines.

- **Le champ β :** représente l'ensemble des contraintes auxquelles est soumis le problème étudié.

- **Le champ γ :** indique le critère à optimiser (tels que ceux cités en 1.2.4).

Comme énoncé ci-dessus, les sous champs α_1 et α_2, indiquent le type de machines utilisées et leur nombre. Ces paramètres peuvent prendre les valeurs des ensembles respectifs suivants $\{1, P, Q, R, F, J, O\}$ et $\{\phi, k\}$. Les désignations de ces notations sont résumées dans la table 1.1.

Notation	Désignation du problème
1	Une seule machine
P	Machines parallèles identiques
Q	Machines parallèles uniformes
R	Machines parallèles générales
F	Machines de type *flow shops*
J	Machines de type *Job shops*
O	Machines de type *Open shops*
ϕ	Un nombre variable de machines
k	Entier positif désignant le nombre de machines

TABLE 1.1 – Désignations des champs $\alpha_1\alpha_2$.

Il est possible de retrouver d'autres symboles à côté de ceux déjà cités, afin de désigner des versions plus complexes. Par exemple, la lettre F peut s'ajouter aux lettres précédentes afin d'indiquer la version flexible du problème. Aussi, un paramètre d'indisponibilité des ressources existe, noté h_k, et désigne les k périodes d'indisponibilité pour les problèmes à une seule machine. Dans le cas général, h_{rk} indique l'indisponibilité de la machine M_r.

Le champ β doit regrouper et informer toutes les contraintes imposées au problème étudié. Les contraintes, ou plutôt leurs désignations par des mots clés, sont regroupées, ordonnées et séparées par des virgules. Nous présentons ci-dessous, des exemples de notations fréquentes pour les paramètres du champ β :

- *pmtn* : indique que la préemption des tâches est autorisée.

- *prec* : indique qu'il existe des contraintes de précédence entre les tâches.

- *tree*, *chains* : indiquent que les relations de précédence entre les tâches peuvent former un arbre ou un ensemble de chaines.

- $p_j = 1$, $p_j = p$ ou $a \leq p_j \leq b$: indiquent que les durées opératoires du problème peuvent être unitaires, identiques et égales à une valeur p ou encore appartenant à un intervalle $[a, b]$.

- r_j (respectivement. d_j) : indique que la tâche T_j possède une date de disponibilité (respectivement. date échue).

- s_j ou $s_{jj'}$: indiquent la présence de temps de préparation propre à la tâche T_j ou relative à la séquence des tâches T_j et $T_{j'}$ respectivement.

- *no-wait* : propre au cas des machines spécialisées, cette contrainte interdit un temps d'attente entre les opérations, ce qui signifie qu'une tâche qui termine son exécution sur une machine passe directement à la machine suivante.

- *split* : signifie, dans le cas de la préemption, que deux parties d'une même tâche peuvent s'exécuter par deux machines différentes et ce au même moment.

- *block* : indique que certaines tâches peuvent occuper une machine même après la fin de leurs traitements. Ce qui bloque la machine quant à l'exécution d'autres tâches.

A titre d'exemple, le $P3|pmtn, prec|L_{max}$ désigne le problème d'ordonnancement d'un ensemble de tâches sur trois machines, sachant que la préemption des tâches est autorisée et que des relations de précédence entre elles sont à respecter, le tout en vue de minimiser le retard maximum.

1.2.6 Représentation d'un ordonnancement

A la suite de la résolution d'un problème d'ordonnancement d'atelier, il est possible de visualiser l'ordonnancement obtenu avec un maximum

d'informations. Plusieurs moyens graphiques sont utilisés afin de schématiser un ordonnancement réalisable. En ce qui nous concerne, nous adoptons la représentation via le diagramme de Gantt[1] et plus précisément, le diagramme de Gantt ressources qui se décrit comme suit :

Le diagramme de Gantt ressources

Proposée par Henri Gantt, cette représentation est considérée comme étant la plus simple et la plus utilisée pour un ordonnancement de tâches (Herrmann, 2006). Le diagramme se compose de plusieurs barres parallèles horizontales, formant deux à deux plusieurs niveaux. Chaque niveau désigne une machine et comprend toutes les tâches dont l'exécution est assignée à la machine correspondante. Les tâches sont alignées et représentées par des rectangles dont la longueur est proportionnelle à la durée opératoire de la tâche qu'ils représentent. Un axe des abscisses est inclue dans la représentation afin de visualiser l'exécution des tâches dans le temps. Les barres verticales des rectangles désignant les tâches indiquent les dates de début et de fin de traitement de chacune d'entre elles. Un exemple du diagramme de Gantt ressources sera présenté pour l'exemple introductif du chapitre suivant (figure 2.1).

Le diagramme de Gantt tâches

Inversement au cas précédent, ce diagramme de Gantt place les tâches sur un axe vertical. Chacune d'entre elles est donc représentée par un niveau formé de deux barres parallèles horizontales (comme c'était le cas pour les machines dans le diagramme Gantt ressources). Par similitude, les longueurs des rectangles, formés dans chaque niveau, indiquent l'occupation dans le temps de chaque machine par la tâche correspondante. Le diagramme de Gantt tâches permet de visualiser les périodes d'attente

1. Henry Laurence Gantt, un ingénieur américain en mécanique (1861-1919). Il était également consultant en management, bouleversant par cette fonction le monde de la gestion de projet. C'est en 1910, que le fameux diagramme portant son nom a vu le jour, et demeure un outil puissant sur lequel repose les logiciels de planification jusqu'à nos jours.

des tâches lorsque celles-ci attendent la libération d'une ressource pouvant les exécuter.

Le graphe potentiels-tâches

Une autre manière de représenter un ordonnancement réalisable est celle qui utilise le graphe potentiels-tâches. Cette méthode repose, dans un premier temps, sur la création d'un graphe orienté, appelé aussi arbre disjonctif, à partir des données du problème étudié. Cette représentation a été initialement développée pour le problème du job shop par Roy & Sussmann (1964).

L'ensemble des sommets correspond à toutes les opérations à ordonnancer aux quels sont ajoutés deux sommets fictifs appelés *source* et *puit*, représentant respectivement le début et la fin de l'ordonnancement. Les opérations consécutives d'une même tâche sont reliées par des arcs dits *conjonctifs* qui indique la relation de précédence entre les deux opérations. Des arcs non orientés dits *disjonctifs* sont également ajoutés entre les opérations de tâches différentes nécessitant une même machine. Une pondération sur les différents arcs est à définir et correspond aux durées des opérations qui sont représentées par les extrémités initiales de chaque arc. Notons que le sommet *source* est relié à toutes les premières opérations des tâches avec des arcs de poids égal à zéro. Les dernières opérations de chaque tâche sont, quant à elles, reliées au sommet *puit* par des arcs de poids égal à leurs durées opératoires.

À la différence du diagramme de Gantt, le graphe potentiel-tâches ne permet pas seulement la visualisation d'une solution réalisable pour un problème donné, mais représente également un outil de modélisation utilisé lors de la résolution de certains problèmes d'ordonnancement de type job shop (Jain & Meeran, 1999; Aggoune, 2002). Pour la résolution, il est question de choisir une orientation des arcs *disjonctifs* de sorte à ce que le graphe résultant soit sans circuits. Une fois que c'est le cas, le graphe est dit arbitré et la durée totale de l'ordonnancement est égale à la longueur du plus long chemin reliant le sommet *source* au sommet *puit*. Des exemples de représentation par le graphe potentiel-tâches sont présentés dans Aggoune (2002).

1.3 Notions sur la complexité des algorithmes et des problèmes

La notion de complexité est une notion fondamentale dans le domaine de l'optimisation combinatoire. Ce concept permet d'une part de définir la nature de tout problème mathématique ou réel et ce, en lui attribuant la caractéristique d'être facile ou difficile à résoudre d'un point de vue mathématique. D'une autre part, la complexité permet, en fonction du degré de difficulté mesuré, de partager les problèmes étudiés en un ensemble de classes de problèmes. Chaque classe regroupe des problèmes qualifiés du même niveau de difficulté, et qui par la suite peuvent être abordés par les mêmes outils de résolution.

En recherche opérationnelle, le terme *complexité* est associé à deux notions qu'il faut absolument distinguer. Dans ce qui suit, nous allons définir les notions citées, à savoir celles de la complexité algorithmique et de la complexité des problèmes.

1.3.1 L'algorithme

Avant de présenter une quelconque définition de complexité, il est nécessaire d'aborder une notion élémentaire traduite par le terme algorithme. Généralement, et même intuitivement, un algorithme désigne une méthode, une description précise ou encore une façon de procéder destinée à résoudre un problème donné. Cette définition combien même très vaste, demeure commune à plusieurs disciplines scientifiques.

En ce qui nous concerne, les définitions les plus proches et les plus représentatives sont celles que l'on retrouve en mathématique mais aussi en informatique. D'ailleurs, plusieurs personnes associent le terme algorithme[1] (plutôt mathématique historiquement) au terme programme (qui lui, est plutôt informatique), et confondent entre leurs deux significations. Néanmoins, nous tenons à signaler qu'il n'existe pas une seule définition

1. Officiellement, nous devons l'apparition du terme algorithme au mathématicien perse Mohammed Ibn Musa-Al Khwarizmi (considéré comme étant le père de l'algèbre) originaire de la région Khawarizm dans l'actuel Ouzbékistan et dont la naissance est estimée vers l'an 820.

du terme algorithme mais plusieurs chacune proche du contexte dans lequel elle est utilisée.

Pour un algorithme, plusieurs définitions ont été rencontrées dans la littérature. Bien que différentes, ces définitions présentent toutes des aspects en commun. Dans ce qui suit, nous présentons quelques-unes à commencer par la plus naturelle, celle que l'on retrouve dans le Larousse :

Définition (Larousse) : *« Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations. »*

Définition (Cormen *et al.* (2001)) : *« De manière informelle, un algorithme est une procédure de calcul bien définie qui à partir d'une valeur, ou d'un ensemble de valeurs, en entrée produit une valeur, ou un ensemble de valeurs, en sortie. Un algorithme est donc une séquence d'étapes de calcul qui transforment une entrée en une sortie. »*

Définition (Skiena (2008)) : *« Un algorithme est une procédure permettant d'accomplir une tâche spécifique. Un algorithme est l'idée qui se trouve derrière tout programme informatique raisonnable. »*

Définition (Modeste (2012)) : *« Un algorithme est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambiguës et organisées, la réponse au problème pour toute instance de cette famille. »*

Définition (Yagouni (2017)) : *« Un algorithme de résolution d'un problème est une séquence complète d'instructions* ***élémentaires****,* ***complémentaires****,* ***logiques*** *et* ***chronologiques****, transformant une chaîne de caractères représentant les données de n'importe quelle instance du problème, en une chaîne de caractères représentant le résultat de sa résolution. »*

1.3.2 Complexité algorithmique

La complexité d'un algorithme peut être définie comme étant une mesure mathématique permettant d'évaluer sa performance en matière de ré-

solution de problèmes. De plus, cette mesure nous offre la possibilité de le comparer à d'autres algorithmes destinés à résoudre un même problème, dans le but de définir le meilleur algorithme à cet effet, indépendamment de la machine qui les exécutent. La complexité d'un algorithme est calculée en fonction de la taille de l'instance à résoudre. Elle peut être mesurée selon deux critères : **le temps** que nécessite l'algorithme pour résoudre l'instance, et l'**espace mémoire** qu'il occupe pour le même objectif.

La notion de complexité temporelle d'un algorithme est exprimée en fonction du nombre d'opérations élémentaires effectuées par ce dernier, pour la résolution d'une instance du problème. Usuellement, c'est le nombre d'opérations pour le pire des cas possibles qui est calculé. Autrement dit, le cas le plus défavorable qui compte le maximum d'opérations éventuellement effectuées pour la résolution d'une instance. Cependant, il est aussi fréquent d'évaluer la complexité moyenne d'un algorithme, qui est obtenue en calculant le nombre moyen d'opérations effectuées pour la résolution d'une même instance.

Un peu moins utilisée que la précédente, la complexité spatiale (qui est relative à l'espace mémoire), désigne la quantité d'informations indispensable au bon déroulement de l'algorithme. Dans un contexte informatique, où un algorithme est confondu avec un programme informatique, cette complexité représente la taille mémoire nécessaire pour stocker toutes les structures de données requises au bon fonctionnement de l'algorithme.

Étant une mesure mathématique, il est indispensable de présenter les définitions formelles sur lesquelles repose la notion de complexité d'algorithme. Il s'agit, entre autre, de la notion du *Grand O*. Ce paramètre, qui est d'ailleurs le plus utilisé, permet de représenter l'évaluation des complexités d'algorithmes de manière commune. Ainsi, on dit qu'un algorithme est de complexité $T(n) = O(f(n))$ (se lit $T(n)$ est de l'ordre de $f(n)$), s'il existe deux constantes positives c et n_0 telles que (Rebaine, 2000) :

$$T(n) \leq cf(n), \quad \forall n \geq n_0$$

Ainsi, cette notion du O, indiquant la complexité $T(n)$ d'un algorithme donné, est définie par l'existence d'une fonction $f(n)$ considérée comme étant une borne supérieure. Cependant, d'autres notations existent dans

la littérature, il s'agit des deux notations Ω et Θ exprimant respectivement les concepts de borne inférieure et celui d'égalité. Ces deux concepts sont définis de façon similaire à celle du O, c'est-à-dire :

Grand Ω . *Un algorithme est de complexité $T(n) = \Omega(f(n))$, pour tout n positif, s'il existe deux constantes positives c et n_0, telles que :*

$$T(n) \geq cf(n), \quad \forall n \geq n_0$$

Grand Θ . Un algorithme est de complexité $T(n) = \Theta(f(n))$, pour tout n positif, s'il existe des constantes positives c_1, c_2 et n_0, telles que :

$$c_1 f(n) \leq T(n) \leq c_2 f(n), \quad \forall n \geq n_0$$

Nous rappelons que la notion de complexité en général, et le temps de calcul en particulier, est exprimée en fonction de la taille de l'instance que l'algorithme vise à résoudre. La taille de l'instance du problème est notée n. Par conséquent, la complexité d'un algorithme est liée non seulement à l'existence des paramètres précédents dont de la fonction $f(n)$, mais aussi à sa nature. Ce qui donne lieu à plusieurs complexités connues comme par exemple :

- **La complexité constate**, notée $O(c)$, où $f(n)$ est une constante c indépendante de n.
- **La complexité logarithmique**, notée $O(log(n))$, où $f(n)$ est une fonction logarithmique.
- **La complexité polynomiale**, notée $O(n^k)$, où $f(n)$ est un polynôme d'ordre k.
- **La complexité exponentielle**, notée $O(k^n)$, où $f(n)$ est une fonction exponentielle, et k est une constante strictement supérieure à 1.
- **La complexité factorielle**, notée $O(n!)$.

Remarque. Un algorithme dont la complexité est d'ordre polynomial est considéré comme un algorithme efficace. Pour abréger, un tel algorithme est dit polynomial. En revanche, cette efficacité ne peut être attribuée à un algorithme de complexité exponentielle à cause de sa consommation importante en espace ou en temps. Pour mieux le voir, prenons

l'exemple d'un algorithme de résolution d'un problème d'optimisation donné, dont la complexité est de l'ordre de $O(2^n)$. Si cet algorithme est exécuté par une machine effectuant chaque opération en 10^{-6} secondes, alors il serait capable de résoudre une instance de ce problème de taille 50, une taille modeste pour un problème réel d'industrie, en pas moins de 35 ans.

1.3.3 Complexité des problèmes

La complexité d'un problème dépend de la nature du problème lui-même et de sa résolution. En théorie de complexité, afin de connaitre la nature d'un problème, il est nécessaire de dissocier la notion du problème d'optimisation de celle du problème de décision qu'il lui est associé. Quant à la résolution, elle est mesurée par rapport au meilleur [1] algorithme connu et conçu pour ce même but.

Un problème de décision, appelé aussi problème de reconnaissance, est un problème pour lequel on ne peut répondre que par *Vrai* ou *Faux* [2] à une question donnée. Notons que pour tout problème d'optimisation, il est possible d'associer un problème de décision. Ainsi, connaitre la complexité d'un problème de décision nous permet d'avoir des informations sur celle du problème d'optimisation associé. Par exemple, le problème de décision associé à un problème d'ordonnancement peut être formulé grâce à la question suivante :

Pour une instance donnée d'un problème d'ordonnancement, existe-il une solution réalisable telle que la valeur de la fonction objectif ne dépasse pas une valeur fixée ?

La résolution d'un problème de décision se fait en deux temps, la recherche d'une solution et la vérification de la solution. La classification de Garey et Johnson (Johnson & Garey, 1979) répertorie les problèmes en deux classes P et NP :

1. En termes de complexité.
2. Oui ou non

- **La classe** P (Polynomial time), regroupe les problèmes de décision pour lesquels, il existe des algorithmes polynomiaux permettant de les résoudre. Ces algorithmes doivent permettre la recherche et la vérification d'une solution réalisable en un temps polynomial. Les problèmes de cette classe sont considérés comme étant facile car résolus de manière efficace.

- **La classe** NP (Non deterministic Polynomial time[1]), contient les problèmes de décision pour lesquels, il existe un algorithme polynomial non-déterministe permettant de les résoudre. La phase de vérification d'une solution réalisable obtenue peut être réalisée par un algorithme polynomial, sans que ça soit le cas pour la recherche d'une telle solution[2].

Plusieurs définitions d'un algorithme non déterministe existent, nous nous contentons de donner la plus simple à notre sens. Un algorithme non déterministe est un algorithme qui comporte l'instruction choix, celle-ci opérant sur un ensemble fini, choisit un élément de cet ensemble mais on ne spécifie pas comment ce choix est effectué (Sakarovitch, 1984).

Certains problèmes appartenant à la classe NP sont dits NP-*complets*. Un problème est dit appartenir à la classe NP-*complet*, si tous les problèmes de NP peuvent se réduire à lui par un principe de réduction polynomiale. On dit d'un problème P qu'il peut être réduit à un problème P', si les données du problème P peuvent être transformées en données du problème P' en temps polynomial. Cette relation apporte une équivalence entre tous ces problèmes en question dans le sens où l'existence d'un algorithme polynomial pour résoudre un problème des NP-*complet*, impliquerait nécessairement l'existence d'un algorithme polynomial pour tout problème de NP. Notons aussi, que pour tout problème de décision appartenant à la classe des problèmes NP-*complets*, le problème d'optimisation associé est dit NP-$difficile$.

Pour certains problèmes non polynomiaux, il existe des algorithmes de résolution polynomiaux dont la complexité est exprimée en fonction

1. À ne pas confondre la classe NP avec Non Polynomial time
2. Une telle action peut se réaliser en énumérant l'ensemble des solutions possibles qui est, au pire des cas, de cardinalité exponentielle

des données de l'instance elle-même. De tels algorithmes sont appelés *pseudo-polynomiaux*. Un problème d'optimisation est dit NP-$difficile$ ***au sens faible*** (*in a weak sense*), s'il peut être résolu par un algorithme pseudo-polynomial. Dans le cas contraire, un tel problème est dit NP-$difficile$ ***au sens fort*** (*in a strong sense*).

1.4 Méthodes de résolution

Dans la section précédente, nous avons exposé les deux classes de problèmes les plus importantes en théorie de complexité. Les problèmes considérés difficiles à résoudre sont donc ceux de la classe NP-*complet* qui n'ont pas pu être résolus efficacement jusqu'à l'heure actuelle. Pour ces problèmes, il est plus précis de dire que l'on ne connait pas encore d'algorithmes polynomiaux pouvant les résoudre. D'ailleurs, à ce propos, une récompense d'un million de dollars est proposée afin de répondre à la fameuse question : *"la classe P, est-elle égale à NP ?"*. À défaut d'être prouvée, toutes les études reposent sur la négation de cette dernière [1]. On se penche donc, plus particulièrement, sur l'étude de la classe des problèmes NP-*complets*, qui sont considérés comme étant les plus difficiles à résoudre, mais qui s'avèrent être les plus intéressants à étudier.

Deux catégories de méthodes existent principalement et interviennent dans la résolution des problèmes non polynomiaux du type NP-*complets*. Il s'agit en premier, des méthodes exactes qui assurent l'obtention de solutions optimales mais qui ne s'avèrent intéressantes que pour des instances de petites tailles dont les solutions peuvent être obtenues en un temps raisonnable. En second, il est question de méthodes approchées qui n'offrent pas cette qualité d'optimalité, mais qui en contrepartie se rattrapent en matière de temps de calcul. Dans cette section, nous passons en revue les méthodes de résolution les plus connues et les plus utilisées pour la résolution des problèmes d'optimisation NP-$difficile$.

D'un point de vue mathématique, nous rappelons que pour un problème d'optimisation, le souhait de toute tentative de résolution est de tomber sur ce que l'on appelle l'optimum global qui peut se définir comme

1. Jusqu'à preuve du contraire.

suit :

Une solution réalisable notée $s*$ est un optimum global si sa fonction objectif est meilleure que celle de toutes les solutions réalisables formant l'espace de recherche S,
i.e.

$$\forall s \in S, f(s*) \leq f(s) \quad \textit{pour un problème de minimisation}$$
$$\forall s \in S, f(s*) \geq f(s) \quad \textit{pour un problème de maximisation}$$

1.4.1 Les méthodes exactes

Les méthodes de résolution exactes garantissent l'obtention de la solution optimale et ce, en explorant de manière intelligente et implicite tout l'espace de recherche. Ces méthodes peuvent visiter l'ensemble des solutions réalisables en utilisant des procédures spécifiques à chaque problème qui permettent d'une part d'orienter la direction de recherche, et d'une autre part, de limiter les directions inutiles ou stériles.

Parmi les méthodes exactes fréquemment utilisées, nous pouvons citer :

- **La méthode de séparation et évaluation** (*branch & bound*) : C'est une méthode qui date de 1954 (Dantzig *et al.*, 1954), proposée pour résoudre le célèbre problème du voyageur de commerce. Le principe de cette méthode est d'accompagner la recherche d'une solution optimale par une représentation arborescente qui énumère toutes les solutions possibles. Chaque noeud de cette arborescence correspond à un choix binaire, qui traduit la réalisation d'une action ou non (comme par exemple, l'affectation d'une tâche ou non à une machine). La séparation consiste à décomposer le problème en une partition de sous problème de tailles inférieurs, et ayant chacun un ensemble de solutions réalisables, de telle sorte à ce que la résolution optimale des sous problèmes engendrerait la résolution optimale du problème initial. Pour chaque sous problème, une borne

inférieure ainsi qu'une borne supérieure sont calculées. L'évaluation permet justement d'éviter une énumération exhaustive de toutes les solutions possibles. Une telle énumération, notamment pour un ensemble de recherche très souvent de cardinal exponentiel, est très coûteuse en terme de temps et souvent inutile. Pour chaque noeud, lorsque la valeur de la borne inférieure dépasse celle de la meilleure borne supérieure calculée, la recherche dans cette direction est inutile et donc suspendue, et le branchement s'arrête pour ce noeud.

- **La programmation linéaire** : Une méthode reposant sur la construction d'un programme linéaire, qui est avant tout un outil mathématique nous permettant de modéliser les problèmes d'optimisation combinatoire. Un programme linéaire comporte une fonction à optimiser et des contraintes exprimées par des équations et des inéquations linéaires. Les solutions réalisables forment une enveloppe convexe, appelé polytope, dont les sommets représentent les solutions optimales du problème. Lorsque toutes les variables de décision du programme linéaire sont de nature entière, le programme linéaire est dit linéaire en nombres entiers (PLNE). Pour ce cas, le polytope engendré est discret et non convexe, ce qui rend les PLNE plus difficile à résoudre. Lorsque cette propriété caractérise certaines variables seulement, le programme est dit linéaire en variables mixtes (PLNE mixte). En fonction de la nature du programme linéaire, plusieurs méthodes ont été développées et destinées à sa résolution à commencer par l'algorithme du simplexe (développé par Dantzing en 1947), qui permet de résoudre un bon nombre de programmes linéaires. D'autres méthodes de résolution ont suivi à l'instar de la méthode ellippsoïdes, la méthode des points intérieurs ou encore les méthodes arborescentes.

- **La programmation dynamique** : Considérée comme étant le fruit de plusieurs recherches menées par l'américain Richard Bellman durant les années 1950 et ce, afin de résoudre des problèmes d'optimisation et d'affectation. Le principe de cette méthode a vu le jour en 1957 grâce au théorème d'optimalité qui part du principe suivant : *"toute politique optimale ne peut être formée que de sous politiques optimales"*. Les travaux de Bellman sont apparus après la découverte du simplexe par Dantzig pour la programmation linéaire, et c'est en partie pour

cette raison que Bellman l'a baptisée "programmation dynamique". L'adjectif dynamique a été employé pour décrire l'aspect séquentiel et récursif de la méthode. La programmation dynamique n'est applicable que pour les problèmes décomposables en phases, où chaque phase ne dépend que de ses voisines les plus proches (antérieure et postérieure). Le principe de la programmation dynamique est de décomposer le problème en sous problèmes, puis d'établir l'équation récursive qui permet d'exprimer la fonction objectif du sous problème d'ordre k en fonction de celle du sous problème d'ordre $k-1$ ou $k+1$. La transition d'un état k à un état $k+1$ est appelée décision, et la valeur de la fonction objectif est évaluée à la prise de chaque décision. Le problème consiste donc à trouver une séquence de décisions (de l'état initial à l'état final) menant à la valeur optimale de la fonction objectif.

Le fait d'énumérer toutes les solutions de l'espace de recherche par les méthodes exactes, a conduit à atteindre très vite une certaine limite face à des problèmes de taille considérable. Effectivement, à mesure que la taille du problème augmente, les délais d'attente des solutions optimales que garantissent ces approches deviennent plus importants. Malheureusement, et en dépit de tous les progrès que ces méthodes réalisent, la contrainte du temps a fait que ces méthodes soient en quelque sorte considérées comme inefficaces face à de problèmes de grande envergure. Cette contrainte a poussé les chercheurs à changer de priorité, quant à leur attente en matière de qualité de solutions, au service du temps de calcul. Cette réflexion a donné lieu à ce que nous appelons méthodes approchées.

1.4.2 Les méthodes approchées

Face à des problèmes d'optimisation de grande taille et de complexité non polynomiale, la recherche s'est orientée vers la conception de nouvelles alternatives aux méthodes exactes jugées trop coûteuses. Ces nouvelles approches n'ont plus l'objectif d'atteindre l'optimalité, mais oeuvrent plutôt à fournir des solutions de bonnes qualités (c'est-à-dire, celles qui sont les plus proches possible des solutions optimales), et ce à moindre coût (c'est-à-dire, s'exécutant en des temps raisonnables).

La performance des méthodes approchées, mesurée par la qualité des solutions qu'elles fournissent, est évaluée pour une pile d'instances moyennant deux critères. Le premier consiste à déterminer l'écart ou la distance entre les solutions obtenues et les solutions optimales, si ces dernières sont calculables[1]. Dans le cas où il n'est pas possible de mesurer l'écart entre les solutions fournies par les méthodes approchées et les solutions optimales, la comparaison peut se faire par rapport à des bornes inférieures calculées pour le problème étudié. Le deuxième critère est bien évidemment le temps de calcul rapporté suite à l'obtention de ces solutions.

Pour les méthodes approchées, on distingue principalement deux catégories de méthodes : **les heuristiques**, qui se présentent sous la forme de procédures simples conçues et destinées à résoudre un problème en particulier, et **les métaheuristiques**[2] qui elles se présentent sous la forme d'ossatures générales représentant un certain concept que l'on adapte spécifiquement au problème que l'on souhaite à résoudre.

Les heuristiques sont donc des méthodes de résolution assez simple à mettre en oeuvre. La conception de telles approches résulte d'une connaissance approfondie du problème considéré et de ses propriétés. Ainsi, le principe de chacune dépend d'un seul problème en particulier afin de proposer des solutions réalisables et au mieux de bonne qualité. Les temps de calcul de ces méthodes sont en général très raisonnables. En fonction des paramètres de départ de ces méthodes, nous pouvons distinguer deux types d'heuristiques, celles qui construisent une solution réalisable et celles qui ont pour but d'améliorer une telle solution.

- **Les heuristiques de construction :** Comme leur nom l'indique, les heuristiques de construction sont des algorithmes de résolution qui pour un problème d'optimisation, construisent une solution réalisable à partir des données de l'instance. Ceci est réalisé en effectuant itérativement des choix partiels menant à la construction de la solution finale. Un exemple de procédures très utilisées pour les

1. L'évaluation se fait entre la valeur de la fonction objectif correspondante à chaque solution et non pas entre les solutions citées.

2. Généralement, le terme *méta* est attribué à un niveau d'abstraction supérieur.

problèmes d'ordonnancement sont les algorithmes de liste, dans lesquels les tâches suivent un ordre convenu est fixé pour démarrer la procédure. À partir de cet ordre, les tâches sont affectées une par une à la première machine libre en mesure de les exécuter. Les algorithmes SPT (Shortest Processing Time), LPT (Longest Processing Time) ou encore l'algorithme EDD (Earliest Due Date) (Jackson, 1955) sont des exemples concrets exprimant ce principe. D'autres algorithme de ce genre sont répandus pour résoudre des problèmes d'optimisation combinatoire comme le voyageur de commerce ou le problème de tournées de véhicules pour lesquels nous pouvons citer les heuristiques du plus proche voisin ou l'algorithme glouton (Talbi, 2009; Yagouni, 2017).

- **Les heuristiques d'amélioration :** Contrairement au cas précédent, le but des algorithmes d'amélioration n'est plus de construire une solution réalisable, mais plutôt d'en améliorer la qualité. C'est à dire de trouver, à partir d'une solution initiale bien évidemment réalisable, une meilleure solution en ayant diminué (pour un problème de minimisation) la valeur de la fonction objectif. Généralement, la notion de voisinage est utilisée par ces méthodes, et a pour objectif de produire une exploration plus large de l'espace de solutions (on espère par ce principe visiter le maximum de solutions possibles). Le principe de recherche locale est généralement associé aux processus d'amélioration en incluant la notion de voisinage. Le voisinage d'une solution réalisable est obtenu en effectuant une transformation élémentaire (appelée aussi, opération élémentaire, mouvement ou aussi perturbation) sur la structure qui représente la solution afin d'obtenir de nouvelles solutions. Ces nouvelles solutions, dites voisines, sont par la suite évaluées et comparées à la solution initiale et seuls les voisins de meilleur qualité sont en premier lieu privilégiés d'une itération à une autre jusqu'à atteindre un certain objectif désignant le critère d'arrêt de la méthode. Il arrive que des solutions de moins bonne qualité soient retenues par ces méthodes dans le but de s'extraire de certaines zones dans lesquelles l'exploration n'est plus fructueuse.

Bien que l'avantage du temps de calcul soit nettement au profit des heuristiques, de construction ou d'amélioration, certaines limites se sont

vite faites remarquer. Parmi ces limites, la notion de portabilité (expliquée par Yagouni (2017)) qui exprime le champ d'application de ces méthodes assez restreint du fait qu'elles aient été conçues pour des problèmes précis. Une heuristique conçue pour un problème d'ordonnancement donné ne peut être appliquée directement à un autre problème d'ordonnancement ou à un autre problème d'optimisation à moins d'en apporter plusieurs modifications si toutefois de telles modifications sont possibles à réaliser.

Les chercheurs ont également discuté la qualité des solutions fournies par les heuristiques des deux catégories. Très souvent, il arrive que les solutions soient de mauvaise qualité, leurs améliorations par des heuristiques spécialisées dépend directement de la solution initiale et du concept de voisinage adopté. Le problème courant dans le choix du voisinage, est que certains conduisent et bloque la recherche à un optimum local propre à ce voisinage. La plus grande limite des heuristiques est justement leur incapacité de s'extraire d'une telle zone au risque de ne pas atteindre l'optimum global.

Afin de pallier aux limites des approches précédentes, de nouvelles méthodes, plus efficaces et tout aussi simples à mettre en oeuvre, ont vu le jour, et ceci par l'incorporation de techniques plus subtiles et plus intelligentes permettant l'obtention de meilleurs résultats. Ces nouvelles méthodes approchées sont donc les Métaheuristiques.

Contrairement aux heuristiques qui, elles sont dédiées à un problème en particulier, les métaheuristiques sont des méthodes génériques qui se présentent sous la forme de mécanismes et de concepts fondamentaux indépendants de tout problème. Ces principes sont adaptables et applicables à bon nombre de problèmes d'optimisation combinatoire. En fonction de leurs caractéristiques ou de leurs origines, plusieurs classifications de ces méthodes existent dans la littérature (Yagouni, 2017; Talbi, 2009).

La plus courante des classifications repose sur le nombre de solutions que la métaheuristique manipule à la fois. En effet, il existe suivant cette classification deux types de métaheuristiques :

Le premier représente les méthodes qui traitent une seule solution à la fois (*Single Based Metaheuristics*). Ces méthodes sont également appe-

lées des méthodes de recherche locale ou méthodes de trajectoire, et ont tendance à intensifier la recherche en explorant "intensément" une partie (une direction) de l'espace de recherche. Ce processus d'intensification est réalisé en partant d'une solution initiale unique, puis en se transportant d'une solution vers une solution voisine dans l'optique d'aboutir en fin de processus de recherche, à une amélioration du critère considéré.

Le second type de méthodes regroupe celles qui manipulent un ensemble de solutions simultanément que l'on appelle *population*. Ces méthodes sont également connues sous le nom d'algorithmes évolutionnistes[1] (*Population Based Metaheuristics*), et ont plutôt tendance à diversifier la recherche en explorant plusieurs parties de l'espace de recherche quitte à tolérer momentanément une dégradation du critère. Cette particularité se traduit par l'acceptation de solutions qui sont de moins bonne qualité. Ce processus de diversification permet surtout à la recherche de s'extraire des zones dans lesquelles elle peut se retrouver piégée notamment dans des optimums locaux.

Dans ce qui suit, nous présentons des aperçus des métaheuristiques les plus utilisées pour résoudre des problèmes d'ordonnancement. Pour plus de détails concernant ces métaheuristiques, et bien d'autres existantes dans la littérature, le lecteur intéressé est prié de consulter l'ouvrage de Talbi (2009).

Le recuit simulé (Simulated Annealing)

Le recuit simulé est une métaheuristique qui s'inspire d'un principe réel de traitement thermique appelé le recuit, utilisé principalement en métallurgie. Cette pratique permet de modifier les caractéristiques physiques d'un matériau afin d'aboutir à un changement de la matière première (réduire sa dureté ou encore corriger sa structure). L'introduction de ce principe sous la forme d'une métaheuristique est due aux trois chercheurs d'IBM[2] : S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983 (Kirkpatrick *et al.*, 1983). L'exploration du voisinage dans le recuit simulé a la

1. En référence à la théorie de l'évolution.
2. De l'anglais *International Business Machines Corporation*, IBM est une multinationale américaine spécialisée dans le matériel, le logiciel et les services informatiques depuis 1911 (source : https ://fr.wikipedia.org/wiki/IBM).

particularité de permettre de se diriger vers une solution voisine de moins bonne qualité avec une certaine probabilité afin d'échapper aux optimums locaux. En plus de cette probabilité, la métaheuristique inclut d'autres paramètres propres au processus du recuit tels que la température, fixée au tout début de l'algorithme et qui décroit tout au long de son exécution. Ainsi, trouver le meilleur réglage possible des paramètres du recuit simulé demeure sans doute la tâche la plus difficile dans l'implémentation de cette métaheuristique. Une preuve de la convergence de cette méthode existe dans la littérature ce qui implique qu'un paramétrage optimal, lorsqu'il est trouvé, mènerait nécessairement à la solution optimale.

La recherche tabou (Tabou Search)

La recherche tabou a été introduite par Fred Glover en 1986 (Glover, 1986, 1990; Glover & Laguna, 1998). Elle représente une des méthodes de recherche locale totalement déterministe ayant comme particularité d'explorer, à chaque itération, tout le voisinage d'une solution courante afin d'obtenir la meilleure solution de ce dernier (même si celle-ci n'est pas de meilleure qualité que la solution courante). La recherche tabou a été mise en place comme une nouvelle stratégie par laquelle Glover a souhaité échapper aux optimums locaux en dotant la méthode d'un mécanisme de mémoire. Concrètement, la mémoire de la méthode est représentée par une liste dite tabou contenant les solutions récemment visitées. Cette liste est mise à jour chaque itération en insérant la nouvelle solution visitée et en supprimant la solution la plus anciennement visitée. Ce principe est mis en place dans le but d'éviter de cycler entre un optimum local et son meilleur voisin et par conséquent, être bloqué dans la même région de l'espace de recherche.

L'algorithme génétiques (Genetic Algorithm)

L'algorithme génétique est une métaheuristique à population de solutions, proposée par John Holland en 1975 (Holland, 1975). Cette méthode tire son fonctionnement du principe de la théorie de l'évolution des espèces dans leur environnement naturel énoncée par Darwin, qui exprime la transmission des gênes entre les parents et les enfants au fil des générations. Tout comme dans le domaine de la génétique, chaque individu de la population (qui correspond à une solution réalisable du problème)

est porteur d'une signature identitaire dite *chromosome* qui contient ses informations génétiques. En implémentation, ce chromosome comprend les caractéristiques de la solution ainsi que sa valeur de la fonction objectif. Par analogie à la théorie de survie de Darwin, seuls les individus les plus forts sont susceptibles de donner lieu à la meilleure descendance possible. Ainsi, l'évolution de la population est établie en deux phases. Dans un premier lieu, la sélection, en se basant sur l'évaluation de la fonction objectif, des meilleurs individus prédisposés à être les meilleurs parents. Puis, la rencontre entre eux, qui se fait moyennant principalement deux opérateurs appelés croisement et mutation. L'idée principale d'un algorithme génétique est donc de faire évoluer une population d'individus (représentant des solutions initiales générées aléatoirement) vers une population d'individus meilleurs, qui comporte ainsi, les meilleures solutions possibles. Une meilleure population de même taille est reconstituée à chaque itération composée des nouveaux meilleurs individus.

Algorithmes de colonies de fourmis (Ant Colony Optimization Algorithms)

Une autre méthode à population de solution est l'algorithme de colonies de fourmis, proposé initialement par Marco Dorigo en 1992 (Dorigo, 1992), et conçu à la base pour résoudre le problème du voyageur de commerce. Le principe de cette méthode s'inspire de l'observation du comportement auto-organisationnel des fourmis lors de leur quête de nourriture. En effet, des observateurs biologistes ont rapporté que les fourmis étaient capables de choisir parmi plusieurs chemins, le chemin le plus court reliant leur nid à leur source de nourriture. Cette faculté est due à une substance chimique odorante, appelée *phéromone*, déposée par leurs congénères lors de leur passage afin de leurs indiquer le meilleur chemin trouvé. Même si, de base, les algorithmes de colonies de fourmis, par leur nature, aient été conçus pour retrouver le plus court chemin dans des problèmes tels que le voyageur de commerce, ces algorithmes ont tout de même pu être généralisés et repris pour résoudre plusieurs autres problèmes d'optimisations (Gagné *et al.*, 2001).

1.4.3 Les algorithmes d'approximation

En plus des deux catégories de méthodes de résolution citées, il existe une autre famille d'algorithmes conçue pour la résolution des problèmes intéressants d'optimisation combinatoire. Bien que moins sollicités que les autres, ces algorithmes représentent une excellente tentative de résolution de par leur particularité. En effet, cette classe de problèmes a la particularité de garantir la qualité des solutions qu'elle fournit. Cette garantie est exprimée par la distance, fixée préalablement, entre la valeur de la solution optimale et celle des solutions fournies par ces algorithmes. Ces algorithmes d'approximation n'ont pas été évoqués lors de la partie introductive des méthodes de résolution car, à notre sens, ces approches représentent un cas particulier des algorithmes approchés avec en plus la particularité énoncée.

En plus des caractéristiques précédentes, cette classe de méthodes de résolution peut être décrite à travers les définitions suivantes (Schuurman & Woeginger, 1999) :

Définition (Algorithmes d'approximation)**.** *Pour un problème de minimisation (respectivement, maximisation) noté X, soient $\epsilon > 0$ et $\rho = 1 + \epsilon$ (respectivement, $\rho = 1 - \epsilon$). Un algorithme A est dit $\rho - approximation$ pour le problème X, si pour toute instance I de X, les solutions fournies par X, de valeurs $A(I)$, vérifient :*

$$|A(I) - OPT(I)| \leq \epsilon.OPT(I)$$

où, $OPT(I)$ représente la valeur de la solution optimale pour l'instance I.

Notez que l'expression précédente peut être formulée autrement, elle devient donc :

$A(I) \leq (1 + \epsilon)OPT(I)$ si X est un problème de minimisation.

$A(I) \geq (1 - \epsilon)OPT(I)$ si X est un problème de maximisation.

À partir de l'expression précédente, nous pouvons définir quelques mesures utilisées dans la littérature permettant d'évaluer la qualité des solutions, et ce, en mesurant la distance de la valeur des solutions fournies

par ses algorithmes par rapport à celle de la solution optimale (Sakarovitch, 1984). Il s'agit du :

- Rapport d'approximation standard de A sur I, connu par l'expression

$$\frac{A(I)}{OPT(I)}$$

- L'erreur absolue de A sur I, notée $r(A)$, et connue par l'expression

$$|A(I) - OPT(I)|$$

- L'erreur relative de A sur I, notée $R(A)$, et connue par les expressions

$$\frac{A(I) - OPT(I)}{OPT(I)} \text{ si } X \text{ est un problème de minimisation.}$$

$$\frac{OPT(I) - A(I)}{OPT(I)} \text{ si } X \text{ est un problème de minimisation.}$$

À présent, nous pouvons définir la notion de schéma d'approximation qui représente une famille d'algorithmes d'approximation pouvant être définis comme suit :

Définition. *Soit X un problème de minimisation (respectivement, maximisation).*

- *Un schéma d'approximation pour X est une famille d'algorithmes, notée A_ϵ, pour laquelle A_ϵ est un $(1+\epsilon) - approximation$ (respectivement, $(1-\epsilon) - approximation$) pour toute valeur de ϵ vérifiant $0 < \epsilon < 1$.*

- *A_ϵ est un schéma d'approximation polynomial PTAS (abréviation de, Polynomial Time Approximation Scheme) pour le problème X si A_ϵ est un schéma d'approximation de complexité polynomiale en la taille de l'instance sur laquelle il est appliqué.*

- *A_ϵ est un schéma d'approximation complètement polynomial FPTAS (abréviation de, Fully Polynomial Time Approximation Scheme) pour le problème X si en plus d'être un schéma d'approximation polynomial, sa complexité est également polynomiale en* $1/\epsilon$.[1]

Après ce rappel concernant les notions de bases nécessaires pour l'étude d'un problème d'ordonnancement, nous présentons dans le chapitre qui suit, une description du problème considéré dans cette étude. Sa définition sera suivie d'un état de l'art relatant les travaux étudiés durant notre phase de recherche.

1. Dans Schuurman & Woeginger (1999), les auteurs ont présenté un historique des travaux traitant des schémas d'approximation depuis leur apparition. Il semblerait que le premier travail comportant un PTAS revient à Graham (1966) pour l'étude d'un problème d'ordonnancement sur machines parallèles identiques. Plusieurs autres travaux, plus récents, ont également été cités.

2 Position du problème et état de l'art

Sommaire

2.1 Introduction **40**
2.2 Description mathématique du problème **41**
2.3 État de l'art **43**
2.3.1 Problèmes sans délais de transport 43
2.3.2 Problèmes avec délais de transport 51

2.1 Introduction

Dans la majorité des problèmes d'ordonnancement classiques, la préemption (lorsqu'elle est autorisée) n'engendre aucun coût financier ou temporel. Les tâches soumises à cette propriété pouvaient reprendre leur traitement aussitôt interrompu et ce même sur une autre machine. Bien que réelle, cette pénalité occasionnée suite à la préemption des tâches a été peu considérée par les chercheurs, et ce malgré son impact logique et avéré sur la durée et le coût des ordonnancements résultants. Dans ce chapitre, nous présentons une description formelle du problème étudié, à savoir

le problème d'ordonnancement d'un ensemble de tâches préemptives sur des ressources du type machines parallèles. Le problème est soumis à la contrainte relative aux temps de transport engendrés par les tâches préemptées. Dans notre cas, ces temps peuvent représenter entre autre la distance entre les machines.

La description du problème est suivie par un état de l'art comportant l'ensemble des travaux rencontrés durant notre étude en rapport avec la problématique énoncée.

2.2 Description mathématique du problème

Soit m machines parallèles et identiques, notées $M_1, M_2, \dots, M_m$, destinées au traitement d'un ensemble de n tâches indépendantes et morcelables, $T_1, \dots, T_n$. Chaque tâche T_j ($j = 1, \dots, n$) est caractérisée par sa durée opératoire p_j et sont toutes disponibles à l'instant $t = 0$. Chaque machine n'exécute qu'une seule tâche à la fois, et toute tâche ne peut être exécutée que par une seule machine en même temps. La préemption étant autorisée, ceci signifie que le traitement de chaque tâche T_j sur une machine M_i peut s'interrompre puis reprendre ultérieurement soit sur la machine M_i, soit sur une autre machine $M_{i'}$. Dans ce cas, la reprise de la tâche préemptée sur une autre machine ne peut se faire qu'après un certain temps que l'on note $d_{ii'}$, ce temps est appelé *délai de transport* des tâches préemptées entre les machines M_i et $M_{i'}$. Le transport des tâches se fait moyennant des ressources de transport (humaines ou matérielles) supposées disponibles à tout instant et de capacité illimitée. Rappelons que les délais de transport dépendent uniquement des distances entre les machines. L'objectif considéré dans ce travail est l'ordonnancement de l'ensemble des tâches tout en minimisant la durée totale de l'ordonnancement (makespan).

Conformément à la notation standard proposée par Graham *et al.* (1979), le problème étudié se note $P|pmtn(delay_{ii'})|C_{max}$. L'exemple suivant illustre une instance du problème étudié :

Exemple. Considérons une instance du problème $P3|pmtn(delay_{ii'})|C_{max}$ comportant un ensemble de 10 tâches dont les durées opératoires sont pré-

sentées dans la table 2.1.

T_j	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_j	2	4	3	5	2	3	1	2	5	3

TABLE 2.1 – Durées opératoires

0	2	3
6	0	5
4	1	0

TABLE 2.2 – Délais de transport

Les délais de transport sont détaillés dans la table 2.2. Ainsi, à titre d'exemple, le transport d'une tâche préemptée T_j depuis la machine M_1 vers la machine M_3 requiert trois unités de temps. La matrice n'est pas nécessairement symétrique, car dans notre exemple le transport depuis la machine M_3 vers la machine M_1 nécessite contrairement au chemin inverse quatre unités de temps.

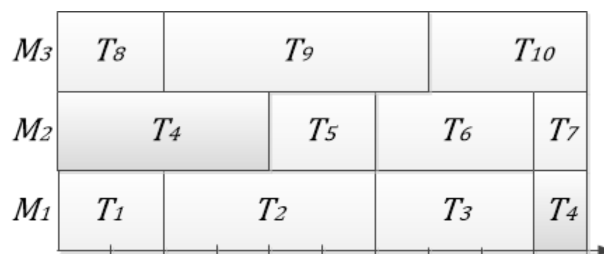


FIGURE 2.1 – Ordonnancement réalisable pour le $P3|pmtn|C_{max}$

La figure 2.1 représente un ordonnancement réalisable (et optimal aussi) pour le problème $P3|pmtn|C_{max}$ "sans considérer les délais de transport", et qui est de longueur $C_{max} = 10$. Nous pouvons constater que la tâche T_4 a été préemptée à l'instant $t = 4$ et a repris son traitement ultérieurement à l'instant $t = 9$, soit cinq unités de temps plus tard. Dans ce cas,

le délais de transport entre les machines M_2 et M_1, qui vaut six unités de temps, n'est pas respecté, ainsi, cette solution n'est pas réalisable pour le problème $P3|pmtn(delay_{ii'})|C_{max}$.

Si les temps de transport sont considérés, la solution précédente peut devenir réalisable, avec un $C_{max} = 11$, en effectuant un décalage de la deuxième partie de la tâche préemptée T_4 qui est exécutée sur la machine M_1. Le début de cette partie devrait, dans ce cas, avoir lieu à partir de l'instant $t = 10$ comme le montre la figure 2.2.

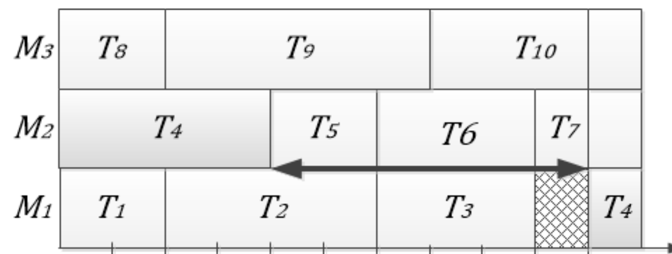


FIGURE 2.2 – Ordonnancement réalisable pour le $P3|pmtn|C_{max}$

2.3 État de l'art

Dans cette partie, nous présentons les principaux travaux rencontrés durant notre phase de recherche. Ces travaux concernent les problèmes d'ordonnancement d'ateliers principalement avec des machines parallèles en vue de minimiser le makespan. La première partie de cet état de l'art est dédiée aux problèmes classiques sans délais de transport entre les machines. La deuxième partie est consacrée aux problèmes d'ordonnancement avec prise en charge de ces délais de transport.

2.3.1 Problèmes sans délais de transport

Naturellement, le premier travail considéré dans cet axe de recherche a été l'étude du problème d'ordonnancement $P|pmtn|C_{max}$ menée par Robert McNaughton (McNaughton, 1959). Il s'agit d'affecter un ensemble de tâches à des machines parallèles identiques en vue de minimiser la durée totale. Le champ β indique que la préemption des tâches est autorisée.

L'auteur a montré qu'il est possible de résoudre le problème en temps polynomial ($O(n)$) en affectant les tâches (prises dans n'importe quel ordre) une par une en commençant par la première machine. Lorsque la charge sur la première machine atteint une valeur Δ, appelée borne de McNaughton et obtenue par le calcul de la formule $\max\{\frac{1}{m}\sum_{j=1}^{n} p_j; p_{max}\}$, deux cas de figures sont possibles. La première situation se présente lorsque la tâche en cours d'exécution achève son traitement à la date Δ, alors la tâche suivante est affectée à la machine d'après et le même processus est répété pour le reste des tâches sur les machines suivantes. Sinon, si le traitement de la tâche n'est pas terminé, ce dernier sera interrompu pour être ensuite repris au début de la machine suivante. Cette description résume les étapes de l'algorithme proposé et qui porte le nom de son auteur. Les ordonnancements optimaux résultants sont de longueur Δ.

L'académicien russe Vyacheslav S. Tanaev a été parmi les premiers à avoir étudié le problème précédent, plus précisément le $P2|pmtn|C_{max}$ et ses variantes (Tanaev, 1973). Cependant les objectifs de cette étude portaient sur les propriétés de la préemption occasionnée dans un ordonnancement optimal, à savoir la limitation du nombre total de préemptions et celle de l'ensemble des moments durant lesquels des préemptions peuvent intervenir.

En présence de contraintes supplémentaires sur les tâches, Cheng & Sin (1991) ont étudié le problème précédent où les tâches sont accompagnées par des dates échues ou des dates de fin au plus tard ("due date" en anglais)[1]. Cheng (1985) présente un algorithme capable de fournir (pour certaines instances) des solutions réalisables, c'est-à-dire, sans aucun retard enregistré. L'existence d'un tel algorithme repose sur certaines conditions posées par Sahni (1979) (probablement le premier à avoir proposé un tel algorithme appelé ONERT). Ainsi, en se basant sur les travaux de ce dernier, Cheng (1985) a mis en place un algorithme, qui, à partir d'un ordonnancement réalisable, réduit successivement les valeurs du C_{max} et régénère de nouvelles dates de fin de traitement, et ce, jusqu'à ce qu'aucune réduction ne soit possible. Cheng (1985) a montré que sur l'ensemble des instances testées (25 au total), 90% des instances résolues ont obtenues

1. Dans leur manuscrit, les auteurs ont adopté la notation suivante $N|M|parralel|C_{max}$.

de C_{max} considérablement meilleurs que ceux obtenus par Sahni (1979).

Suite aux résultats des travaux précédents, Cheng & Sin (1991) ont proposé un nouvel algorithme fonctionnant en deux étapes. Le principe de la première étape découle de l'algorithme de Sahni (1979) (précédemment cité) consacré à la génération d'un ordonnancement réalisable (appelé aussi DD-schedule) respectant toutes les dates échues des tâches. La deuxième phase vise à minimiser le C_{max} en utilisant un principe de recherche binaire. D'autres problèmes incluant des dates échues sont présentés dans (Cheng & Sin, 1990; Gordon *et al.*, 2002).

Dans ce même contexte, de meilleurs résultats ont été obtenus par l'algorithme de Cheng & Sin (1991) dans le cas où des ordonnancements réalisables existent. L'amélioration a été enregistrée suite à une réduction importante sur le nombre d'itérations. Les auteurs ont consacré la dernière partie de leur travail à l'analyse du nombre de préemptions ainsi que les coûts engendrés par ce dernier. Il est tout de même utile de mentionner que le premier travail sur machines parallèles en présence de dates échues revient à Horn (1974), qui a étudié le problème en vue de minimiser le retard maximum.

Il existe dans la littérature des problèmes d'ordonnancement avec une généralisation de la notion classique des dates échues. Lorsque le traitement d'une tâche doit être achevé dans un intervalle de temps donné (au lieu d'une date précise), nous parlons de problème d'ordonnancement avec des *fenêtres de temps échues*. Pour plus de détails sur la définition, la classification et la résolution de ses problèmes, nous renvoyons le lecteur intéressé au travail de synthèse effectué par Janiak *et al.* (2015) ainsi qu'aux états de l'art figurant dans les articles de Wu *et al.* (2015); Yue & Wan (2016); Liu *et al.* (2017).

Outre les dates échues, des études ont également porté sur les problèmes d'ordonnancement de tâches munies de dates de disponibilité ("release date" en anglais). Pour ces problèmes, Du *et al.* (1990) ont montré que lorsqu'il s'agit de minimiser la somme des dates de fin de traitement des tâches (soit le problème noté $Pm|pmtn, r_j|\sum C_j$), le problème est NP-difficile. Ce problème étant initialement posé par Lawler (1983). Baptiste *et al.* (2008) ont montré que pour des temps de traitement égaux, le pro-

blème $Pm|pmtn, r_j, p_j = p|\sum C_j$ devient polynomial, et ce, en proposant une formulation du problème en programme linéaire. Ils ont également étudié et fourni plusieurs bornes inférieures pour d'autres versions de ce cas, notamment le cas non préemptif $Pm|r_j|\sum C_j$ et sa version pondérée $Pm|r_j|\sum W_jC_j$.

Prot *et al.* (2013) se sont également penchés sur un problème combinant préemption et date de disponibilité dans le cas de la minimisation d'un critère régulier. Nous rappelons qu'un critère f est dit régulier, lorsque le coût d'une solution est une fonction croissante des dates de fin de traitement (Savourey, 2006; Baker, 1974). Pour ce problème, les auteurs ont réussi à étendre les résultats polynomiaux précédents (Baptiste *et al.*, 2008) pour d'autres problèmes (plus précisément d'autres critères d'optimisation, à savoir le retard total et le nombre de tâches en retard). Ils ont également fourni une structure d'ordonnancement dominante pour tous les problèmes de la forme $Pm|pmtn, r_j|f$. Ces problèmes admettent des solutions optimales dans lesquelles les dates de fin de traitement conservent le même ordre que les dates de disponibilité.

La résolution du $Pm|pmtn, r_j|\sum C_j$ a récemment été prise en charge par Liaw (2016) en proposant d'abord une heuristique efficace destinée à résoudre les instances pratiques de grande taille. L'heuristique a été utilisée, par la suite, pour calculer des valeurs de bornes supérieures pour un algorithme exact du type branch and bound. Cette dernière a permis de résoudre des instances de 11 tâches exécutées sur 4 machines en un temps acceptable.

Certains chercheurs se sont intéressés à la présence de relations de dépendance entre des parties de différentes tâches préemptées. En effet, introduit par Muntz & Coffman (1969), le problème présente des contraintes de précédence entre des parties de deux tâches différentes, les parties d'une même tâche préemptée sont quant à elles indépendantes. Ces contraintes de précédence, lorsqu'elles existent, sont représentées par un graphe orienté. Les auteurs ont développé un algorithme polynomial pour le problème sur deux machines s'exécutant en $O(n^2)$ et produisant exactement nm préemptions. Lorsque le nombre de machines s'élève à trois, un algorithme a également été présenté fournissant de "*bons*" ordonnancements mais sans aucune garantie quant à la valeur du C_{max}.

Lorsque le graphe de précédence présente une structure de forêt, Gonzalez & Johnson (1980) ont introduit l'algorithme *Fast schedule by weight* afin de comparer leurs résultats avec ceux obtenus pour un graphe quelconque (Muntz & Coffman, 1969). Ce nouvel algorithme, plus rapide, s'exécute en $O(nlogm)$ et produit des ordonnancements contenant au plus $n-2$ préemptions. Une version légèrement modifiée de cet algorithme appelée *critical weight algorithm*, a également été proposée pour un nombre de machine $m \geq 2$. Cette dernière s'opère en $O(nm)$ et produit des ordonnancements avec $2nm - 2n - m + 2$ préemptions au total.

Les problèmes présentés jusque-là sont presque tous accompagnés par des résultats positifs, c'est-à-dire, des algorithmes polynomiaux ont pu être conçus pour leurs résolutions. Cependant, les problèmes pour lesquels de tels résultats n'ont pu être développés sont aussi nombreux . En effet, suite aux travaux précédents, Ullman (1975) a étudié quelques problèmes d'ordonnancement NP-$complets$. En particulier, il a montré que le cas général du problème d'ordonnancement sur m machines parallèles en présence de contraintes de dépendance entre les tâches préemptives est dans NP, y compris entre des tâches unitaires ou encore des tâches avec des temps de traitement appartenant à l'ensemble $\{1, 2\}$. L'auteur a aussi classifié les problèmes qu'il a étudiés en présentant des transformations polynomiales à partir de problèmes d'ordonnancement connus et prouvés être $Np - complet$ ainsi qu'à partir d'autres problèmes connus dans la littérature à l'exemple du problème de 3-SAT (Garey & Johnson, 2002).

Lorsqu'une tâche est préemptée, ces différentes portions ne peuvent être exécutées en même temps sur deux machines différentes (Leung, 2004). Cette hypothèse a été levée pour donner lieu à une nouvelle possibilité d'ordonnancement incluant des tâches dites *split jobs*. La notation du problème est passée de $P|pmtn|Cmax$ à $P|split|Cmax$ pour les machines parallèles. Cette notion est parfaitement envisageable lorsque par exemple le traitement de la tâche représente la demande d'un produit. Un cas réel présent dans un atelier de textile a été présenté et étudié par Serafini (1996). Les auteurs ont présenté une description de l'atelier concerné implanté au nord de l'Italie et dans lequel il est question de fabriquer plusieurs types de tissu. L'objectif principal pour ces producteurs est de respecter au maximum les dates échues de fabrication en sachant que les différents produits

ne sont pas tous de la même importance. Afin de répondre aux préoccupations du client, les auteurs ont présenté plusieurs formulations et modélisations en des problèmes d'ordonnancement considérant différents objectifs. Xing & Zhang (2000) ont présenté des algorithmes polynomiaux pour le cas des machines parallèles avec des *split jobs* et ce pour plusieurs critères : C_{max}, L_{max}, $\sum C_j$, $\sum U_j$, $\sum T_j$, $\sum w_j C_j$, $\sum w_j U_j$. Le problème devient $NP\text{-}difficile$ en injectant aux instances des temps de préparations (setup times en anglais) relatives aux tâches (Xing & Zhang, 1998). La résolution de ce dernier a fait l'objet de récentes recherches à commencer par l'heuristique de Xing & Zhang (2000), la méthode SearchCol[1] de Florêncio (2013) et les méthodes hybrides de Wang *et al.* (2013). Pour plus de détails, nous invitons le lecteur à consulter également les travaux suivants : Florêncio *et al.* (2015); Wiens & Ullrich (2016); Wang *et al.* (2016); Liu *et al.* (2018).

Lorsque la préemption des tâches n'est pas autorisée, nous parlons du problème classique $Pm||C_{max}$, qui pour $m = 2$ est NP-difficile (Karp, 1972). Lorsque le nombre de machines n'est pas fixé, le problème $P||C_{max}$ est NP-difficile au sens fort (Garey & Johnson, 1975). Ce problème peut se présenter sous la forme d'un programme linéaire en nombre entier en considérant une variable x_{ij} qui prend la valeur 1 si la Tâche T_j est exécutée sur la machine M_i et prend la valeur 0 dans le cas contraire. Le programme peut s'écrire de la manière suivante :

$$
\begin{array}{llll}
min & C_{max} & & \\
s.c. & \sum_{j=1}^{n} x_{ij} p_j \leq & C_{max} & i = 1, \ldots, m \\
 & \sum_{i=1}^{m} x_{ij} = & 1 & j = 1 \ldots, n \\
 & & x_{ij} \in \{0, 1\} \quad i = 1, \ldots, m & j = 1 \ldots, n
\end{array}
$$

Il est à retenir que, de nos jours, le $P||C_{max}$ représente un problème classique voir même de référence pour un bon nombre de problèmes d'or-

1. Le principe de la méthode repose sur la résolution d'un programme linéaire à variables mixtes par la décomposition de Dantzing-wolfe. La résolution des sous programmes engendrés se fait par l'hybridation de deux méthodes de résolution : la génération de colonnes et la recherche par les métaheuristiques MIP et VNS.

donnancement plus complexes ou plus récents, et ceci dans le sens où les bornes développées pour le $P||C_{max}$ restent valides et sont utilisées pour les autres problèmes.

Le $P||C_{max}$ est aussi utilisé pour illustrer des problèmes de la vie réelle même si en apparence ces problèmes décrivent des situations qui semblent être assez éloignées des problèmes d'ordonnancement. Comme c'est le cas pour le problème des m voleurs qui se décrit comme suit : un groupe formé de m voleurs s'organisent la nuit pour cambrioler des villas. Les objets volés (d'une valeur c_j chacun) sont partagés équitablement en fin d'opération. Cette situation peut se formuler comme un $P||C_{max}$ si l'on considère les m voleurs comme étant les m machines identiques, et les objets volés comme étant les n tâches à ordonnancer. Le partage du butin est considéré équitable si le makespan est minimisé. Bien évidemment, si les voleurs ne souhaitent pas être capturés, le partage doit se faire en utilisant l'algorithme le plus rapide possible (Houcine, 2006).

Pour la résolution du problème général, nous citons la plus ancienne des études qui revient à Rothkopf (1966), dans laquelle il a proposé un programme dynamique d'une complexité de $O(nB^m)$, où B est une borne supérieure pour le C_{max} [1]. Un algorithme du même type a également été proposé par Schuurman & Woeginger (2007) pour la résolution exacte du cas de deux machines.

Le coût des méthodes exactes conçues pour résoudre le $P||C_{max}$ a poussé les chercheurs à se tourner naturellement vers une résolution approchée. Les méthodes de ce type ont suscité un réel intérêt pour le problème en question. Les heuristiques les plus fréquentes sont celles comportant des Algorithmes de Liste plus communément appelés : les heuristiques FAM (pour First Available Machine). Ces méthodes, dont le principe est d'affecter des tâches se présentant suivant un ordre de priorité donné, une par une à la première machine disponible, sont connues par leurs rapport de performance $\frac{C_{max}(FAM)}{C_{max}(opt)} \leq (2 - \frac{1}{m})$ au plus mauvais cas. Ce rapport devient inférieur ou égal à $(\frac{4}{3} - \frac{1}{3m})$ lorsque les tâches sont ordonnées selon la règle LPT (par ordre décroissant des durées opératoires) (Graham *et al.*, 1979).

1. Une description de ce programme dynamique est présente dans Błażewicz (1987).

Une autre famille d'heuristique a été utilisée à cette même fin, il s'agit des algorithmes multifit (Alvim & Ribeiro, 2004; Dell'Amico *et al.*, 2008) qui tirent leur principe du problème du sac à dos (Kellerer *et al.*, 2004). Les tâches sont ordonnées suivant la règle FFD (First Fit Decreasing) qui affecte la plus grande tâche (ayant la plus grande durée opératoire) à la première machine pouvant la recevoir (en tenant compte de la capacité restante de la machine). Pour plus de détails sur le principe et l'application de ces algorithmes ainsi que sur l'analyse de leurs rapports de performance, nous invitons le lecteur à consulter les études effectuées par Friesen & Langston (1986); Yue (1990); Chang & Hwang (1999), et particulièrement sur celle présentée récemment par Laha & Behera (2017) qui passe en revue les méthodes précédentes ainsi que d'autres combinaisons, et leurs évaluations pour le problème d'ordonnancement sur machines parallèles. Une hybridation de plusieurs anciennes méthodes a été présentée bien avant (Lee & Massey, 1988), et a fourni de meilleurs résultats que ceux obtenus par l'exécution de chaque méthode individuellement.

En plus des deux types de méthodes précédemment citées (algorithmes de liste et algorithmes multifit) considérées comme étant des heuristiques constructives, d'autres heuristiques d'amélioration ont été présentées par Ho & Wong (1995) et França *et al.* (1994).

Concernant les métaheuristiques, Davis (1987) a proposé une adaptation du recuit simulé dont les résultats ont été comparés (plusieurs années après) à ceux obtenus par Min & Cheng (1999) grâce à son adaptation d'une variante d'un algorithme génétique. La variante de l'algorithme génétique proposée s'est avérée plus adaptée pour la résolution des instances du $P||C_{max}$ de taille importante. Toutefois, les résultats les plus serrés sont proposés par Hochbaum & Shmoys (1987) en proposant un schéma d'approximation polynomial obtenu par l'application de nouveaux types d'algorithme que les auteurs ont appelé : Algorithmes à double approximation (dual approximation algorithms). Un schéma d'approximation complètement polynomial a été construit auparavant par Sahni (1976) pour ce même problème en fixant le nombre de machines m (le problème étant NP-difficile au sens faible). D'autres algorithmes, plus récents, sont aussi présents dans l'étude de Costa *et al.* (2002), mettant en

oeuvre un Système Immunitaire Artificiel[1]. Plus précisément, ce nouvel algorithme de recherche est basé sur le fonctionnement des vertèbres chez les animaux.

Il existe dans la littérature une généralisation du problème $P||C_{max}$, il s'agit du problème noté $P|\# \leq k|C_{mac}$. Cette variante comporte une contrainte additionnelle sur le nombre de tâches pouvant être assignées à la même machine et qui est au plus égal au paramètre k. Selon Dell'Amico & Martello (2001), le $P|\# \leq k|C_{mac}$ est NP-difficile pour $k > 2$. Tandis que pour $k = 2$, le problème est solvable en $O(nlogn)$. Une formulation mathématique, des bornes inférieures ainsi que plusieurs algorithmes de résolution sont proposés par Dell'Amico *et al.* (2004).

2.3.2 Problèmes avec délais de transport

Comme il a été mentionné au tout début, la reprise d'une tâche préemptée sur une autre machine pouvait se faire à l'instant même de son interruption. Il semblerait que Rayward-Smith (1987) ait été le premier à avoir considéré des temps additionnels à respecter lorsqu'une tâche subit une préemption. L'étude présentée a été développée dans un contexte informatique (i.e. où il est question d'optimiser l'exécution des tâches par des processeurs), dans lequel ces temps sont appelés : *temps de communication*. L'auteur a montré que pour des temps de communication unitaires, le problème peut se résoudre en un temps polynomial en proposant un algorithme de résolution de la même complexité. En revanche, pour des délais supérieurs ou égaux à deux, le problème est lui NP-*complet*[2].

En 2005, Fishkin *et al.* (2005) se sont intéressés à cette problématique telle que nous la décrivons. Pour les auteurs, la préemption d'une tâche puis son déplacement vers une autre machine est appelé une *migration*. Dans cette étude, les auteurs ont considéré le cas de délais de migration identiques (soient, égaux à une valeur d). Pour une instance donnée, les auteurs ont borné, des deux côtés, de la longueur d'un ordonnancement optimal en utilisant Δ, la borne inférieure de McNaughton (McNaughton,

1. Pour plus de détails sur les SIA(s) notamment appliqués aux problèmes d'ordonnancement, se référer aux travaux de Mori *et al.* (1997); Hofmeyr & Forrest (2000) et de Dasgupta (2012).
2. Preuve établie en utilisant le problème de 3-partitions.

1959) pour la même instance du problème en omettant la notion de transport (c'est-à-dire l'instance correspondante du $P|pmtn|C_{max}$). Il a également été démontré que, pour des valeurs de délai de transport inférieures ou égales à $\Delta - p_{max}$, aussi, pour des valeurs de délai de transport inférieures ou égales à $\Delta - p_{max}$, le problème devient polynomial et se résout en $O(n)$ en utilisant l'algorithme de McNaughton. En plus des algorithmes polynomiaux présentés pour certains sous problèmes, l'étude présente une preuve de complexité pour des sous problèmes dans lesquels la valeur de d est minorée par celle de $(1+\epsilon)*(\Delta-p_{max})$. Pour ces mêmes valeurs de d, les auteurs ont aussi montré, en utilisant une réduction à partir du problème α-partition, que le problème étudié est NP-$difficile$ au sens fort pour toute constante $\epsilon > 0$.

Les auteurs ont également attiré l'attention du lecteur quant à une équivalence entre le problème d'ordonnancement étudié (en présence de délais de migration) et un autre problème d'ordonnancement non préemptif. En effet, il s'agit du problème d'ordonnancement sur machines parallèles avec des tâches unitaires et des contraintes de précédence entre les tâches. Ce problème (noté $P|prec, p_j = 1, c_{jk} = d|C_{max}$) est une version équivalente du problème initial. Si l'on considère que la contrainte de précédence existe entre deux tâches T_j et T_k, alors la tâche T_k ne peut commencer son traitement qu'après la fin de traitement de la tâche T_j, et au minimum après c_{jk} unités de temps si les deux tâches sont exécutées sur deux machines différentes. L'intérêt d'avoir montré la relation entre ces deux problèmes revient aux nombreux travaux traitant le $P|prec, p_j = 1, c_{jk} = 1|C_{max}$, et notamment en matière d'algorithmes d'approximation établis pour sa résolution (Hoogeveen *et al.*, 1994; Schuurman & Woeginger, 1999; Hanen & Munier, 2001; Engels *et al.*, 2001; Afrati *et al.*, 2005).

Les mêmes auteurs se sont également intéressés à l'existence d'ordonnancements optimaux et à leurs propriétés pour les problèmes d'ordonnancement avec deux et trois machines. En effet, il a été démontré que pour toute instance du problème à deux machines, il existe un ordonnancement optimal avec au plus une migration. Ce cas se caractérise soit par l'absence de préemption, soit par la préemption d'une seule tâche, et ce, une seule fois. Une telle tâche est donc scindée en deux parties qui sont exécutées sur les extrémités de deux machines. De même, les auteurs ont également prouvé que pour le cas de trois machines, il existe un ordon-

nancement optimal avec au plus deux migrations. Dans ce cas, les deux migrations peuvent être effectuées soit par une seule tâche exécutée sur les trois machines, soit par deux tâches différentes et qui sont chacune exécutée sur deux machines différentes.

Suite aux résultats précédents, deux résultats concernant l'existence d'ordonnancements optimaux ont figuré dans la même étude. En effet, une conjecture a été établie pour le problème à m machines. Dans ce cas, un ordonnancement optimal existerait pour toute instance du $P|pmtn(delay_{ii'} = d)|C_{max}$ avec au plus $(m-1)$ migrations. De plus, il a été montré que pour toute instance de ce problème, il existe un ordonnancement optimal dans lequel aucune machine ne présente un temps mort pendant son fonctionnement, et ce, jusqu'à la fin de traitement de l'ensemble des tâches .

En 2009, Boudhar & Haned (2009) ont présenté leur premier travail traitant explicitement le problème d'ordonnancement préemptif sur machine parallèles en présence de délais de transport variables. Les auteurs ont précisé que les temps à respecter lors de la préemption d'une tâche dépendaient du transport de celle-ci et plus précisément de la distance entre les machines. La complexité du problème a été abordée dans ce travail en montrant que le problème sur deux machines était NP-$difficile$. Une formulation mathématique linéaire a également été proposée ainsi que des heuristiques de résolution.

En se penchant d'avantage sur cette même problématique, Les auteurs se sont par la suite intéressés à l'étude de cas particuliers. De nouveaux résultats de complexité ont été présentés. En ce qui concerne la résolution, le problème sur deux machines a été traité par l'élaboration d'un programme dynamique. Un algorithme d'approximation complètement polynomial a également été proposé à cette même fin (Haned *et al.*, 2012; Haned, 2012). Quant à la résolution approchée du problème cité par des métaheurisiques, le lecteur peut se référer aux premières adaptations proposées par Haned et al. dans les références suivantes : Haned *et al.* (2011); Haned & Boudhar (2013a,b).

Par la suite, Shams & Salmasi (2014) ont étendu les travaux précédents en discutant quelques hypothèses sur lesquelles a été basée la formulation mathématique de Boudhar & Haned (2009). Les auteurs pré-

sentent une preuve de la validité de la conjecture posée dans Fishkin *et al.* (2005) concernant le problème général sur m machines. Shams et Salmasi ont également proposé une formulation mathématique linéaire du problème d'ordonnancement préemptif en présence de délais de transport identiques ainsi qu'un algorithme exact pour la résolution de petites instances. L'étude comporte à sa fin une comparaison des performances des deux approches proposées.

Il est vrai que notre recherche s'est basée sur trois axes principaux qui sont : les machines parallèles, la préemption et les délais de transport. Cependant, la notion de délais entre les parties des tâches préemptées (plus précisément entre les opérations d'une même tâche) est apparue à nous dans d'autres types d'ateliers d'ordonnancement. Il est question des ateliers d'ordonnancement avec machines dédiées (ou spécialisées). Nous allons présenter brièvement dans le paragraphe suivant quelques travaux rencontrés.

La plus ancienne étude croisée revient à Maggu & Das (1980). Cette étude traite un flow-shop à deux machines en vue de minimiser le makespan en ne considérant aucune limite relative au nombre de transporteurs. Les auteurs ont proposé une généralisation de l'algorithme de Johnson (Johnson, 1954) pour résoudre le problème. Un flow-shop a aussi été abordé par Soukhal & Martineau (2005), et pour lequel un modèle mathématique en nombre entier a été proposé ainsi qu'une adaptation de l'algorithme génétique pour la résolution d'instances de grande taille. En incluant la notion de transporteur et en le limitant à un seul, un flow-shop a été traité par Stern & Vitner (1990). Pour plus de détails, le lecteur est invité à consulter la section état de l'art de Lee & Chen (2001).

3

Le cas de trois machines

Sommaire

3.1 Introduction **55**
3.2 Programme dynamique **57**
3.3 Résolution pseudo-polynomiale **59**
3.4 Schéma d'approximation complètement polynomial . . **64**
3.5 Expérimentation numérique **69**

3.1 Introduction

Dans ce chapitre, nous présentons les travaux réalisés pour la résolution d'un cas particulier du problème d'ordonnancement préemptif avec délai de transport, en l'occurrence le cas sur trois machines, et ce, en considérant des temps de transport identiques. Le problème en question se note $P3|pmtn(delay_{ii'} = d)|C_{max}$ où d représente la valeur du délai nécessaire au transport d'une tâche préemptée de la machine M_i vers la machine $M_{i'}$. Le contenu de ce chapitre fait suite aux travaux de Haned *et al.* (2012) ayant abordé le cas du problème étudié sur deux machines.

Fishkin et al (Fishkin *et al.*, 2005) se sont intéressés au problème noté $P|pmtn(delay_{ii'} = d)|C_{max}$. Il s'agit donc du problème d'ordonnancement de tâches préemptives sur machines parallèles avec des délais de transport qui sont identiques et de valeurs d. Suite à une préemption, le transport d'une tâche est appelé par les auteurs *migration* et les délais de transport sont appelés *temps de migration* .

Pour le cas de deux et de trois machines, les auteurs ont présenté les résultats suivants :

Théorème 3.1.1 (Fishkin et al.)**.** *Pour toute instance du $P2|pmtn(delay_{ii'} = d)|C_{max}$, il existe un ordonnancement optimal avec au plus une migration.*

Théorème 3.1.2 (Fishkin et al.)**.** *Pour toute instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$, il existe un ordonnancement optimal avec au plus deux migrations.*

Suite à cela, Haned *et al.* (2012) se sont intéressés à l'étude du premier cas cité, c'est-à-dire celui sur deux machines en considérant des délais de transport variables[1]. Les auteurs ont présenté une méthode de résolution reposant sur l'implémentation d'un programme dynamique destiné à résoudre le $P2||C_{max}$, à partir de laquelle nous nous sommes inspirés pour la résolution du sous problème que nous considérons.

Afin d'exploiter la caractérisation de Fishkin *et al.* (2005), et en s'inspirant du travail de Haned *et al.* (2012), nous avons tenté d'étendre les résultats obtenus pour le cas de deux machines au problème $P3|pmtn(delay_{ii'} = d)|C_{max}$, cette volonté a donné lieu au contenu de ce chapitre. Dans ce dernier, nous proposons un algorithme pseudo polynomial pour la résolution du $P3|pmtn(delay_{ii'} = d)|C_{max}$. Cet algorithme inclut l'exécution d'un programme dynamique dont la description et le fonctionnement seront expliqués par la suite. À la fin de ce chapitre, nous proposons un algorithme d'approximation pour résoudre le sous problème considéré.

1. Bien que l'existence d'un ordonnancement optimal avec au plus une migration ait été démontré pour des temps de transport identiques, les auteurs ont étendu leur étude en considérant des délais de transport variables. Il est facile de démontrer, pour le cas de deux machines, qu'un tel ordonnancement optimal existe, ce qui n'est pas le cas pour le problème à trois machines.

3.2 Programme dynamique

Dans cette partie, nous décrivons un programme dynamique que l'on peut utiliser pour résoudre des instances du problème $P3||C_{max}$ et qui, rappelons-le, est un problème NP-$difficile$ (Karp, 1972). Le problème dynamique, noté DP, repose sur le calcul d'une fonction principale $F(n, P_1, P_2)$ relative à la meilleure affectation d'un ensemble de n tâches aux machines M_1, M_2 et M_3 tel que P_1, P_2 représentent respectivement les charges sur les machines M_1 et M_2. Par charge sur les machines M_1 et M_2, nous entendons la somme des temps de traitement des tâches exécutées sur ces deux machines respectivement. Le calcul récursif de la valeur de cette fonction principale conduit à la connaissance (à chaque instant) de l'état des machines et de leurs charges menant à la fin vers la détermination de la valeur de la fonction objectif C_{max} du problème $P3||C_{max}$.

Le programme dynamique DP est décrit par la formulation suivante :

$$F(j, P_1, P_2) = \begin{cases} 0 & j = 0 & \text{et} & P_1 \geq 0 \text{ et } P_2 \geq 0 \\ \sum_{k=1}^{j} p_k & j > 0 & \text{et} & P_1 = 0 \text{ et } P_2 = 0 \\ +\infty & j \in \{1, .., n\} & \text{et} & P_1 < 0 \text{ ou } P_2 = 0 \\ +\infty & j \in \{1, .., n\} & \text{et} & P_1 > P \text{ ou } P_2 > P \\ +\infty & j > n & \text{et} & \forall P_1 \text{ et } \forall P_2 \\ F_1 & j \in \{1, .., n\} & \text{et} & P_1 = 0 \text{ et } P_2 \in \{1, .., P\} \\ F_2 & j \in \{1, .., n\} & \text{et} & P_1 \in \{1, .., P\} \text{ et } P_2 = 0 \\ F & j \in \{1, .., n\} & \text{et} & P_1 \in \{1, .., P\} \text{ et } P_2 \in \{1, .., P\} \end{cases}$$

Où :

$$F_1 = min\{F(j-1, 0, P_2 - p_j); F(j-1, 0, P_2)\} + p_j$$
$$F_2 = min\{F(j-1, P_1 - p_j, 0); F(j-1, P_1, 0) + p_j\} + p_j$$
$$F = min\{F(j-1, P_1 - p_j, P_2); F(j-1, P_1, P_2 - p_j); F(j-1, P_1, P_2) + p_j\}$$

P est une borne supérieure pour le problème (par exemple, $P = \sum_{k=1}^{n} p_k$).

Plus explicitement, chaque état de la fonction récursive $F(j, P_1, P_2)$ représente la meilleure affectation d'une tâche T_j à l'une des trois machines.

Ainsi, pour trouver la meilleure affectation de la tâche en question, il suffit de calculer la fonction $F(j, P_1, P_2)$ pour toutes les valeurs possibles de P_1 et de P_2. Par conséquent, pour affecter une tâche T_j, il est question d'effectuer un des trois choix suivants :

- La tâche T_j est affectée à la machine M_1, si la meilleure valeur de $F(j, P_1, P_2)$ correspond à celle obtenue par $F(j-1, P_1 - p_j, P_2)$.
- La tâche T_j est affectée à la machine M_2, si la valeur de la fonction principale correspond à la valeur $F(j-1, P_1, P_2 - p_j)$.
- De même, la tâche T_j est assignée à la troisième machine, si la valeur de la fonction principale $F(j, P_1, P_2)$ est obtenue par la valeur $F(j-1, P_1, P_2) + p_j$.

Notons que lorsque $P_1 = 0$, le traitement de la tâche T_j est assigné soit à la machine M_2 avec une valeur de la fonction objectif obtenue à partir de $F(j-1, 0, P_2 - p_j)$, ou bien à la machine M_3 si la valeur de l'expression $F(j-1, 0, P_2)\} + p_j$ est meilleure (donc plus petite). Le même principe est bien évidemment considéré pour le cas où $P_2 = 0$.

La longueur de l'ordonnancement optimal est obtenue après avoir calculé toutes les valeurs possibles de la fonction dynamique F pour l'ensemble des n tâches et pour toutes les valeurs possibles de P_1 et P_2. Cette valeur est déterminée par l'expression suivante :

$$C_{max}* = \min_{\substack{1 \leq P_1 \leq P \\ 1 \leq P_2 \leq P}} \{max\{P_1, P_2, F(n, P_1, P_2)\}\}$$

.

Quant à l'ordonnancement optimal, il peut être obtenu à partir de cette valeur optimale du $C_{max}*$ en utilisant une procédure de retour en arrière. Toutes les affectations des tâches sont obtenues en pistant toutes les valeurs de la fonction $F(n, P_1, P_2)$ ayant menées à cette valeur optimale du C_{max}. Ce programme dynamique permet d'obtenir un ordonnancement optimal pour l'instance du problème sur trois machines sans préemption en $O(nP^2)$. Un exemple démonstratif qui permet de mieux visualiser le fonctionnement de ce programme dynamique pour une instance du problème $P_3||C_{max}$, est présenté en Annexe (voir 5.7).

3.3 Résolution pseudo-polynomiale

Après la description du programme dynamique précédent, nous proposons dans cette partie une approche de résolution pour le problème $P3|pmtn(delay_{ii'} = d)|C_{max}$. Cette approche, que nous avons nommée $DPX3$ et qui s'inspire de l'algorithme DPX (Haned *et al.*, 2012), inclut l'utilisation du programme dynamique DP qui sera appliqué sur un ensemble de $(n-1)$ ou de $(n-2)$ tâches. À cette étape, une nouvelle instance du problème initialement considéré (à savoir le $P3|pmtn(delay_{ii'} = d)|C_{max}$) est créée à partir de ces $(n-1)$ ou $(n-2)$ tâches. Cette instance correspond à une instance du problème $P3||C_{max}$ dans laquelle ni la préemption ni les délais de transport ne sont considérés. Le principe de l'algorithme proposé est le suivant :

Chaque itération de l'algorithme $DPX3$ correspond à la sélection de : soit une tâche T_j, soit deux tâches T_j et $T_{j'}$. Les tâches en questions sont sélectionnées en vue d'être exclues provisoirement de l'ensemble des tâches T. À partir de cela, l'exécution de cette méthode pour chaque cas de figure est expliquée ci-dessous :

Une tâche T_j est sélectionnée Une fois cette tâche sélectionnée, le programme dynamique DP est appliqué à l'ensemble des $(n-1)$ tâches restantes. Le résultat du programme dynamique DP correspond à un ordonnancement optimal des tâches pour lesquelles il est appliqué (c'est-à-dire $T - \{T_j\}$). L'ordonnancement optimal du problème $P3|pmtn(delay_{ii'} = d)|C_{max}$ est obtenu par l'ajout de la tâche T_j à l'ordonnancement partiel dont le C_{max} est fourni par le programme dynamique DP. L'injection de la tâche ignorée T_j à cette itération se fait suivant la caractérisation de l'ordonnancement optimal de Fishkin *et al.* (2005) (voir figure 3.1).

Les tâche T_j et $T_{j'}$ sont sélectionnées Le même principe précédent est appliqué lorsque deux tâches sont ignorées. Le programme dynamique DP est appliqué sur l'ensemble des $(n-2)$ tâches restantes. Le résultat du programme dynamique DP correspond à un ordonnancement optimal des tâches pour lesquelles il est appliqué (soit $T - \{T_j, T_{j'}\}$). L'ordonnancement optimal du problème $P3|pmtn(delay_{ii'} = d)|C_{max}$

est obtenu par l'ajout des tâche T_j et $T_{j'}$ à l'ordonnancement partiel dont le C_{max} est fourni par le programme dynamique DP. De la même manière, l'injection de ces deux tâches ignorées à cette itération se fait suivant la caractérisation de l'ordonnancement optimal de Fishkin *et al.* (2005) (voir figure 3.2).

Le procédé de sélection des tâches à ignorer est appliqué tantôt pour toute tâche T_j de l'ensemble des tâches T, puis pour toutes les combinaisons de tâches $\{T_j, T_{j'}\}$ du même ensemble de tâches.

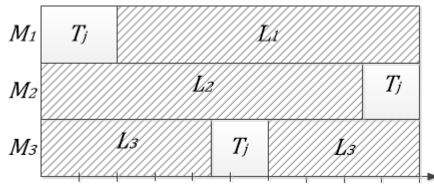


FIGURE 3.1 – Cas 1 : une tâche T_j est ignorée

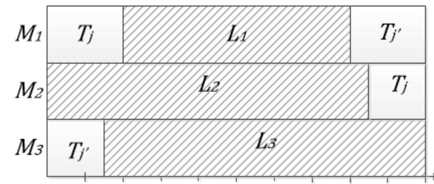


FIGURE 3.2 – Cas 2 : deux tâches T_j, T_j' sont ignorées

Dans les figures précédentes, L_1, L_2 et L_3 représentent respectivement les charges sur les machines M_1, M_2 et M_3. Par la charge L_i, nous désignons la somme des temps de traitements des tâches exécutées entièrement sur la machine M_i. Sans perte de généralité, il a été assumé que pour ces valeurs : $L_1 \leq L_2 \leq L_3$. À cette étape, les machines sont prêtes à accueillir les tâches sélectionnées selon l'ordre suivant : M_2, M_1 puis M_3.

Dans le 1er cas, la tâche T_j commence son traitement à l'instant $t = 0$ sur M_1, puis elle est transportée sur M_2 où elle est exécutée à partir de $t = L2$. Le reste de la tâche T_j (s'il existe) est exécuté sur la machine M_3 après exactement d unités de temps de la part précédente. Les tâches déjà en place subissent éventuellement le même type de décalage.

Dans le cas où deux tâches T_j et $T_{j'}$ sont sélectionnées, la tâche avec la plus grande durée opératoire est injectée en premier lieu à la fin de la machine M_2 puis au début de la machine M_1. Après le décalage des tâches de M_1, la première partie de la tâche $T_{j'}$ est exécutée à la fin de la machine M_1 et le reste (s'il existe) au début de la machine M_3 en prévoyant le décalage

des tâches déjà affectées à la machine M_3. Pour plus de détails sur le principe de ce procédé, nous invitons le lecteur à consulter la preuve fournie par Fishkin *et al.* (2005).

Le raisonnement précédent a donné lieu à un algorithme de résolution pour le problème initialement étudié, à savoir le $P3|pmtn(delay_{ii'} = d)|C_{max}$, et qui sera présenté ultérieurement. Avant de le présenter, nous attirons l'attention du lecteur qu'il est parfaitement possibles que des solutions optimales soient rencontrées par une application simple de la règle de McNaughton dont les étapes ont été décrites dans la partie 2.3.1. En effet, il est facile de voir que les instances pour lesquelles $d \leq \Delta - p_{max}$, où $p_{max} = \max_{j=1,n}\{p_j\}$, sont résolues de manière optimale en temps polynomial donnant ainsi lieu à des ordonnancements de longueur Δ[1]. Dans ce cas, nous avons utilisé une simple procédure *"test"* dont l'exécution précède celle de l'algorithme $DPX3$. Les méthodes que nous venons de citer nous amènent aux énoncés des deux algorithmes 1 et 2.

Algorithme 1: *Pseudo-algorithmes test*

Entrées : une instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$

1 Calculer la valeur de la borne inférieure de McNaughton $\Delta = max\{\frac{1}{3}\sum_{i=1}^{n} p_j; p_{max}\}$;

2 **si** $\Delta - p_{max} \geq d$ **alors**

3 La solution de McNaughton est optimale;

4 Arrêt de la procédure;

5 **si** $\Delta - p_j \geq d_{ii'}$, *pour toute tâche T_j préemptée et transportée de M_i à $M_{i'}$* **alors**

6 La solution de McNaughton est optimale ;

Pour certaines instances du $P3|pmtn(delay_{ii'} = d)|C_{max}$, l'algorithme *test* présente une résolution efficace en fournissant des ordonnancements

1. Cette propriété est très simple à vérifier. Il s'agit de considérer le pire des cas, c'est-à-dire celui impliquant la tâche avec la plus grande durée opératoire. Le délai de transport étant respecté en cas de préemption de celle-ci suivant la règle de McNaughton. L'ordonnancement est par conséquent optimal.

optimaux par la simple application de l'algorithme de McNaughton. Bien évidemment nous nous intéressons peu à de telles instances. Pour les autres instances, plus intéressantes à résoudre, les étapes de résolution sont décrites par l'algorithme $DPX3$.

Algorithme 2: *Algorithme $DPX3$*

Entrées : Une instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$
Sorties : Un Ordonnancement des n tâches sur trois machines

1 Appliquer l'heuristique FAM à l'instance du $P3||C_{max}$ correspondante. Soit C_{max}^{FAM} la longueur de cet ordonnancement;
2 Poser, $P1 = P2 = C_{max}^{FAM}$;
3 Résoudre le problème $P3|\,|C_{max}$ par DP;
4 **pour** $j \leftarrow 1$ **à** n **faire**
5 Considérer l'ensemble de tâches $T'=T-\{T_j\}$;
6 Résoudre le Problème $P3|\,|C_{max}$ avec T' en utilisant le programme dynamique DP;
7 Trouver la meilleure solution pour le problème $P3|pmtn(delais_{ii'} = d)|C_{max}$, en intégrant la tâche T_j;
8 Choisir la meilleure solution parmi les n solutions résultantes;
9 **pour** $j \leftarrow 1$ **à** n **faire**
10 **pour** $j' \leftarrow 1$ **à** n **faire**
11 Considérer l'ensemble de tâches $T'=T-\{T_j,T_{j'}\}$;
12 Résoudre le Problème $P3|\,|C_{max}$ avec T' en utilisant le programme dynamique DP;
13 Trouver la meilleure solution pour le problème $P3|pmtn(delais_{ii'} = d)|C_{max}$, en intégrant les tâches $T_j, T_{j'}$;
14 Choisir la meilleure solution parmi les $n(n-1)$ solutions résultantes;
15 Garder la meilleure solution parmi celles obtenues aux étapes 3, 8 et 14;

Pour une instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$, la première étape de l'algorithme DPX consiste en l'application d'un algorithme de liste (en l'occurrence, l'algorithme FAM cité en 2.3.1). Dans notre cas, les tâches

sont ordonnées par ordre décroissant de leurs temps de traitement. La longueur de l'ordonnancement obtenu est utilisée comme borne supérieure pour la suite de l'exécution de $DPX3$ (étape 2). En utilisant cette valeur, le programme dynamique DP est appliqué sur l'instance du $P3||C_{max}$ qui considère les mêmes tâches du problème initial tout en interdisant la préemption de celles-ci. Par conséquent, les délais de transport ne sont plus considérés. L'étape 4 consiste en la suppression temporaire d'une tâche T_j de l'ensemble T. Les tâches restantes sont ordonnancées par l'algorithme dynamique DP (étape 6). La tâche ignorée T_j est injectée à l'ordonnancement partiel (comportant les $(n-1)$ tâches) obtenu précédemment selon la preuve et la caractérisation de Fishkin *et al.* (2005) de l'ordonnancement optimal, soit conformément à la représentation de la figure 3.1 (étape 7). Par la suite et à partir de l'étape 9, le même procédé est appliqué pour tout couple de tâches $\{T_j, T_{j'}\}$ de l'ensemble initial. Les tâches ignorées sont à leur tour injectées à l'ordonnancement des $(n-2)$ tâches ordonnancées par DP en se basant sur la caractérisation de Fishkin *et al.* (2005) comme l'illustre la figure 3.2. Le meilleur des ordonnancements obtenus à la fin des étapes 3, 8 et 14 est retenu pour le problème $P3|pmtn(delay_{ii'} = d)|C_{max}$.

Théorème 3.3.1. *Le problème $P3|pmtn(delay_{ii'} = d)|C_{max}$ est résolu par l'algorithme DPX en $O(n^3P^2)$ où P est une borne supérieure.*

Démonstration. La première partie de l'algorithme $DPX3$ consiste en l'application du programme dynamique pour résoudre une instance du problème $P3||C_{max}$, et ce pour un ensemble de n tâches. Cette étape est caractérisée par le calcul de toutes les valeurs $F(n, P_1, P_2)$ de la fonction F, et ce pour toutes les valeurs de P_1 et de P_2 variant dans l'ensemble $\{1, \dots, P\}$. Ce calcul se fait moyennant $O(nP^2)$ opérations. Pour la seconde partie de l'algorithme, indiquée par le bloc d'instructions 4-7, chaque tâche T_j est omise de l'ensemble des n taches. Les $(n-1)$ autres tâches sont ordonnancées par le programme DP en $O((n-1)P^2)$. T_j est ajoutée à l'ordonnancement partiel suivant la caractérisation indiquée en vue de minimiser le C_{max}, un processus répété n fois. L'ordonnancement final est donc obtenu après $n(n-1)P^2$ opérations. Le même calcul est effectué pour la dernière phase, indiquée par le bloc d'instructions 9-13, qui consiste à ignorer un couple de tâches de l'ensemble T. Pour chaque cas des $\frac{n(n-1)}{2}$ possibilités, les $(n-2)$ tâches restantes sont ordonnancées par DP en $O((n-2)P^2)$.

L'ordonnancement obtenu après la réintégration des tâches ignorées afin de minimiser le C_{max} se fait ainsi en $O(n(n-1)(n-2)P^2)$. Au final, une solution optimale du $P3|pmtn(delay_{ii'} = d)|C_{max}$ est calculée en $O(n^3P^2)$. □

3.4 Schéma d'approximation complètement polynomial

Pour un problème d'optimisation, l'idée principale d'un schéma d'approximation est de transformer une instance initiale en une autre instance pour laquelle il est plus simple d'obtenir une solution optimale par un algorithme exact. L'objectif escompté est que la solution optimale de la nouvelle instance et celle de l'instance initiale soient les plus proches possible. Ainsi, le plus difficile pour concevoir un tel algorithme est d'arriver à trouver la bonne transformation permettant d'atteindre un tel objectif. Pour ce faire, il existe plusieurs façons de procéder, d'ailleurs des exemples de transformations ainsi que des preuves d'existence pour de tels algorithmes sont présentés pour plusieurs problèmes d'optimisation dans Schuurman & Woeginger (1999).

Pour résumer et mieux comprendre le principe ainsi que le fonctionnement des algorithmes d'approximation, nous reprenons (à notre façon) un schéma assez simple proposé par Schuurman & Woeginger (1999) et présenté dans la figure 3.3. Pour cela, nous adoptons les notations suivantes :

Soit I une instance d'un problème d'optimisation NP-$difficile$. Soit $I^\#$ la nouvelle instance modifiée (ou transformée) de l'instance initiale I via l'algorithme d'approximation proposé. À priori, l'instance modifiée doit être plus simple à traiter (d'où sa forme plus régulière) et donc plus simple à résoudre par un algorithme exact. Concrètement, la nouvelle instance doit être résolue de manière optimale en temps polynomial. Compte tenu de la nature du problème et donc de la difficulté à obtenir directement la solution optimale $OPT(I)$, le principe est de passer par la solution optimale de l'instance modifiée, $OPT(I^\#)$, et de retrouver par la transformation inverse de celle-ci une solution approchée (notée $APP(I)$) que l'on espère assez proche de $OPT(I)$.

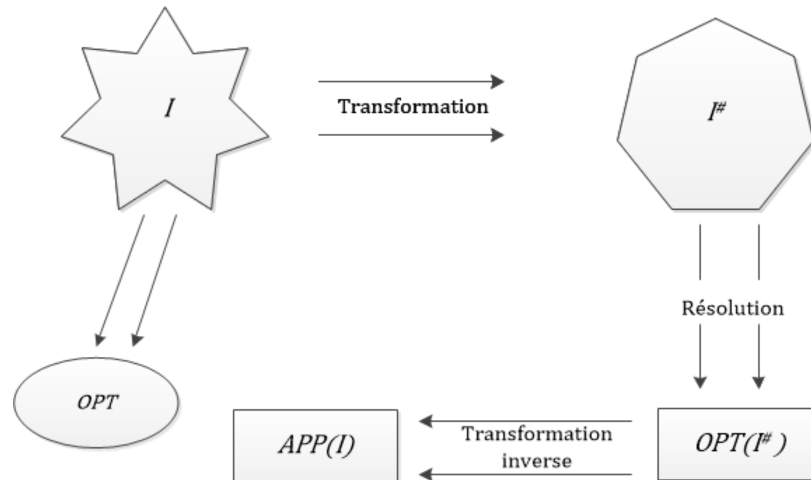


FIGURE 3.3 – Représentation du principe des algorithmes d'approximation

Pour le $P3|pmtn(delay_{ii'} = d)|C_{max}$ nous proposons, dans ce qui suit, un schéma d'approximation complètement polynomial (noté FPTAS, abréviation de l'anglais Fully polynomial-time approximation scheme) qui, rappelons-le, est une famille d'algorithmes d'approximation que l'on note A_ϵ fournissant des solutions proches de la solution optimale d'une distance qui dépend de ϵ. Plus précisément, l'existence de ce type d'algorithmes pour une instance I du problème initial fournit des solutions dont la valeur de la fonction objectif est majorée par la valeur de l'expression $(1+\epsilon)OPT(I)$. L'algorithme proposé doit impérativement être polynomial en la taille du problème ainsi qu'en $1/\epsilon$). Nous rappelons également qu'un FPTAS a été auparavant proposé par Haned *et al.* (2012) pour le problème d'ordonnancement sur deux machines en considérant des délais de transport variables[1]. Les étapes du FPTAS proposé sont décrites dans l'algorithme 3.

1. Dans Haned *et al.* (2012), les auteurs ont cité quelques exemples d'algorithmes d'approximation polynomiaux pour des problèmes NP-difficiles à deux et à plusieurs machines impliquant différentes contraintes à l'exemple des contraintes de précédence et de dates échues. Les intéressés peuvent consulter les travaux suivants : Kovalyov (1995); Ji & Cheng (2008); Woeginger (2009); Agnetis *et al.* (2010); MICHAEL (2018)

Algorithme 3: FPTAS Pour $P3|pmtn(delay_{ii'} = d)|C_{max}$.

Entrées : Une instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$
Sorties : Ordonnancement des n tâches sur trois machines
1. Soient $\epsilon > 0$ et $K = \frac{\epsilon LB}{n}$;
2. Création de l'instance $I^{\#}$: pour toute tâche $T_j \in T$, $p_j^{\#} = \lfloor \frac{p_j}{K} \rfloor$;
3. Appliquer $DPX3$ sur l'instance modifiée $I^{\#}$;
4. Retrouver l'affectation des tâches de l'instance originale à partir de l'ordonnancement résultant de l'étape précédente;

L'algorithme proposé considère une instance du $P3|pmtn(delay_{ii'} = d)|C_{max}$ comme paramètre en entrée. Les étapes 1 et 2 représentent les étapes de transformation de l'instance initiale en la nouvelle instance modifiée. Cette nouvelle instance du problème est définie comme suit :

Étant donné une valeur de $\epsilon > 0$, posons $K = \frac{\epsilon LB}{n}$ où LB est une borne inférieure du C_{max}. Nous concernant, la borne inférieure utilisée est celle de McNaughton appliquée au cas de trois machines. Pour toute tâche T_j de l'ensemble T, un nouveau temps de traitement lui est attribué et est définie par la nouvelle valeur $p_j^{\#} = \lfloor \frac{p_j}{K} \rfloor$. Pour cette instance, nous appliquons l'algorithme $DPX3$ et la solution de l'instance originale est obtenue à partir de la solution obtenue en maintenant la même affectation des tâches aux machines pour l'instance originale.

Théorème 3.4.1. *L'algorithme 3 est un FPTAS pour le $P3|pmtn(delay_{ii'} = d)|C_{max}$ opérant en $O(\frac{n^3}{\epsilon})$.*

Démonstration. Pour cette preuve nous adoptons les notations suivantes :

Soit $K = \frac{\epsilon LB}{n}$.

Les nouveaux temps de traitement des tâches relatifs à l'instance transformée sont $p_j^{\#} = \lfloor \frac{p_j}{K} \rfloor$, $\forall j = \overline{1, n}$.

Soient σ^* et $\sigma^{\#}$ des ordonnancements optimaux pour le problème original et sa version transformée respectivement.

Pour tout ordonnancement réalisable σ, soient $C_{max}(\sigma)$ la longueur de cet ordonnancement avec les temps de traitement du problème original, et $C^{\#}_{max}(\sigma)$ sa longueur en considérant les temps de traitement après transformation de l'instance ($p^{\#}_j$, $\forall j = \overline{1, n}$).

$c_j(\sigma)$ et $c^{\#}_j(\sigma)$ correspondent, respectivement, à la date de fin de traitement de la tâche T_j dans l'ordonnancement σ en considérant les temps de traitement de l'instance originale et de l'instance transformée respectivement.

Par définition nous avons :

$$C_{max}(\sigma^*) \leq C_{max}(\sigma^{\#}) \tag{3.1}$$

$$C^{\#}_{max}(\sigma^{\#}) \leq C^{\#}_{max}(\sigma^*) \tag{3.2}$$

Pour toute tâche T_j de l'ensemble T, nous avons :

$$Kp^{\#}_j \leq p_j \tag{3.3}$$

$$p_j \leq K(p^{\#}_j + 1) \tag{3.4}$$

Considérons S_j comme étant l'ensemble des tâches exécutées avant la tâche T_j sur la même machine

$$\begin{aligned}
(3.3) \Rightarrow K. \sum_{J_k \in S_j} p^{\#}_k \leq \sum_{J_k \in S_j} p_k && (3.5)\\
\Rightarrow K.c^{\#}_j(\sigma) \leq c_j(\sigma) && (3.6)\\
\Rightarrow K.C^{\#}_{max}(\sigma) \leq C_{max}(\sigma) && (3.7)
\end{aligned}$$

$$\begin{aligned}
(3.4) \Rightarrow \sum_{J_k \in S_j} p_k \leq K \sum_{J_k \in S_j} p^{\#}_k + K \mid S_j \mid && (3.8)\\
\Rightarrow \sum_{J_k \in S_j} p_k \leq K \sum_{J_k \in S_j} p^{\#}_k + Kn && (3.9)\\
\Rightarrow c_j(\sigma) \leq Kc^{\#}_j(\sigma) + Kn && (3.10)\\
\Rightarrow C_{max}(\sigma) \leq KC^{\#}_{max}(\sigma) + Kn && (3.11)
\end{aligned}$$

Compte tenu de la transformation de l'instance originale effectuée à l'étape 2 de l'algorithme 3, l'inégalité (3.11) reste valide pour l'ordonnancement $\sigma^{\#}$:

$$C_{max}(\sigma^{\#}) \leq KC^{\#}_{max}(\sigma^{\#}) + Kn \tag{3.12}$$

$$(3.12) \Rightarrow C_{max}(\sigma^{\#}) \leq KC^{\#}_{max}(\sigma^{*}) + Kn \qquad (voir\ (3.2)) \tag{3.13}$$

$$(3.7) \Rightarrow C_{max}(\sigma^{\#}) \leq C_{max}(\sigma^{*}) + Kn \tag{3.14}$$

$$\Rightarrow C_{max}(\sigma^{\#}) \leq C_{max}(\sigma^{*}) + \epsilon LB \tag{3.15}$$

$$\tag{3.16}$$

Ce qui signifie que l'ordonnancement $\sigma^{\#}$, résultat de l'algorithme 3 satisfait

$$C_{max}(\sigma^{\#}) \leq (1+\epsilon)C_{max}(\sigma^{*})$$

Si l'on note UB comme étant une borne supérieure pour le problème étudié, la complexité de l'algorithme 3 est :

$$O(n^3.UB^{\#^2}) = O(n^3\frac{UB^2}{K^2}) \tag{3.17}$$

$$= O(n^3 UB^2 \frac{n^2}{\epsilon^2 LB^2}) \tag{3.18}$$

$$= O(\frac{n^5}{\epsilon^2}) \tag{3.19}$$

Qui est donc une complexité polynomiale en n (taille de l'instance) et aussi en $\frac{1}{\epsilon}$, ce qui fait de l'algorithme 3 un schéma d'approximation de type FPTAS pour le problème considéré, à savoir le $P3|pmtn(delay_{ii'} = d)|C_{max}$. □

Remarque. Comme c'est le cas pour le problème sur trois machines, ainsi que précédemment sur deux machines, nous pouvons étendre les résultats cités pour le problème général à m machines. Ainsi, il existe pour ce dernier un ordonnancement optimal avec au plus $m-1$ préemptions et transports de ces tâches préemptées. Une telle solution peut être obtenue par l'algorithme $DPX3$ en $O(n^m UB^{m-1})$. Et dans le même contexte, le $Pm|pmtn(delay_{ii'} = d)|C_{max}$ admet un FPTAS s'exécutant en $O(\frac{n^{2m-1}}{\epsilon^{m-1}})$.

3.5 Expérimentation numérique

Cette section présente une analyse empirique des résultats numériques obtenus après l'implémentation et l'exécution des algorithmes $DPX3$ et $FPTAS$. Les algorithmes sont implémentés en langage C et exécutés sur un ordinateur doté d'un processeur Intel(R) core (TM) i5-3230M, d'une fréquence de base de 2.60 GHz avec 4 GB de RAM et sous Windows 10.

En l'absence d'instances de référence pour le problème étudié, une batterie d'instances a été générée aléatoirement suivant une loi uniforme. Les instances ont été générées de telle sorte à ce que l'algorithme de McNaughton ne puisse fournir de solutions optimales pour ces dernières.

Générer de telles instances se fait en fixant un intervalle possible pour le délai de transport. Ce dernier est donc généré dans l'intervalle $[\Delta - p_{min}, C_{max}^{FAM} - p_{min}]$, où p_{min} représente le plus petit temps de traitement de l'instance et C_{max}^{FAM} représente la longueur d'un ordonnancement réalisable obtenu par l'application de l'heuristique FAM combinée à un tri des tâches de type LPT.

Le nombre de tâches n évolue dans l'ensemble $\{50, 70, 80, 100\}$. Les temps de traitement sont générés dans les intervalles $[1, 50]$, $[10, 50]$ et $[20, 50]$. Pour chaque valeur de n, nous générons 100 instances à tester. Nous obtenons un total de 1200 instances.

Le $FPTAS$ a, quant à lui, été exécuté et testé pour plusieurs valeurs de ϵ. Dans un premier temps, seules les valeurs significatives ont été rapportées à savoir $\{1\%, 5\%, 10\%\}$.

Les tables 3.1, 3.2 et 3.3 sont relatives aux trois intervalles des temps de traitement générés. Ces tables affichent les résultats obtenus en termes de déviation moyenne de toutes les exécutions du $FPTAS$ ainsi que de l'heuristique FAM par rapport à la valeur optimale des instances obtenues grâce à l'algorithme DPX, et aussi en terme de déviation maximale des méthodes citées. Nous rappelons que cette déviation, qui n'est autre que la mesure de l'erreur relative décrite auparavant, est définie par $\frac{C_{max}^{FPTAS} - C_{max}^{DPX3}}{C_{max}^{DPX3}}$. Le temps d'exécution moyen est également présenté (en secondes) pour

chacun des algorithmes cités. Nous signalons que nous avons limité le temps d'exécution des algorithmes implémentés à une heure de calcul.

n	Temps d'exécution moyen (s)				Déviation moyenne				Déviation maximum			
	DPX3	FPTAS			FPTAS			FAM	FPTAS			FAM
		$\epsilon(\%)$			$\epsilon(\%)$				$\epsilon(\%)$			
		1%	5%	10%	1%	5%	10%		1%	5%	10%	
50	8.17	20.24	0.63	0.07	0.00	0.32	0.39	0.11	0.00	0.82	1.78	0.48
70	45.26	51.32	1.28	0.09	0.00	0.30	0.24	0.04	0.00	0.88	1.38	0.34
80	87.19	76.82	1.88	0.11	0.00	0.27	0.22	0.03	0.14	0.87	1.49	0.16
100	32.2	229.4	2.78	0.12	0.02	0.25	0.18	0.01	0.13	0.74	1.49	0.12

TABLE 3.1 – Résultats des mesures rapportées pour les instances dont les $p_i \in [1, 50]$

n	Temps d'exécution moyen (s)				Déviation moyenne				Déviation maximum			
	DPX3	FPTAS			FPTAS			FAM	FPTAS			FAM
		ϵ			ϵ				ϵ			
		1%	5%	10%	1%	5%	10%		1%	5%	10%	
50	28.73	21.58	2.08	0.20	0.00	0.27	0.65	0.66	0.00	0.79	1.59	1.07
70	113.10	56.54	5.33	0.54	0.00	0.32	0.72	0.85	0.00	0.85	1.70	1.10
80	204.20	87.03	3.61	0.81	0.00	0.29	0.65	0.39	0.00	0.74	1.74	0.51
100	795.75	271.16	15.42	1.57	0.02	0.37	0.80	0.59	0.10	0.89	1.78	0.73

TABLE 3.2 – Résultats des mesures rapportées pour les instances dont les $p_i \in [10, 50]$

Depuis les trois tables, nous pouvons voir que le temps d'exécution moyen de l'algorithme $DPX3$, ayant permis la détermination des solutions optimales, augmente proportionnellement avec la taille du problème n à l'exception du cas où $n = 100$ pour les instances avec des p_i générées dans l'intervalle $[1, 50]$. Ce temps moyen enregistré de $32s$ nous a poussé à analyser de plus près cette catégorie d'instances. Nous avons constaté que pour 96% des instances testées, la solution a été obtenue, en atteignant la borne inférieure utilisée Δ, très vite et ce par la première étape de $DPX3$, plus exactement par l'application de l'algorithme FAM. Pour ces instances

n	Temps d'exécution moyen (s)				Déviation moyenne				Déviation maximum			
	DPX3	FPTAS			FPTAS			FAM	FPTAS			FAM
		$\epsilon(\%)$			$\epsilon(\%)$				$\epsilon(\%)$			
		1%	5%	10%	1%	5%	10%		1%	5%	10%	
50	34.65	20.89	0.88	0.20	0.00	0.00	0.36	0.28	0.00	0.52	1.14	6.20
70	235.03	58.69	2.71	0.55	0.00	0.00	0.37	0.06	0.00	0.49	1.10	3.10
80	406.80	87.20	7.72	0.81	0.00	0.00	0.41	0.05	0.11	0.45	0.95	1.60
100	1175.27	273.14	15.59	1.59	0.01	0.02	0.42	0.08	0.08	0.42	1.07	0.30

TABLE 3.3 – Résultats des mesures rapportées pour les instances dont les $p_i \in [20, 50]$

de 100 tâches, nous avons remarqué que le nombre de petites tâches (voire unitaires) était plus important que pour les instances où $n \geq 80$. Le principe de FAM en présence de telles tâches fait que ces petites tâches (ordonnancées à la fin) aient un rôle de rééquilibrage des charges sur les machines causées par l'ordonnancement des grandes tâches en premier, la préemption est par conséquent évitée. Nous avons aussi relevé une augmentation du temps de calcul de $DPX3$ en fonction de longueur des tâches, comme le montre la figure 3.4.

Dans un premier temps, le FPTAS a été testé pour des valeurs de ϵ allants de 5% à 10%. En parallèle, une colonne rapportant les résultats d'une exécution rapide de FAM a été ajoutée afin de comparer les résultats d'un des algorithmes connu pour sa rapidité avec ceux de l'algorithme d'approximation proposé qui offre la possibilité d'obtenir des solutions assez proches de l'optimale. Nous n'avons pas été surpris que FAM ai été très rapide à résoudre le problème en quelques secondes offrant des solutions relativement bonnes si l'on juge de sa déviation moyenne qui n'excède pas 0.85. Les solutions ont été assez proches des solutions du FPTAS et donc de bonne qualité pour l'ensemble des instances lorsque ϵ variait autour de 10%. Néanmoins, ce dernier a enregistré les plus mauvaises solutions avec une déviation maximale égale à 6.20 pour des instances comportant 50 tâches de grande durée (celles de la table 3.3). Ce qui est très normal pour des ordonnancements avec de longues tâches, pour qui la préemption n'est pas autorisée. Cependant, la réflexion du paragraphe précédent

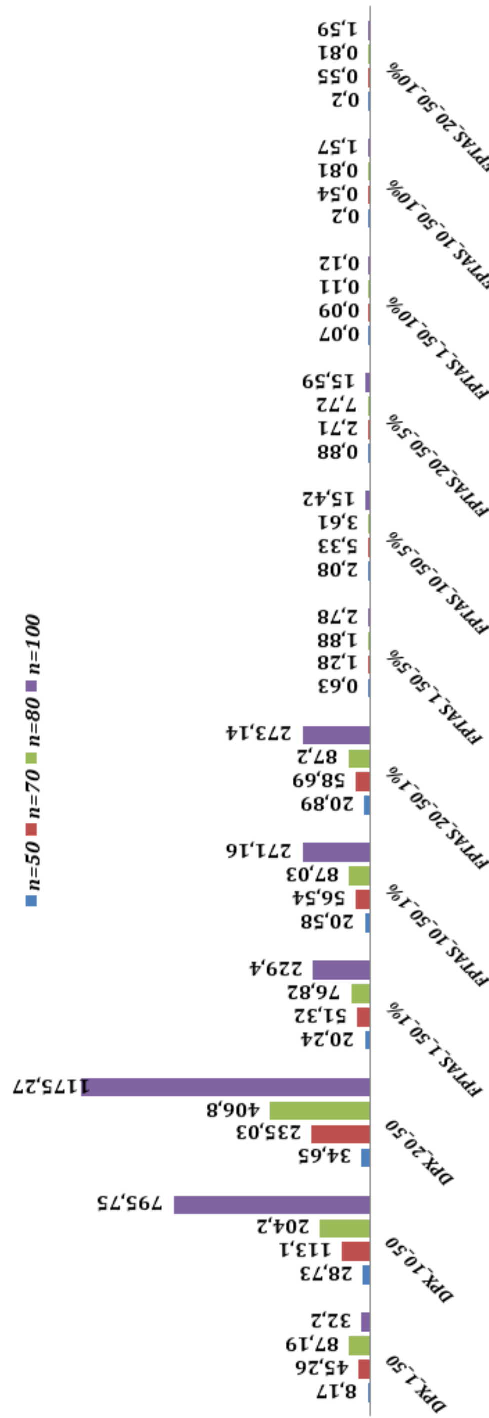


FIGURE 3.4 – Comparaison des temps de calcul pour les différentes exécutions d'algorithmes

en rapport avec les plus petites tâches (unitaires surtout) est confirmée par les déviations (moyennes et maximales) rapportées dans la table 3.1.

Ainsi, à partir de la table 3.1, l'algorithme FAM a été satisfaisant pour des instances comportant des tâches de taille variée, et ce, en offrant de bonnes solutions dans le sens où ces dernières ont été de qualité proche de celles obtenues par le FPTAS lorsque ϵ variait autour des 5%.

Pour des instances de tâches moyennes, avec des p_i dans $[10, 50]$, les algorithmes demeurent équivalents en terme de qualité de solutions obtenues lorsque le FPTAS est exécuté avec $\epsilon = 10\%$, et ce au vue des valeurs enregistrées par leurs déviation moyennes et maximales. Bien évidemment, ce dernier est meilleur lorsqu'il est exécuté avec $\epsilon = 5\%$. Pour le cas des tâches de grande taille, avec des pi_i dans $[20, 50]$ (table 3.3, les plus mauvaises solutions obtenues sont dues à l'exécution de FAM en observant les valeurs des déviations maximales enregistrées. Cependant, la qualité de ses résultats reste correcte comparée à celle des solutions du FPTAS, encore une fois lorsqu'il est exécuté pour $\epsilon = 10\%$, mais pas assez lorsque l'on pousse le FPTAS à fournir des solutions avec $\epsilon = 5\%$ de précision.

Il est évident que les résultats énoncés n'atteignent pas la satisfaction espérée suite à l'implémentation des différentes méthodes. L'aspect mitigé de ces dernier nous a incité à vouloir mieux exploiter le FPTAS proposé en le testant pour des valeurs plus précises de ϵ, c'est-à-dire $\leq 5\%$. Un objectif à atteindre avec l'ambition de garantir, bien évidemment, la même (ou presque) vitesse d'exécution de ce dernier. Cependant, pour faire face à sa complexité spatiale, l'utilisation d'une ressource plus importante a été primordiale. Pour ce faire, nous nous somme pourvus d'une mémoire vive de $30GB$, une extension qui nous a permis de tester le FPTAS pour différentes valeurs allant jusqu'à $\epsilon = 1\%$.

Au même titre que les temps d'exécution du FPTAS, jugés très satisfaisants (d'après 3.4), nous pouvons aussi observer que, pour presque toutes les instances, les solutions obtenues sont de plus en plus proches des solutions optimales. Effectivement, pour $\epsilon = 1\%$ de meilleurs ordonnancements ont été obtenus ce qui est traduit par des valeurs infimes de déviations moyennes et maximales. Un autre gain est considéré comme étant

plus favorable à l'exécution du FPTAS pour une valeur de $\epsilon = 1\%$, il s'agit du temps de calcul nécessaire à l'obtention de ces solutions. Donc, si une solution optimale pour une instance de 100 tâches est obtenue au bout de $1175sec$, il est aussi possible de l'approcher avec une précision de 1% et l'obtenir en $273sec$ grâce à une telle exécution du FPTAS. Le même constat peut être observé pour 80 tâches où il est parfaitement possible d'avoir de bonnes solutions en $87sec$ au lieu de $406sec$.

Le contenu de ce chapitre a fait l'objet d'une participation à une conférence internationale (Badaoui *et al.*, 2015).

4

Modélisation mathématique

Sommaire

4.1 Introduction 75
4.2 Le modèle mathématique 78
4.3 Détails du modèle mathématique 78
4.4 Expérimentation et résultats numériques 84

4.1 Introduction

Dans ce chapitre, nous proposons une nouvelle formulation mathématique du problème général sur m machines en considérant des délais de transport variables. Cette nouvelle formulation a été élaborée suite à l'ajout d'une contrainte relative à la préemption des tâches. Nous expliquons tout d'abord les raisons ayant motivé ce contenu, puis nous proposons ladite formulation, qui sera suivie par les résultats numériques obtenus.

En effet, cette partie est dédiée au problème général étudié qui se note $Pm|pmtn(delay_{ii'})|C_{max}$, pour lequel un modèle mathématique linéaire a

déjà été proposé par Boudhar & Haned (2009). Le modèle en question comportait des variables réelles et binaires. Quelques années plus tard, Shams & Salmasi (2014) ont critiqué le modèle cité précédemment et ont montré que lors de la préemption d'une tâche, la formulation précédente ne représentait pas les solutions dans lesquelles une tâche préemptée reprenait son traitement sur la même machine sans qu'elle ne soit transportée ailleurs. Pour ce faire, les auteurs ont présenté un exemple d'instance d'un $P3|pmtn(delay_{ii'})|C_{max}$ comportant quatre tâches à exécuter, et dont les durées opératoires ainsi que les délais de transport entre les trois machines sont regroupés dans les tables 4.1a et 4.1b respectivement.

T_j	T_1	T_2	T_3	T_4
	60	60	60	30

(a) Durées opératoires.

0	70	20
70	0	20
20	20	0

(b) Délais de transport.

TABLE 4.1 – Contre-exemple proposé par Shams & Salmasi (2014) .

Les auteurs ont montré qu'une solution optimale pour cette instance est de longueur 70 contre celle obtenue par le modèle mathématique de Boudhar & Haned (2009), et dont le C_{max} valait 75 unités de temps. Les deux ordonnancements cités sont représentés dans les figures 4.1 et 4.2 respectivement.

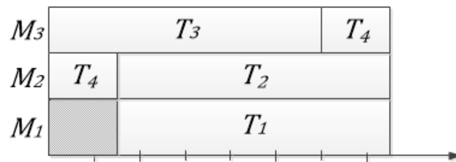


FIGURE 4.1 – Ordonnancement obtenu par le modèle de Boudhar & Haned (2009).

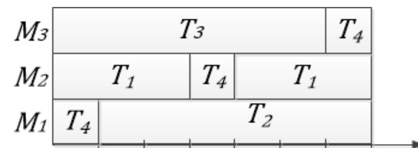


FIGURE 4.2 – Ordonnancement optimal.

Compte tenu de la nature des variables utilisées par Boudhar & Haned (2009), le modèle mathématique ne peut fournir de telles solutions optimales, à savoir un ordonnancement dans lequel une tâche préemptée

pouvait reprendre son traitement après un certain temps sur la même machine (comme c'est le cas pour la tâche T_1 dans l'ordonnancement optimal représenté par la figure 4.2, et qui est exécutée à deux reprises sur la machine M_2). Cette situation peut survenir lorsqu'une tâche qui est exécutée sur plusieurs machines requiert, après son transport, l'interruption d'une autre tâche en cours d'exécution, en l'occurrence, il s'agit ici de la tâche T_4.

Comme énoncé plus haut, nous proposons dans cette partie une nouvelle formulation mathématique pour le $Pm|pmtn(delay_{ii'})|C_{max}$ et dont les notations, paramètres ainsi que les variables de décision se définissent comme suit :

Paramètres :

n	Nombre de tâches.
m	Nombre de machines.
p_j	Durée opératoire de la tâche T_j.
$d_{ii'}$	Délai de transport entre la machine M_i et la machine $M_{i'}$.
M	Un nombre assez grand ($M \gg 0$).
r	Nombre maximum de parts d'une tâche préemptée.

Variables de décision :

$$C_{max} = \text{La valeur du Makespan}$$

$$q_{jk}(\geq 0) = \text{La durée de la part } k \text{ de la tâche } T_j$$

$$t_{jk}(\geq 0) = \text{La date de début de traitement de la part } k \text{ de la tâche } T_j$$

$$x_{jki} = \begin{cases} 1 \text{ si la part } k \text{ de la tâche } T_j \text{ est exécutée sur la machine } M_i \\ 0 \text{ sinon} \end{cases}$$

$$\alpha_{jkj'k'} = \begin{cases} 1, & \text{si la part } k \text{ de la tâche } T_j \text{ est exécutée avant la part } k' \text{ de la tâche } T_{j'} \\ 0, & \text{sinon} \end{cases}$$

$$(j, j' = 1, \ldots, n; k, k' = 1, \ldots, r \text{ et } i = 1, \ldots, m)$$

4.2 Le modèle mathématique

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

(2) $\sum_{k=1}^{r} q_{jk} = p_j; \quad j = 1, \ldots, n$

(3) $\sum_{i=1}^{m} x_{jki} \leq 1; \quad j = 1, \ldots, n; k = 1, \ldots, r$

(4) $\sum_{i=1}^{m} x_{jki} \leq q_{jk}; \quad j = 1, \ldots, n; k = 1, \ldots, r$

(5) $q_{jk} \leq M \sum_{i=1}^{m} x_{jki}; \quad j = 1, \ldots, n; k = 1, \ldots, r$

(6) $$\begin{cases} t_{jk} + q_{jk} - t_{jk+1} \leq (2 - x_{jki} - x_{jk+1i})M; \\ j = 1, \ldots, n; k = 1, \ldots, r-1; i = 1, \ldots, m \end{cases}$$

(7) $$\begin{cases} t_{jk} + q_{jk} + (x_{jki} + x_{jk+1i'} - 1)d_{ii'} - t_{jk+1} \leq (2 - x_{jki} - x_{jk+1i'})M; \\ j = 1, \ldots, n; k = 1, \ldots, r-1; i, i' = 1, \ldots, m; i \neq i' \end{cases}$$

(8) $$\begin{cases} t_{jk} + q_{jk} - t_{j'k'} \leq (3 - x_{jki} - x_{j'k'i} - \alpha_{jkj'k'})M; \\ t_{j'k'} + q_{j'k'} - t_{jk} \leq (2 - x_{jki} - x_{j'k'i} + \alpha_{jkj'k'})M; \\ j, j' = 1, \ldots, n; k, k' = 1, \ldots, r; i = 1, \ldots, m; j < j'; (j,k) \neq (j',k') \end{cases}$$

(9) $\sum_{i=1}^{m} x_{jki} \leq \sum_{i=1}^{m} x_{jk+1i}; \quad j = 1, \ldots, n; k = 1, \ldots, r$

(10) $t_{jk} \leq t_{jk+1}; \quad j = 1, \ldots, n; k = 1, \ldots, r-1$

(11) $t_{jr} + q_{jr} \leq C_{max}; \quad j = 1, \ldots, n$

4.3 Détails du modèle mathématique

Les différentes équations du modèle mathématique présenté définissent l'objectif et les contraintes du problème étudié, et sont citées et expliquées dans ce qui suit :

- Notre objectif qui consiste à minimiser la durée totale de l'ordonnancement est exprimé par la contrainte (1).

- L'ensemble des contraintes de type (2) indique que pour chaque tâche T_j, la somme de toutes ses parts exécutées sur les différentes machines doit être exactement égale à sa durée opératoire p_j.

- Les contraintes de type (3) indiquent que toute part k d'une tâche T_j , si elle existe, ne peut être exécutée que par une seule machine. Notons que ces contraintes n'interdisent pas le traitement de deux parts différentes k et k' d'une même tâche sur la même machine.

- Pour toute tâche T_j, lorsque q_{jk} est nulle cela signifie que la k^{me} part de cette tâche n'existe pas (en d'autre terme cette part n'est traitée par aucune machine), ce qui implique que la variable de décision qui lui correspond doit également être nulle (d'où $\sum_{i=1}^{m} x_{jki} = 0$). Ceci est représenté par les contraintes de type (4).

- Par ailleurs, lorsqu'une part k d'une tâche T_j est de longueur $q_{jk} \neq 0$, cette part doit exister et être exécutée par au moins l'une des machines, donc $\sum_{i=1}^{m} x_{jki}$ doit être égale à 1. Ce qui est exprimé justement par les contraintes de type (5).

- Les contraintes de type (6) garantissent le non chevauchement des parts successives k et $k+1$ d'une même tâche préemptée T_j exécutées sur une même machine M_i. En effet, si de telles parts existent et sont traitées par la machine M_i (c'est-à-dire, $x_{jki} = 1$ et $x_{jk+1i} = 1$), la part $k + 1$ ne peut démarrer qu'à la fin de traitement de la part k. Le traitement de la part suivante ne peut démarrer qu'à partir de la date exprimée par $t_{jk} + q_{jk}$.

- Les contraintes de type (7) interdisent le chevauchement des parts d'une tâche préemptée T_j, traitées par différentes machines tout en assurant le respect des délais de transport entre celles-ci. En effet, lorsque T_j est exécutée par M_i pendant q_{jk} unités de temps, puis est préemptée et transportée vers $M_{i'}$, la part suivante $k + 1$ ne peut commencer son traitement qu'après $d_{ii'}$ unités de temps à compter de l'achèvement du traitement de la k^{e} part traitée sur M_i. Ainsi, la date de début de traitement de la part $k+1$ ne peut être programmée avant la date $t_{jk} + q_{jk} + d_{ii'}$, et ce, bien entendu lorsque $x_{jki} = 1$ et $x_{jk+1i'} = 1$.

- En plus de l'ensemble des contraintes de type (6) concernant le chevauchement des parts d'une même tâche sur une machine M_i, les contraintes de types (8) contrôlent l'absence de ce type de conflit entre toutes les parts des différentes tâches exécutées par M_i. Ainsi, pour toute paire de tâches $(T_j, T_{j'})$ traitées par la machine M_i, soit la tâche T_j précède la tâche $T_{j'}$ soit l'inverse . Dans ce cas-là, et pour éviter le chevauchement des parts, si $\alpha_{jkj'k'} = 1$ alors la part k' de la tâche $T_{j'}$ ne peut démarrer son traitement qu'après la fin de celui de la k^{e} part de T_j ($t_{jk} + q_{jk} \leq t_{j'k'}$). Inversement, lorsque $\alpha_{jkj'k'} = 0$, signifiant que $T_{j'}$ précède T_j, le traitement de la k^{e} part de T_j ne peut commencer avant la fin de traitement de la part $k+1$ de la seconde tâche (à savoir, $t_{j'k'} + q_{j'k'} \leq t_{jk}$.

- Les contraintes de type (9) concernent toutes les tâches préemptées, et donc exécutées en plusieurs parts. Pour toutes ces tâches, nous exprimons via ces contraintes le principe d'avoir imposé aux parts d'être successives et de se suivre à la fin. Par exemple, si une tâche T_j est préemptée une seule fois et exécutée sur deux machines et que $r = 3$, les parts existantes de cette tâche auront 2 et 3 pour indice. En d'autres termes, ces parts seront de longueurs respectives q_{j2} et q_{j3}. Cette convention est justifiée par le fait que les délais de transport aient été imposés pour toutes les parts successives des tâches par les contraintes de type (7), et donc cet ensemble de contraintes a été mis en place afin d'éviter la configuration suivante : $q_{jk} \neq 0$, $q_{jk+1} = 0$ et $q_{jk+2} \neq 0$.

- Les contraintes de type (10) ont été instaurées pour toute tâche préemptée afin que ses parts soient ordonnées par ordre croissant de leurs dates de début de traitement.

- Les contraintes de type (11) expriment le fait que le traitement de toutes les tâches doit être achevé avant la fin de l'ordonnancement. Pour notre cas, il suffit de le vérifier uniquement pour les dernières parts des tâches (elles existent surement).

Le modèle mathématique proposé comporte $nmr + nr(n-1)(r-1)$ variables binaires, $2nr$ variables réelles et un nombre équivalent à $2n + 4nr + n(r-1)[1 + m + m(m-1)] + \frac{1}{2}(n-1)(n-2)mr^2$ contraintes. Si l'on considère le cas de délais de transport identiques ($Pm|pmtn(delay_{ii'} = d)|C_{max}$),

cette nouvelle formulation mathématique améliore celle de Shams & Salmasi (2014) dans le sens où elle contient moins de variables binaires. La taille du modèle a été réduite de $O(n^6)$ à $O(n^4)$ et ce après l'injection de certaines contraintes ayant réduit le domaine de solutions.

Nous rappelons qu'en 2005, le problème d'ordonnancement préemptif en présence de délais de transport identiques a été abordé par Fishkin *et al.* (2005). Dans cette étude, les auteurs ont caractérisé un ordonnancement optimal pour le cas de deux et de trois machines. Pour les ordonnancements à m machines, qui nous intéressent dans ce chapitre, les auteurs ont posé la conjecture suivante :

Conjecture. *Pour toute instance du problème $Pm|pmtn(delay_{ii'} = d$[1]$)|C_{max}$, il existe un ordonnancement optimal avec au plus (m-1) préemptions avec transport des tâches entre les machines (migrations).*

La preuve de cette conjecture, apportée par Shams & Salmasi (2014), a permis aux auteurs d'évaluer le nombre maximum de parts possibles d'une tâche pouvant s'exécuter par une même machine. Ce nombre de parts vaut $\frac{m-1}{2} + 1$, et a été utilisé pour évaluer le modèle mathématique proposé pour la résolution d'instances de taille modérée du problème étudié avec des temps de transport identiques ($Pm|pmtn(delay_{ii'} = d)|C_{max}$).

Cependant, contrairement au cas du problème avec des délais de transport identiques, le nombre de parts qui peuvent former une tâche n'est pas connu pour le cas du problème que nous étudions et qui considère des délais de transport variables. Tous les tests que nous avons effectués pour tenter de l'évaluer nous ont cependant permis de constater que pour tous les ordonnancements résultants, le nombre de préemptions impliquant des transports de tâches entre les machines demeure inférieur ou égal à $m - 1$. Par ailleurs, cette borne n'a été atteinte que pour des instances que nous avons construites dans ce même but, ces instances sont de la forme suivante :

1. Où d représente le délai de transport des tâches préemptées entre les machines M_i et $M_{i'}$

$$\begin{cases} n = m + 1 \\ C_{max} = c + (m-1).d \\ p_1 = c \\ p_j = C_{max} - 1 & j = 2, \dots, n \end{cases}$$

Où c est une constante quelconque.

Un exemple d'ordonnancement optimal pour ce type d'instances extrêmes est montré dans la figure 4.3. Il s'agit d'une instance de six tâches exécutées sur 5 machines et dont les durées opératoires ainsi que les délais de transport sont résumés dans les tables 4.2a et 4.2b respectivement.

$Jobs$	J_1	J_2	J_3	J_14	J_5	J_6
P_j	5	12	12	12	12	12

(a) Durées opératoires.

0	2	15	16	10
9	0	2	10	10
5	3	0	2	10
6	7	15	0	2
10	9	15	16	0

(b) Délais de transport.

TABLE 4.2 – Instance particulière dont le nombre de migrations est égal à $m-1$.

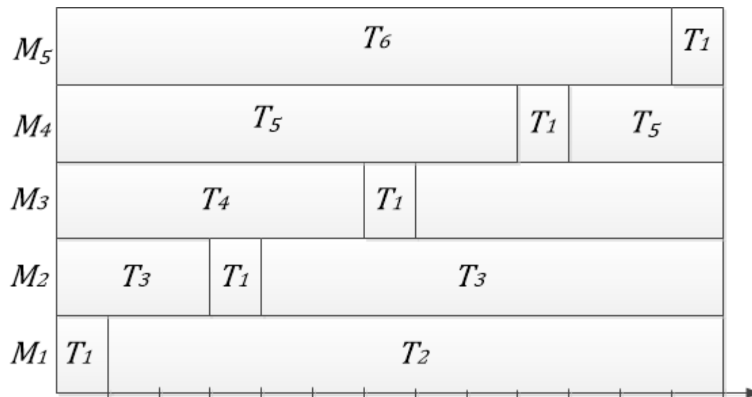


FIGURE 4.3 – Ordonnancement optimal d'une instance particulière

En se basant sur les ordonnancements optimaux résultants pour ce type d'instances construites et ceux des instances générées aléatoirement, nous avons remarqué qu'à partir d'une valeur minimale du nombre de parts nécessaires à une tâche préemptée, la valeur du C_{max} devient stationnaire. Pour l'exemple de l'instance précédente, la table 4.3 rapporte les variations du C_{max} en fonction des valeurs incrémentées de r. Ces valeurs sont représentées par une courbe décroissante dans la figure 4.4, où l'on peut voir que les valeurs du C_{max} décroissent en fonction des valeurs ascendantes de r.

r	C_{max}
1	17
2	14
3	13.6667
4	13.25
5	13

TABLE 4.3 – Valeurs de C_{max} calculées pour différentes valeurs de r.

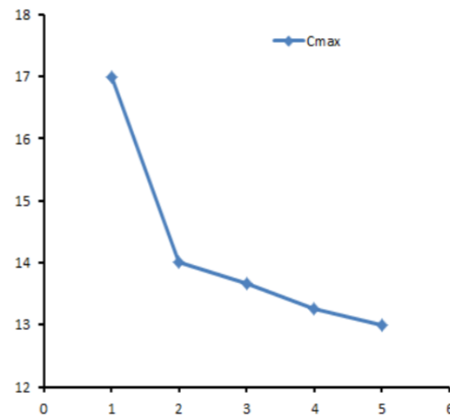


FIGURE 4.4 – Représentation du C_{max} en fonction de r.

Nous expliquons maintenant la démarche suivie pour obtenir les solutions optimales avec différentes valeurs de r. Pour ce faire, nous avons utilisé les notations de valeurs suivantes :

- $\boldsymbol{UB}$, représente une borne supérieure utilisée. Initialement, la borne utilisée est celle calculée par l'application d'un algorithme de type FAM en ordonnant les tâches par ordre décroissant de leurs durées opératoires.
- $\boldsymbol{\Delta}$, représente la borne inférieure de McNaughton.
- $\boldsymbol{LB(r)}$, représente une borne inférieure calculée lorsque au moins une

des tâches est exécutée en $\boldsymbol{r}$ parts. Cette valeur est calculée par l'expression $p_{min} + (r-1)d_{min}$, où $p_{min} = \min_{1 \leq j \leq n} \{p_j\}$ et $d_{min} = \min_{1 \leq i,i' \leq m} \{d_{ii'}\}$.

- $\boldsymbol{C_{max}(r)}$, représente la meilleure valeur de makespan obtenue lorsqu'au moins une tâche est exécutée en r parts.

Il est clair que les valeurs de r influent directement sur la taille du modèle mathématique proposé, du fait qu'il intervient directement dans le nombre de contraintes. Le nombre de contraintes du modèle croit proportionnellement avec les valeurs de r. C'est pour cette raison que la première exécution du modèle mathématique par le solveur est effectuée pour $r = 1$ avec les valeurs initialisées des paramètres Δ et UB. Lorsqu'aucune solution optimal n'est retrouvée (c'est-à-dire $C_{max}(r) \neq \Delta$ et $LB(r) \leq UB$), la valeur de r est incrémentée d'une unité, et la valeur de UB est mise à jour en lui affectant la meilleure valeur de C_{max} obtenue jusqu'à cette étape (c'est-à-dire, $UB = C_{max}(r-1)$).

Pour l'exemple précédent, la valeur de r n'a cessé d'être incrémentée tant que le C_{max} obtenu n'avait pas encore atteint la valeur de $\Delta = 13$, à condition que cette valeur (r) n'engendre pas de mauvaises solutions (c'est-à-dire, avec des longueurs qui dépassaient les nouvelles valeurs de la borne supérieure, mise à jours grâce aux ordonnancements réalisables obtenus). Pour l'exemple précédent, il n'est pas possible d'incrémenter r à la valeur 6 car tous les ordonnancements comportant au moins une tâche exécutée en six parts auraient au mieux été de longueur égale à 15.

4.4 Expérimentation et résultats numériques

Pour évaluer le modèle mathématique proposé dans ce chapitre, nous l'avons testé pour un ensemble de 270 instances générées aléatoirement en fonction des intervalles considérés par Shams & Salmasi (2014). Ces instances sont générées comme suit :

Le nombre de machines m est généré dans trois intervalles différents $[2, 5]$, $[6, 10]$ et $[11, 15]$. Le nombre de tâches est généré dans les intervalles suivants $[2, 20]$, $[21, 50]$ et $[51, 100]$. Pour chaque configuration $(n * m)$, 30

instances sont créées comportant des tâches de tailles différentes appartenant aux intervalles suivants $[1, 10]$, $[1, 40]$ et $[1, 90]$. De la même manière que l'expérimentation effectuée dans le chapitre précédent, les délais de transport sont générés de telle sorte que l'application de l'algorithme de McNaughton ne puisse fournir de solutions optimales à savoir dans l'intervalle $[\Delta - p_{min}, Cmax(FAM) - p_{min}]$. Le solveur ILOG CPLEX 12.6 a été utilisé pour tester cette nouvelle formulation sur un ordinateur personnel d'une fréquence de base de 2.60 GHz avec 4 GB de RAM et sous Windows 10.

Les résultats obtenus sont présentés dans la table 4.4. La colonne $Nopt$ indique le nombre d'instances ayant atteint la borne inférieure du problème. La colonne suivante rapporte le temps d'exécution moyen évalué en secondes. Les colonnes restantes expriment les propriétés des ordonnancements optimaux fournis par le modèle mathématique, et ce en affichant le nombre de parts maximum des tâches préemptées. Une heure de temps a été fixée pour l'exécution du modèle. Au-delà de cette durée, la résolution est stoppée et l'instance est considérée comme étant non résolues.

m	n	$Nopt$	Temps d'exécution moyen (s)	Nombre de parts		
				$r = 1$	$r = 2$	$r = 3$
[2,5]	[2,20]	29	2.00	11	18	0
	[21,50]	30	66.98	28	2	0
	[51,100]	26	302.37	23	3	0
[6,10]	[2,20]	29	34.47	7	22	0
	[21,50]	20	286.20	12	8	0
	[51,100]	5	443.04	5	0	0
[11,15]	[2,20]	30	43.07	0	30	1*
	[21,50]	17	1066.53	3	14	1*
	[51,100]	1	467.39	1	0	0
Total		187 sur 270	353.27	90	97	2

(*) Une solution ayant la même valeur que celles obtenues avec $r = 2$.

TABLE 4.4 – Résultats de l'exécution du modèle par ILOG CPLEX 12.6

La table 4.4 montre que sur un total de 270 instances générées, le modèle mathématique a pu fournir 187 solutions optimales, soit environ 69% des instances considérées en un temps moyen d'environs 353 secondes. Cependant, en augmentant la taille du problème, à savoir son nombre de

tâches et de machines, le modèle mathématique n'arrive pas à fournir des solutions optimales en moins d'une heure de temps. Comme c'est le cas, lorsque $m \geq 11$ et $n \geq 58$, où une seule instance seulement a pu être résolue de manière optimale.

En plus du temps d'exécution et du nombre de solutions optimales obtenues, nous nous sommes particulièrement intéressés au nombre de parts des tâches préemptées dans les ordonnancements optimaux. Nous avons constaté (voir la figure 4.5) que pour 48.13% des solutions, les ordonnancements obtenus ne sont pas préemptifs, les tâches sont donc exécutées entièrement sans préemption. Pour le reste, 51.87% des ordonnancements comportent au moins une tâche préemptée une seule fois, et donc exécutée en deux parts ($r = 2$). Ainsi, nous notons que presque la totalité des ordonnancements optimaux sont caractérisés par la présence d'au moins une tâche pour laquelle $r \leq 2$. Pour les instances dont le nombre de parts r est égal à 3, des ordonnancements optimaux ont également été obtenus avec $r = 2$ en plus d'une heure de résolution. C'est d'ailleurs sur ce constat que repose le contenu du chapitre suivant.

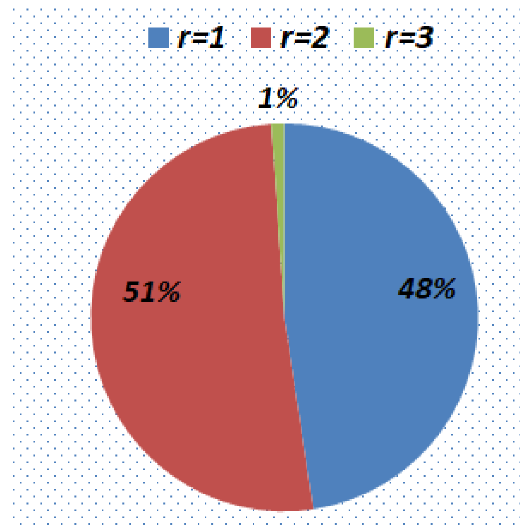


FIGURE 4.5 – Répartitions des valeurs de r dans les ordonnancements optimaux.

5

Méthodes Approchées pour le problème à m machines

Sommaire

5.1 Introduction **87**

5.2 Recherche locale pour déterminer la séquence de machines **89**

5.3 Évaluation des ordonnancements **91**

5.4 L'heuristique H-*conj* pour déterminer une séquence de tâches **93**

5.5 Le Recuit Simulé (RS) pour déterminer une séquence de tâches **95**

5.6 La Recherche à Voisinage Variable pour déterminer une séquence de tâches **99**

5.7 Expérimentation numérique **103**

5.1 Introduction

Le problème considéré dans ce chapitre est le problème d'ordonnancement préemptif sur m machines parallèles en présence de délais de trans-

port entre celles-ci. Dans cette partie, nous exposons les méthodes approchées, conçues et mises en oeuvre pour tenter de résoudre le cas général du problème étudié dans cette thèse. En effet, différentes méthodes de type heuristique sont abordées dans ce chapitre, à l'exemple de la Recherche Locale, ou encore de type métaheuristique à savoir le Recuit Simulé et la Recherche à Voisinage Variable.

En 2005, Fishkin *et al.* (2005) ont été les premiers à établir la complexité du problème d'ordonnancement parallèles de tâches préemptives en présence de délais de transport identiques. Les auteurs ont montré que le problème est NP-$difficile$ au sens fort en présentant une réduction polynomiale au problème α-Partition. Le cas sur deux machines avait également été traité en utilisant le même problème de partition pour $\alpha = 2$ (Boudhar & Haned, 2009). La complexité du problème étudié en considérant des délais de transport variable est donc à déduire d'autant plus que sa variante comportant des temps de traitement identiques a été prouvée comme étant NP-$difficile$ (Haned *et al.*, 2012).

Par conséquent, il est naturel d'envisager la conception de méthodes approchées pour tenter de résoudre le cas général (c'est-à-dire le problème $Pm|pmtn(delay_{ii'})|C_{max}$) d'autant plus que toutes les méthodes exactes développées dans ce sens, n'ont pu résoudre que des instances de petite et moyenne taille ne dépassant pas les 100 tâches, il s'agit essentiellement des travaux de Shams & Salmasi (2014).

Le présent chapitre a non seulement été motivé par la complexité du problème, mais également par le manque de méthodes de résolutions approchées qui lui sont destinées (hormis ceux citées dans la partie état de l'art (voir section 2.3.2). Ainsi, et comme souligné dans la conclusion de Shams & Salmasi (2014), une résolution de ce type doit être développée afin de traiter des instances de taille plus importante. Notre objectif est de contribuer, par la présente étude, au développement de méthodes de résolution approchées, destinées à la résolution du problème d'ordonnancement parallèles en présence de délais de transport **variables**. Dans ce chapitre, nous décrivons une heuristique conçue à cet effet, ainsi que l'adaptation de quelques métaheuristiques comme le recuit simulé et la recherche à voisinage variable.

Les méthodes citées pour tenter de résoudre le $Pm|pmtn(delay_{ii'})|C_{max}$ sont organisées sous la forme d'une stratégie de résolution que nous allons décrire dans ce qui suit. L'idée de cette approche est de déterminer, dans un premier temps, une séquence ou un ordre de machines suivant lequel les tâches vont par la suite être ordonnancées. Cette séquence indique par la même occasion les directions suivant lesquelles les transports des tâches préemptées sont effectués.

La seconde étape de la stratégie consiste, quant à elle, à déterminer la meilleure affectation possible des tâches aux machines engendrant le plus petit coût possible (en termes de durée total d'ordonnancement). Les tâches seront ordonnancées selon la séquence déterminée lors de l'étape précédente, et en cas de préemptions, celles-ci se feront en respectant les délais de transport entre les machines intervenant dans cette même séquence. Cette étape est concrétisée par l'adaptation des deux métaheuristiques connues et citées précédemment.

5.2 Recherche locale pour déterminer la séquence de machines

Comme énoncé plus haut, la première étape est caractérisée par l'application d'une méthode de type recherche locale (Local Search (LS)) afin de déterminer la meilleure séquence de machines possible. Ce type de méthode est, selon Talbi (2009), sans doute la plus ancienne et la plus simple des métaheuristiques à implémenter manipulant une solution unique à la fois. Son principe est de passer successivement d'une solution à une autre appartenant à son voisinage tout en améliorant la valeur de la fonction objectif, et ce, jusqu'à ce qu'aucune amélioration ne soit possible. Rappelons que le voisinage d'une solution, que l'on note x est noté $N(x)$ et représente l'ensemble de toutes les solutions voisines de x, qui peuvent être obtenues par l'application d'une opération élémentaire (appelée aussi mouvement, transformation admissible). Ces opérations permettent de générer, pour une solution courante réalisable x, des solutions voisines $x' \in N(x)$ tout en préservant leur faisabilité et ceci en appliquant sur x une « légère » modification. Ces opérations sont également appelées des transformations admissibles. Le principe de la recherche locale est décrit par le pseudo-

algorithme suivant :

Algorithme 4: Pseudo-code de RL

```
1 Initialisation de la solution courante x = x*, où x* est une solution
  initiale;
2 tant que Critère d'arrêt non satisfait faire
3 |  Générer N(x) le voisinage de la solution x;
4 |  si Aucune solution x' ∈ N(x) n'est meilleure que x alors
5 |  |  Arrêt de la recherche ;
6 |  sinon
7 |  |  x = x' /* la solution voisine est maintenue pour la poursuite
  |  |  de la recherche*/ ;
8 |  fin
9 fin
```

Cette première étape, caractérisée par l'exécution de la méthode de recherche, vise à fournir la meilleure séquence de machines possible suivant laquelle les tâches vont être exécutées. En raison du nombre de machines relativement raisonnable intervenant dans les instances testées, notre implémentation de cette recherche locale a donné lieu à une exécution efficace est rapide. Durant sa mise en oeuvre, nous avons eu à manipuler un certain nombre de paramètres qui ont été réglés de la manière suivante :

- **La représentation de la solution :** nous avons adopté un codage réel pour la représentation des solutions qui indiquent les séquences de machines. La structure de données utilisée est un vecteur de taille m dans lequel, chaque case a pour valeur l'indice i relatif à la machine M_i.

- **Solution initiale :** afin de démarrer l'exécution de RL, une solution initiale réalisable a été générée aléatoirement.

- **Les transformations admissibles :** nous utilisons à cette étape certains opérateurs de voisinage (ou mouvement) connu dans la littérature. Il s'agit entre autres de :
  - *insertion*, qui consiste à déplacer le contenu d'une case de la solution codée depuis sa position initiale, qui est choisie aléatoirement,

ailleurs vers une autre position également piochée aléatoirement.
  - *i-swap*, qui effectue la permutation de i éléments choisis aléatoirement dans la représentation de la solution. Pour notre cas, nous nous sommes contentés de l'utilisation de *2-swap*, *3-swap* et *4-swap*.

- **L'évaluation d'une solution :** chaque vecteur représente une séquence de machines, et toute succession de deux machines (i, i') implique un temps représentatif du délai de transport entre les deux machines M_i et $M_{i'}$ obtenu à partir de la matrice des délais. Par conséquent, le coût, que nous souhaitons minimiser, de toute solution est le plus grand délai de transport entre deux machines successives dans la dite séquence. Ces délais de transport seront appliqués aux tâches qui seront éventuellement préemptées.

- **Fonction objectif :** comme il vient d'être mentionné concernant l'évaluation de la qualité de toute solution, il est question de fournir la meilleure séquence possible de machines suivant laquelle les tâches vont être préemptées. Par la meilleure séquence, nous parlons de celle avec le moindre coût, à savoir le plus petit délai de transport possible intervenant dans une telle séquence. Formellement, soit une séquence de machines que l'on note S, et soient M_i et $M_{i'}$ deux machines successives dans S représentées par les indices (i, i'). Notre objectif est de minimiser le maximum des délais de transport de S, ce qui peut être formulé par l'expression $minimiser\ \{max\{d_{ii'}\}, \forall (i, i') \in S\}$.

Une fois la séquence de machines déterminée, nous passons à présent à la détermination des meilleures affectations des tâches aux machines en commençant par la dernière machine de la séquence obtenue, et en suivant l'ordre inverse de cette séquence. Par cet ordre, les délais de transport intervenant lors d'éventuelles préemptions de tâches, seront ceux qui appartiennent à la séquence, et qui ont été sélectionnés par la méthode de la recherche locale de cette première étape.

5.3 Évaluation des ordonnancements

Évaluer une affectation des tâches permet de déterminer la valeur de la fonction objectif du problème étudié, c'est-à-dire la longueur de l'or-

donnancement. Pour y parvenir, nous nous reposons sur les résultats et les remarques posés à la fin du chapitre précédent. Rappelons que, pour l'ensemble des instances générées et résolues par l'exécution du modèle mathématique, les ordonnancements obtenus comportaient au moins une tâche préemptée une fois. Ces derniers ont été formés par des tâches exécutées au maximum en deux parts. En se basant sur cette caractéristique, nous avons adopté deux modes d'évaluation que nous désignons par *horizontal* et *vertical*, et qui pourront à travers leurs fonctionnements n'engendrer que des tâches s'exécutant au plus en deux parts.

Le premier mode d'évaluation est dit *horizontal*, noté *Hor*, illustre la construction d'un ordonnancement suivant l'algorithme de McNaughton. Les tâches sont affectées l'une après l'autre aux machines de la séquence S en commençant par la dernière machine. Sur chaque machine, à chaque fois que la borne de McNaughton Δ est atteinte, deux scénarios se présentent. Le premier est rencontré lorsqu'une tâche, en cours d'exécution, termine son traitement à la date Δ. Dans ce cas, la tâche suivante sera exécutée sur la machine suivante, c'est-à-dire celle qui précède la machine actuelle dans la séquence S. Le deuxième cas est celui où la borne Δ est atteinte avant la fin de traitement de la tâche en cours d'exécution. Dans ce cas, la partie effectuée est maintenue sur la machine actuelle et le reste de la tâche est affecté à la machine suivante (toujours celle qui précède la machine actuelle dans la séquence). Les tâches préemptées dans les ordonnancements obtenus par cette méthode seront exécutées sur deux parts au maximum.

Par ailleurs, l'autre mode d'évaluation est dit *vertical*, et est noté *Ver*, applique quant à lui le principe des Algorithmes de Liste présentés dans la partie 2.3.1, et plus particulièrement ceux appliquant le principe du First Available Machine. Rappelons que ces algorithmes produisent des ordonnancements sans tâches préemptées, ce qui signifie que chaque tâche est exécutée entièrement sur la machine à laquelle elle est affectée. Le principe est très simple, les tâches étant ordonnées sont ordonnancées les unes après les autres sur la première machine disponible pouvant les exécuter.

5.4 L'heuristique H-*conj* pour déterminer une séquence de tâches

H-*conj* est une heuristique conçue dans le but de fournir des ordonnancements réalisables pour le problème $Pm|pmtn(delay_{ii'})|C_{max}$, et dont le principe s'inspire de la conjecture de Fishkin *et al.* (2005). En effet, H-*conj* permet de construire et de fournir des ordonnancements avec la particularité de contenir chacun au plus $(m-1)$ préemptions impliquant des transports de tâches. Chaque transport concerne une tâche préemptée une seule fois et donc exécutée en deux parts différentes. Les tâches qui ne sont pas préemptées, et par conséquent exécutées entièrement sur une machine, sont elles ordonnancées en utilisant un principe d'algorithme approché de type FAM.

Dans l'heuristique H-*conj*, les délais de transport présents dans la séquence S sont ordonnés par ordre décroissant. Par ailleurs, les tâches sont elles ordonnées par ordres croissant de leurs temps de traitement, et ce sont les premières tâches (à savoir, les plus petites) qui seront par la suite sujettes à la préemption.

Pour chaque paire de machines, représentée par une certaine valeur D_k indiquant le délai de transport entre ces deux ressources, nous affectons la première tâche destinée à être préemptée et n'ayant pas encore été affectée à aucune machine. Une telle tâche va être traitée en deux parts de telle sorte à ce qu'elle soit transportée entre les deux machines auxquelles elle a été affectée.

Prenons pour exemple le cas de ces deux paramètres : d_{12} est fourni par la séquence S et T_1 est une tâche sélectionnée pour être préemptée et transportée depuis la machine M_1 vers M_2. Dans ce cas, La première part de T_1 est exécutée sur la machine M_2 vers la fin ce qui signifie après la dernière tâche exécutée entièrement sans préemption sur M_2, et ce, jusqu'à atteindre la valeur Δ. Le reste de la tâche T_1 est exécutée entièrement, au début de la machine M_1. Les tâches déjà présentes sur cette machine vont être décalées.

L'intérêt quant à une telle réorganisation des tâches et des délais de

transport concernés, est d'essayer d'éviter les plus mauvaises situations qui mèneraient à des longueurs d'ordonnancement plus importantes. Ces cas de figure sont rencontrés lorsque les tâches préemptées de durée importante sont transportées sur les plus longues distances. Pour éviter cela, les grandes tâches à préempter sont éventuellement associées aux plus petits délais de transport, et ce, en cas de transport effectif. Inversement, les plus petites tâches sont, quant à elles, associées à des délais de transport plus grands dans le but d'avoir les meilleures valeurs possibles du C_{max}. Les étapes de l'heuristique H-*conj* sont regroupées et présentées dans l'algorithme 5.

Algorithme 5: L'heuristique H-*conj*

Entrées : Une instance du $Pm|pmtn(delay_{ii'})|C_{max}$
Une séquence de machines S.

Sorties : Un ordonnancement des n tâches sur les m machines

1 Ordonnancer toutes les tâches en utilisant un algorithme approché de type FAM.
2 Ordonner les tâches en utilisant la règle dite SPT, soit $p_1 \leq p_2 \leq \ldots \leq p_n$.
3 Ordonner les valeurs des délais de transport figurant dans la séquence S par ordre décroissant, posons $D_1 \geq D_2 \geq \ldots \geq D_{m-1}$ où D_k correspond à une certaine valeur $d_{ii'} \in S$.
4 **pour** $k \leftarrow 1$ **à** $m - 1$ **faire**
5   Ordonnancer les $(n - k)$ dernières tâches (i.e $J_n, J_{n-1}, \ldots, J_{n-k}$) en utilisant le principe d'un algorithme FAM en utilisant l'ordre LPT pour ces tâches.
6   **pour** $s \leftarrow 1$ **à** k **faire**
7   Injecter J_s dans l'ordonnancement partiel en la préemptant et la transportant suivant le chemin D_s.
8 Choisir le meilleur ordonnancement à partir de ceux construits à la fin des étapes 1 et 7.

Nous passons à présent à la présentation d'une autre méthode de résolution qui appartient à cette seconde phase, à savoir celle de l'affectation des tâches. Il s'agit de notre adaptation de la métaheuristique Recuit Si-

mulé pour la résolution du problème général en considérant des délais de transport variables.

5.5 Le Recuit Simulé (RS) pour déterminer une séquence de tâches

Très souvent adapté et utilisé pour la résolution des problèmes d'optimisation complexes, le recuit simulé est l'une des métaheuristiques les plus connues parmi celles utilisant une solution unique à la fois. Cette approche a été développée à partir de l'algorithme de Metropolis *et al.* (1953), qui décrit l'évolution d'un système thermodynamique en métallurgie.

Aperçu du phénomène. En métallurgie, le recuit est un processus thermodynamique qui décrit la manière avec laquelle un métal est chauffé puis refroidi, dans le but de modifier ses caractéristiques (principe de cristallisation). Plus exactement, le métal pur est soumis à un cycle de chauffage qui le maintient, d'abord à une forte température, puis contrôle son refroidissement progressif. Ce dernier est manipulé d'abord sous sa forme solide, qui sous l'effet de la chaleur, développe une énergie assez suffisante aux atomes de ce métal pour briser leurs liaisons chimiques afin retrouver leur caractère mobile. L'effet de la chaleur sur les atomes permet ainsi, de modifier leurs positions donnant lieu à de nouvelles combinaisons chimiques entre ces derniers. Si le cycle de chauffage est baissé lentement, on peut observer des mouvements d'atomes moins aléatoires accompagnés d'une énergie assez basse menant à un état d'équilibre qui correspond à chaque température. Cet équilibre permet d'aboutir à une forme du métal plus souple avec moins d'irrégularités. Par contre, si le refroidissement est brusque ou trop rapide, les atomes risquent de se figer et se retrouver bloqués dans des configurations qui ne sont ni initiales, ni celles que l'on souhaite atteindre. Ainsi, le refroidissement du système doit se faire lentement afin d'aboutir à ces états d'équilibre.

Les expériences menées par Metropolis *et al.* (1953) ont abouti à un algorithme permettant de simuler l'évolution d'un système physique instable vers un état d'équilibre thermique à une température fixée t. L'algorithme proposé génère une suite d'états successifs, résultant chacun d'un

déplacement d'atomes. Le passage d'un état vers un état suivant, se fait, soit en diminuant l'énergie du système, soit en considérant une certaine probabilité.

Si l'on considère une analogie entre l'énergie du système et la fonction objectif d'un problème d'optimisation d'une part, et entre les états du système et ses solutions réalisables d'une autre part, nous pouvons définir la notion du Recuit Simulé (RS) en tant qu'algorithme de résolution en optimisation combinatoire. Une solution appartenant au voisinage d'une solution courante n'est autre qu'un état atteignable depuis un état courant suite aux déplacements des atomes. Ainsi, une solution voisine, relative à un état prochain, n'est retenue que si cette dernière améliore la qualité de la fonction objectif, qui est relative à l'énergie de l'état. Dans le cas contraire, la solution voisine est acceptée avec une probabilité qui dépend de la détérioration engendrée et de la température actuelle. Ses transitions sont indiquées dans la figure 5.1, qui schématise la variation des valeurs d'une fonction objectif à minimiser, engendrées par une exécution du Recuit Simulé.

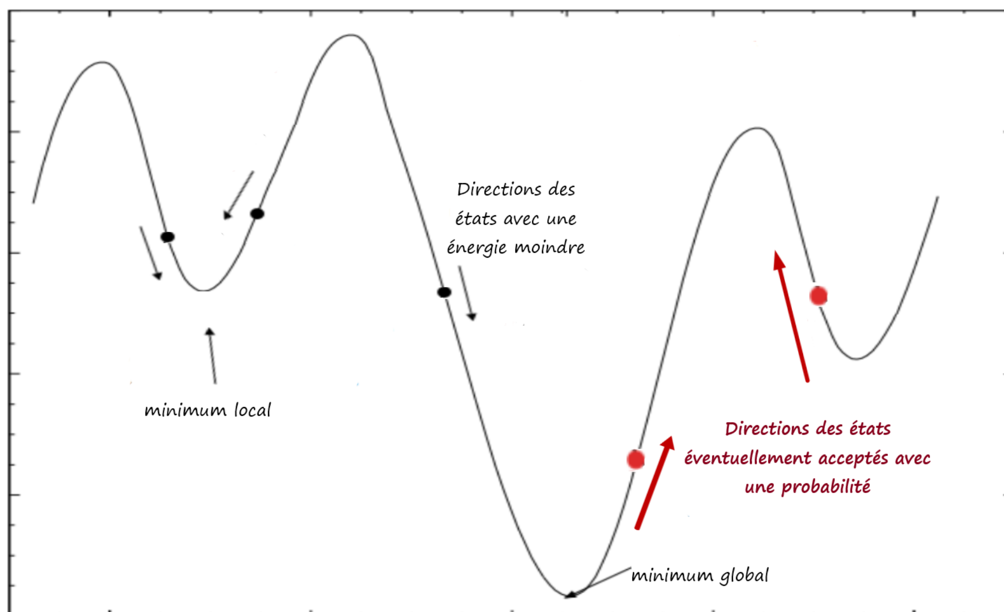


FIGURE 5.1 – Variation des énergies d'un système par un Recuit Simulé.

À très haute température, la probabilité d'accepter des solutions de moins bonne qualité est proche de 1. Ceci permet, dans un premier temps, de visiter un maximum de solutions parfois même suivant un schéma aléatoire. L'objectif de converger vers une meilleure solution peut, malheureusement, coincer le système dans un minimum local, éventuellement global. Le point fort du Recuit Simulé est justement sa capacité de s'extraire de telles zones. Les changements de température sont effectués par palier à chaque fois qu'un certain nombre d'itérations est effectué ou lorsqu'aucune amélioration n'a été enregistrée depuis un certain moment à cette température. Ces deux derniers paramètres représentent, dans la majorité des cas, les conditions d'arrêt qui permettent de baisser la température du système. Le schéma général du Recuit Simulé est présenté par le pseudo algorithme 6.

Algorithme 6: Pseudo algorithme du RS

```
1  Initialisation des paramètres : solution initiale x = x_0 et température
   initiale t = t_0;
2  répéter
3  |  répéter
4  |  |  Générer x' une solution voisine de x;
5  |  |  si x' est meilleure que x alors
6  |  |  |_ x = x' ;
7  |  |  Accepter x' avec une probabilité ≥ exp^((C_max(x') − C_max(x))/t);
8  |  jusqu'à atteindre une condition d'équilibre relative à la température
   |  actuelle t;
9  |  Mise à jour de la température;
10 jusqu'à critère d'arrêt;
```

Comme toute approche de type métaheuristique, notre adaptation du principe du RS s'est reposée sur l'implémentation des paramètres de cette méthode de la manière suivante :

- **La représentation de solutions :** comme pour les machines, un codage réel de la solution a été adopté en utilisant un vecteur réel de taille n comportant les indices des tâches.

- **La solution initiale :** il est connu pour ces méthodes, que le choix de la solution initiale n'est pas sans impact sur la qualité de la solution fournie à la fin de l'exécution. Pour cette raison, nous avons implémenté plusieurs méthodes fournissant des ordonnancements réalisables en guise de solutions de départ pour les méthodes approchées. Ces ordonnancements sont obtenus à partir de procédures de types FAM basées sur des ordres de priorité tels que LPT, SPT ou encore une génération aléatoire. Par la suite, nous avons tenté de générer, à partir des solutions précédentes, d'autres ordonnancements qui sont de bonne ou de mauvaise qualité. De tels ordonnancements sont obtenus en considérant les tâches déjà ordonnées suivant les ordres précédents. Une fois la charge sur la machine actuelle a atteint la valeur de la borne inférieure de McNaughton, la tâche en cours d'exécution est permutée avec la dernière tâche de la liste non encore exécutée. Les tâches sont :
  - Ou bien, ordonnées par ordre décroissant de leurs durées opératoires si l'on souhaite obtenir des solutions que l'on espère de bonne qualité, et ce, en interrompant l'exécution des tâches de petite taille.
  - Ou bien, ordonnées par ordre croissant de leurs durées opératoires si l'on souhaite avoir des solutions de mauvaise qualité en préemptant les tâches de grande taille.

- **Les transformations admissibles :** afin de développer le voisinage des solutions réalisables et d'en explorer le maximum de solutions voisines , nous avons implémenté et testé certains opérateurs de voisinages décrits comme suit :
  - ***insertion***, consiste à insérer une tâche choisie aléatoirement à une position aléatoire.
  - ***i-swap***, consiste à permuter i éléments désignés à leurs tours aléatoirement.
  - ***improv-swap***, effectue des permutations impliquant forcément un changement dans la valeur du C_{max} et évitant ainsi, la redondance des solutions visitées. Ces opérations sont soit effectuées pour deux tâches obligatoirement préemptées, ou bien pour une tâche préemptée et une autre tâche qui ne l'est pas.

Le schéma de refroidissement du Recuit Simulé exprime la manière avec laquelle nous décidons de mettre à jour les températures (ligne 9).

Dans Talbi (2009), nous pouvons retrouver plusieurs fonctions permettant cette action. Ces fonctions reposent sur la définition d'un facteur α, appelé facteur de décroissance de la température, dont la valeur doit être comprise entre 0 et 1. Si la valeur de α est assez proche de 1, le refroidissement est considéré lent, et permet d'avoir assez de temps afin de rencontrer de bonnes solutions (ce qui est relatif à l'aboutissement à un état d'équilibre thermodynamique). Cependant, plus la valeur de α est proche de 0, plus le refroidissement est considéré rapide ou brusque, ce qui pourrait mener la recherche vers des minimums locaux (Yagouni, 2017). Dans notre cas, les valeurs retenues pour les différents paramètres du Recuit Simulé, figurent dans la partie expérimentale.

5.6 La Recherche à Voisinage Variable pour déterminer une séquence de tâches

La Recherche à Voisinage Variable, ou VNS de l'anglais Variable Neighborhood Search, est également une métaheuristique manipulant une solution unique, qui a été proposée par Mladenović & Hansen (1997) pour la résolution du problème du voyageur de commerce. VNS est non seulement connue pour son efficacité avérée ces dernières années, et exprimée par le nombre de travaux scientifique l'ayant utilisée, mais aussi pour sa simplicité d'implémentation et d'adaptation à de nombreux problèmes d'optimisation combinatoires et d'ordonnancement plus particulièrement.

Depuis sa formalisation, nombreuses variantes de la métaheuristique VNS ont vu le jour, et plusieurs travaux passent en revue bon nombre d'entre elles. D'ailleurs, la thèse de Mjirda (2014) a particulièrement retenu notre attention, et c'est dans ce sens que nous invitons le lecteur à consulter sa partie introductive. Pour les problèmes d'ordonnancement, nous tenons également à citer les travaux suivants : Gagné *et al.* (2005); Sevkli & Aydin (2006); De Paula *et al.* (2007); Behnamian *et al.* (2009); Klemmt *et al.* (2009); Costa *et al.* (2012).

Cette méthode a la particularité de considérer, pour une solution actuelle, plusieurs voisinages définis obligatoirement au début de la méthode, et qui seront visités l'un après l'autre. Le passage d'un voisinage

à un autre se poursuit lorsque le processus de recherche se retrouve coincé dans un optimum local, et ce, suite à l'utilisation d'un principe de recherche locale.

Plus explicitement, la méthode VNS repose principalement sur deux phases. La première consiste à générer, pour une solution courante x, une nouvelle solution voisine x' piochée aléatoirement à partir du voisinage courant $N_k(x)$ (un des voisinages définis). Cette phase est généralement appelée phase de perturbation ou phase de secousse (*shaking* en anglais). La seconde phase consiste en l'application d'une recherche locale, à partir de la solution x', et fournissant comme résultat une nouvelle solution notée x". Si cette nouvelle solution est de meilleure qualité que x', elle sera retenue comme solution actuelle, et le rang du voisinage sera réinitialisé. Dans le cas contraire, la recherche locale, à partir de la solution x' est relancée dans le voisinage suivant $N_{k+1}(x)$. Ces étapes sont résumées par l'algorithme 7.

Algorithme 7: Pseudo algorithme de VNS

1 Initialisation des paramètres : Solution initiale $x = x_0$ et un ensemble de structures de voisinage N_k, $k = 1, \ldots, k_{max}$;
2 **répéter**
3 $k = 1$;
4 **répéter**
5 Sélectionner une solution voisine x' in $N_k(x)$;
6 Appliquer une recherche locale à partir de x', soit x" la solution obtenue;
7 **si** *x" est de meilleure qualité que x* **alors**
8 La nouvelle solution est retenue $x = x$" et $k = 1$;
9 $k = k + 1$;
10 **jusqu'à** $k = k_{max}$;
11 **jusqu'à** *Critère d'arrêt vérifié*;

Comme indiqué dans cet algorithme, la définition des structures de voisinage est un paramètre à effectuer au début de la méthode. Pour notre cas, nous avons développé trois structures de voisinage ($k_{max} = 3$) propres à chaque mode d'évaluation utilisé pour la construction d'un ordonnan-

cement réalisable. Les solutions initiales ont été construites, soit aléatoirement, soit à partir d'une exécution du Recuit Simulé (décrit en section précédente).

Pour une évaluation horizontale, les structures de voisinage considérées et utilisées sont les suivantes :

- **La permutation de deux tâches :** deux tâches aléatoirement sélectionnées sont permutées. Toutes les permutations de deux tâches sont possibles, à l'exception du cas où les deux tâches en question sont exécutées entièrement sur une même machine.

- **La permutation de tâches traitées par deux machines différentes :** deux machines différentes sont sélectionnées aléatoirement. Toutes les permutations des tâches exécutées par chacune d'entre elles sont permises.

- **L'insertion d'une tâche sur une autre machine :** une tâche est sélectionnée aléatoirement. Tous les transferts de cette tâche vers les autres machines (désignées aléatoirement) sont permis, à l'exception de la machine par qui elle est traitée au départ.

De la même manière, les structures de voisinage développées lorsque l'évaluation est verticale sont les suivantes :

- **La permutation de deux tâches :** deux tâches aléatoirement sélectionnées sont permutées, et toutes les permutations de deux tâches sont possibles.

- **L'inversion de l'ordre des tâches :** deux indices de tâches sont générés aléatoirement. L'ordre de toutes les tâches présentes entre ces deux tâches sélectionnées est inversé.

- **La permutation des tâches de deux machines spécifiques :** il s'agit particulièrement des deux machines présentant le plus grand écart de charges calculées à partir des tâches qu'elles exécutent. Il s'agit donc de la machine la moins et la plus chargée de l'ordonnancement actuel. Toutes les permutations de tâches entre ces deux machines sont pos-

sibles afin d'obtenir des ordonnancements plus équilibrés dans l'optique d'aboutir à de meilleurs ordonnancements.

La recherche locale, appliquée à l'étape 6, a également été prévue et conçue pour chaque structure de voisinage et pour chaque mode d'évaluation. Pour une évaluation horizontale, les procédures mises en oeuvre pour chaque structure sont respectivement désignées par les termes LS1-Hor, LS2-Hor et LS3-Hor. Ces dernières sont désignées par les termes LS1-Ver, LS2-Ver et LS3-Ver respectivement dans le cas d'une évaluation verticale. Plus précisément, ces procédures développées chacune de la manière suivante :

LS1-Hor :

Pour chaque tâche $T_{j_1} \in T$ et $T_{j_2} \in T$ ($j_1 \neq j_2$ et hormis la situation interdite[1]),
Faire
Si *la solution obtenue par la permutation de* T_1 *et* $T_2 <$ *la solution courante,*
Alors permuter T_1 et T_2.

LS2-Hor :

Pour chaque machine $M_1 \in M$ et $M_2 \in M$ ($M_1 \neq M_2$),
Pour chaque tâche T_1 traitée par M_1, et chaque tâche T_2 traitée par M_2,
Faire
Si *la solution obtenue par la permutation de* T_1 *et* $T_2 <$ *la solution courante,*
Alors permuter T_1 et T_2.

LS3-Hor :

Pour chaque tâche $T_1 \in T$
(a) Trouver une machine M_1 traitant T_1.

1. Dans laquelle les deux tâches désignées sont exécutées entièrement sur la même machine.

(b) **Pour** toute machine $M_2 \in M$ ($M_1 \neq M_2$) et toute position valide p sur M_2,
Faire Si *la solution dans laquelle la tâche T_1 est exécutée à la position p par la machine $M_2 <$ la solution courante,*
Alors transférer la tâche T_1 de M_1 vers M_2 à la position p.

LS1-Ver :

Pour toute tâche $T_1 \in T$ et $T_2 \in T$ ($j_1 \neq j_2$),
Faire Si *la solution obtenue en permutant T_1 et $T_2 <$ la solution courante,*
Alors Permuter les tâches T_1 et T_2

LS2-Ver :

Pour toute position p_1 et toute position p_2 ($p_1 < p_2$), **Faire Si** *la solution obtenue en inversant toutes les tâches qui se trouvent entre les positions p_1 et $p_2 <$ la solution courante,*
alors inverser l'ordre d'exécution de toutes ces tâches comprises entre p_1 et p_2.

LS3-Ver :

1. Trouver la machine la plus chargée de la solution actuelle, soit M_i
2. Trouver la machine la moins chargée de la solution actuelle, soit $M_{i'}$
3. **Pour** toute tâche T_1 traitée par M_i, et toute tâche T_2 traitée par $M_{i'}$,
Faire Si *la solution obtenue par la permutation des tâches T_1 et $T_2 <$ la solution actuelle,*
Alors Permuter les tâches T_1 et T_2.

5.7 Expérimentation numérique

Nous présentons dans cette section les résultats numériques obtenus après plusieurs exécutions des méthodes approchées décrites dans ce chapitre. L'implémentation des méthodes a été réalisée moyennant les mêmes caractéristiques logicielles et ressources informatiques utilisées dans les chapitres 3 et 4.

Comme mentionné au début de ce chapitre, il a été question de mettre en oeuvre une stratégie de résolution pour le $Pm|pmtn(delay_{ii'})|C_{max}$. Cette approche est conçue sur deux phases : la première concerne la détermination de la meilleure séquence de machines possible, et ce, en utilisant une procédure de recherche locale. Ladite séquence sera utilisée comme une information en entrée pour la deuxième phase afin d'indiquer l'ordre des machines suivant lequel les tâches vont être ordonnancées. La seconde phase concerne l'affectation des tâches aux machines, et est réalisée par l'exécution des méthodes approchées décrites plus haut, à savoir l'heuristique H-*conj* ainsi que l'adaptation des deux métaheuristiques le Recuit Simulé et la Recherche à Voisinage Variable.

En l'absence d'instances de type Benchmark pour le problème considéré, une pile d'instance a été générée aléatoirement suivant une loi uniforme. Les délais de transport de toutes les instances ont été générés dans le même intervalle que celui présenté en chapitre 3, de telle sorte à avoir des instances non solvables en temps polynomial par l'algorithme de McNAUGHTON. Quant aux autres caractéristiques des instances, leur génération est décrite comme suit :

- n généré dans l'ensemble $\{50, 70, 100, 200, 250, 500, 700\}$.
- m généré dans l'ensemble $\{5, 10, 15, 20, 25\}$.
- $\forall j = \overline{1, n}$, p_j est généré dans les deux intervalles $[1, 100]$ et $[50, 100]$.

L'analyse de la qualité des solutions fournies par les méthodes implémentées repose sur le calcul de l'écart (déviation) moyen et maximum. Rappelons que cet indicateur est calculé à partir des valeurs des solutions obtenues par la méthode approchée (notées $C_{max}(MA)$), et celle de la solution optimale ($C_{max}(OPT)$), dans notre cas, celle calculée par la borne inférieure de McNaughton. Cet écart est calculé par la formule mathématique $\frac{C_{max}(MA)-C_{max}(OPT)}{C_{max}(OPT)}$.

Aussi, et en plus du temps de calcul estimé en secondes, nous avons également considéré le critère $Nopt$ qui désigne le nombre totale d'instances pour lesquelles, les solutions fournies par les méthodes approchées ont atteint la borne inférieure.

Paramétrage des méthodes

Plusieurs valeurs ont été testées pour les paramètres du Recuit Simulé. Les exécutions maintenues sont celles débutant l'algorithme avec une température initiale t_0 égale à 500. La mise à jour de la température est effectuée suivant une fonction géométrique en fixant le coefficient de refroidissement α à 0.98. Pour échapper à un piégeage de l'algorithme tout en sauvegardant la valeur de l'optimum local rencontré, le nombre d'itérations maximum toléré avant de baisser la température actuelle a été fixé à 5000 itérations. L'exécution du recuit simulé prend fin lorsque la température du système atteint la valeur finale fixée à 0.00001. Concernant la recherche à voisinage variable, les trois voisinages ont été utilisés ainsi que les procédures de recherche locale conçue pour chacun d'entre eux. La condition d'arrêt de la méthode est représentée par le nombre maximum d'itérations, qui a été limité à 2000 itérations.

Résultats et discussion

Tous les résultats numériques sont rapportés dans les tables 5.1, 5.2, 5.3 et 5.4. Ces derniers comprennent les temps d'exécution, le critère $Nopt$, la déviation moyenne et la déviation maximum de toutes les instances générées.

Les tables 5.1 et 5.2 présentent les résultats obtenus par l'application des différentes méthodes sur des instances de taille appartenant à l'intervalle $[1, 100]$. La première colonne de la table 5.1 affiche les temps de calcul de la première phase de la stratégie de résolution, des temps négligeables compte tenu du nombre réduit de machines. En termes de temps de calcul, jugés tout de même raisonnable pour l'ensemble des implémentations, il est clair que l'heuristique H-*conj* est la plus rapide de toutes les méthodes implémentées avec des solutions fournies en un maximum de temps, évalué à 0.02 secondes, pour des instances à 700 tâches. Ceci est expliqué par le fonctionnement lui-même de la méthode peu coûteux et peu gourmand en mémoire, et qui est dû au nombre réduit de possibilités d'ordonnancements possibles comportant au plus $(m - 1)$ préemptions. De plus, l'ordonnancement partiel avec les $n - m + 1$ tâches non préemptées n'est construit qu'une seule fois. D'ailleurs, les temps d'exécution des

différentes méthodes ainsi que les déviations moyennes sont représentés dans les figures 5.2 et 5.3.

$n * m$	Temps d'exécution moyen (s)						$Nopt$				
	LS	H-*conj*	SA		VNS		H-*conj*	SA		VNS	
			Hor	Ver	Hor	Ver		Hor	Ver	Hor	Ver
50*5	0.010	0.011	2.669	6.511	0.041	0.030	18	100	100	62	60
70*5	0.006	0.008	3.104	8.718	0.110	0.062	26	100	100	74	67
100*5	0.005	0.006	3.906	12.764	0.258	0.153	34	100	100	80	65
50*10	0.005	0.008	2.751	8.762	0.107	0.034	0	99	94	0	0
100*10	0.011	0.013	4.121	30.264	0.628	0.238	6	100	96	1	1
150*10	0.006	0.008	5.550	22.612	1.863	0.708	20	100	99	5	1
70*15	0.010	0.013	3.675	14.361	0.412	0.128	0	84	56	0	0
100*15	0.009	0.012	4.501	20.408	0.985	0.324	3	100	60	0	0
200*15	0.007	0.011	7.135	64.603	6.805	2.627	14	100	81	0	0
100*20	0.013	0.017	5.326	47.058	1.364	0.438	0	89	18	0	0
250*20	0.007	0.010	9.119	63.376	16.967	7.005	13	100	76	0	0
500*20	0.007	0.011	35.130	117.660	63.367	51.577	40	100	99	0	0
200*25	0.016	0.020	19.303	56.946	4.872	4.529	1	99	33	0	0
500*25	0.007	0.012	39.897	152.344	96.114	76.640	22	100	94	0	0
700*25	0.007	0.013	22.086	296.767	177.757	250.037	47	100	97	0	0

TABLE 5.1 – Temps de calcul moyen et nombre de solutions optimales pour les instances avec $p_j \in [1, 100]$

$n * m$	Déviation moyenne					Déviation maximum				
	H-*conj*	SA		VNS		H-*conj*	SA		VNS	
	(%)	Hor(%)	Ver(%)	Hor(%)	Ver(%)	(%)	Hor(%)	Ver(%)	Hor(%)	Ver(%)
50*5	0.47	0.00	0.00	1.37	0.12	1.60	0.00	0.00	10.20	1.16
70*5	0.18	0.00	0.00	0.83	0.06	0,96	0.00	0.00	10.04	0.72
100*5	0.10	0.00	0.00	0.32	0.06	0,63	0.00	0.00	3.79	0.65
50*10	1.86	0.00	0.02	18.93	1.51	04.49	0.37	0.41	36.29	5.74
100*10	0.42	0.00	0.00	8.13	0.67	1.13	0.00	0.20	15.04	2.95
150*10	0.18	0.00	0.00	4.78	0.41	0.53	0.00	0.12	12.50	1.83
70*15	2.17	0.35	0.18	26.66	2.77	5.11	5.62	0.50	39.90	6.61
100*15	0.96	0.00	0.11	18.81	2.00	2.74	0.00	0.34	27.76	5.11
200*15	0.18	0.00	0.02	8.41	0.87	0.63	0.00	0.15	13.88	2.33
100*20	2.02	0.21	0.32	29.32	3.60	4.54	3.00	0.45	39.67	7.66
250*20	0.20	0.00	0.03	11.20	1.23	0.58	0.00	0.17	15.30	3.40
500*20	0.04	0.00	0.00	5.53	0.59	0.15	0.00	0.08	7.25	1.40
200*25	0.69	0.00	0.16	19.40	2.80	1.59	0.74	0.48	23.31	5.22
500*25	1.24	0.00	0.00	7.39	1.03	0.28	0.00	0.10	9.60	1.85
700*25	0.03	0.00	0.00	5.45	0.73	0.14	0.00	0.07	7.12	1.58

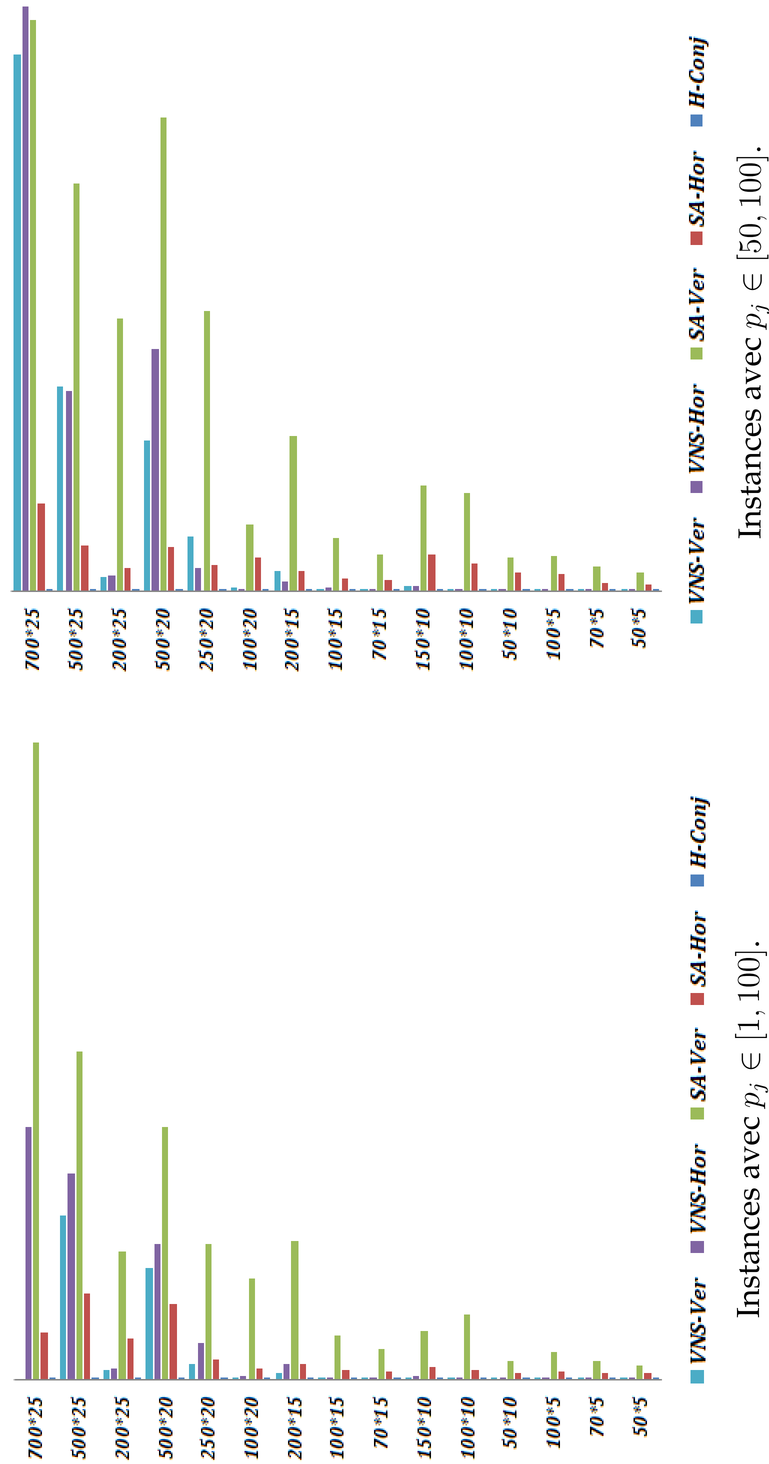
TABLE 5.2 – Déviation moyenne et maximum pour les instances avec $p_j \in [1, 100]$.

Sauf que, en marge de sa rapidité, les solutions fournies ne sont pas à la hauteur de celles obtenues par l'adaptation du Recuit Simulé en termes de nombre de solutions optimales. Néanmoins, elle propose de bien meilleures solutions que ceux obtenues par la Recherche à Voisinage Variable, notamment à partir de 150 tâches et 10 machines, où l'on obtient un total de 160 solutions optimales contre 5 seulement obtenues par la Recherche à Voisinage Variable.

En se basant sur l'ensemble des critères considérés, l'évaluation horizontale des listes fournies par le Recuit Simulé a été la plus efficace de toutes, en fournissant un nombre considérable de solutions optimales (exactement 1471 solutions optimale sur un total de 1500 instances testées), en des temps très raisonnables qui ne dépassent pas les 40 secondes. Pour les autres cas, n'ayant pas atteint les valeurs de la borne inférieure, de très bonnes solutions réalisables ont été obtenues en enregistrant, dans le pire des cas, des déviations moyennes (autour de 0.35%) bien inférieures à 1% comme c'est le cas pour la configuration $(70 * 15)$. Les plus mauvaises solutions sont attribuées à la recherche à Voisinage Variable en observant des solutions réalisables très éloignées avec en moyenne des déviations maximums pouvant atteindre 39.67 (figure 5.3(a)).

Depuis la table 5.3, nous pouvons remarquer que, pour les instances avec des temps de traitement générés dans l'intervalle $[50, 100]$, l'heuristique H-*conj*, enregistre les mêmes observations que celles du cas précédent. Les temps sont aussi rapides pour ces tâches (n'excèdent pas les $0.014s$). L'heuristique se comporte aussi moyennement par rapport au nombre de solutions optimales, d'ailleurs ce nombre a été revu à la hausse (475 solutions optimales au total contre 244 pour le cas précédent) notamment lorsque n et m dépassent les valeurs de 500 et 20 respectivement. De même pour le reste des instances, les solutions réalisables sont aussi de bonne qualité et assez proches de l'optimum avec une seule mauvaise valeur de déviation maximum enregistrée pour la configuration $(70, 15)$ (voir table 5.4).

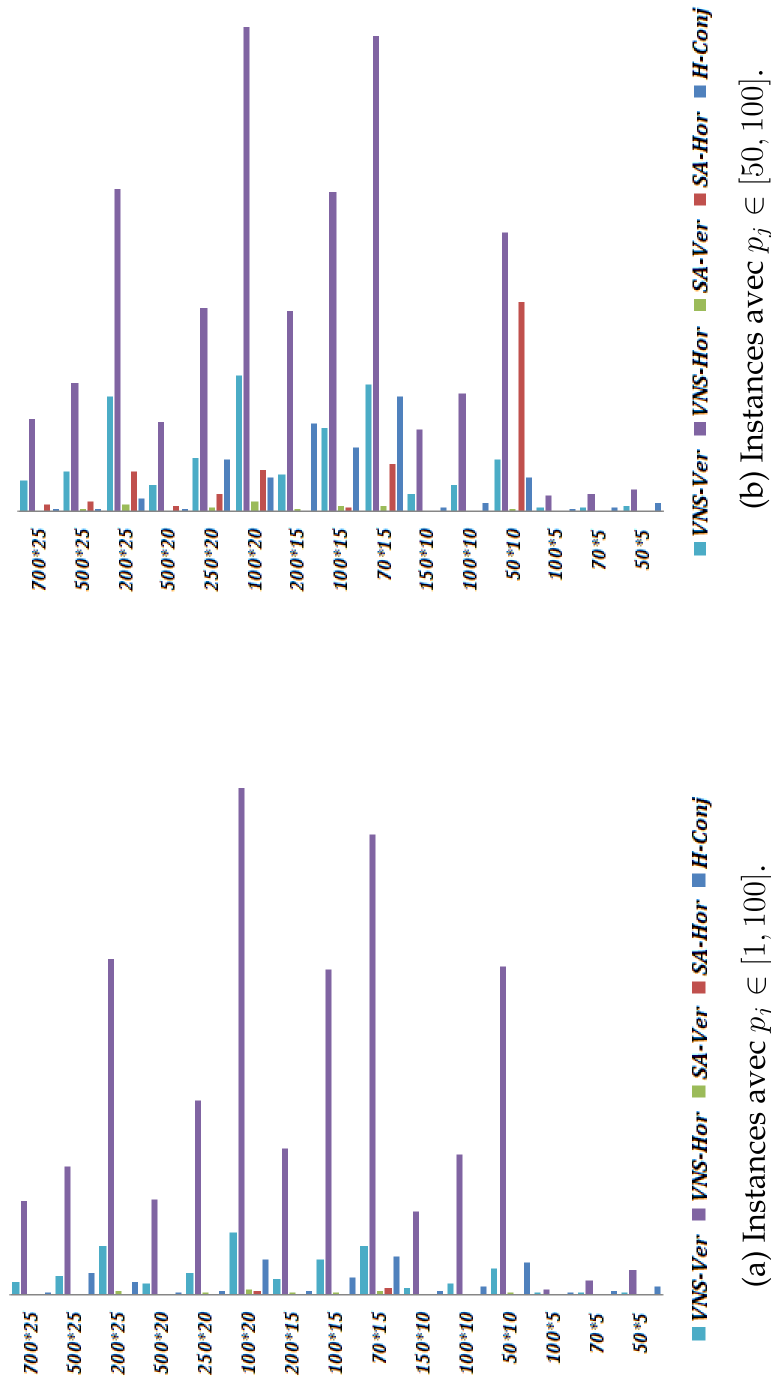
Contrairement aux précédentes instances, les tables 5.3 et 5.4 montrent que l'évaluation verticale des listes fournies par le Recuit Simulé est meilleure que sa version horizontale, et bien meilleure que les deux évaluations obtenues par la Recherche à Voisinage Variable, même si cette dernière



Instances avec $p_j \in [1, 100]$.

Instances avec $p_j \in [50, 100]$.

FIGURE 5.2 – Représentations des temps d'exécution moyens (en secondes)



(a) Instances avec $p_j \in [1, 100]$.

(b) Instances avec $p_j \in [50, 100]$.

FIGURE 5.3 – Représentations des déviations moyennes (%).

s'est sensiblement améliorée pour ce type d'instances. Nous observons pour elle, un total de 1119 solutions optimales obtenues (contre 875 pour le cas horizontal), et des déviations moyennes et maximales ne dépassant jamais les 1%.

$n * m$	Temps d'exécution moyen (s)						$Nopt$				
	LS	H-*conj*	SA		VNS		H-*conj*	SA		VNS	
			Hor	Ver	Hor	Ver		Hor	Ver	Hor	Ver
50*5	0.004	0.005	2.3760	6.8547	0.0390	0.0267	27	100	100	44	56
70*5	0.0041	0.0055	2.9146	8.8016	0.0934	0.0741	40	100	100	49	60
100*5	0.0048	0.0062	5.9564	12.5359	0.2375	0.2678	70	100	100	37	64
50*10	0.0052	0.0075	6.6468	12.4403	0.0936	0.0906	7	90	92	0	1
100*10	0.0052	0.0080	9.9392	35.7369	0.5782	0.6708	27	99	98	1	2
150*10	0.0051	0.0077	13.3402	38.7827	1.8531	1.9501	49	100	93	0	2
70*15	0.0073	0.0105	3.9176	13.4833	0.4169	0.3316	0	32	66	0	0
100*15	0.0075	0.0109	4.4600	19.5019	1.0375	0.9175	0	92	56	0	0
200*15	0.0077	0.0111	7.0039	56.6595	3.2229	7.0006	0	100	64	0	0
100*20	0.0103	0.0142	12.4727	24.0165	0.6199	1.3298	3	11	27	0	0
250*20	0.0086	0.0120	9.1818	102.0953	8.1347	19.79	0	41	41	0	0
500*20	0.0064	0.0106	16.2658	173.2043	88.3763	55.1809	81	9	84	0	0
200*25	0.0070	0.0115	8.2879	99.2426	5.8590	5.0386	13	0	28	0	0
500*25	0.0070	0.0120	16.7314	148.7207	73.1928	74.5991	70	0	81	0	0
700*25	0.0072	0.0124	31.6880	208.5003	213.6951	196.2718	88	1	89	0	0

TABLE 5.3 – Temps de calcul moyen et nombre de solutions optimales pour les instances avec $p_j \in [50, 100]$

L'adaptation de la Recherche à Voisinage Variable, présentée dans cette thèse, pour la résolution du problème $Pm|pmtn(delay_{ii'})|C_{max}$ n'a été intéressante que pour des instances de petite taille (dont le nombre de tâches ne dépasse pas les 100). Cependant, une légère amélioration a été remarquée dans le traitement d'instances comportant de grandes tâches ($p_j \geq 50$). Les solutions réalisables fournies pour de telles instances ont été de meilleure qualité avec au pire des cas une déviation moyenne aux environs de 6.14% (au lieu de 29.32%), et une déviation maximum égale à 14% (au lieu de 36.29%).

Nous avons donc remarqué que pour des instances comportant de grandes tâches ($50 \leq p_j \leq 100$), la version verticale (n'engendrant pas de préemptions de tâches) est plus favorable (figure 5.3(b)). Pour de grandes tâches, la préemption a engendré, sur la quasi-totalité des instances, automatiquement un décalage des parties des tâches à transporter, ce qui

$n * m$	Déviation moyenne					Déviation maximum				
	H-*conj*	SA		VNS		H-*conj*	SA		VNS	
	(%)	Hor(%)	Ver(%)	Hor(%)	Ver(%)	(%)	Hor(%)	Ver(%)	Hor(%)	Ver(%)
50*5	0.16	0.00	0.00	0.40	0.09	1.02	0.00	0.00	2.41	0.82
70*5	0.07	0.00	0.00	0.33	0.07	0.28	0.00	0.00	4.21	0.79
100*5	0.02	0.00	0.00	0.30	0.06	0.14	0.00	0.00	2.98	0.67
50*10	0.64	0.04	0.02	5.32	0.99	2.08	0.80	0.27	11.19	2.60
100*10	0.14	0.00	0.00	2.24	0.49	0.54	0.13	0.13	5.92	2.39
150*10	0.05	0.00	0.00	1.55	0.33	0.18	0.00	0.09	3.74	1.35
70*15	2.17	0.89	0.09	9.06	2.41	3.75	2.82	0.29	14.00	5.68
100*15	1.20	0.05	0.08	6.08	1.58	2.05	0.99	0.20	9.91	3.33
200*15	1.67	0.00	0.03	3.81	0.70	2.01	0.00	0.10	5.03	1.49
100*20	0.64	0.78	0.19	9.22	2.58	1.58	1.85	0.27	13.12	5.13
250*20	0.97	0.31	0.06	3.86	1.02	2.58	0.96	0.11	5.26	2.57
500*20	0.01	0.08	0.00	1.69	0.48	0.05	0.21	0.05	2.62	1.02
200*25	0.23	0.74	0.12	6.14	2.17	0.66	1.85	0.33	8.23	4.34
500*25	0.02	0.19	0.01	2.44	0.75	0.07	0.33	0.06	3.24	1.41
700*25	0.01	0.12	0.00	1.75	0.57	0.05	0.23	0.04	2.40	1.32

TABLE 5.4 – Déviation moyenne et maximum pour les instances avec $p_j \in [50, 100]$.

conduit à une augmentation de la longueur de l'ordonnancement. Par ailleurs, les instances pour lesquelles les longueurs des tâches ont été générées dans l'intervalle $[1, 100]$, et dans lesquelles nous avons remarqué une forte présence de tâches unitaires subissaient moins de décalages donnant lieu à des ordonnancements de plus courtes durées. En termes de temps de calcul également, l'ordonnancement selon le mode horizontal en présence de tâches unitaires est moins coûteux que le mode vertical à cause du nombre réduit de préemptions et de décalages possibles.

Les résultats présentés dans ce chapitre ainsi que dans le précédent ont fait l'objet d'un article publié (Badaoui *et al.*, 2020).

Conclusion générale

Le travail présenté dans cette thèse s'inscrit dans le domaine de l'ordonnancement. Nous nous sommes intéressés à l'étude du problème d'ordonnancement des tâches sur des machines parallèles identiques en vue de minimiser la durée totale de l'ordonnancement. En particulier, nous nous sommes penchés sur le cas des ordonnancements préemptifs en présence de délais de transport. Nous avons supposé tout au long de cette thèse, que le transport des tâches préemptées est régi par la présence de délais minimums à respecter qui dépendent uniquement de la distance entre les machines.

Le problème étudié est particulièrement important vu qu'il modélise le fonctionnement de certains ateliers de production établis en plusieurs sites industriels, où la production des tâches peut s'effectuer sur un ensemble de sites. Dans ce cas, la notion de distance entre les multiples points de production n'est pas à ignorer car elle intervient directement sur la durée de fin de traitement de chaque tâche transportée, et par conséquent intervient sur la durée du projet total.

Dans le premier chapitre, nous avons présenté quelques définitions et notions de base utiles et nécessaires à la compréhension de tout problème d'ordonnancement d'atelier. Puis nous avons présenté quelques notions de complexité des problèmes d'optimisation afin de situer le problème

étudié par rapport à sa difficulté. En fonction de sa complexité, nous avons cité quelques exemples d'approches possibles pour sa résolution.

Dans le chapitre 2, nous avons présenté une description mathématique du problème d'ordonnancement préemptif sur machines parallèles en présence de délais de transport, ainsi qu'un exemple introductif sur quelques ordonnancements réalisables ou pas. Nous avons présenté également une étude bibliographique qui porte sur les problèmes d'ordonnancement sur machines parallèles d'abord, incluant ou pas la notion de préemption ainsi que des études rencontrées avec et sans prise en charge des délais de transport. Notre étude a montré que, comparé à la littérature dédiée aux problèmes classiques d'ordonnancement, les travaux traitant des délais de transport plus particulièrement dans le cas des machines parallèles ne sont pas très nombreux. En effet, la majorité des études menées dans ce sens supposent que la reprise du traitement d'une tâche préemptée sur une autre machine peut débuter immédiatement après son interruption, ce qui n'est pas très réaliste.

Le chapitre 3 représente l'étude d'un cas particulier, celui du problème d'ordonnancement préemptif en présence de délais de transport identiques, et ce, sur un nombre de machines égal à trois. Cette étude représente une généralisation du cas de deux machines présenté auparavant. Dans ce chapitre, nous décrivons un algorithme de résolution exacte de complexité pseudo-polynomiale, et nous montrons également que le cas traité dans ce chapitre admet, lui aussi, un algorithme d'approximation complètement polynomial.

Dans le chapitre 4, nous nous sommes intéressés au problème général sur m machines en considérant des délais de transport **variables**. Nous avons présenté, suite à certains résultats publiés récemment, une nouvelle formulation mathématique en nombre entiers et binaires. Nous avons décrit dans un premier temps, l'ensemble des contraintes exprimées dans ce modèle ainsi que le principe de son exécution. À partir des résultats obtenus et présentés à la fin du chapitre, nous pouvons constaté une amélioration, d'abord en termes de taille du modèle proposé par rapport au dernier modèle existant dans la littérature, ainsi qu'en termes du nombre d'ordonnancements optimaux obtenus dans la limite du temps imparti. Nous avons présenté également un constat sur l'aspect des ordonnance-

ments optimaux, plus précisément sur le nombre maximal de préemptions pour un ordonnancement optimal et qui nous a été utile pour le chapitre suivant.

Dans le chapitre 5, nous nous sommes penchés sur la résolution approchée du problème général sur m machines en considérant des délais de transport variables. Les étapes ont été présentées sous la forme d'une stratégie de résolution établie en deux phases. La première a consisté en la détermination d'une séquence de machines suivant laquelle les tâches vont par la suite être ordonnancées, et ceci, en utilisant une procédure de recherche locale assez rapide. L'affectation des tâches aux machines est présentée en seconde étape, et est établie par une heuristique dont le principe s'inspire d'une conjecture précédemment posée, et par l'adaptation de deux métaheuristiques, le Recuit Simulé et la Recherche à Voisinage Variable.

Les méthodes citées ont été implémentées et testées sur des instances générées aléatoirement. L'expérimentation et les résultats numériques ont montré que les méthodes approchées mises en oeuvre pour la résolution du problème étudié ont, en général, toutes fourni des solutions de bonnes qualités au vu des déviations moyennes et maximales rapportées. Bien que l'heuristique utilisée a été la plus rapide à fournir de telles solutions, l'adaptation du Recuit Simulé a fourni, en des temps aussi rapides, les meilleurs ordonnancements en terme de qualité et de nombre de solutions optimales obtenues. La recherche à Voisinage Variable a été meilleure en traitant des instances comportant de grandes tâches. Nous avons également remarqué, que la façon d'ordonnancer les tâches impliquant des préemptions ou pas, pouvait affecter la qualité des solutions en fonction de la nature des tâches. Dans ce sens, nous avons constaté que pour certains cas comportant un nombre considérable de tâches de grande taille, il est préférable de ne pas favoriser leurs préemptions quel que soit la séquence de machines afin d'éviter tout décalage et tout temps mort, qui pourraient engendrer des valeurs plus grandes de *makespan*. Dans le cas opposé, l'ordonnancement des tâches suivant la règle de McNaughton était meilleur et pouvait fournir d'avantage d'ordonnancements optimaux.

Le travail proposé dans cette thèse a fait apparaitre de nombreuses perspectives de recherche touchant aussi bien à l'aspect théorique, qu'au

volet expérimental. Ainsi, dans la continuité de notre travail, nous pensons qu'il serait intéressant de considérer les axes de recherche suivants :

D'abord, nous pouvons nous intéresser au problème étudié en se penchant sur les modalités de transport. Ceci peut se traduire par l'ajout de nouvelles contraintes relatives aux opérateurs de transport, à leurs nombres, leurs capacités et à leurs disponibilités. Nous pensons également, qu'il serait possible de considérer d'autres contraintes sur les tâches, telles que les dates de disponibilité et/ou les dates butoirs, ou encore d'envisager de traiter le problème de transport avec d'autres critères d'optimisation.

Aussi, nous pensons qu'il serait particulièrement intéressant de considérer, à partir du problème étudié, une nouvelle version dont laquelle les temps, à respecter en cas de préemption, soient relatifs à chaque tâche. Des temps pareils peuvent représenter des temps de transport ou des temps de préparation.

Concernant les méthodes de résolution, nous pensons qu'il serait possible d'améliorer les approches implémentées, en proposant de nouvelles adaptations (ou hybridations) de métaheuristiques pouvant traiter des instances de taille plus importante.

Annexe

Dans ce qui suit, nous proposons un exemple démonstratif qui permet de mieux visualiser le fonctionnement du programme dynamique DP, pour une instance du problème $P_3||C_{max}$.

Exemple démonstratif. Considérons une petite instance du problème $P3||C_{max}$ composée de six tâches. Les temps de traitement de l'ensemble des tâches sont présentés dans la table 5.

T_j	T_1	T_2	T_3	T_4	T_5	T_6
p_j	8	6	5	4	5	4

TABLE 5 – Durées opératoires des tâches.

Les tables qui vont suivre comportent toutes les valeurs calculées de la fonction F. Pour une représentation à deux dimensions, nous convenons que chaque table soit relative à une valeur de P_1. Pour chaque valeur de celle-ci, une matrice est créée dans laquelle chaque ligne j correspond à l'affectation d'une tâche T_j. Les colonnes représentent, quant à elles, les valeurs de P_2. Pour que le déroulement de l'exemple puisse être le plus clair possible, les valeurs de P_1 et de P_2 doivent varier jusqu'à atteindre la valeur maximale. Pour cela, nous utilisons une borne supérieure pour le problème qui vaut 12.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	**8**	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	**8**	8	6	6	6	6	6
3	19	19	19	19	19	14	13	13	11	11	11	8	8
4	23	23	23	23	19	18	17	17	15	14	13	12	11
5	28	28	28	28	24	23	22	22	20	19	18	17	16
6	32	32	32	32	28	27	26	26	24	23	22	21	20

TABLE 6 – Valeurs de la fonction F calculées pour $P_1 = 0$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	8	8	6	6	6	6	6
3	19	19	19	19	19	14	13	13	11	11	11	8	8
4	23	23	23	23	19	18	17	17	15	14	13	12	11
5	28	28	28	28	24	23	22	22	20	19	18	17	16
6	32	32	32	32	28	27	26	26	24	23	22	21	20

TABLE 7 – Valeurs de la fonction F calculées pour $P_1 = 1$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	8	8	6	6	6	6	6
3	19	19	19	19	19	14	13	13	11	11	11	8	8
4	23	23	23	23	19	18	17	17	15	14	13	12	11
5	28	28	28	28	24	23	22	22	20	19	18	17	16
6	32	32	32	32	28	27	26	26	24	23	22	21	20

TABLE 8 – Valeurs de la fonction F calculées pour $P_1 = 2$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	8	8	6	6	6	6	6
3	19	19	19	19	19	14	13	13	11	11	11	8	8
4	23	23	23	23	19	18	17	17	15	14	13	12	11
5	28	28	28	28	24	23	22	22	20	19	18	17	16
6	32	32	32	32	28	27	26	26	24	23	22	21	20

TABLE 9 – Valeurs de la fonction F calculées pour $P_1 = 3$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	8	8	6	6	6	6	6
3	19	19	19	19	19	14	13	13	11	11	11	8	8
4	19	19	19	19	19	14	13	13	11	11	11	8	8
5	24	24	24	24	24	19	18	18	16	16	14	13	13
6	28	28	28	28	24	23	22	22	20	19	18	17	16

TABLE 10 – Valeurs de la fonction F calculées pour $P_1 = 4$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	14	14	14	14	14	14	8	8	6	6	6	6	6
3	14	14	14	14	14	14	**8**	8	6	6	6	6	6
4	18	18	18	18	14	14	**12**	12	10	10	8	8	6
5	23	23	23	23	19	18	17	17	15	14	13	**12**	11
6	27	27	27	27	23	22	21	21	19	18	17	16	15

TABLE 11 – Valeurs de la fonction F calculées pour $P_1 = 5$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	8	8	8	8	8	8	8	8	0	0	0	0	0
3	13	13	13	13	13	8	8	8	5	5	5	5	5
4	17	17	17	17	13	12	12	12	9	8	8	8	5
5	22	22	22	22	18	17	17	17	14	13	12	12	10
6	26	26	26	26	22	21	21	21	18	17	16	16	14

TABLE 12 – Valeurs de la fonction F calculées pour $P_1 = 6$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	8	8	8	8	8	8	8	0	0	0	0	0
2	8	8	8	8	8	8	8	8	0	0	0	0	0
3	13	13	13	13	13	8	8	8	5	5	5	5	5
4	17	17	17	17	13	12	12	12	9	8	8	8	5
5	22	22	22	22	18	17	17	17	14	13	12	12	10
6	26	26	26	26	22	21	21	21	18	17	16	16	14

TABLE 13 – Valeurs de la fonction F calculées pour $P_1 = 7$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	6	6	6	6	6	0	0	0	0	0	0	0
3	11	11	11	11	11	6	5	5	5	5	5	0	0
4	15	15	15	15	11	10	9	9	9	6	5	4	4
5	20	20	20	20	16	15	14	14	14	11	10	9	9
6	24	24	24	24	20	19	18	18	16	15	14	13	13

TABLE 14 – Valeurs de la fonction F calculées pour $P_1 = 8$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	6	6	6	6	6	0	0	0	0	0	0	0
3	11	11	11	11	11	6	5	5	5	5	5	0	0
4	14	14	14	14	11	10	8	8	6	6	5	4	4
5	19	19	19	19	16	14	13	13	11	11	10	8	8
6	23	23	23	23	19	18	17	17	15	14	13	**12**	11

TABLE 15 – Valeurs de la fonction F calculées pour $P_1 = 9$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	6	6	6	6	6	0	0	0	0	0	0	0
3	11	11	11	11	11	6	5	5	5	5	5	0	0
4	13	13	13	13	11	8	8	8	5	5	5	4	4
5	18	18	18	18	14	13	12	12	10	10	8	8	6
6	22	22	22	22	18	17	16	16	14	13	12	12	10

TABLE 16 – Valeurs de la fonction F calculées pour $P_1 = 10$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	6	6	6	6	6	0	0	0	0	0	0	0
3	8	8	8	8	8	6	5	5	0	0	0	0	0
4	12	12	12	12	8	8	8	8	4	4	4	4	0
5	17	17	17	17	13	12	12	12	9	8	8	8	5
6	21	21	21	21	17	16	16	16	13	12	12	12	9

TABLE 17 – Valeurs de la fonction F calculées pour $P_1 = 11$.

p_j \ P_2	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	6	6	6	6	6	0	0	0	0	0	0	0
3	8	8	8	8	8	6	5	5	0	0	0	0	0
4	11	11	11	11	8	6	5	5	4	4	4	0	0
5	16	16	16	16	13	11	10	10	9	8	6	5	5
6	20	20	20	20	16	15	14	14	13	11	10	9	9

TABLE 18 – Valeurs de la fonction F calculées pour $P_1 = 12$.

De chaque table, nous pouvons extraire le meilleur ordonnancement possible **pour les valeurs en question**. Les détails des ordonnancements résultants sont rassemblés et présentés dans la table19.

Ordonnancements	Charge M_1	Charge M_2	Charge M_3	C_{max}
ordonnancement 1	0	12	20	20
ordonnancement 2	1	12	20	20
ordonnancement 3	2	12	20	20
ordonnancement 4	3	12	20	20
ordonnancement 5	4	12	16	16
ordonnancement 6	5	12	15	15
ordonnancement 7	6	12	14	14
ordonnancement 8	7	12	14	14
ordonnancement 9	8	11	13	13
ordonnancement 10	**9**	**11**	**12**	**12**
ordonnancement 11	10	10	12	**12**
ordonnancement 12	11	9	12	**12**
ordonnancement 13	12	9	11	**12**

TABLE 19 – Détails d'ordonnancements obtenus.

La table 19 décrit les meilleurs ordonnancements obtenus par le calcul de toutes les valeurs possibles de la fonction F en utilisant le programme dynamique DP. La première colonne répertorie l'ensemble des ordonnancements obtenus où chacun d'entre eux est relatif à une valeur de P_1 (c'est-

à-dire pour des valeurs de P_1 allant de 0 jusqu'à 12). La deuxième, troisième et quatrième colonne indiquent les charges sur les machines M_1, M_2 et M_3 respectivement pour chaque ordonnancement. La dernière colonne est consacrée à la longueur de chacun.

Quatre ordonnancements optimaux ont été obtenus avec un $C_{max}* = 12$. L'affectation des tâches aux machines est donc obtenue en remontant en arrière à partir de la valeur du $C_{max}* = 12$, et ce, en passant par toutes les affectations optimales (mentionnées en caractère gras) des tâches ayant mené à un ordonnancement de longueur égale à 12.

Pour l'ordonnancement sélectionné (le n° 10) relatif à $P_1 = 9$, le $C_{max}*$ est obtenu à partir des charges sur les machines mentionnées dans la table 19, à savoir des charges égales à 9, 11 et 12 pour les machines M_1, M_2 et M_3 respectivement. Pour connaitre l'affectation de la tâche T_6 de longueur 4, il suffit de savoir si cette valeur a été obtenue à partir de $F(j-1, P_1-p_j, P_2) = F(j-1, 9-4, 11)$ ou à partir de $F(j-1, P_1, P_2-p_j) = F(j-1, 9, 12-4)$ ou plutôt à partir de $F(j-1, P_1, P_2) + p_j = F(j-1, 9, 11) + 4$. Dans ce cas, il s'agit de la première expression $F(j-1, 9-4, 11) = 12$ (voir la table 11), et par conséquent la tâche T_6 est exécutée par la machine M_1. Le même principe est utilisé par la procédure de retour en arrière pour retrouver toutes les affectations des tâches restantes et qui sont d'ailleurs les valeurs marquées en caractère gras dans les tables 15, 11 et 6. De ce fait, la tâche T_3 est également traitée par la machine M_1. Le traitement des tâches T_5 et T_2 est assigné à la machine M_2. Les tâches restantes sont exécutées par la machine M_3. L'ordonnancement résultant est présenté dans la figure 4.

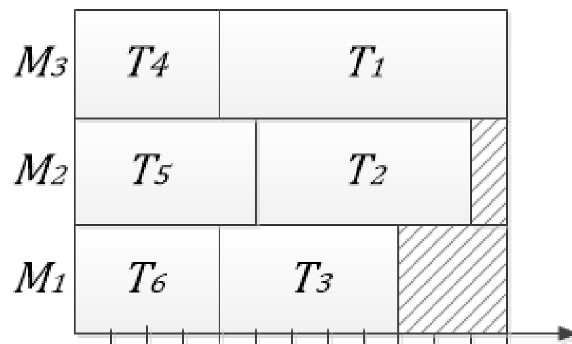


FIGURE 4 – Ordonnancement optimal de longueur 12 obtenu par DP pour l'instance de la table 5

Bibliographie

Afrati, Foto, Bampis, Evripidis, Finta, Lucian, & Milis, Ioannis. 2005. Scheduling trees with large communication delays on two identical processors. *Journal of Scheduling*, **8**(2), 179–190.

Aggoune, Riad. 2002. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. Ph.D. thesis, Université de Metz.

Agnetis, Alessandro, Flamini, Marta, Nicosia, Gaia, & Pacifici, Andrea. 2010. Scheduling three chains on two parallel machines. *European Journal of Operational Research*, **202**(3), 669–674.

Alvim, Adriana, & Ribeiro, Celso. 2004. A hybrid bin-packing heuristic to multiprocessor scheduling. *Experimental and Efficient Algorithms*, 1–13.

Artigues, Christian. 1997. *Ordonnancement en temps réel d'ateliers avec temps de préparation des ressources*. Ph.D. thesis, Université Paul Sabatier-Toulouse III.

Badaoui, Ryma zineb, Boudhar, Mourad, & Dahane, Mohammed. 2015. Algirithmes exactes et approchés pour le probléme d'ordonnancement préemptif sur machines paralléles avec délais de transport. *In : 6th Operational Research Practice in Africa conference*.

Badaoui, Ryma zineb, Boudhar, Mourad, & Dahane, Mohammed. 2020.

Preemptive scheduling with transportation delays between machines. *Journal of Modelling in Management*, **15**(3), 829–847.

Baker, Kenneth R. 1974. *Introduction to sequencing and scheduling*. John Wiley & Sons.

Baptiste, Philippe, Jouglet, Antoine, & Savourey, David. 2008. Lower bounds for parallel machine scheduling problems. *International Journal of Operational Research*, **3**(6), 643–664.

Behnamian, Javad, Zandieh, Mostafa, & Ghomi, SMT Fatemi. 2009. Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. *Expert Systems with Applications*, **36**(6), 9637–9644.

Ben Hmida, Abir. 2009. *Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible*. Ph.D. thesis, Université de Toulouse.

Błażewicz, Jacek. 1987. Selected topics in scheduling theory. *Pages 1–59 of : North-Holland Mathematics Studies*, vol. 132. Elsevier.

Blazewicz, Jacek, Ecker, Klaus H, Pesch, Erwin, Schmidt, Gunter, & Weglarz, Jan. 2013. *Scheduling computer and manufacturing processes*. springer science & Business media.

Botta-Genoulaz, Valèrie. 1996. *Planification et ordonnancement d'une succession d'ateliers avec contraintes : vers un atelier de génie décisionnel pour le pilotage des sytèmes de production*. Ph.D. thesis, Lyon 1.

Boudhar, Mourad, & Haned, Amina. 2009. Preemptive scheduling in the presence of transportation times. *Computers & Operations Research*, **36**(8), 2387–2393.

Brucker, Peter, & Schlie, Rainer. 1990. Job-shop scheduling with multipurpose machines. *Computing*, **45**(4), 369–375.

Carlier, Jacques, Chrétienne, Philippe, Hanen, C, Lopez, P, Munier, A, Pinson, E, Portmann, Mc, Prins, C, Proust, C, & Villon, P. 1988. Les problèmes d'ordonnancement. *Modèlisation, Complexité, Algorithmes, Editions Massons, Paris*.

Chang, Soo Y, & Hwang, Hark-Chin. 1999. The worst-case analysis of the MULTIFIT algorithm for scheduling nonsimultaneous parallel machines. *Discrete Applied Mathematics*, **92**(2), 135–147.

Cheng, TCE. 1985. A note on preemptive-due-date scheduling with minimum Makespan. *International journal on policy and information*, **9**(2), 137–140.

Cheng, TCE, & Sin, CCS. 1990. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, **47**(3), 271–292.

Cheng, TCE, & Sin, CCS. 1991. An algorithm for the N/M/parallel/Cmaxpreemptive due-date scheduling problem. *Engineering costs and production economics*, **21**(1), 43–49.

Chu, Chengbin, & Proth, Jean-Marie. 1996. *L'ordonnancement et ses applications*. Masson.

Conway, Richard Walter, Maxwell, William L, & Miller, Louis W. 2003. *Theory of scheduling*. Courier Corporation.

Conway, RW, Maxwell, WL, & Miller, LW. 1967. *Theory of scheduling Addison*.

Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L, & Stein, Clifford. 2001. *Introduction to algorithms : Second Edition*. McGraw-Hill Book Company.

Costa, AM, Vargas, PA, Von Zuben, FJ, & Franca, PM. 2002. Makespan minimization on parallel processors : an immune-based approach. *Pages 920–925 of : Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 1. IEEE.

Costa, Wagner Emanoel, Goldbarg, Marco César, & Goldbarg, Elizabeth G. 2012. New VNS heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, **39**(9), 8149–8161.

Dantzig, George, Fulkerson, Ray, & Johnson, Selmer. 1954. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, **2**(4), 393–410.

Dasgupta, Dipankar. 2012. *Artificial immune systems and their applications*. Springer Science & Business Media.

Dauzère-Pérès, Stéphane, Roux, W, & Lasserre, Jean B. 1998. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, **107**(2), 289–305.

Davis, Lawrence. 1987. Genetic algorithms and simulated annealing.

De Paula, Mateus Rocha, Ravetti, Martín Gómez, Mateus, Geraldo Robson, & Pardalos, Panos M. 2007. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, **18**(2), 101–115.

Dell'Amico, & Martello, Silvano. 2001. Bounds for the cardinality constrained P||Cmax problem. *Journal of scheduling*, **4**(3), 123–138.

Dell'Amico, Mauro, Iori, Manuel, & Martello, Silvano. 2004. Heuristic algorithms and scatter search for the cardinality constrained P|| C max problem. *Journal of Heuristics*, **10**(2), 169–204.

Dell'Amico, Mauro, Iori, Manuel, Martello, Silvano, & Monaci, Michele. 2008. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, **20**(3), 333–344.

Dorigo, Marco. 1992. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*.

Du, Jianzhong, Leung, Joseph Y-T, & Young, Gilbert H. 1990. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, **75**(3), 347–355.

Engels, Daniel W, Feldman, Jon, Karger, David R, & Ruhl, Matthias. 2001. Parallel processor scheduling with delay constraints. *Pages 577–585 of : Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics.

Erschler, Jacques. 1976. *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*. Ph.D. thesis, Université Paul Sabatier, Toulouse.

Esquirol, P, & Lopez, P. 1999. L'ordonnancement, Série : Production et techniques quantitatives appliquées à la gestion, collection Gestion. *Economica, Paris*.

Fishkin, Aleksei V, Jansen, Klaus, Sevastyanov, Sergey V, & Sitters, René. 2005. Preemptive scheduling of independent jobs on identical parallel machines subject to migration delays. *Pages 580–591 of : European Symposium on Algorithms*. Springer.

Florêncio, Luís, Pimentel, Carina, & Alvelos, Filipe. 2015. An Exact and a Hybrid Approach for a Machine Scheduling Problem with Job Splitting. *Pages 191–212 of : Operational Research*. Springer.

Florêncio, Luís Filipe Pacheco. 2013. *A searchcol algorithm for the unrelated parallel machine scheduling problem with job splitting*. Ph.D. thesis, Universidade do Minho.

França, Paulo M, Gendreau, Michel, Laporte, Gilbert, & Müller, Felipe M. 1994. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers & operations research*, **21**(2), 205–210.

French, Simon. 1982. Sequencing and scheduling. *An Introduction to the Mathematics of the Job-shop*.

Friesen, Donald K., & Langston, Michael A. 1986. Evaluation of a MULTIFIT-based scheduling algorithm. *Journal of Algorithms*, **7**(1), 35–59.

Gagné, Caroline, Price, WL, & Gravel, M. 2001. Optimisation par colonie de fourmis pour problème d'ordonnancement industriel avec temps de réglages dépendants de la séquence. *In : 3e Conférence Francophone de Modélisation et Simulation MOSIM01, Troyes (France)*.

Gagné, Caroline, Gravel, Marc, & Price, Wilson L. 2005. Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems. *Journal of the Operational Research Society*, **56**(6), 687–698.

Garey, Michael R, & Johnson, David S. 1975. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, **4**(4), 397–411.

Garey, Michael R, & Johnson, David S. 2002. *Computers and intractability*. Vol. 29. wh freeman New York.

Glover, Fred. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, **13**(5), 533–549.

Glover, Fred. 1990. Tabu search : A tutorial. *Interfaces*, **20**(4), 74–94.

Glover, Fred, & Laguna, Manuel. 1998. Tabu search. *Pages 2093–2229 of : Handbook of combinatorial optimization*. Springer.

Gonzalez, Teofilo F, & Johnson, Donald B. 1980. A new algorithm for preemptive scheduling of trees. *Journal of the ACM (JACM)*, **27**(2), 287–312.

Gordon, Valery, Proth, Jean-Marie, & Chu, Chengbin. 2002. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, **139**(1), 1–25.

Graham, Ronald L. 1966. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, **45**(9), 1563–1581.

Graham, Ronald L, Lawler, Eugene L, Lenstra, Jan Karel, & Kan, AHG. 1979. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete Mathematics*, **5**, 287–326.

Haned, A, & Boudhar, M. 2013a. A genetic algorithm for scheduling problem with preemption and transportation delays. *In : 8e conférence internationale Sur La conception et Production Intégré CPI'13, Tlemcen (Algérie)*.

Haned, A, & Boudhar, M. 2013b. Metaheuristics for the resolution of a scheduling problem with preemption and transportation delays. *In : International conference on Industrial Engineering and Manufacturing, ICIEM'14, Batna (Algeria)*.

Haned, A, Boudhar, M, & Soukhal, A. 2011. Resolution of the parallel machines scheduling problem with preemption and transportation delays.

In : 8e édition du Colloque sur l'Optimisation et les systèmes d'Information COSI'11, Guelma (Algérie).

Haned, Amina. 2012. *Ordonnancement sous contraintes additionnelles : préemption et transport*. Ph.D. thesis, Université des sciences et de la technologie Houari Boumedienne.

Haned, Amina, Soukhal, Ameur, Boudhar, Mourad, & Tuong, Nguyen Huynh. 2012. Scheduling on parallel machines with preemption and transportation delays. *Computers & Operations Research*, **39**(2), 374–381.

Hanen, Claire, & Munier, Alix. 2001. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, **108**(3), 239–257.

Herrmann, Jeffrey W. 2006. *Handbook of production scheduling*. Vol. 89. Springer Science & Business Media.

Ho, Johnny C, & Wong, Johnny S. 1995. Makespan minimization for m parallel identical processors. *Naval Research Logistics (NRL)*, **42**(6), 935–948.

Hochbaum, Dorit S, & Shmoys, David B. 1987. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, **34**(1), 144–162.

Hofmeyr, Steven A, & Forrest, Stephanie. 2000. Architecture for an artificial immune system. *Evolutionary computation*, **8**(4), 443–473.

Holland, John. 1975. Adaptation in natural and artificial systems : an introductory analysis with application to biology. *Control and artificial intelligence*.

Hoogeveen, JA, Lenstra, Jan Karel, & Veltman, Bart. 1994. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, **16**(3), 129–137.

Horn, WA. 1974. Some simple scheduling algorithms. *Naval Research Logistics (NRL)*, **21**(1), 177–185.

Houcine, Boumedienne Merouane. 2006. *Les problemes d'ordonnancement a machines paralleles de taches independantes*. Ph.D. thesis, Universite SAAD DAHLEB BLIDA.

Hsu, Chou-Jung, Ji, Min, Guo, Jia-Yuarn, & Yang, Dar-Li. 2013. Unrelated parallel-machine scheduling problems with aging effects and deteriorating maintenance activities. *Information Sciences*, **253**, 163–169.

Jackson, James R. 1955. Scheduling a production line to minimize maximum tardiness. *management science research project*.

Jain, Anant Singh, & Meeran, Sheik. 1999. Deterministic job-shop scheduling : Past, present and future. *European journal of operational research*, **113**(2), 390–434.

Janiak, Adam, Janiak, Władysław A, Krysiak, Tomasz, & Kwiatkowski, Tomasz. 2015. A survey on scheduling problems with due windows. *European Journal of Operational Research*, **242**(2), 347–357.

Jansen, Klaus, Mastrolilli, Monaldo, & Solis-Oba, Roberto. 2005. Approximation algorithms for flexible job shop problems. *International Journal of Foundations of Computer Science*, **16**(02), 361–379.

Ji, Min, & Cheng, TC Edwin. 2008. An FPTAS for parallel-machine scheduling under a grade of service provision to minimize makespan. *Information Processing Letters*, **108**(4), 171–174.

Johnson, David S, & Garey, Michael R. 1979. *Computers and intractability : A guide to the theory of NP-completeness*. WH Freeman.

Johnson, Selmer Martin. 1954. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, **1**(1), 61–68.

Karp, Richard M. 1972. Reducibility among combinatorial problems. *Pages 85–103 of : Complexity of computer computations*. Springer.

Kellerer, Hans, Pferschy, Ulrich, & Pisinger, David. 2004. Introduction to NP-Completeness of knapsack problems. *Pages 483–493 of : Knapsack problems*. Springer.

Kirkpatrick, Scott, Gelatt, C Daniel, & Vecchi, Mario P. 1983. Optimization by simulated annealing. *science*, **220**(4598), 671–680.

Klemmt, Andreas, Weigert, Gerald, Almeder, Christian, & Mönch, Lars. 2009. A comparison of MIP-based decomposition techniques and VNS approaches for batch scheduling problems. *Pages 1686–1694 of : Winter Simulation Conference*. Winter Simulation Conference.

Kovalyov, Mikhail Y. 1995. Improving the complexities of approximation algorithms for optimization problems. *Operations Research Letters*, **17**(2), 85–87.

Laha, Dipak, & Behera, Dhiren Kumar. 2017. A comprehensive review and evaluation of LPT, MULTIFIT, COMBINE and LISTFIT for scheduling identical parallel machines. *International Journal of Information and Communication Technology*, **11**(2), 151–165.

Lawler, EL. 1983. Recent results in the theory of machine scheduling. *Pages 202–234 of : Mathematical programming the state of the art*. Springer.

Lee, Chung-Yee, & Chen, Zhi-Long. 2001. Machine scheduling with transportation considerations. *Journal of scheduling*, **4**(1), 3–24.

Lee, Chung-Yee, & Massey, J David. 1988. Multiprocessor scheduling : combining LPT and MULTIFIT. *Discrete applied mathematics*, **20**(3), 233–242.

Lenstra, Jan Karel, & Shmoys, David B. 1995. Computing near-optimal schedules. *Pages 1–14 of : Scheduling theory and its applications*. Wiley.

Leung, Joseph YT. 2004. *Handbook of scheduling : algorithms, models, and performance analysis*. CRC Press.

Liaw, Ching-Fang. 2016. A branch-and-bound algorithm for identical parallel-machine total completion time scheduling problem with preemption and release times. *Journal of Industrial and Production Engineering*, **33**(6), 383–390.

Linn, Richard, & Zhang, Wei. 1999. Hybrid flow shop scheduling : a survey. *Computers & industrial engineering*, **37**(1-2), 57–61.

Liu, Changchun, Wang, Chenjie, Zhang, Zhi-hai, & Zheng, Li. 2018. Scheduling with job-splitting considering learning and the vital-few law. *Computers & Operations Research*, **90**, 264–274.

Liu, Lu, Wang, Jian-Jun, Liu, Feng, & Liu, Ming. 2017. Single machine due window assignment and resource allocation scheduling problems with learning and general positional effects. *Journal of Manufacturing Systems*, **43**, 1–14.

Lopez, P, & Roubelat, F. 2000. Ordonnancement de la production. Lavoisier.

Maccarthy, Bart L, & Liu, Jiyin. 1993. Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling. *The International Journal of Production Research*, **31**(1), 59–79.

Maggu, PL, & Das, G. 1980. On 2$\times$ n sequencing problem with transportation times of jobs. *Pure and Applied Mathematika Sciences*, **12**(1), 6.

McNaughton, Robert. 1959. Scheduling with deadlines and loss functions. *Management Science*, **6**(1), 1–12.

Metropolis, Nicholas, Rosenbluth, Arianna W, Rosenbluth, Marshall N, Teller, Augusta H, & Teller, Edward. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, **21**(6), 1087–1092.

MICHAEL, L PINEDO. 2018. *Scheduling : theory, algorithms, and systems*. Springer.

Min, Liu, & Cheng, Wu. 1999. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, **13**(4), 399–403.

Mjirda, Anis. 2014. *Recherche à voisinage variable pour des problèmes de routage avec ou sans gestion de stock*. Ph.D. thesis, Université de Valenciennes et du Hainaut-Cambresis.

Mladenović, Nenad, & Hansen, Pierre. 1997. Variable neighborhood search. *Computers & operations research*, **24**(11), 1097–1100.

Modeste, Simon. 2012. *Enseigner l'algorithme pour quoi ?* Ph.D. thesis, UniversitÃ© de Grenoble.

Mori, Kazuyuki, Tsukiyama, Makoto, & Fukuda, Toyoo. 1997. Artificial immunity based management system for a semiconductor production line. *Pages 851–855 of : 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 1. IEEE.

Muntz, Richard R, & Coffman, EG. 1969. Optimal preemptive scheduling on two-processor systems. *IEEE Transactions on Computers*, **100**(11), 1014–1020.

Pinedo, Michael. 2000. *Scheduling : theory, algorithms, and systems. 1995.*

Pinedo, Michael. 2012. *Scheduling*. Vol. 29. Springer.

Prot, Damien, Bellenguez-Morineau, Odile, & Lahlou, Chams. 2013. New complexity results for parallel identical machine scheduling problems with preemption, release dates and regular criteria. *European Journal of Operational Research*, **231**(2), 282–287.

Rayward-Smith, Victor J. 1987. The complexity of preemptive scheduling given interprocessor communication delays. *Information processing letters*, **25**(2), 123–125.

Rebaine, DjamelS. 2000. *Une introduction á l'analyse des algorithmes*. ENAG Editions.

Rinnoy Kan, Alexander HG. 1976. Machine scheduling problems : classification, complexity, and computations. *PhD thesis, University of Amsterdam*.

Rothkopf, Michael H. 1966. Scheduling independent tasks on parallel processors. *Management Science*, **12**(5), 437–447.

Roy, Bernard, & Sussmann, B. 1964. Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds*, **9**.

Sahni, Sartaj. 1979. Preemptive scheduling with due dates. *Operations Research*, **27**(5), 925–934.

Sahni, Sartaj K. 1976. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, **23**(1), 116–127.

Sakarovitch, Michel. 1984. *Optimisation combinatoire : Programmation discrète*. Vol. 2. Editions Hermann.

Savourey, David. 2006. *Ordonnancement sur machines parallèles : minimiser la somme des coûts.* Ph.D. thesis, Université de Technologie de Compiègne.

Schuurman, Petra, & Woeginger, Gerhard J. 1999. Polynomial time approximation algorithms for machine scheduling : Ten open problems. *Journal of Scheduling*, **2**(5), 203–213.

Schuurman, Petra, & Woeginger, Gerhard J. 2007. *Approximation Schemes – A Tutorial*.

Serafini, Paolo. 1996. Scheduling jobs on several machines with the job splitting property. *Operations Research*, **44**(4), 617–628.

Sevkli, Mehmet, & Aydin, M Emin. 2006. A variable neighbourhood search algorithm for job shop scheduling problems. *Pages 261–271 of : European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer.

Shams, Hesam, & Salmasi, Nasser. 2014. Parallel machine scheduling problem with preemptive jobs and transportation delay. *Computers & Operations Research*, **50**, 14–23.

Skiena, Steven S. 2008. *The algorithm design manual : Second Edition*. Vol. 2. Springer Science & Business Media.

Soukhal, Ameur, & Martineau, Patrick. 2005. Resolution of a scheduling problem in a flowshop robotic cell. *European Journal of Operational Research*, **161**(1), 62–72.

Stern, Helman I, & Vitner, Gad. 1990. Scheduling parts in a combined production-transportation work cell. *Journal of the Operational Research Society*, **41**(7), 625–632.

Talbi, El-Ghazali. 2009. *Metaheuristics : from design to implementation*. Vol. 74. John Wiley & Sons.

Tanaev, Vjačeslav S, Gordon, VS, & Šafranskij, JM. 1994. *Scheduling theory : single-stage theory*. Kluwer.

Tanaev, VS. 1973. Interruptions in Deterministic Service Systems with Identical Parallel Machines. *Izv. Akad. Nauk BSSR, Ser. Fiz.-mat. Nauk*, 44–48.

Toumi, Fatma. 2007. *Ordonnancement Dynamique dans les Industries Agroalimentaires*. Ph.D. thesis, UNIVERSITE DE TUNIS EL MANAR.

Tsoukiàs, Alexis. 2006. De la thèorie de la décision à l'aide à la décision. *D. Bouyssou, D. Dubois, M. Pirlot, H. Prade, Concepts et méthodes pourl'aide à la décision*, **1**.

Ullman, Jeffrey D. 1975. NP-complete scheduling problems. *Journal of Computer and System sciences*, **10**(3), 384–393.

Wang, Chenjie, Liu, Changchun, Zhang, Zhi-hai, & Zheng, Li. 2016. Minimizing the total completion time for parallel machine scheduling with job splitting and learning. *Computers & Industrial Engineering*, **97**, 170–182.

Wang, Wan-Liang, Wang, Hai-Yan, Zhao, Yan-Wei, Zhang, Li-Ping, & Xu, Xin-Li. 2013. Parallel machine scheduling with splitting jobs by a hybrid differential evolution algorithm. *Computers & Operations Research*, **40**(5), 1196–1206.

Wiens, Tobias, & Ullrich, Christian Alexander. 2016. Scheduling with Team Production Effects.

Woeginger, Gerhard J. 2009. A comment on parallel-machine scheduling under a grade of service provision to minimize makespan. *Information Processing Letters*, **109**(7), 341–342.

Wu, Yu-Bin, Wan, Long, & Wang, Xiao-Yuan. 2015. Study on due-window assignment scheduling based on common flow allowance. *International Journal of Production Economics*, **165**, 155–157.

Xing, Wenxun, & Zhang, Jiawei. 2000. Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, **103**(1), 259–269.

Xing, WENXUN, & Zhang, JIWEI. 1998. Splitting parallel machine scheduling. *OR TRANSACTIONS*, **3**, 005.

Yagouni, Mohammed. 2017. *Contributionà la Résolution des Problèmes Complexes de l'Optimisation Combinatoire*. Ph.D. thesis, Université de la Science et de la Technologie Houari Boumedienne.

Yue, Minyi. 1990. On the exact upper bound for the multifit processor scheduling algorithm. *Annals of Operations Research*, **24**(1), 233–259.

Yue, Qing, & Wan, Guohua. 2016. Single machine SLK/DIF due window assignment problem with job-dependent linear deterioration effects. *Journal of the Operational Research Society*, **67**(6), 872–883.