République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université YAHIA Fares de Médéa

Faculté des Sciences

Département de Mathématiques

MÉMOIRE DE MASTER

En : Mathématiques

Option : Recherche Opérationnelle

Thème

Contribution à l'étude de certains problèmes d'ordonnancement préemptif

Présenté par

HAMADI Samah

Soutenu pupliquement devant le Jury composé de :

Mr N. IKHLEF ESCHOUF	Professeur	U.Y.F.M	Président
Mr A. BENKACI	MAA	U.Y.F.M	Examinateur
Mme K. SERIER ABDALLAH	MAA	U.Y.F.M	Examinatrice
Mme S. KOUIDER AIAD	MAA	U.Y.F.M	Promotrice

Année universitaire : 2023/2024

Résumé

La théorie d'ordonnancement est une branche de la recherche opérationnelle, qui consiste à ordonner un ensemble d'opérations tout en satisfaisant un ensemble de contraintes et en optimisant un ou plusieurs objectifs. Dans l'ordonnancement préemptif, l'exécution d'une tâche peut être interrompue par l'exécution d'une autre tâche. Elle sera alors terminée plus tard, éventuellement après d'autres interruptions. Dans ce travail, nous nous intéressons à étudier quelques problèmes d'ordonnancement avec préemption, à savoir trois problèmes faciles et un problème $\mathcal{NP}$-difficile [26]. Nous avons spécifiquement examiné le problème $Pm/pmtn, c_{j,j'}/C_{\max}$, pour lequel nous avons élaboré des heuristiques et mené une expérimentation numérique.

Mots–clés :Ordonnancement, préemptif, NP-difficile, heuristique.

Abstract

Scheduling theory is a branch of operations research that deals with the ordering of a set of operations while satisfying a set of constraints and optimizing one or more objectives. In preemptive scheduling, the execution of a task can be interrupted by the execution of another task. It will then be completed later, possibly after further interruptions. In this work, we are interested in studying a few scheduling problems with preemption, namely three easy problems and one $\mathcal{NP}$-hard problem [26]. In particular, we consider the problem $Pm/pmtn, c_{j,j'}/C_{\max}$, for which we have developed heuristics and conducted numerical experiments.

Keywords : Scheduling, preemptive, NP-hard, heuristics.

Remerciements

Je remercie ALLAH le tout puissant de m'avoir donné le courage, la volonté et la patience de mener à terme le présent travail.

Mes premiers remerciements vont naturellement à Madame KOUIDER AIAD Soumia, Maître assistante à l'Université de Médéa et promotrice de ce mémoire. Je tiens à exprimer ma profonde gratitude pour ses conseils pertinents, sa disponibilité sans faille, et surtout pour son encouragement constant tout au long de cette période de travail. Sa patience infinie et sa compréhension ont été des éléments essentiels qui m'ont permis de surmonter les problèmes rencontrés et de m'épanouir dans ce projet.

Je tiens également à exprimer ma reconnaissance envers Monsieur IKHLEF ESCHOUF Noureddine, Professeur à l'Université de Médéa, pour avoir accepté la lourde responsabilité de présider ce jury, ainsi qu'à Monsieur BENKACI Azzedine, Maître assistant à l'Université de Médéa, pour ses contributions inestimables en tant qu'examinateur, qui ont enrichi ce travail et méritent une mention spéciale. J'adresse aussi ma gratitude à Madame SERIER ABDALLAH Khadidja, Maître assistante à l'Université de Médéa, pour ses conseils et son aide précieuse en tant qu'examinatrice. Leur présence et leur engagement ont été d'une importance capitale dans cette étape cruciale.

Je tiens enfin à remercier tous les enseignants du département de mathématiques, ainsi que tous les employés de l'administration, et tous mes proches et amis pour leur soutien et leur encouragement tout au long de la réalisation de ce mémoire.

Dédicaces

À tous ceux que j'aime et qui m'aiment,

À l'âme de ma chère grand-mère,

Je dédie ce modeste travail.

Table des matières

Page

Résumé i

Abstract ii

Remerciements iii

Dédicaces v

List of Figures ix

List of Tables x

Notations et abréviations xii

Introduction générale 1

Chapitre

1 Généralités sur l'ordonnancement 3

1.1 Introduction 3

1.2 Présentation d'un problème d'ordonnancement . . . 3

1.2.1 Définition de l'ordonnancement 3

1.2.2 Définition d'un problème d'ordonnancement 4

1.3 Formulation d'un problème d'ordonnancement . . . 5

1.3.1 Tâches 5

1.3.2 Ressources 7

1.3.3 Contraintes 8

1.3.4 Objectifs de l'ordonnancement 8

1.4 Ordonnancement dans différents types d'ateliers . 9

1.4.1 Problème à machines parallèles (parallel machines problem) 10

1.4.2 Problème à machines spécialisées (dedicated machines problem) . . . 10
1.5 Préemption . . . 11
1.6 Classification d'un problème d'ordonnancement . . . 12
1.6.1 Champ α : Ressources . . . 12
1.6.2 Champ β : Contraintes . . . 13
1.6.3 Champ γ : Critère d'optimisation . . . 14
1.7 Représentation des problèmes d'ordonnancement par un diagramme de Gantt . . . 14
1.8 Théorie de complexité . . . 15
1.8.1 Principes fondamentaux de complexité . . . 15
1.8.2 Classes de complexité . . . 17
1.8.3 Réduction entre problème d'ordonnancement . . . 18
1.9 Conclusion . . . 19
2 Étude de certains problèmes d'ordonnancement préemptif 20
2.1 Introduction . . . 20
2.2 Problème $1/pmtn, r_i/\overline{C}$ . . . 20
2.2.1 Description du problème . . . 20
2.2.2 Complexité du problème . . . 21
2.2.3 Résolution du problème . . . 21
2.3 Problème $1/prec, pmtn, r_i/L_{\max}$ . . . 23
2.3.1 Description du problème . . . 23
2.3.2 Complexité du problème . . . 23
2.3.3 Résolution du problème . . . 23
2.4 Problème $Pm/pmtn/C_{\max}$ . . . 26
2.4.1 Description du problème . . . 26
2.4.2 Complexité du problème . . . 26
2.4.3 Résolution du problème . . . 26
2.5 Problème $Pm/pmtn, c_{j,j'}/C_{\max}$ . . . 28
2.5.1 Description du problème . . . 28
2.5.2 Complexité du problème . . . 29
2.6 Conclusion . . . 30

3 Heuristiques proposées pour la résolution du problème $Pm/pmtn, c_{j,j'}/C_{\max}$ 31
3.1 Introduction 31
3.2 Représentation du problème 31
3.2.1 Position du problème 31
3.2.2 Détermination d'une borne inférieure 32
3.3 Heuristiques proposées pour résoudre $Pm/pmtn, c_{j,j'}/C_{\max}$ 32
3.3.1 Heuristiques proposées 32
3.4 Conclusion 45
4 Expérimentations numériques 46
4.1 Introduction 46
4.1 Langage Python 46
4.2 Implémentation des heuristiques proposées sur Python 47
4.3 Exécution des algorithmes 47
4.3.1 Algorithme de la règle LPT 47
4.3.2 Algorithme de l'heuristique H1 48
4.3.3 Algorithme de la procédure RANGEMENT 48
4.4 Déroulement des tests 49
4.5 Comparaison des performances des heuristiques . . . 50
4.6 Conclusion 51
Conclusion générale 52

Liste des figures

1.1 Représentation d'une tâche 7

1.2 Représentation de quelques réductions élémentaires entre les critères de performance d'ordonnancement 19

2.1 Solution optimale de l'exemple 2.1 22

2.2 Contraintes de précédence du problème $1/prec, pmtn, r_i/L_{\max}$ 24

2.3 Solution optimale de l'exemple 2.2 25

2.4 Solution générée par l'algorithme de Mc Naughton 28

3.1 Solution générale de l'exemple par l'heuristique H1 39

3.2 Solution générale de l'exemple par l'heuristique H2 42

3.3 Solution générale de l'exemple par l'heuristique H3 45

4.1 Exemple d'exécution de l'algorithme de la règle LPT 47

4.2 Exemple d'exécution de l'algorithme H1 48

4.3 Exemple d'exécution de l'algorithme de la procédure RANGEMENT 48

Liste des tables

2.1 Données de l'exemple 2.1 21

2.2 Résolution de l'exemple 2.1 22

2.3 Données de l'exemple 2.2 24

2.4 Modification des dates échues d_i de l'exemple 2.2 24

2.5 Données de l'exemple 2.3 27

3.1 Temps de changement de machines $c_{j,j'}$ 33

3.2 Temps de traitement de l'exemple d'application 37

3.3 Temps de transport de l'exemple d'application 37

4.1 Résultats des tests 49

4.2 Calcul du ratio d'approximation R 50

Liste des algorithmes

2.1 Algorithme de Mc Naughton 27
3.1 Algorithme de H1 34
3.2 Procédure RANGEMENT 35
3.3 Algorithme de H2 36
3.4 Algorithme de H3 36

Notations et abréviations

n	Nombre de tâches (jobs,traveaux).
m	Nombre de machines.
i, k	Indice des tâches.
j	Indice des machines.
$T = \{T_1, T_2, \ldots, T_n\}$	Ensemble des tâches.
$M = \{M_1, M_2, \ldots, M_m\}$	Ensemble des machines.
$\mathcal{P}$	Polynomial.
$\mathcal{NP}$	Non polynomial.
$pmtn$	Préemption.
$prec$	Précédence quelconque.
r_i	Date de disponibilité de la tâche T_i .
p_i	Temps de traitement de la tâche T_i .
d_i	Date échue de la tâche T_i.
$\widetilde{d}_i$	Date limite de la tâche T_i.
$C_{\max}$(makespan)	Date de fin de traitement de toutes les tâches.
$L_{\max}$	Grande tardivité.
$T_{\max}$	Grand retard.
$\overline{C}$	Temps de fin de traitement moyen.
F	Ensemble des tâches disponibles.

st	Séquence des tâches.
sm	Séquence des machines.
SRPT	Shortest Remaining Processing Time.
EDD	Earliet Due Date.
LPT	Longest Processing Time.
$c_{j,j'}$	Temps de changement de machines (temps de trasport)

Introduction générale

La recherche opérationnelle constitue une discipline fondamentale visant à fournir aux gestionnaires des outils scientifiques afin de prendre des décisions éclairées. Initialement développée pendant la Seconde Guerre mondiale pour résoudre des problèmes militaires, son champ d'application s'est depuis étendu pour englober divers domaines tels que l'économie et l'industrie [2].

La théorie de l'ordonnancement, intégrée à la recherche opérationnelle, s'attache à élaborer des modèles mathématiques et des méthodes de résolution efficaces pour les problèmes rencontrés. Fondamentalement, un problème d'ordonnancement implique l'allocation temporelle de ressources pour mener à bien un ensemble d'activités. Il répond principalement à deux interrogations essentielles :

- Sur quelle ressource doit-on exécuter une tâche ?
- Quand faudra-t-il l'exécuter ?

En raison de leur pertinence industrielle, les problèmes d'ordonnancement suscitent un intérêt au sein de la communauté scientifique, engendrant ainsi une abondante production de travaux de recherche et demeurant un domaine d'études inlassablement exploré.

À travers ce mémoire, nous allons traiter quelques problèmes d'ordonnancement préemptif, une méthode de gestion des tâches où l'exécution d'une tâche peut être interrompue pour permettre l'exécution d'une autre tâche, la tâche interrompue reprenant ultérieurement après une ou plusieurs interruptions possibles.

Ce mémoire est organisé comme suit :

Chapitre 1 intitulé "*Généralités sur l'ordonnancement*", présentons de généralités sur la théorie d'ordonnancement. Nous y retrouvons 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.

Chapitre 2 intitulé "*Étude de certains problèmes d'ordonnancement préemptif*", présente les différentes études ayant traité des probèmes d'ordonnancement avec préemption à savoir trois problèmes faciles et un problème $\mathcal{NP}$-difficile [26], la modélisation et l'étude de la complexité de ces problèmes, des exemples illustratifs et explicatifs de la problématique sont également présentés.

Chapitre 3 intitulé "*Heuristiques proposées pour résoudre le problème*

$Pm/pmtn, c_{j,j'}/C_{\max}$", Une borne inférieure ainsi que des heuristiques de résolution avec des exemples illustratifs sont proposés.

Chapitre 4 intitulé "*Expérimentations numériques*", comprend la programmation des heuristiques développées, des tests sur leur performance, ainsi qu'une comparaison des résultats obtenus à partir des expérimentations numériques.

Une conclusion générale s'achève ce travail, donnant une récapitulation des résultats obtenus, ainsi que des perspectives pour de nouvelles recherches.

Chapitre 1

Généralités sur l'ordonnancement

1.1 Introduction

Dans ce chapitre, nous définissons les principaux outils mathématiques utilisés tout au long de ce mémoire. Une présentation d'un problème d'ordonnancement est donnée. Nous présentons la formulation et la classification de ce problème. L'ordonnancement dans différents types d'ateliers est mentionné, ainsi que la définition de la préemption et de la complexité.

1.2 Présentation d'un problème d'ordonnancement

1.2.1 Définition de l'ordonnancement

Définition 1.1 Ordonnancer c'est programmer dans le temps, l'exécution d'une réalisation décomposable en tâches, en attribuant des ressources à ces tâches et en fixant en particulier leurs dates de début d'exécution tout en respectant des contraintes portant sur l'utilisation et la disponibilité des ressources requises par les tâches, afin de satisfaire un ou plusieurs critères d'optimisation [14, 21].

- Un ordonnancement est une solution réalisable au problème. C'est donc un calendrier possible.

- Un ordonnancement optimal est une solution optimale. C'est un calendrier d'exécution qui conduit à la durée totale la plus courte [15].

Un calendrier se compose de trois processus principaux suivants :

- L'affectation : il s'agit d'attribuer une machine à chaque tâche à exécuter.
- Le séquençage : il s'agit d'attribuer un ordre d'exécution des tâches sur chaque machine.
- La programmation : il s'agit d'attribuer une date de début et une date de fin à chaque tâche [3].

1.2.2 Définition d'un problème d'ordonnancement

Les problèmes d'ordonnancement font partie de notre vie quotidienne.

Qui ne s'est jamais trouvé avec une liste de "choses à faire" sans savoir par où commencer ?

Ces problèmes concernent bien évidemment aussi de multiples secteurs économiques : l'informatique (tâches=jobs, produits, pièces... etc. Et ressources=processeurs, mémoire, machines... etc.), la construction (suivi de projet), l'industrie (problèmes d'ateliers, gestion de production), l'administration (emplois du temps) [27].

Il existe de nombreuses définitions pour le problème d'ordonnancement, mais toutes mettent en avant l'idée d'attribuer des ressources aux tâches pour optimiser leur réalisation. L'une de ces définitions pourrait être la suivante :

Définition 1.2 Face à un ensemble de tâches à réaliser, l'ordonnancement vise à définir les opérations à exécuter, en leur assignant des dates et des ressources. L'objectif est d'achever les tâches aussi efficacement que possible, en respectant les délais, en minimisant les coûts et en garantissant les meilleures conditions d'exécution [13].

Définition 1.3 Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises [31].

Les problèmes d'ordonnancement sont généralement très complexes en raison de leur nature combinatoire. Cependant, ces problèmes sont largement étudiés dans la littérature en tant que problèmes de référence pour les méthodes d'optimisation, en plus de leurs propres applications pratiques. Les approches traditionnelles pour résoudre les problèmes d'ordonnancement utilisent la simulation, des modèles analytiques, des heuristiques ou une combinaison de ces méthodes [37].

En termes généraux, un problème d'ordonnancement se pose lorsqu'il y a :

1. Une série de tâches à exécuter.
2. Un ensemble de ressources à utiliser par ces tâches.
3. Un programme à déterminer, pour allouer convenablement les ressources aux tâches [13].

1.3 Formulation d'un problème d'ordonnancement

Les différentes composantes d'un problème d'ordonnancement sont les tâches, les contraintes, les ressources et les fonctions objectives. Ces éléments clés d'un problème de programmation sont présentés ci-après.

1.3.1 Tâches

Une tâche est, par définition, une entité élémentaire localisée dans le temps par une date de début et/ou de fin, et par une durée d'exécution.

Par ailleurs, les tâches nécessitent l'utilisation de ressources. Les tâches sont, le plus souvent, liées entre elles par des conditions d'origines diverses ; on dit alors que les tâches sont dépendantes. Dans le cas contraire, elles sont dites indépendantes. Dans certains problèmes, les tâches peuvent être interruptibles ce qui signifie, en pratique, qu'elles peuvent être exécutées par morceaux par une ou plusieurs ressources ; dans d'autres cas contraires, on ne peut interrompre une tâche une fois celle-ci commencée. On parle respectivement de problèmes préemptif et non préemptif [21, 35].

Une tâche T_i est définie par ses caractéristiques temporelles :

p_i : temps de traitement de la tâche T_i.

S_i : temps de préparation de la tâche T_i.

r_i : date de disponibilité de la tâche T_i.

d_i : date échue de la tâche T_i.

C_i : date de fin de traitement de la tâche T_i.

w_i : poids associé à la tâche T_i.

$l_i = C_i - d_i$: tardiveté de la tâche T_i.

$t_i = max\{l_i, 0\}$: retard de la tâche T_i.

U_i : indicateur de retard de la tâche $T_i = \begin{cases} 1 \text{ si } C_i > d_i \\ 0 \text{ sinon} \end{cases}$.

$f_i = C_i - r_i$: durée de flow.

$\widetilde{d_i}$: date limite (la tâche doit être obligatoirement achevé avant la date $\widetilde{d_i}$).

L'ensemble des n tâches est noté $T = \{T_1, T_2, \ldots, T_n\}$.

L'ensemble des m machines est noté $M = \{M_1, M_2, \ldots, M_m\}$[36].

De manière plus schématique, la Figure 1.1 illustre

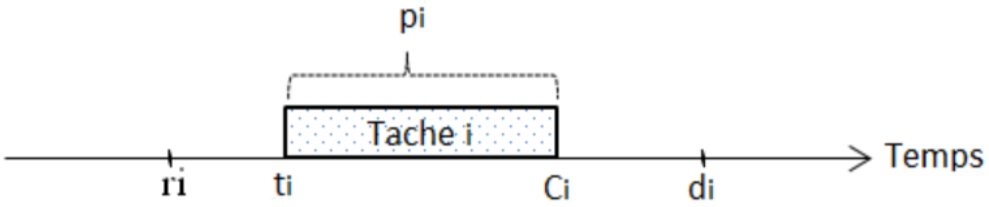


Figure 1.1 – Représentation d'une tâche

1.3.2 Ressources

Une ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'au moins une tâche et disponible en quantité limitée, appelée capacité de la ressource. On distingue différents types de ressources. Elles peuvent être classées selon leurs disponibilités au cours du temps et on parle dans ce cas de ressources renouvelables et consommables, ou selon leurs capacités et on parle dans ce cas de ressources disjonctives (non partageables) et cumulatives (partageables).

- **Ressource renouvelable :** il s'agit d'une ressource pouvant redevenir disponible en même quantité après avoir été allouée à une tâche (les machines, les hommes, l'équipement,... etc).
- **Ressource consommable :** il s'agit d'une ressource dont la disponibilité décroît après avoir été allouée à une tâche, par exemple la matière première.
- **Ressource disjonctive :** il s'agit d'une ressource qui ne peut exécuter qu'une seule tâche à la fois.
- **Ressource cumulative :** il s'agit d'une ressource qui peut être utilisée simultanément par plusieurs tâches [7, 31].

1.3.3 Contraintes

Suivant la disponibilité des ressources et suivant l'évolution temporelle, deux types de contraintes peuvent être distingués :

1. Contraintes de ressources : Plusieurs types de contraintes peuvent être induites par la nature des ressources. À titre d'exemple, la capacité limitée d'une ressource implique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cette ressource. Les contraintes relatives aux ressources peuvent être disjonctives, induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource, ou cumulatives impliquant la limitation du nombre de tâches à réaliser en parallèle.
2. Contraintes temporelles : elles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement. Ces contraintes peuvent être :
   (a) Contraintes de dates butoirs, certaines tâches doivent être achevées avant une date préalablement fixée.
   (b) Contraintes de précédence, une tâche T_i doit précéder la tâche T_k, des contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches [16].

1.3.4 Objectifs de l'ordonnancement

Généralement, l'objectif d'un problème d'ordonnancement consiste à optimiser un ou plusieurs critères ou à trouver une solution admissible qui respecte les contraintes du problème.

Les critères liés aux paramètres temporels caractérisant les tâches, les plus utilisés sont :

- La longueur d'ordonnancement : $C_{\max} = \max_{1 \leqslant i \leqslant n} \{C_i\}$.
- Le temps de fin de traitement moyen : $\overline{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$.
- Le temps de fin de traitement moyen pondéré : $\overline{C_w} = \frac{\sum_{i=1}^{n} w_i C_i}{\sum_{i=1}^{n} w_i}$.
- La grande tardivité : $L_{\max} = \max_{1 \leqslant i \leqslant n} \{l_i\}$.
- La tardivité moyenne : $\overline{L} = \frac{1}{n} \sum_{i=1}^{n} l_i$.
- La tardivité moyenne pondérée : $\overline{L_w} = \frac{\sum_{i=1}^{n} w_i l_i}{\sum_{i=1}^{n} w_i}$.
- Le grand temps de flow : $F_{\max} = \max_{1 \leqslant i \leqslant n} \{f_i\}$.
- Le temps de flow moyen : $\overline{F} = \frac{1}{n} \sum_{i=1}^{n} f_i$.
- Le temps de flow moyen pondéré : $\overline{F_w} = \frac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$.
- Le grand retard : $T_{\max} = \max_{1 \leqslant i \leqslant n} \{t_i\}$.
- Le retard moyen : $\overline{T} = \frac{1}{n} \sum_{i=1}^{n} t_i$.
- Le retard moyen pondéré : $\overline{T_w} = \frac{\sum_{i=1}^{n} w_i t_i}{\sum_{i=1}^{n} w_i}$.
- Le nombre de tâches en retard : $N_T = \sum_{i=1}^{n} u_i$ [25].

1.4 Ordonnancement dans différents types d'ateliers

On peut classifier les problèmes d'ordonnancement en atelier en fonction du nombre et du type de machines, ainsi que de la séquence des opérations (gamme de fabrication). Deux catégories principales de problèmes d'ordonnancement émergent. La première concerne les situations où chaque tâche nécessite une seule opération. La seconde concerne les situations où chaque tâche nécessite plusieurs opérations.

1.4.1 Problème à machines parallèles (parallel machines problem)

Les problèmes d'atelier à machines parallèles sont une généralisation des problèmes à une machine. Ce type d'atelier se caractérise par le fait que chaque opération peut être réalisée par n'importe quelle machine, disposée en parallèle, mais n'en nécessite qu'une seul. Le problème d'ordonnancement consiste donc à déterminer l'affectation des opérations aux machines puis le séquencement de ces opérations sur chaque machine.

Dans le dernier cas, il est possible de distinguer trois classes de machines :

- Machines parallèles identiques (identical parallel machines) : la durée d'exécution des opérations est la même sur toutes les machines.
- Machines parallèles uniformes (uniform parallel machines) : la durée d'exécution des opérations varie uniformément en fonction de la performance de la machine choisie.
- Machines parallèles indépendantes ou générales (unrelated parallel machines) : les durées opératoires dépendent complètement des machines utilisées [30].

1.4.2 Problème à machines spécialisées (dedicated machines problem)

Les problèmes de la deuxième catégorie sont dits problèmes d'atelier du fait de la nécessité du passage de chaque tâche sur deux ou plusieurs machines dédiées. Suivant le mode de passage des opérations sur les différentes machines, trois types d'ateliers sont distingués à savoir :

- Problèmes Flow-Shop : les ateliers de type Flow-Shop appelés également ateliers à cheminement unique, il s'agit d'un ensemble de m machines disposées

en séries. Toutes les opérations de toutes les tâches passent par les machines dans le même ordre (flot unidirectionnel).

- Problèmes Job-Shop : dans cette classe d'ateliers, appelés aussi ateliers à cheminements multiples, chaque opération passe sur les machines dans un ordre fixé, mais à la différence du Flow-Shop, cet ordre peut être différent pour chaque tâche (flot multidirectionnel).
- Problèmes Open-Shop : dans cette classe d'ateliers, appelés aussi ateliers à cheminement libre, les gammes opératoires des différentes tâches ne sont pas fixées a priori contrairement au problème d'atelier Job-Shop. Les opérations d'une même tâche peuvent donc être exécutées dans un ordre quelconque. Le problème consiste d'une part à déterminer le cheminement de chaque tâche et d'autre part à ordonnancer les tâches en tenant compte des gammes trouvées [30].

1.5 Préemption

Dans certains cas, une tâche peut être exécutée par morceaux (par exemple, dans un système multiprogrammé où l'on peut morceler l'exécution d'une tâche) ; dans ce cas, on dit que la tâche est interruptible ou encore que la «préemption» est possible : une tâche T_i est «préemptée» par une tâche T_k à l'époque t si T_i était en cours d'exécution avant t et employait une ressource qu'on alloue à la tâche T_k à t, provoquant ainsi l'interruption de T_i.

Souvent la possibilité de préempter rend les problèmes plus faciles, ainsi, pourra-t-on autoriser la préemption pour calculer des bornes de la fonction objectif [14].

1.6 Classification d'un problème d'ordonnancement

Pour des raisons de clarté et de compréhension, il est important d'adopter un descripteur qui soit basé sur l'existant et homogénéisant les notations actuelles.

La notation $(\alpha/\beta/\gamma)$ introduite par *Graham* [24] est couramment utilisée pour distinguer les différents problèmes d'ordonnancement entre eux et les classifier.

1.6.1 Champ α : Ressources

Renseigne les informations relatives aux ressources, à savoir le nombre de machines, type de machines et type d'ateliers. Il est composé de deux sous champs et désigné par $\alpha_1\alpha_2$:

- $\alpha_1 \in \{\phi, P, Q, R, F, O, J\}$: Type de machines.

ϕ : Une seule machine.

P : Machine parallèle identique.

Q : Machine parallèle uniforme.

R : Machine parallèle générale.

F : Machine spécialisée (système flow-shop).

O : Machine spécialisée (système open-shop).

J : Machine spécialisée (système job-shop).

- $\alpha_2 \in \{\phi, m\}$: Nombre de machines.

$\alpha_2 = \phi$: Le nombre de machines est variable.

$\alpha_2 = m$: Le nombre de machines est égal à m [9].

1.6.2 Champ β : Contraintes

Permet d'obtenir des informations sur les contraintes prises en compte aussi bien sur les tâches qui doivent être ordonnancées. Il est composé de 6 sous champs $\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6$:

- $\beta_1 \in \{\phi, pmtn\}$: Possibilité de préemption.

$\beta_1 = \phi$: La préemption n'est pas autorisée.

$\beta_1 = pmtn$: La préemption est autorisée.

- $\beta_2 \in \{\phi, prec, tree, chain, etc...\}$: Type de contraintes de précédence.

$\beta_2 = \phi$: Pas de contraintes de précédence.

$\beta_2 = prec$: Contraintes de précédence quelconques.

$\beta_2 = tree$: Les contraintes de précédence sont données sous forme d'un arborescence.

$\beta_2 = chain$: Les contraintes de précédence sont données sous forme de chaines.

- $\beta_3 \in \{\phi, r_i\}$: Les dates de disponibilité.

$\beta_3 = \phi$: Toutes les dates de disponibilité sont nulles.

$\beta_3 = r_i$: Il existe des dates de disponibilité pour les tâches non identiques.

- $\beta_4 \in \left\{\phi, p, \underline{p} \leq p_i \leq \overline{p}\right\}$.

$\beta_4 = \phi$: Les temps de traitement des tâches sont quelconques et non identiques.

$\beta_4 = p$: Les temps de traitement des tâches sont égaux à p (constante).

$\beta_4 = \underline{p} \leq p_i \leq \overline{p}$: Le temps de traitement de chaque tâche est compris entre $\underline{p}$ et $\overline{p}$.

- $\beta_5 \in \left\{\phi, d_i, \widetilde{d_i}\right\}$.

$\beta_5 = \phi$: Il n'y a pas des dates échues pour les tâches, ou il existe des dates échues pour les tâches dans le cas où le critère d'optimisation est en fonction de ces derniers.

$\beta_5 = d_i$: Il existe des dates échues pour les tâches.

$\beta_5 = \widetilde{d_i}$: Il existe une date limite pour chaque tâche.

- $\beta_6 \in \{\phi, S_i\}$.

$\beta_6 = \phi$: Il n'existe pas de temps de préparation .

$\beta_6 = S_i$: Il existe un temps de préparation [32].

1.6.3 Champ γ : Critère d'optimisation

Le troisième champ décrit le critère d'optimalité.

On cherche à minimiser Y, avec $Y \in \left\{C_{\max}, \overline{C}, \overline{C_w}, T_{\max}, \overline{T}, \overline{T_w}, L_{\max}, \overline{L}, \overline{L_w}, ...\right\}$ [11].

1.7 Représentation des problèmes d'ordonnancement par un diagramme de Gantt

Le diagramme de *Gantt*, du nom de son développeur *Henry Gantt* (1861-1919), est le moyen le plus simple et le plus utilisé pour représenter un ordonnancement, a la fois dans la littérature et dans les entreprises.

Ce diagramme de *Gantt* est un graphique, également appelé chronogramme, qui permet de représenter chronologiquement les tâches en fonction des contraintes techniques de succession, autrement dit, les contraintes d'antériorités. L'axe horizontal

des abscisses symbolise le temps, tandis que l'axe vertical des ordonnées représente les différentes machines. Chaque tâche est symbolisée par un segment de droite dont la longueur est proportionnelle à sa durée. L'origine du segment correspond à la date de début au plus tôt de l'opération, tandis que son extrémité indique la fin de la tâche.

De plus, il offre une vue d'ensemble des tâches à réaliser. Il permet de visualiser rapidement la date de début et la date de fin de chaque tâche, la durée prévue pour chaque tâche, les éventuels chevauchements entre les tâches et leurs durées, ainsi que la date de début et la date de fin globale du projet ou de l'ordonnancement [5, 34].

1.8 Théorie de complexité

La théorie de la complexité a été crée dans le but de fournir des informations sur la difficulté théorique des problèmes que l'on peut être amené à résoudre [22].

1.8.1 Principes fondamentaux de complexité

1.8.1.1 Algorithme : Un algorithme est dit polynomial si son temps d'exécution théorique est borné par une fonction polynomiale de la taille de l'instance. On dit qu'il a une complexité en $O(p(n))$, où p est un polynôme et n la taille de l'instance d'entrée. Dans le cas contraire, il est dit exponentiel [19].

On peut différencier les divers algorithmes de résolution par le moyen des critères suivants :

- L'efficacité de l'algorithme en terme de durée d'exécution : un algorithme est dit plus efficace qu'un autre si pour les mêmes données, il s'exécute en un laps de temps plus court.

- L'efficacité de l'algorithme en espace mémoire de stockage : un algorithme est dit plus efficace qu'un autre si pour résoudre le même problème, il utilise moins d'espace mémoire.
- La fiabilité de l'algorithme : plus un programme est complexe, plus il y a des risques d'existence de bugs, les bugs étant des erreurs plus ou moins évidentes qui se manifestent lors de la mise en exploitation d'un programme. Un programme est jugé plus fiable ou plus stable qu'un autre s'il présente moins de bugs.
- La robustesse de l'algorithme : elle mesure son degré de tolérance aux erreurs des utilisateurs et sa résistance aux attaques des pirates. Un programme est plus robuste qu'un autre s'il résiste mieux aux erreurs de manipulations des utilisateurs plus ou moins bien attentionnés [1].

1.8.1.2 Définition de complexité : La définition de la complexité d'un problème découle de la définition de la complexité des algorithmes. Informellement, la complexité d'un problème est mesurée par la complexité du meilleur programme résolvant ce problème de façon optimale [19].

1.8.1.3 Problème d'optimisation : Les problèmes d'optimisation sont des problèmes dont la résolution consiste à trouver parmi un ensemble de solutions celle qui répond le mieux à certains critères décrits sous forme d'une fonction objectif. Lorsque le domaine de solutions est discret, on parle alors de problèmes d'optimisation combinatoire. Les problèmes d'ordonnancement font partie des problèmes d'optimisation combinatoire.

Selon *Cook* [18], la théorie de la complexité a pour but d'analyser les coûts de résolution, notamment en terme de temps de calcul, des problèmes d'optimisation combinatoire [30].

1.8.1.4 Problème de décision : Selon *Garey* et *Johnson* [23], la théorie de complexité d'un problème d'optimisation se limite à la seule étude de problème de décision. Celui-ci, aussi appelé problème de reconnaissance, est un problème dont la résolution se limite à la réponse par «oui» ou par «non» à la question de savoir s'il existe une solution à un problème donné.

Pour chaque problème d'optimisation, on peut associer un problème de décision défini par la question suivante : «Existe-t-il une solution telle que son évaluation est majorée (Resp. minorée) par une constante fixée ?». Ceci est pour un problème d'optimisation, dont la fonction objectif est la minimisation (resp. maximisation).

Alors si la réponse à la question existe et est obtenue grâce à un algorithme polynomial alors il existera un algorithme polynomial résolvant le problème d'optimisation associé. Cette propriété nous permet de classer les problèmes d'optimisation grâce à leurs problèmes de décision. Ces problèmes peuvent être classés en deux classes principales : la classe $\mathcal{P}$ et la classe $\mathcal{NP}$ [30].

1.8.2 Classes de complexité

On cherche à déterminer "mathématiquement", si le problème étudié est plutôt "facile" ou bien s'il est plutôt "difficile". Des classes de problèmes ont été définies, et certaines propriétés qui sont associées à ces classes aident à répondre à de telles questions. Pratiquement la connaissance de telles propriétés permet souvent de guider dans le choix de l'algorithme à développer.

1.8.2.1 Classe $\mathcal{P}$: Un problème de décision est dans $\mathcal{P}$ s'il existe un algorithme déterministe pour le résoudre en temps polynomial par rapport à la taille des données. Le problème de cette classe est dit facile [4].

1.8.2.2 Classe $\mathcal{NP}$: Cette classe a un nom trompeur, $\mathcal{NP}$ ne correspond pas à Non Polynomial, mais à Polynomial Non-déterministe. Cette classe est une extension de la classe $\mathcal{P}$, elle représente la classe des problèmes de décision pour lesquels un algorithme non-déterministe peut vérifier en temps polynomial la validité d'une solution du problème traité [30].

1.8.2.3 Classe $\mathcal{NP}$-Complet : Parmi l'ensemble des problèmes appartenant à $\mathcal{NP}$, il existe un sous-ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes $\mathcal{NP}$-complet.

Un problème de décision est dit $\mathcal{NP}$-complet s'il est dans $\mathcal{NP}$ et que tout problème de la classe $\mathcal{NP}$ est réductible polynomialement en ce problème.

La résolution en temps polynomial d'un problème $\mathcal{NP}$-complet entraine la résolution en temps polynomial de tout problème dans la classe $\mathcal{NP}$.

Remarque 1.1 Un problème d'optimisation est $\mathcal{NP}$-difficile, si le problème de décision qui lui est associé est $\mathcal{NP}$-complet [28].

1.8.3 Réduction entre problème d'ordonnancement

Pour permettre une classification assez précise des problèmes d'ordonnancement, la réduction polynômiale est utilisée pour prouver d'une façon formelle qu'un problème est plus difficile à résoudre qu'un autre. Ceci permet l'élaboration d'une hiérarchie partielle d'où la possibilité de réutilisation des approches de résolutions développées pour un problème après quelques modifications mineures pour d'autres problèmes. La figure suivante présente un arbre de réduction des critères d'optimisation en ordonnancement [10].

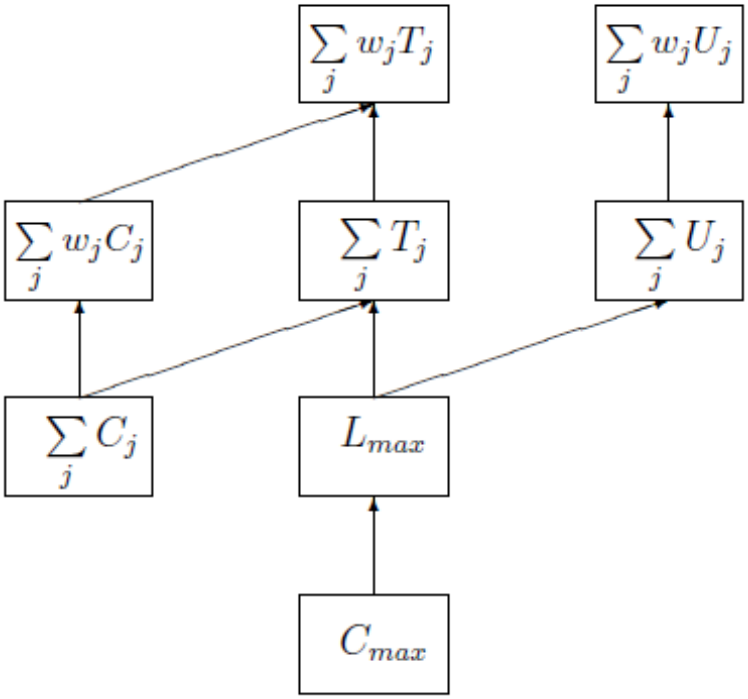


Figure 1.2 – Représentation de quelques réductions élémentaires entre les critères de performance d'ordonnancement

1.9 Conclusion

Dans ce chapitre, nous avons présenté les notions et les caractéristiques générales des problèmes d'ordonnancement, ainsi que les différentes méthodes pouvant être utilisées pour leur présentation et leur classification. En outre, nous avons défini la complexité des problèmes d'ordonnancement.

Chapitre 2

Étude de certains problèmes d'ordonnancement préemptif

2.1 Introduction

Dans ce chapitre, notre attention se focalise sur l'étude de certains problèmes d'ordonnancement, en mettant particulièrement l'accent sur les aspects de préemption. Nous explorerons trois problèmes faciles, ainsi qu'un problème $\mathcal{NP}$-difficile qui nécessite une approche approfondie.

2.2 Problème $1/pmtn, r_i/\overline{C}$

2.2.1 Description du problème

Le problème à une machine pour ordonnancer un ensemble de tâches indépendantes, les tâches sont préemptibles et caractérisées par des temps de traitement et des dates de disponibilité arbitraire.

L'objectif est la minimisation du temps de fin de traitement moyen.

2.2.2 Complexité du problème

Le problème $1/pmtn, r_i/\overline{C}$ est résolu en $O(nlogn)$ [17].

2.2.3 Résolution du problème

Ce problème peut être résolu en construisant un ordonnancement de gauche à droite en utilisant :

2.2.3.1 Règle de Smith modifiée (SRPT) : À chaque date de disponibilité ou une date de fin de traitement d'une tâche, ordonnancer parmi les tâches restantes la tâche disponible ayant le plus petit temps de traitement restant.

Théorème 2.1 *[12] Un ordonnancement construit par la règle de Smith modifiée est optimal pour le problème* $1/pmtn, r_i/\overline{C}$.

Exemple 2.1 Étant donné le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5	T_6
p_i	7	8	2	6	4	3
r_i	4	0	9	15	20	18

Tableau 2.1 – Données de l'exemple 2.1

La règle de Smith modifiée conduit à :

p_i	p_1	p_2	p_3	p_4	p_5	p_6	p_i	p_1	p_2	p_3	p_4	p_5	p_6
$t=0$	7	**8**	2	6	4	3	$t=17$	0	0	0	6	4	3
$t=4$	7	**4**	2	6	4	3	$t=18$	0	0	0	5	4	**3**
$t=8$	**7**	0	2	6	4	3	$t=20$	0	0	0	5	4	**1**
$t=9$	6	0	**2**	6	4	3	$t=21$	0	0	0	5	**4**	0
$t=11$	**6**	0	0	6	4	3	$t=25$	0	0	0	**5**	0	0
$t=15$	**2**	0	0	6	4	3	$t=30$	0	0	0	0	0	0

Tableau 2.2 – Résolution de l'exemple 2.1

La séquence obtenue par SRPT est : $T_2T_1T_3T_1T_4T_6T_5T_4$, sa représentation est illustrée dans la Figure 2.1 :

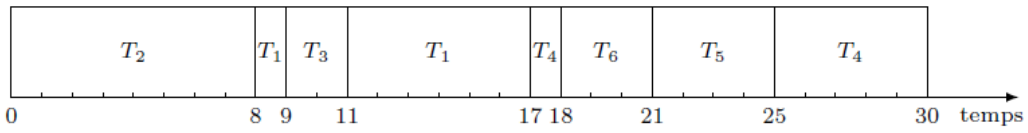


Figure 2.1 – Solution optimale de l'exemple 2.1

Avec $\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$, alors $\overline{C} = \frac{1}{6}(17+8+11+30+25+21)$, ce qui donne : $\overline{C} = 18,67$.

2.3 Problème $1/prec, pmtn, r_i/L_{\max}$

2.3.1 Description du problème

Le problème à une machine pour ordonnancer un ensemble de tâches dépendantes, les tâches sont préemptibles et caractérisées par des temps de traitement, des dates de disponibilité et des dates échues arbitraire. L'objectif est la minimisation de la grande tardivité.

2.3.2 Complexité du problème

Le problème $1/prec, pmtn, r_i/L_{\max}$ est résolu en $O(n^2)$ [29].

2.3.3 Résolution du problème

2.3.3.1 La règle EDD préemptive : Ordonnancer les tâches en commençant à la plus petite valeur de r_i. à chaque point de décision t donné par une date de disponibilité ou par la fin d'une tâche, ordonnancer une tâche T_i telle que : $r_i \leq t$, tous les prédécesseurs de T_i sont ordonnancés, la tâche T_i a la plus petite date échue modifiée.

2.3.3.2 La règle de la modification des dates échues : Nous faisons des modifications aux dates échues, de telle sorte qu'elles soient compatibles avec les contraintes de précédence : si $T_i \rightarrow T_k$ et $d_i < d_k$,alors $d_i^{'} = \min\{d_i, d_k - p_k\}$ avec $k = n \; bas \; 1$.

Théorème 2.2 *[12] Un ordonnancement construit par la règle EDD préemptive est optimal pour le problème* $1/prec, pmtn, r_i/L_{\max}$.

Exemple 2.2 Soit à ordonnancer 5 tâches T_1, T_2, T_3, T_4 et T_5 avec :

T_i	T_1	T_2	T_3	T_4	T_5
r_i	4	0	4	3	1
p_i	4	3	2	4	5
d_i	8	2	5	6	3

Tableau 2.3 – Données de l'exemple 2.2

Les contraintes de précédence sont représentées dans la Figure 2.2 :

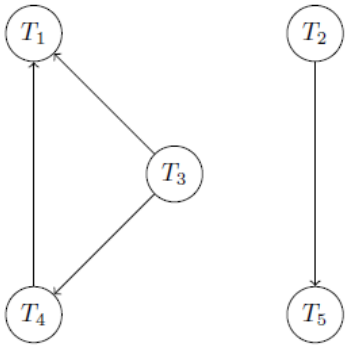


Figure 2.2 – Contraintes de précédence du problème $1/prec, pmtn, r_i/L_{\max}$

Nous faisons la modification des dates échues d_i par la règle de modification 2.3.3.2, nous obtenons :

T_i	T_1	T_2	T_3	T_4	T_5
d_i'	8	-2	2	4	3

Tableau 2.4 – Modification des dates échues d_i de l'exemple 2.2

Nous appliquons la règle EDD préemptive, sachant que :

F : L'ensemble des tâches disponibles.

st : La séquence des tâches déjà ordonnancées.

- $\mathbf{t = 0} \rightarrow F = \{T_2\}$ donc, $st = \{T_2\}$.
- $\mathbf{t = 1} \rightarrow F = \{T_2, T_5\}$ et nous avons le prédécesseur de T_5 n'est pas complètement ordonnancé donc, $st = \{T_2\}$.
- $\mathbf{t = 3} \rightarrow F = \{T_5, T_4\}$ et nous avons le prédécesseur de T_4 n'est pas ordonnancé donc, $st = \{T_2, T_5\}$.
- $\mathbf{t = 4} \rightarrow F = \{T_5, T_4, T_3, T_1\}$, nous avons les prédécesseurs de T_4 et T_1 ne sont pas ordonnancés et $d_3^{'} < d_5^{'}$ donc, $st = \{T_2, T_5, T_3\}$.
- $\mathbf{t = 6} \rightarrow F = \{T_5, T_4, T_1\}$, nous avons le prédécesseur de T_1 n'est pas ordonnancé et $d_5^{'} < d_4^{'}$ donc, $st = \{T_2, T_5, T_3, T_5\}$.
- $\mathbf{t = 10} \rightarrow F = \{T_4, T_1\}$ et nous avons le prédécesseur de T_1 n'est pas ordonnancé donc $st = \{T_2, T_5, T_3, T_5, T_4\}$.
- $\mathbf{t = 14} \rightarrow F = \{T_1\}$ donc, $st = \{T_2, T_5, T_3, T_5, T_4, T_1\}$.
- $\mathbf{t = 18}$ →Les tâches sont toutes ordonnancées.

La séquence obtenue est : $T_2T_5T_3T_5T_4T_1$, sa représentation est illustrée dans la Figure 2.3 :

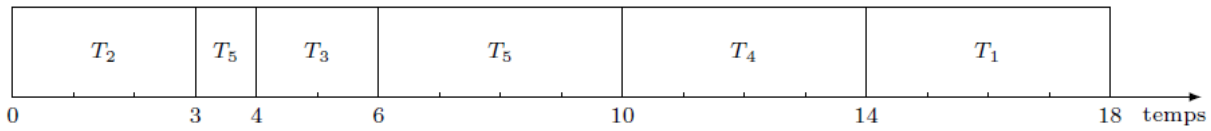


Figure 2.3 – Solution optimale de l'exemple 2.2

Calculons la grande tardivité, ce qui revient à déterminer $L_{\max}$ qui maximise $(C_i - d_i)$ donc, $L_{\max} = 10$.

2.4 Problème $Pm/pmtn/C_{\max}$

2.4.1 Description du problème

Le problème consiste à ordonnancer un ensemble de n tâches indépendantes et préemptives de durée p_i $(i = \overline{1,n})$ sur m machines parallèles identiques afin de minimiser la durée totale $C_{\max}$.

2.4.2 Complexité du problème

Le problème $Pm/pmtn/C_{\max}$ est résolu en un temps polynomial $(O(n))$ [33].

2.4.3 Résolution du problème

Théorème 2.3 *[33] Le problème $Pm/pmtn/C_{\max}$ est résolu par l'algorithme de Mc Naughton (1959).*

2.4.3.1 Description de l'algorithme de Mc Naughton L'algorithme permet de construire un ordonnancement en commençant par calculer la borne inférieure $\overline{B}$ sur la durée totale, telle que $\overline{B} = \max\left\{\max_{1\leq i\leq n}\{p_i\}, \left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil\right\}$, cette borne fournit une valeur optimale pour le C_{max}.

Une fois la borne déterminée, nous sélectionnons une tâche T_i, de l'ensemble des tâches non encore traitées et la traite sur la première machine à l'instant $t = 0$. Nous traitons sur la même machine une autre tâche T_k non encore traitée à l'instant $(t + p_i)$ et ainsi de suite tant que $(t + p_i < \overline{B})$. Sinon, la tâche T_i est interrompue et nous transfèrons le reste de cette tâche à la machine suivante.

Cet enchaînement est répété autant de fois jusqu'à ce que toutes les tâches soient traitées.

L'algorithme est le suivant :

Algorithme 2.1 Algorithme de Mc Naughton

Début
$\overline{B} \leftarrow \max\left\{\max_{1\leq i\leq n}\{p_i\}, \left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil\right\}$;
$t \leftarrow 0$; $i \leftarrow 1$; $j \leftarrow 1$;
Répéter
Si $(t + p_i < \overline{B})$ **alors**
Affecter la tâche T_i à la machine M_j à l'instant t ;
$t \leftarrow t + p_i$;
$i \leftarrow i + 1$;
Sinon si $(t + p_i = \overline{B})$ **alors**
Affecter la tâche T_i à la machine M_j à l'instant t ;
$t \leftarrow t + p_i$; $i \leftarrow i + 1$; $j \leftarrow j + 1$;
Sinon traiter T_i sur la machine M_j à l'instant t pour $(\overline{B} - t)$ unités de temps et le reste de la tâche $(p_i - (\overline{B} - t))$ sur la machine suivante ;
$p_i \leftarrow p_i - (\overline{B} - t)$; $t \leftarrow 0$; $j \leftarrow j + 1$;
Fin si ;
Fin si ;
Jusqu'à $i \leftarrow n$;
$C_{\max} \leftarrow \overline{B}$;
Fin.

Quand une tâche est morcelée, les deux morceaux (parts) de la tâche sont placés en début et fin de l'ordonnancement ce qui permet d'éviter les chevauchements. Nous remarquons aussi que le nombre de préemptions est au plus égal à $(m - 1)$.

Exemple 2.3 Nous souhaitons traiter 10 tâches sur 4 machines, les temps de traitement sont représentés par le tableau ci-dessous :

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
p_i	2	10	3	16	5	8	19	13	13	7

Tableau 2.5 – Données de l'exemple 2.3

En appliquant l'algorithme de *Mc Naughton*, nous trouvons la solution représentée par le diagramme de la Figure ci-dessous 2.4, avec $C_{max} = max\{19, 24\} = 24$.

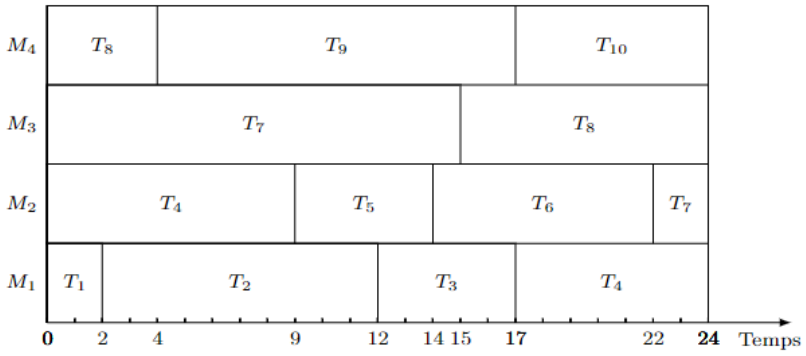


Figure 2.4 – Solution générée par l'algorithme de Mc Naughton

Nous avons interrompu au maximum 3 tâches T_4, T_7 et T_8.

Remarque 2.4 La tâche T_4 commence son traitement sur la machine M_2 à l'instant $t = 0$ et est interrompu à l'instant $t = 9$ pour terminer son traitement sur la machine M_1 à l'instant $t = 17$.

2.5 Problème $Pm/pmtn, c_{j,j'}/C_{\max}$

2.5.1 Description du problème

Soit le problème qui consiste à ordonnancer un ensemble de n tâches $\{T_1, ..., T_n\}$ indépendantes et préemptives de durée p_i $(i = \overline{1, n})$ sur m machines parallèles identiques $\{M_1, ..., M_m\}$ afin de minimiser la durée totale $C_{\max}$. Si une tâche T_i traitée sur une machine M_j est interrompue pour terminer son traitement sur une autre machine $M_{j'}$ alors son transport nécessite $c_{j,j'}$ unités de temps, que nous allons appeler temps de changement de machines.

Nous supposons d'une part que toutes les tâches sont disponibles à l'instant $t = 0$, et que toutes les machines ne présentent pas de période d'indisponibilité durant tout l'ordonnancement.

D'autre part le temps de changement de machine $c_{j,j'}$ ($\forall\ j, j' et\ j \neq j'$) ne dépend pas des tâches.

2.5.2 Complexité du problème

Le problème, avec seulement deux machines et des temps de changement de machines constants, est $\mathcal{NP}$-difficile.

Théorème 2.5 *[26] Le problème $P2/pmtn, c_{j,j'}/C_{\max}$ est $\mathcal{NP}$-difficile.*

Preuve. Réduction du problème de 2-partition qui est $\mathcal{NP}$-complet.

Soit le problème de 2-partition suivant :

"Soient n entiers positifs $a_1, ..., a_n$, existe-t-il un sous ensemble $J \subset I = \{1, ..., n\}$, tel que $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$?"

Montrons que le problème de 2-partition se réduit polynomialement au problème d'ordonnancement O suivant :

Étant données n tâches $T_1, ..., T_n$, indépendantes et morcelables de durées $p_i = a_i$ ($\forall i = \overline{1, n}$), à traiter sur deux machines parallèles identiques M_1 et M_2. Le temps de changement de machines est $c_{j,j'} = c = \frac{1}{2}\sum_{i=1}^{n} a_i, \forall j, j' \in \{1, 2\}$ et $j \neq j'$. Nous avons aussi $c_{11} = c_{22} = 0$.

Soit $k = \frac{1}{2}\sum_{i=1}^{n} a_i$, "existe-t-il un ordonnancement des tâches $T_1, ..., T_n$, sur M_1 et M_2, de durée $\leq k$?"

Le problème O est dans $\mathcal{NP}$, en effet, nous pouvons vérifier en temps polynomial qu'une solution quelconque vérifie toutes les contraintes.

Supposons que le problème de 2-partition a une solution, nous avons donc un sous ensemble $J \subset I$, tel que : $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$.

À partir de cette solution, nous pouvons construire un ordonnancement pour le problème O en traitant les $|J|$ tâches de J sur M_1 et les $|I \setminus J|$ tâches restantes sur M_2. Dans ce cas aucune tâche n'est interrompue, et par conséquent aucune tâche n'est transportée d'une machine à une autre, donc la durée de l'ordonnancement est égale à $\frac{1}{2}\sum_{i=1}^{n} a_i$.

Supposons maintenant qu'il existe une solution pour le problème O de durée $\leq \frac{1}{2}\sum_{i=1}^{n} a_i$.

Donc aucune tâche n'a été interrompue pour reprendre son traitement sur une autre machine car dans le cas contraire nous aurions $C_{max} > \frac{1}{2}\sum_{i=1}^{n} a_i$ (nous pouvons interrompre une tâche pour la reprendre sur la même machine, dans ce cas les temps de transport sont nuls et la préemption n'est pas intéressante).

Par conséquent chaque tâche n'est traitée que sur l'une des deux machines. Ainsi le problème de 2-partition a une solution en considérant l'ensemble J comme étant l'ensemble des tâches traitées sur M_1 et l'ensemble $I \setminus J$ celui des tâches traitées sur M_2. □

Par conséquent, le problème général $Pm/pmtn, c_{j,j'}/C_{\max}$ est $\mathcal{NP}$-difficile [26].

2.6 Conclusion

Dans ce chapitre, nous avons abordé plusieurs problèmes d'ordonnancement préemptif, en les classant selon leur niveau de difficulté. Nous avons exposé des problèmes faciles, en fournissant des algorithmes pour les résoudre, accompagnés d'exemples d'application. En ce qui concerne le problème difficile, nous avons démontré son $\mathcal{NP}$-difficulté en établissant sa complexité.

Chapitre 3

Heuristiques proposées pour la résolution du problème $Pm/pmtn, c_{j,j'}/C_{\max}$

3.1 Introduction

Les problèmes d'ordonnancement sont classés parmi les problèmes d'affectation et en tant que problèmes $\mathcal{NP}$-difficiles, ils sont généralement résolus par des méthodes heuristiques ou des méthodes exactes pour des problèmes de petite taille. Puisque le problème $Pm/pmtn, c_{j,j'}/C_{\max}$ est $\mathcal{NP}$-difficile, nous envisageons de le résoudre à l'aide des heuristiques.

3.2 Représentation du problème

3.2.1 Position du problème

Le problème considéré est l'ordonnancement de tâches indépendantes et préemptives sur des machines parallèles identiques afin de minimiser la durée totale. Les tâches peuvent être préemptées et transportées d'une machine à une autre, avec des temps de changement de machine spécifiés (ou temps de transport). Le transport des tâches est effectué par un transporteur disponible et de capacité suffisante.

Les contraintes portent sur le mode de traitement, autorisant la préemption des tâches et leur transport entre les machines avec des temps de changement de machine dépendant uniquement de la distance entre les machines. Toutes les tâches sont disponibles à l'instant $t = 0$, et que toutes les machines ne présentent pas de période d'indisponibilité durant l'ordonnancement.

Dans ce problème, la préemption d'une tâche sur la même machine n'est pas considérée, car il suffit de réorganiser la séquence des tâches sur la même machine pour regrouper les parties de la tâche préemptée.

3.2.2 Détermination d'une borne inférieure

Dans le cas général $\overline{B} := \max\left\{\max_{1\leq i\leq n}\{p_i\}, \left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil\right\}$ est une borne inférieure pour le C_{max}. Si au moins une tâche est interrompue pour être traitée sur une autre machine, nous proposons la borne inférieure $\overline{B'} := \min_{1\leq i\leq n}\{p_i\} + \min_{j,j' \text{et } j\neq j'}\{c_{j,j'}\}$, par conséquent $\overline{\overline{B}} = \max\left\{\overline{B}, \overline{B'}\right\}$ est une borne inférieure pour le C_{max}.

3.3 Heuristiques proposées pour résoudre $Pm/pmtn, c_{j,j'}/C_{\max}$

Définition 3.1 Une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile [8].

3.3.1 Heuristiques proposées

Nous allons proposer trois heuristiques pour résoudre le problème considéré.

3.3.1.1 Description de l'heuristique H1

Principe : Cette heuristique est inspirée de l'algorithme de *Mc Naughton* 2.1, en attribuant les tâches à la première machine jusqu'à atteindre $\overline{\overline{B}} = \max\left\{\overline{B}, \overline{B'}\right\}$, Si la dernière tâche n'est pas entièrement traitée sur la machine M_1, elle sera transportée à la machine suivante à partir du $t = 0$ pour une période de $p_i - (\overline{\overline{B}} - D_1)$ unités (D_1 est la date de début de la tâche T_i sur la machine M_1). Si $p_i - (\overline{\overline{B}} - D_1) + c_{2,1} > D_1$, alors un décalage est réalisé pour que le début de la tâche T_i sur la machine M_1 soit exactement $p_i - (\overline{\overline{B}} - D_1) + c_{2,1}$. $\overline{\overline{B}}$ sera égale à $p_i + c_{2,1}$ ($\overline{\overline{B}} = p_i + c_{2,1}$). Le traitement des tâches non encore traitées sur la machine M_2 se poursuit, et si une tâche est interrompue, le même raisonnement est suivi que précédemment.

Notons que les $c_{j,j'}$ ($\forall\ j, j' et\ j \neq j'$) sont représentés sous forme d'un tableau de dimension $(m \times m)$.

$M_j/M_{j'}$	M_1	M_2	...	M_m
M_1	0	$c_{1,2}$	...	$c_{1,m}$
M_2	$c_{2,1}$	0	...	$c_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
M_m	$c_{m,1}$	$c_{m,2}$	...	0

Tableau 3.1 – Temps de changement de machines $c_{j,j'}$

Notations utilisées dans l'algorithme H1 :

k	Indice de la position de la tâche T_i sur la machine M_j.
$d_{j,k}$	Date de début de traitement d'une tâche sur la machine M_j à la position k.
$f_{j,k}$	Date de fin de traitement d'une tâche sur la machine M_j à la position k.
r	Temps restant pour atteindre la borne inférieure.
nb_j	Nombre total de tâches traitées sur la machine M_j.
$c_{j,j-1}$	Temps de changement de la machine M_j à la machine M_{j-1}.

Les différentes étapes de cette heuristique sont résumées dans l'algorithme suivant :

Algorithme 3.1 Algorithme de H1

Début
$\overline{\overline{B}} \leftarrow \max\{\overline{B}, \overline{B'}\}$;
$t \leftarrow 0$; $i \leftarrow 1$; $j \leftarrow 1$; $k \leftarrow 1$;
Répéter
Si $(t + p_i < \overline{\overline{B}})$ **alors**
Affecter la tâche T_i à la machine M_j à l'instant t ;
$d_{j,k} \leftarrow t$; $f_{j,k} \leftarrow t + p_i$;
$t \leftarrow t + p_i$; $i \leftarrow i + 1$; $k \leftarrow k + 1$;
Sinon si $(t + p_i = \overline{\overline{B}})$ **alors**
Affecter la tâche T_i à la machine M_j à l'instant t ;
$d_{j,k} \leftarrow t$; $f_{j,k} \leftarrow t + p_i$;
$nb_j \leftarrow k$;
$t \leftarrow 0$; $i \leftarrow i + 1$; $j \leftarrow j + 1$; $k \leftarrow 1$;
Sinon affecter la tâche T_i à la machine M_j à l'instant t pour $(\overline{\overline{B}} - t)$ unités du temps ;
$r \leftarrow \overline{\overline{B}} - t$;
$d_{j,k} \leftarrow t$; $f_{j,k} \leftarrow \overline{\overline{B}}$;
$p_i \leftarrow p_i - r$; $nb_j \leftarrow k$;
$t \leftarrow 0$; $j \leftarrow j + 1$; $k \leftarrow 1$;
Affecter le reste de la tâche T_i à la machine M_j à l'instant $t = 0$;
$d_{j,k} \leftarrow t$; $f_{j,k} \leftarrow t + p_i$;
$i \leftarrow i + 1$; $k \leftarrow k + 1$; $t \leftarrow t + p_i$;
Si $(d_{j-1,nb_{j-1}} - f_{j,k-1} < c_{j,j-1})$ **alors**
Changer la date de début sur la machine M_{j-1} ;
$d_{j-1,nb_{j-1}} \leftarrow f_{j,k-1} + c_{j,j-1}$;
$\overline{\overline{B}} \leftarrow d_{j-1,nb_{j-1}} + r$;
Fin si ;
Fin si ;
Fin si ;
Jusqu'à $i = n$;
$C_{\max} \leftarrow \overline{\overline{B}}$;
Fin.

3.3.1.2 Description de l'heuristique H2 : Nous définissons d'abord la procédure RANGEMENT.

Définition 3.2 Cette procédure a pour objectif de trouver une bonne séquence des machines sm de manière à minimiser les temps de changement de machines $c_{j,j'}$ $(\forall\ j, j' et\ j \neq j')$.

Algorithme 3.2 Procédure RANGEMENT

Début

1. Trouver le minimum de la matrice des temps de transport $\min\{c_{j,j'} \mid j \neq j'\}$, soit $c_{k,l}$ ce minimum ;
2. A savoir que la tâche T_i traitée sur une machine M_k est interrompue pour terminer son traitement sur une autre machine M_l, donc la machine rangée est M_l ;
3. Dans un tableau de dimension $(1 \times m)$ sauvegarder l'indice de la machine rangée dans la case i avec $i \leftarrow 1$;
4. Supprimer la colonne et la ligne l ;
5. Déterminer le minimum de la colonne k, soit $c_{h,k}$ ce minimum, donc la machine rangée est M_k ;
6. Dans la case $i \leftarrow i + 1$ sauvegarder l'indice de la machine rangée ;
7. Supprimer la colonne et la ligne k ;
8. Poser $k \leftarrow h$ et aller à l'étape 3 ;
9. Répéter les étapes 3, 4 et 5 jusqu'à l'élimination de toutes les machines et $(i \leftarrow m)$;

Fin.

La séquence sm obtenue est celle qui est sauvegardée dans le tableau. Ainsi que, les valeurs des minimums Min_k de la matrice des temps de changement de machines figurant dans la séquence sm sont sauvegardées dans un tableau de dimension $(1 \times m - 1)$, où Min_k correspond à une certaine valeur $c_{j,j'} \in sm$.

Principe : Cette heuristique permet de construire un ordonnancement en commençant par ranger les machines selon la procédure RANGEMENT, et puis traiter les tâches, sur la première machine dans l'ordre et en remplissant successivement les machines jusqu'à atteindre $\overline{\overline{B}} = \max\left\{\overline{B}, \overline{B'}\right\}$. Si une tâche est interrompue, nous vérifions si le temps de transport $Min_k(k := \overline{1, m-1})$ est respecté. Sinon, nous faisons un décalage.

Soit l'algorithme :

Algorithme 3.3 Algorithme de H2
Début
Ranger les machines selon la procédure RANGEMENT ;
Faire un appel à l'heuristique H1 (remplacer $c_{j,j-1}$ par Min_{j-1}) ;
Fin.

3.3.1.3 Description de l'heuristique H3 :

Nous définissons d'abord la règle LPT.

Définition 3.3 LPT : Classer les tâches par ordre décroissant des temps de traitement.

Principe : Consiste à assigner les tâches, classées selon l'ordre LPT, aux machines de manière à minimiser les temps de transport. L'affectation des tâches suit le même principe que celui de l'algorithme de *Mc Naughton*. De plus, un test est effectué pour vérifier que les temps de transport sont respectés. Si ce n'est pas le cas, un décalage est appliqué. Les différentes étapes de cette heuristique sont résumées dans l'algorithme suivant :

Algorithme 3.4 Algorithme de H3
Début
Ranger les tâches dans l'ordre LPT ;
Faire un appel à l'heuristique H2 ;
Fin.

3.3.1.4 Exemple d'application :

Considérons l'ordonnancement de 9 tâches sur 4 machines parallèles identiques. Les temps de traitement et les temps de transport sont donnés ci-dessous :

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
p_i	2	9	4	16	5	8	12	13	4

Tableau 3.2 – Temps de traitement de l'exemple d'application

$M_j/M_{j'}$	M_1	M_2	M_3	M_4
M_1	0	2	15	10
M_2	6	0	14	8
M_3	12	3	0	3
M_4	6	17	20	0

Tableau 3.3 – Temps de transport de l'exemple d'application

Nous appliquons cet exemple sur les trois heuristiques proposées précédemment.

La borne inférieure :

$\overline{\overline{B}} = \max\left\{\overline{B}, \overline{B'}\right\}.$

$\overline{\overline{B}} = \max\left\{\max\left\{\max\limits_{1\leq i\leq n}\left\{p_i\right\}, \left\lceil \frac{1}{m}\sum\limits_{i=1}^{n} p_i \right\rceil\right\}, \min\limits_{1\leq i\leq n}\left\{p_i\right\} + \min\limits_{j,j' \text{et } j\neq j'}\left\{c_{j,j'}\right\}\right\}.$

$\overline{\overline{B}} = \max\left\{\max\left\{16, \left\lceil \frac{1}{4}(73)\right\rceil\right\}, 2+2\right\}.$

$\overline{\overline{B}} = \max\left\{19, 4\right\}.$

$\overline{\overline{B}} = 19.$

A. Application de la $1^{ère}$ heuristique

- **Initialisation :**

$\overline{\overline{B}} = 19; \quad t = 0; \quad i = 1; \quad j = 1; \quad k = 1;$

- **Itération 01 :**

$t + p_1 = 2 < 19$; affecter la tâche T_1 à la machine M_1 à l'instant $t = 0$;

$d_{1,1} = 0; \quad f_{1,1} = 2; \quad t = 2; \quad i = 2; \quad k = 2;$

- **Itération 02 :**

$t + p_2 = 11 < 19$; affecter la tâche T_2 à la machine M_1 à l'instant $t = 2$;

$d_{1,2} = 2; \quad f_{1,2} = 11; \quad t = 11; \quad i = 3; \quad k = 3;$

- **Itération 03 :**

$t + p_3 = 15 < 19$; affecter la tâche T_3 à la machine M_1 à l'instant $t = 11$;

$d_{1,3} = 11; \quad f_{1,3} = 15; \quad t = 15; \quad i = 4; \quad k = 4;$

- **Itération 04 :**

$t + p_4 = 31 > 19$; affecter la tâche T_4 à la machine M_1 à l'instant $t = 11$ pour 4 unités;

$r = 4; \quad d_{1,4} = 15; \quad f_{1,3} = 19; \quad p_4 = 12; \quad nb_1 = 4; \quad t = 0; \quad j = 2; \quad k = 1;$

Affecter le reste de la tâche T_4 à la machine M_2 à l'instant $t = 0$;

$d_{2,1} = 0; \quad f_{2,1} = 12; \quad t = 12; \quad i = 5; \quad k = 2;$

$d_{1,4} - f_{2,1} < c_{2,1} \rightarrow 3 < 6$ **alors** changer la date de début sur la machine M_1;

$d_{1,4} = f_{2,1} + c_{2,1} = 12 + 6 = 18; \quad \overline{\overline{B}} = d_{1,4} + r = 18 + 4 = 22;$

- **Itération 05 :**

$t + p_5 = 17 < 22$; affecter la tâche T_5 à la machine M_2 à l'instant $t = 12$;

$d_{2,2} = 12; \quad f_{2,2} = 17; \quad t = 17; \quad i = 6; \quad k = 3;$

- **Itération 06 :**

$t + p_6 = 25 > 22$; affecter la tâche T_6 à la machine M_2 à l'instant $t = 17$ pour 5 unités ;

$r = 5; \quad d_{2,3} = 17; \quad f_{2,3} = 22; \quad p_6 = 3; \quad nb_2 = 3; \quad t = 0; \quad j = 3; \quad k = 1;$

Affecter le reste de la tâche T_6 à la machine M_3 à l'instant $t = 0$;

$d_{3,1} = 0; \quad f_{3,1} = 3; \quad t = 3; \quad i = 7; \quad k = 2;$

$d_{2,3} - f_{3,1} > c_{3,2} \rightarrow 14 > 3;$

- **Itération 07 :**

$t + p_7 = 15 < 22$; affecter la tâche T_7 à la machine M_3 à l'instant $t = 3$;

$d_{3,2} = 3; \quad f_{3,2} = 15; \quad t = 15; \quad i = 8; \quad k = 3;$

- **Itération 08 :**

$t + p_8 = 28 > 22$; affecter la tâche T_8 à la machine M_3 à l'instant $t = 15$ pour 7 unités;

$r = 7; \quad d_{3,3} = 15; \quad f_{3,3} = 22; \quad p_8 = 6; \quad nb_3 = 3; \quad t = 0; \quad j = 4; \quad k = 1;$

Affecter le reste de la tâche T_8 à la machine M_4 à l'instant $t = 0$;

$d_{4,1} = 0; \quad f_{4,1} = 6; \quad t = 6; \quad i = 9; \quad k = 2;$

$d_{3,3} - f_{4,1} < c_{4,3} \rightarrow 9 < 20$ **alors** changer la date de début sur la machine M_3;

$d_{3,3} = f_{4,1} + c_{4,3} = 9 + 20 = 29; \quad \overline{\overline{B}} = d_{3,3} + r = 29 + 7 = 36;$

- **Itération 09 :**

$t + p_9 = 10 < 36$; affecter la tâche T_9 à la machine M_4 à l'instant $t = 6$;

$d_{4,2} = 6; \quad f_{4,2} = 10; \quad t = 10; \quad i = 10; \quad k = 3;$

Résultat : $C_{\max} = 36$.

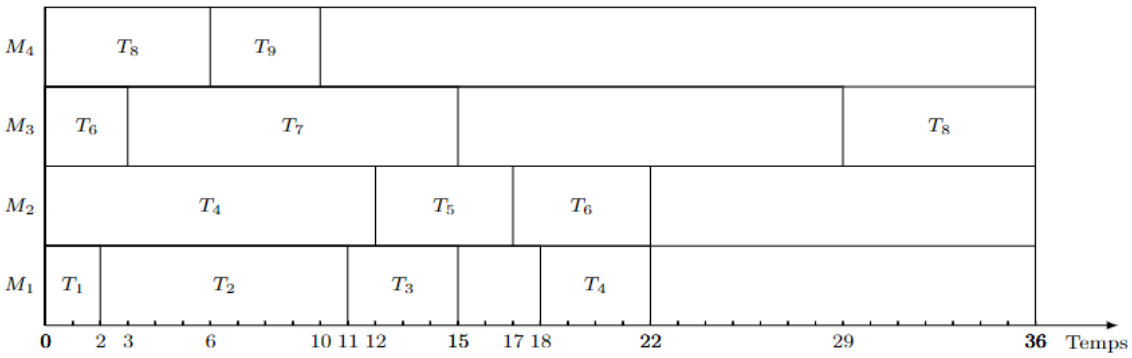


Figure 3.1 – Solution générale de l'exemple par l'heuristique H1

B. Application de la $2^{ème}$ heuristique

La procédure RANGEMENT

- **(1) :** $\min\limits_{j,j' \text{ et } j\neq j'} \{c_{j,j'}\} = 2 \rightarrow c_{1,2} = 2$, la $1^{ère}$ machine rangée est $M_2; i = 1$;
- **(2) :** Suppression de la colonne et la ligne 2 ;
- **(3) :** $\min\limits_{j,1 \text{ et } j\neq 1} \{c_{j,1}\} = 6 \rightarrow c_{4,1} = 6$, la $2^{ème}$ machine rangée est $M_1; i = 2$;
- **(4) :** Suppression de la colonne et la ligne 1 ;
- **(5) :** $\min\limits_{j,4 \text{ et } j\neq 4} \{c_{j,4}\} = 3 \rightarrow c_{3,4} = 3$, la $3^{ème}$ machine rangée est $M_4; i = 3$;
- **(6) :** Suppression de la colonne et la ligne 4 ;
- **(7) :** $\min\limits_{j,3 \text{ et } j\neq 3} \{c_{j,3}\}$ n'existe pas, la $4^{ème}$ machine rangée est $M_3; i = 4$;
- **(8) :** Suppression de la colonne et la ligne 3 ;

La séquence sm obtenue : M_2, M_1, M_4, M_3.

Les temps de transport obtenus : $Min_1 = 2, \quad Min_2 = 6, \quad Min_3 = 3.$

Déroulement de l'algorithme :

- **Initialisation :**

$\overline{\overline{B}} = 19; \quad t = 0; \quad i = 1; \quad j = 1; \quad k = 1;$

- **Itération 01 :**

$t + p_1 = 2 < 19$; affecter la tâche T_1 à la machine M_2 à l'instant $t = 0$;

$d_{2,1} = 0; \quad f_{2,1} = 2; \quad t = 2; \quad i = 2; \quad k = 2;$

- **Itération 02 :**

$t + p_2 = 11 < 19$; affecter la tâche T_2 à la machine M_2 à l'instant $t = 2$;

$d_{2,2} = 2; \quad f_{2,2} = 11; \quad t = 11; \quad i = 3; \quad k = 3;$

- **Itération 03 :**

$t + p_3 = 15 < 19$; affecter la tâche T_3 à la machine M_2 à l'instant $t = 11$;

$d_{2,3} = 11; \quad f_{2,3} = 15; \quad t = 15; \quad i = 4; \quad k = 4;$

- **Itération 04 :**

$t + p_4 = 31 > 19$; affecter la tâche T_4 à la machine M_2 à l'instant $t = 11$ pour 4 unités;

$r = 4; \quad d_{2,4} = 15; \quad f_{2,3} = 19; \quad p_4 = 12; \quad nb_2 = 4; \quad t = 0; \quad j = 2; \quad k = 1;$

Affecter le reste de la tâche T_4 à la machine M_1 à l'instant $t = 0$;

$d_{1,1} = 0; \quad f_{1,1} = 12; \quad t = 12; \quad i = 5; \quad k = 2;$

$d_{2,4} - f_{1,1} > Min_1 \rightarrow 3 > 2;$

- **Itération 05 :**

$t + p_5 = 17 < 19$; affecter la tâche T_5 à la machine M_1 à l'instant $t = 12$;

$d_{1,2} = 12; \quad f_{1,2} = 17; \quad t = 17; \quad i = 6; \quad k = 3;$

- **Itération 06 :**

$t + p_6 = 25 > 19$; affecter la tâche T_6 à la machine M_1 à l'instant $t = 17$ pour 2 unités ;

$r = 2; \quad d_{1,3} = 17; \quad f_{1,3} = 19; \quad p_6 = 6; \quad nb_1 = 3; \quad t = 0; \quad j = 3; \quad k = 1;$

Affecter le reste de la tâche T_6 à la machine M_4 à l'instant $t = 0$;

$d_{4,1} = 0; \quad f_{4,1} = 6; \quad t = 6; \quad i = 7; \quad k = 2;$

$d_{1,3} - f_{4,1} > Min_2 \rightarrow 11 > 6;$

- **Itération 07 :**

$t + p_7 = 18 < 19$; affecter la tâche T_7 à la machine M_4 à l'instant $t = 6$;

$d_{4,2} = 6; \quad f_{4,2} = 18; \quad t = 18; \quad i = 8; \quad k = 3;$

- **Itération 08 :**

$t + p_8 = 31 > 19$; affecter la tâche T_8 à la machine M_4 à l'instant $t = 18$ pour 1 unité;

$r = 1$; $d_{4,3} = 18$; $f_{4,3} = 19$; $p_8 = 12$; $nb_4 = 3$; $t = 0$; $j = 4$; $k = 1$;

Affecter le reste de la tâche T_8 à la machine M_3 à l'instant $t = 0$;

$d_{3,1} = 0$; $f_{3,1} = 12$; $t = 12$; $i = 9$; $k = 2$;

$d_{4,3} - f_{3,1} > Min_3 \rightarrow 6 > 3$;

- **Itération 09 :**

$t + p_9 = 16 < 19$; affecter la tâche T_9 à la machine M_3 à l'instant $t = 12$;

$d_{3,2} = 12$; $f_{3,2} = 16$; $t = 16$; $i = 10$; $k = 3$;

Résultat : $C_{\max} = 19$.

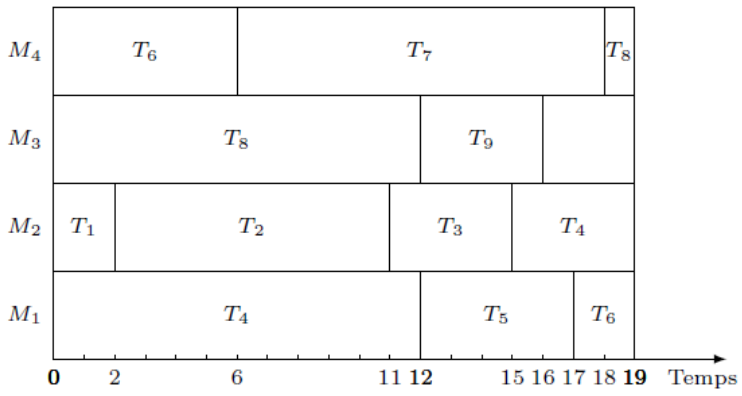


Figure 3.2 – Solution générale de l'exemple par l'heuristique H2

C. Application de la $3^{ème}$ heuristique

La règle LPT :

Nous trouvons la liste $L : \{T_4, T_8, T_7, T_2, T_6, T_5, T_3, T_9, T_1\}$

Déroulement de l'algorithme :

- **Initialisation :**

$\overline{\overline{B}} = 19; \quad t = 0; \quad i = 1; \quad j = 1; \quad k = 1;$

- **Itération 01 :**

$t + p_4 = 16 < 19$; affecter la tâche T_4 à la machine M_2 à l'instant $t = 0$;

$d_{2,1} = 0; \quad f_{2,1} = 16; \quad t = 16; \quad i = 2; \quad k = 2;$

- **Itération 02 :**

$t + p_8 = 29 > 19$; affecter la tâche T_8 à la machine M_2 à l'instant $t = 16$ pour 3 unités;

$r = 4; \quad d_{2,2} = 16; \quad f_{2,3} = 19; \quad p_8 = 10; \quad nb_2 = 2; \quad t = 0; \quad j = 2; \quad k = 1;$

Affecter le reste de la tâche T_8 à la machine M_1 à l'instant $t = 0$;

$d_{1,1} = 0; \quad f_{1,1} = 10; \quad t = 10; \quad i = 3; \quad k = 2;$

$d_{2,2} - f_{1,1} > Min_1 \rightarrow 6 > 2;$

- **Itération 03 :**

$t + p_7 = 22 > 19$; affecter la tâche T_7 à la machine M_1 à l'instant $t = 10$ pour 9 unités;

$r = 9; \quad d_{1,2} = 10; \quad f_{1,2} = 19; \quad p_7 = 3; \quad nb_1 = 2; \quad t = 0; \quad j = 3; \quad k = 1;$

Affecter le reste de la tâche T_7 à la machine M_4 à l'instant $t = 0$;

$d_{4,1} = 0; \quad f_{4,1} = 3; \quad t = 3; \quad i = 4; \quad k = 2;$

$d_{1,2} - f_{4,1} > Min_2 \rightarrow 7 > 6;$

- **Itération 04 :**

$t + p_2 = 12 < 19$; affecter la tâche T_2 à la machine M_4 à l'instant $t = 3$;

$d_{4,2} = 3; \quad f_{4,2} = 12; \quad t = 12; \quad i = 5; \quad k = 3;$

- **Itération 05 :**

$t + p_6 = 20 > 19$; affecter la tâche T_6 à la machine M_4 à l'instant $t = 12$ pour 7 unités ;

$r = 7; \quad d_{4,3} = 12; \quad f_{4,3} = 19; \quad p_6 = 1; \quad nb_4 = 3; \quad t = 0; \quad j = 4; \quad k = 1;$

Affecter le reste de la tâche T_6 à la machine M_3 à l'instant $t = 0$;

$d_{3,1} = 0; \quad f_{3,1} = 1; \quad t = 1; \quad i = 6; \quad k = 2;$

$d_{4,3} - f_{3,1} > Min_3 \rightarrow 11 > 3;$

- **Itération 06 :**

$t + p_5 = 6 < 19$; affecter la tâche T_5 à la machine M_3 à l'instant $t = 1$;

$d_{3,2} = 1; \quad f_{3,2} = 6; \quad t = 6; \quad i = 7; \quad k = 3;$

- **Itération 07 :**

$t + p_3 = 10 < 19$; affecter la tâche T_3 à la machine M_3 à l'instant $t = 6$;

$d_{3,3} = 6; \quad f_{3,3} = 10; \quad t = 10; \quad i = 8; \quad k = 4;$

- **Itération 08 :**

$t + p_9 = 14 < 19$; affecter la tâche T_9 à la machine M_3 à l'instant $t = 10$;

$d_{3,4} = 10; \quad f_{3,4} = 14; \quad t = 14; \quad i = 9; \quad k = 5;$

- **Itération 09 :**

$t + p_1 = 16 < 19$; affecter la tâche T_1 à la machine M_3 à l'instant $t = 14$;

$d_{3,5} = 14; \quad f_{3,5} = 16; \quad t = 16; \quad i = 10; \quad k = 6;$

Résultat : $C_{\max} = 19$.

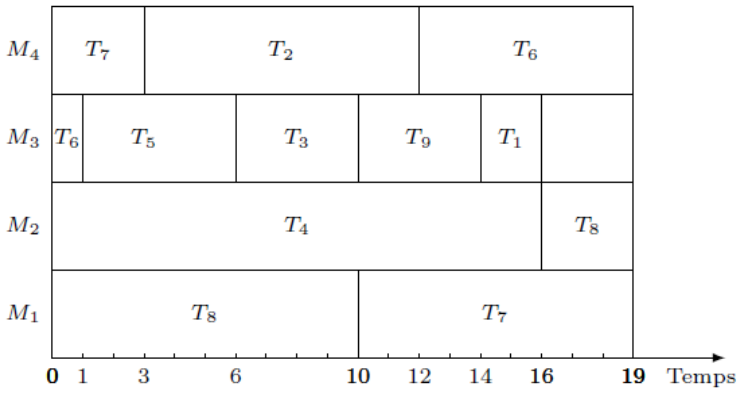


Figure 3.3 – Solution générale de l'exemple par l'heuristique H3

3.4 Conclusion

Le problème étudié concerne l'ordonnancement sur des machines parallèles identiques afin de minimiser la durée totale. Nous nous sommes concentrés particulièrement sur les cas où les tâches sont préemptées et transférées vers une autre machine. Les temps de changement de machine sont pris en compte, nous avons également proposé trois heuristiques et les appliquons sur un exemple.

Chapitre 4

Expérimentations numériques

4.1 Introduction

Dans ce chapitre, nous exposons notre contribution à l'implémentation du problème $Pm/pmtn, c_{j,j'}/C_{\max}$. Nous comparons aussi des résultats numériques obtenus en testant nos heuristiques proposées dans le troisième chapitre sur des exemples aléatoires en utilisant le langage de programmation Python.

4.1 Langage Python

Python est un langage de programmation généraliste. Il permet d'écrire des applications de types variés et de programmer suivant le paradigme impératif (comme en C) ainsi que suivant le paradigme orienté objets (comme en C++ et Java). La richesse conceptuelle de Python facilite la transition vers d'autres langages comme C, C++ ou Java. Python est le langage informatique le plus populaire pour le traitement de Big Data, l'exécution de calculs mathématiques et le Machine Learning.

Le choix de Python tient à plusieurs raisons :

- C'est un langage de haut niveau : les considérations techniques, notamment la gestion de la mémoire, sont gérées automatiquement, ce qui permet un développement plus naturel, rapide et moins sujet aux erreurs.

- Simple et néanmoins puissant et rapide, il a un fonctionnement interactif facilitant l'apprentissage (rappelant MATLAB).
- Il dispose de nombreux modules très élaborés pour des usages plus spécialisés, notamment, en ce qui nous concerne, en calcul scientifique (outils et méthodes numériques, représentations graphiques), analyse de données, analyse d'images, intelligence artificielle, etc [38].

4.2 Implémentation des heuristiques proposées sur Python

Pour programmer nos heuristiques, nous avons utilisé le langage Python. L'algorithme H1 a été implémenté avec une procédure de base. Pour H2, nous avons ajouté une procédure de rangement à l'algorithme de H1. H3 reprend les principes de H2 en y ajoutant la règle LPT. Ainsi, H2 et H3 sont des extensions de H1, intégrant respectivement des procédures de rangement et la règle LPT. Voici les algorithmes et leurs exécutions, permettant d'évaluer leurs performances comparatives.

4.3 Exécution des algorithmes

4.3.1 Algorithme de la règle LPT

```
Entrez le nombre de tâches : 20

Temps de traitement des tâches :
[20, 20, 16, 4, 10, 8, 17, 21, 15, 17, 24, 15, 29, 26, 23, 7, 4, 22, 20, 12]

Temps de traitement des tâches (triés selon la règle LPT) :
[29, 26, 24, 23, 22, 21, 20, 20, 20, 17, 17, 16, 15, 15, 12, 10, 8, 7, 4, 4]
```

Figure 4.1 – Exemple d'exécution de l'algorithme de la règle LPT

4.3.2 Algorithme de l'heuristique H1

```
Entrez le nombre de tâches : 20
Entrez le nombre de machines : 5
Temps de traitement des tâches :
[18  8 14  2 24 20 20 24  6 11 14 22  6  9 27  6 15 14  3 15]
Temps de changement de machines :
[[ 0 36 48  8 37]
 [32  0 32 38 23]
 [41 28  0 35  6]
 [11 50 33  0  4]
 [50 15 33 26  0]]
La borne inférieure B est : 56
Cmax obtenu : 56
```

Figure 4.2 – Exemple d'exécution de l'algorithme H1

4.3.3 Algorithme de la procédure RANGEMENT

```
Entrez le nombre de machines : 12
Matrice des temps de changement de machines (avant traitement) :
[[ 0 10 45 18 16 42 49 36 14 32 45  7]
 [ 9  0  6 17 44  2 19 36  9 44  3 10]
 [ 3 20  0 35 37  9 35 42 34 23 42 39]
 [ 1  8 48  0 49 46 29 44 16 41 30 17]
 [30 16 47 37  0  2 42 18 26  5 22 25]
 [ 1 16 26 15 42  0  1 22 35 41 46 14]
 [49 40  3 23 43 44  0 41 21  1 40 35]
 [44 34 47  7  3 49 24  0 20  7 46 30]
 [46 14 13 18  2 10 15 40  0 12 45 15]
 [29  6  6 23 19 45 46 43  5  0  8  4]
 [10 23 47 37 22 32  3 34 30 40  0  3]
 [46 18 25 33 39 43 39 50 39 19 38  0]]

Séquence des machines rangées : [1, 4, 8, 6, 2, 10, 9, 5, 7, 3, 12, 11]
Valeurs minimales correspondantes : [1, 7, 18, 2, 6, 1, 16, 2, 1, 25, 3]
```

Figure 4.3 – Exemple d'exécution de l'algorithme de la procédure RANGEMENT

4.4 Déroulement des tests

Pour tester les heuristiques proposées $H1$, $H2$ et $H3$, nous procèdons comme suit :

1. **Génération d'instances :** Pour chaque combinaison de m prenant l'une des valeurs suivantes : $\{5, 10, 15\}$ et n prenant l'une des valeurs suivantes : $\{10, 15, 20, 35, 50, 75, 100\}$, nous génèrons une instance aléatoire avec des durées opératoires et des temps de changement de machine tirés uniformément dans $[1, 30]$ et $[1, 50]$ respectivement.
2. **Résolution :** Nous résolvons l'instance avec $H1$, puis $H2$ et enfin $H3$.
3. **Évaluation :** Pour chaque heuristique, nous calculons la borne inférieure $\overline{\overline{B}}$ et nous trouvons le temps de fin de traitement "$C_{\max}$" du problème $Pm/pmtn, c_{j,j'}/C_{\max}$.

   Les résultats obtenus sont présentés dans le tableau suivant. La première ligne indique le nombre de machines, la deuxième ligne le nombre de tâches, et les lignes suivantes montrent la borne inférieure $\overline{\overline{B}}$ ainsi que le temps de fin de traitement $C_{\max}$ trouvé par chaque heuristique.

m	5			10			15		
n	10	15	20	20	35	50	50	75	100
$\overline{\overline{B}}$	30	53	53	26	45	75	48	85	93
$H1$	60	70	53	74	58	75	72	85	93
$H2$	38	58	53	46	45	75	48	85	93
$H3$	30	53	53	35	45	75	48	85	93

Tableau 4.1 – Résultats des tests

4.5 Comparaison des performances des heuristiques

Nous comparons les $C_{\max}$ obtenus par chaque heuristique par rapport à la borne inférieure $\overline{\overline{B}}$ pour déterminer laquelle est la plus efficace. Pour cela nous allons calculer le ratio d'approximation R avec : $R = \frac{C_{\max}(H)}{\overline{\overline{B}}}$

Lorsque R tend vers 1 alors l'approximation est de bonne qualité. Où $C_{\max}(H)$ est la longueur d'ordonnancement obtenue à partir de l'heuristique et la borne inférieure $\overline{\overline{B}} = \max\left\{\max\left\{\max\limits_{1\leq i\leq n}\{p_i\}, \left\lceil\frac{1}{m}\sum\limits_{i=1}^{n}p_i\right\rceil\right\}, \min\limits_{1\leq i\leq n}\{p_i\} + \min\limits_{j,j' \text{et } j\neq j'}\{c_{j,j'}\}\right\}$

Les résultats obtenus sont affichés dans le tableau ci-dessous :

m	5			10			15		
n	10	15	20	20	35	50	50	75	100
$\overline{\overline{B}}$	30	53	53	26	45	75	48	85	93
$R1 = \frac{C_{\max}(H1)}{\overline{\overline{B}}}$	2	1.3207	1	2.8461	1.2888	1	1.5	1	1
$R2 = \frac{C_{\max}(H2)}{\overline{\overline{B}}}$	1.2667	1.0943	1	1.7692	1	1	1	1	1
$R3 = \frac{C_{\max}(H3)}{\overline{\overline{B}}}$	1	1	1	1.3461	1	1	1	1	1

Tableau 4.2 – Calcul du ratio d'approximation R

Examinons les performances des trois heuristiques. Il est notable que la première heuristique $H1$ produit des solutions moins satisfaisantes lorsque le nombre de tâches n est faible par rapport au nombre de machines m, en raison de décalages importants qui allongent les temps morts. Pour pallier cela, il serait pertinent d'interdire la préemption des tâches nécessitant plus de temps de transport que de traitement sur une même machine, ce qui améliorerait le processus.

En comparant les heuristiques $H2$ et $H3$, il est évident que les résultats obtenus avec $H3$ se rapprochent davantage de l'optimal. En effet, cette heuristique, basée sur l'ordonnancement des tâches selon la règle LPT avec des machines classées, génère

généralement des solutions efficaces pour les problèmes de petite taille. Quant aux problèmes de taille plus importante, ils peuvent être résolus avec l'heuristique $H2$.

Un autre constat récurrent pour les trois heuristiques est que la borne inférieure est systématiquement atteinte. Cela découle du fait que lorsque le nombre de tâches est considérable, cela entraîne une augmentation de la moyenne du nombre de tâches par machine.

En résumé, $H3$ est privilégiée car elle s'appuie sur un ordonnancement des tâches permettant une meilleure performance. Elle commence par traiter en priorité les tâches de longue durée, et même si elles sont interrompues, leur transport est peu chronophage car les machines sont classées par temps de transport minimal. Les tâches plus courtes sont traitées en dernier, ce qui réduit la nécessité de préemption.

4.6 Conclusion

Ce chapitre porte sur l'évaluation numérique de la performance des heuristiques développées au chapitre précédent. Après avoir programmé et testé les trois heuristiques en utilisant Python, nous avons analysé les résultats obtenus à partir de diverses instances de test. Chaque heuristique présente ses propres forces et faiblesses, donc elles sont incomparables.

Conclusion générale

La théorie de l'ordonnancement, au cœur de la recherche opérationnelle, offre des outils essentiels pour la prise de décision dans divers domaines, de l'industrie à l'économie. à travers ce mémoire, intitulé "Contribution à l'étude de certains problèmes d'ordonnancement préemptif", nous nous sommes plongés dans l'étude des problèmes d'ordonnancement préemptif, une méthode dynamique de gestion des tâches.

Le premier chapitre présente les bases en présentant les concepts fondamentaux de l'ordonnancement, la complexité des problèmes d'optimisation combinatoire, et leur classification. Cette exploration initiale a posé le cadre nécessaire à la compréhension des problèmes à venir.

Le deuxième chapitre a approfondi notre compréhension en examinant divers problèmes d'ordonnancement préemptif, mettant en lumière leur complexité et proposant des modèles pour leur résolution. Des exemples concrets ont illustré les enjeux et les méthodes envisagées pour aborder ces problèmes.

Le troisième chapitre a porté la proposition d'heuristiques pour résoudre le problème spécifique $Pm/pmtn, c_{j,j'}/C_{\max}$. Ces heuristiques ont été développées avec une approche pragmatique, offrant des solutions pratiques et efficientes.

Enfin, le quatrième chapitre a mené des expérimentations numériques pour évaluer la performance des heuristiques proposées. Ces tests ont permis une analyse

approfondie des résultats obtenus, mettant en évidence les forces et les limites de chaque approche.

La conclusion générale de ce travail offre une récapitulation des résultats obtenus, espérant qu'il a une importance et contribue à la théorie de l'ordonnancement préemptif. En fournissant des outils concrets pour la résolution de problèmes complexes.

Ce mémoire ouvre la voie à de nouvelles recherches et à des applications pratiques dans divers domaines industriels à savoir la proposition d'autres heuristiques sur le problème traité ou sur d'autres problèmes $\mathcal{NP}$-difficile.

Références

[1] **A. Abdelaziz, A. Messai**, "*Aide à la décision pour la résolution du problème d'ordonnancement Job shop sous contrainte de ressources consomables*", Mémoire de Fin d'Études de Master en Génie Industriel université Abou Bekr Belkaid de Tlemcen, 2017.

[2] **L. Andrieu**, **"***La Recherche Opérationnelle en France : La science du management quantitatif",* livre réalisé par la Société Française de Recherche Opérationnelle et d'Aide à la Décision ROADEF, 2011.

[3] **M. Bachir**, "*Ordonnancement tolérant aux fautes pour les systèmes distribués temps réel embarqués*", Thèse de Doctorat en Informatique université de Batna 2, 2019.

[4] **A. Bechir**, "*Résolution de problèmes d'optimisation par les systèmes multi-agents et les approches évolutionnaires*", Mémoire de Fin d'Études de Magister en Informatique université Mohamed khider de Biskra, 2016.

[5] **N. Belkhira,I. Tabani**, "*Gestion d'un processus industriel au niveau de CIT Tiaret*", Mémoire de Fin d'Études de Master en Génie Mécanique : Production université Ibn-Khaldoun de Tiaret, 2016.

[6] **M. Belkhous, S. Bourahla**, "*Modélisation d'un processus de fabrication et résolution d'un problème d'ordonnancement de type Flow-Shop. Cas «ENIEM»*",

Mémoire de Fin d'Études de Master en Mathématiques Appliquées à la Gestion université Mouloud Mammeri de Tizi-Ouzou, 2017.

[7] **A. Ben Hmida**, "*Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible*", Automatique/Robotique, INSA de Toulouse, 2009.

[8] **A. Benkaci**, Note de cours "*Problèmes d'optimisation combinatoire*" $2^{ème}$ année Master Recherche Opérationelle univesité Yahia Fares de Médéa, 2016.

[9] **I. Benkalai**, "*Ordonnancement d'ateliers en présence d'opérateurs*", Thèse de Doctorat en Informatique université du Québec à Chicoutimi, 2018.

[10] **T. Bentrcia**, "*Ordonnancement des systèmes de production sous contraintes technologiques ou contextuelles : Modélisation, étude de complexité et approches intégrées de résolution*", Thèse de Doctorat en Génie Industriel université de Batna 2, 2017.

[11] **M. Boukendou**, "*Contribution à l'application de la robustesse et de la flexibilité en ordonnancement*", Mémoire de Fin d'Études de Magistère en Recherche Opérationnelle université Saad Dahleb de Blida, 2010.

[12] **P. Brucker**, "*Complex scheduling*", édition Spring, 2006.

[13] **J. Carlier et al**, "*Les problèmes d'ordonnancement*", par le groupe GOThA RAIRO-Operations Research, Tome 27 n° 1 ,1993.

[14] **J. Carlier, P. Chrétienne**, "*Un domaine très ouvert : les problèmes d'ordonnancement* ", RAIRO-Operations Research, Tome 16 n° 3 ,1982.

[15] **A. Charles**, leçons 5 du cours "*Modèles d'aide à la décision*", univesité de Paris-dauphine, 2008.

[16] **A. Chergui, A. Dahmani, A. Adda Abbou**, "*Ordonnancement d'un flow-shop par métaheuristique hybride*", Mémoire de Fin d'Études de Master en Génie industriel université Abou Bekr Belkaid de Tlemcen, 2017.

[17] **C. Chu**, "*A branch and bound algorithm to minimize total tardiness with different release dates*", Naval Research Logistics 39 :265–283, 1992.

[18] **S.A. Cook**, "*The complexity of theorem proving procedures*", In Association of Computing Machinery (pp. 151–158) New York : Proceedings of the third annual ACM symposium on the theory of computing, 1971.

[19] **S. Djellouli**, "*Approche par coopération inter machine pour un ordonnancement distribué*", Mémoire de Fin d'Études de Magistère en Automatique École Nationale Polytechnique, 2007.

[20] **M. El-Mansouri**, "*Le jeu vidéo didactique ou serious game : processus de conception, ingénierie didactique et game design*", Thèse de Doctorat en Sciences de l'Education université Côte d'Azur, 2019.

[21] **P. Esquirol, P. Lopez**, "*L'ordonnancement*", Economia, Paris, 1999.

[22] **C. Esswein**, "*Un apport de flexibilité séquentielle pour l'ordonnancement robuste*", Thèse de Doctorat université François Rabelais Tours, 2003.

[23] **M.R. Garey, D.S. Johnson**, "*Computers and intractability : A guide of the theory of $\mathcal{NP}$-completeness*", San Fransisco : W.H. freeman and Company, 1979.

[24] **R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H. Rinnooy Kan**, "*Optimization and approximation in deterministic sequencing and scheduling : a survey*", Annals of Discrete Mathematics 5 287-326, 1979.

[25] **M. Habib-Zahmani**, "*Contribution à l'ordonnancement dynamique : proposition d'une approche guidée par effet de Simulation/Datamining*", Thèse de Doctorat en Informatique :Apprentissage Automatique et Web Intelligence université Abdelhamid Ibn Badis de Mostaganem, 2018.

[26] **A. Haned, A. Soukhal, M. Boudhar, N. Huynh Tuong**, "*Scheduling on parallel machines with preemption and transportation delays*", Computers and Operations Research 39 (2) pp. 374-381, 2012.

[27] **J. Kaabi-Harrath**, "*Contribution à l'ordonnancement des activités de maintenance dans les systèmes de production*", Thèse de Doctorat en Automatique et Informatique université de Franche-Comté, 2004.

[28] **S. Kouider Aiad**, Note de cours "*Formulation des problèmes d'ordonnancement*", $2^{ème}$ année Master Recherche Opérationelle univesité Yahia Fares de Médéa, 2023.

[29] **J. Lageweg, J.K. Lenstra, A.H. Rinnooy Kan**, "*Minimizing maximum lateness on one machine : computational experience and some application*", Statistica Neerlandica 30 25-41, 1976.

[30] **I. Laribi**, chapitre 2 du cours "*Ordonnancement de la production*", $2^{ème}$ année Licence Génie Industriel université de Tlemcen, 2019.

[31] **P. Lopez**, **F. Roubellat**, "*Ordonnancement de la production*", Paris Hermès Sciences, IC2 Production, 2001.

[32] **S. Mammeri**, "*Problème à acheminement multiple dans un environnement dynamique*", Mémoire de Fin d'Études de Master en Recherche Opérationnelle université Saad Dahleb de Blida, 2020.

[33] **R. Mc Naughton**, "*Scheduling with deadlines and loss functions"*, Management Science volume 6, 1959.

[34] **N. Moussouni, I. Ouzzar**, "*Heuristique pour la minimisation de la variance des dates de fin d'exécution des tâches dans les problèmes d'ordonnancement à une seule machine*", Mémoire de Fin d'Études de Master en Recherche Opérationnelle univesité Yahia Fares de Médéa, 2022.

[35] **M. Pinedo**, "*Scheduling-Theory, Algorithms, and Systems*", Prentice Hall, Englewood Cliffs, 1995.

[36] **M. Shahzad**, "*Une approche hybride de simulation-optimisation basée sur la fouille de données pour les problèmes d'ordonnancement*", Thèse de Doctorat en Automatique et Génie Informatique université de Nantes, 2011.

[37] **E. Talbi**, "*Sélection et réglage de paramètres pour l'optimisation de logiciels d'ordonnancement industriel*", Thèse de Doctorat en Informatique Industrielle Institut National Polytechnique de Toulouse, 2004.

[38] **L. Tournier**, Note de cours "*Introduction à Python pour les mathématiques*", $1^{ère}$ année Master Mathématiques univesité Sorbone Paris Nord, 2019.