République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Yahia Farès de Médéa

Faculté des Sciences
Département des Mathématiques et Informatique

M É M O I R E

Présenté en vue de l'obtention du

Diplôme de Master en Mathématiques

Spécialité : Recherche opérationnelle

Par

BEY Mustapha Abd Elmadjid & **BOUKENDAKDJI Kenza**

Thème

Problème de minimisation de la longueur d'ordonnancement sur machines parallèles identiques

Devant le jury composé de :

M*me*	K. SERIER ABDELLAH	Maitre assistant	U.D.Y.F-Médéa	Présidente
M*r*	K. TABLENNEHAS	Maitre assistant	U.D.Y.F-Médéa	Examinateur 1
M*r*	A. SAHRAOUI	Maitre assistant	U.D.Y.F-Médéa	Examinateur 2
M*me*	S. KOUIDER AIAD	Maitre assistant	U.D.Y.F-Médéa	Promotrice

Date de soutenance 07 juillet 2019

Remerciement

Tout d'abord, nous remercions Dieu le tout puissant, pour la volonté, la santé, la patience et le courage qu'il nous a offert durant la réalisation de ce modeste travail de recherche ...

Nous tenons tout particulièrement à remercier et à exprimer notre profonde gratitude à notre promotrice madame KOUIDER AIAD Soumia pour sa générosité, ses précieux conseils, sa disponibilité et son encadrement fort bénéfique ...

Nous tenons à exprimer également nos plus vifs remerciements et toute ma gratitude à

***Madame** K. SERIER ABDELLAH Maître assistant à l'Université Dr. Yahia Farès de Médéa, de m'avoir fait l'honneur de présider le jury de ce mémoire ...*

***Monsieur** K. TABLENNEHAS, Maître assistant à l'Université Dr. Yahia Farès de Médéa, pour avoir accepté de participer au jury de ce mémoire et pour sa disponibilité, son aide et ses précieux conseils tout au long de ce travail ...*

***Monsieur** A. SAHRAOUI, Maître assistant à l'Université Dr. Yahia Farès de Médéa, pour avoir accepté de participer au jury de ce mémoire...*

Nos remerciements vont également à l'ensemble de nos enseignants de master recherche opérationnelle pour leurs efforts fournis durant notre formation universitaire ...

Toutes nos infinies gratitudes à tout l'ensemble du jury qui nous ont honorés de leur présence pour juger notre travail ...

Enfin nous devons remercier si particulièrement nos très chers parents pour leur soutien et l'encouragement qu'ils nous ont donné afin de continuer nos études et nous remercions tous ceux qui de près ou de loin ont contribué à l'élaboration de ce mémoire ...

Dédicace

Avec l'expression de ma reconnaissance je dédie ce mémoire à tous ceux qui me sont chers

À mon cher père

Qui est toujours à mes côtés pour me soutenir et m'encourager. Que ce travail traduit ma gratitude et mon affection, ma précieuse offre du dieu, qui doit ma vie, ma réussite et tout mon respect. Puisse Dieu vous donne santé, bonheur, courage et surtout réussite ...

À mon adorable mère

Je ne saurai point te remercier comme il se doit. Ton affection me couvre, ta bienveillance me guide et ta présence à mes côtés a toujours été ma source de force pour affronter les différents obstacles ...

À mes chers parents

Qui n'ont jamais cessé, de formuler des prières à mon égard, de me soutenir et de m'épauler pour que je puisse atteindre mes objectifs ...

À ma chère sœur

Qui n'a pas cessée de me conseiller, encourager et soutenir tout au long de mes études. Que Dieu la protège et leurs offre la chance et le bonheur...

À mes chers frères

Qui ont partagé avec moi tous les moments d'émotion lors de la réalisation de ce travail. Ils m'ont chaleureusement supporté et encouragé tout au long de mon parcours ...

À mon binôme

Qui a collaboré efficacement pour la réalisation de notre mémoire ...

À tous mes enseignants du notre section

Qui ont contribué à notre formation ...

Mustapha...

Dédicace

Du profond de mon cœur je dédie ce modeste travail à tous ceux qui me sont chers

À mes chers parents

Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que vous avez consenti pour mon instruction et mon bien-être .je vous remercie pour tout le soutien et l'amour que vous me portez depuis mon enfance et j'espère que votre bénédiction m'accompagnera toujours et que cet humble travail soit l'exaucement de vos vœux et le fruit de vos maintes sacrifices ...

Puisse Dieu, le très haut vous accorde santé, bonheur et longue vie et faire en sorte que jamais je ne vous déçoive ...

À mon cher et unique frère

Merci énormément pour ton soutien plus que précieux, pour ton grand cœur, supportant mon impatiente et mes sautes d'humeurs sans jamais te lasser...

À toute ma famille

Pour leur soutien tout au long de mon parcours ...

À mon binôme

Qui a collaboré efficacement pour la réalisation de notre mémoire ...

À mes enseignants mes amies et amis

Qui n'ont cessé de m'encourager lors de mon parcours ...

Kenza...

Résumé

Dans ce travail, nous nous intéressons au problème d'ordonnancement de tâches indépendantes sur des machines parallèles identiques pour minimiser la date de fin de traitement de l'ensemble des tâches et implémenter des heuristiques proposées.

Mots clés : Ordonnancement, machines parallèles identiques, complexité.

Abstract

In this work, we are interested in the problems of scheduling independent tasks on identical parallel machines to minimize the end date of processing of all jobs and implement proposed heuristics.

Keywords: Scheduling, identical parallel machines, complexity.

ملخـــص

وفي هذه المذكرة سوف نهتم بمشاكـل الجدولـة حيث تكون القيود فيها مستقلة على آلات مطابقة ومتشابهة و لها نفس سرعة التنفيذ والهدف من هذه الدراسة هو الاقتصـاد في وقـت إنهـاء عمليات تنفيذ المهام و تطبيق استدلالات مقترحـة.

كلمـات مفتاحيـة : الجدولـــــة, آلات مطابقـــــة و متشابهـــة, التعقيـــــد.

Table des matières

Introduction générale **1**

1 Généralité sur l'ordonnancement **4**

1.1 **Introduction** . . . . . T

1.2 **Les données d'un problème d'ordonnancement** . . . . . T

1.2.1 **Formulation d'un problème d'ordonnancement** . . . . . T

1.3 **Caractérisation des machines** X

1.3.1 **Les types de machines** X

1.3.2 **Notation générale** $(\alpha/\beta/\gamma)$ . . . . . Y

1.4 **Classification des problèmes d'ordonnancement** . . . . . ŁŁŁQR

1.4.1 **Modèle à une opération** . . . . . ŁŁŁQR

1.4.2 **Modèle à plusieurs opérations** . . . . . ŁŁŁQS

1.5 **Représentation des problèmes d'ordonnancement sur les machines parallèles** . . . . . ŁŁŁQU

1.5.1 **Diagramme de Gantt** . . . . . ŁŁŁQU

1.5.2 **Graphe potentiel-tâches** . . . . . ŁŁŁQV

1.5.3 **Méthode PERT** . . . . . ŁŁŁQW

1.5.4 **Les anomalies de GRAHAM** . . . . . ŁŁŁQY

1.6 **La complexité des problèmes d'ordonnancement** . . . . . ŁŁŁRQ

1.6.1 **Notion de complexité de problèmes** . . . . . ŁŁŁRR

1.6.2 **Problème de Décision** . . . . . ŁŁŁRR

1.6.3 **Problème d'optimisation combinatoire** . . . . . ŁŁŁRS

1.6.4 **Classes de complexité** . . . . . ŁŁŁRS
1.6.5 **Réduction entre problèmes d'ordonnancement** . . . . . ŁŁŁRU
1.7 **Conclusion** . . . . . ŁŁŁRV

2 Les méthodes de résolution **28**
2.1 **Introduction** . . . . . ŁŁŁRX
2.2 **Les méthodes exactes** . . . . . ŁŁŁRX
2.2.1 **La méthode de Branch and Bound** . . . . . ŁŁŁRY
2.2.2 **La programmation dynamique** . . . . . ŁŁŁSQ
2.2.3 **La programmation par contrainte** . . . . . ŁŁŁSS
2.2.4 **La programmation linéaire** . . . . . ŁŁŁST
2.3 **Les méthodes approchées** . . . . . ŁŁŁSU
2.3.1 **Les heuristiques** . . . . . ŁŁŁSU
2.3.2 **Les méta-heuristiques** . . . . . ŁŁŁSU
2.4 **Les méta-heuristiques à solution unique** . . . . . ŁŁŁSW
2.4.1 **La méthode de descente** . . . . . ŁŁŁSW
2.4.2 **Le recuit simulé** . . . . . ŁŁŁSW
2.4.3 **La recherche TABOU** . . . . . ŁŁŁTP
2.5 **Les méta-heuristiques à population de solutions** . . . . . ŁŁŁTQ
2.5.1 **Les algorithmes génétiques** . . . . . ŁŁŁTQ
2.5.2 **L'optimisation par colonie de fourmis** . . . . . ŁŁŁTR
2.6 **Les avantages et les inconvénients des heuristiques et méta-heuristiques45**
2.6.1 **Les avantages des heuristiques** . . . . . ŁŁŁTU
2.6.2 **Les inconvénients des heuristiques** . . . . . ŁŁŁTU
2.6.3 **Les avantages des méta-heuristiques** . . . . . ŁŁŁTU
2.6.4 **Les inconvénients des méta-heuristiques** . . . . . ŁŁŁTU
2.7 **La comparaison entre les méthodes exactes et les méthodes approchées** . . . . . ŁŁŁTV
2.8 **Conclusion** . . . . . ŁŁŁTV

3 La modélisation des problèmes d'ordonnancement sur les machines parallèles identiques **48**

3.1 **Introduction** . . . . . .ŁŁŁTX

3.2 **Les problèmes d'ordonnancement sur des machines parallèles identiques** . . . . . .ŁŁŁTX

3.2.1 **Le Makespan ($\mathbf{C_{max}}$)** . . . . . .ŁŁŁTX

3.2.2 **La modélisation** . . . . . .ŁŁŁTY

3.2.3 **La formulation** . . . . . .ŁŁŁTY

3.2.4 **Les problèmes mono-objectifs** . . . . . .ŁŁŁTY

3.3 **Minimisation du Makespan avec l'ordonnancement de type préemptif** **49**

3.3.1 **Le problème ($\mathbf{P/Pmtn/C_{max}}$)** . . . . . .ŁŁŁTY

3.4 **Minimisation du Makespan avec l'ordonnancement de type non-préemptif** . . . . . .ŁŁŁUR

3.4.1 **Le problème ($\mathbf{P//C_{max}}$)** . . . . . .ŁŁŁUR

3.4.2 **Le sous problème ($\mathbf{Pm//C_{max}}$) et sa difficulté** . . . . . .ŁŁŁUT

3.4.3 **Le sous problème ($\mathbf{P2//C_{max}}$) et sa difficulté** . . . . . .ŁŁŁUW

3.4.4 **Le problème ($\mathbf{Pm/split, S_i, r_i\ /C_{max}}$)** . . . . . .ŁŁŁUX

3.4.5 **Le problème ($\mathbf{Pm/S_i, k/C_{max}}$)** . . . . . .ŁŁŁVR

3.4.6 **Le sous problème ($\mathbf{P2/p_i = p, S_i, k = 2/C_{max}}$) et sa difficulté** .ŁŁŁVS

3.4.7 **Le sous problème ($\mathbf{P2/S_i = s, k = 2/C_{max}}$) et sa difficulté** . . .ŁŁŁVT

3.4.8 **Le sous problème ($\mathbf{P2/p_i = p, S_i = s, k \leq 2/C_{max}}$) et sa difficulté** **65**

3.5 **Minimisation le Makespan dans l'ordonnancement du graphe de précédence** . . . . . .ŁŁŁVW

3.5.1 **Le problème ($\mathbf{Pm/prec/C_{max}}$)** . . . . . .ŁŁŁVW

3.5.2 **Le problème ($\mathbf{Pm/prec,\widetilde{d_i}\ /C_{max}}$)** . . . . . .ŁŁŁVY

3.6 **Conclusion** . . . . . .ŁŁŁWP

4 Implémentation du problème ($\mathbf{P3//C_{max}}$) **72**

4.1 **Introduction** . . . . . .ŁŁŁWR

4.2 **MATLAB** . . . . . .ŁŁŁWR

4.3 **Définition du problème ($\mathbf{P3//C_{max}}$)** . . . . . .ŁŁŁWS

4.4 **Les heuristiques proposées** . . . . . ŁŁŁWS
4.4.1 **Principe de la 1ère heuristique** . . . . . ŁŁŁWT
4.4.2 **Principe de la 2ème heuristique** . . . . . ŁŁŁWT
4.4.3 **Principe de la 3ème heuristique** . . . . . ŁŁŁWU
4.5 **Exemple d'application** . . . . . ŁŁŁWV
4.5.1 **Application de la 1ère heuristique** . . . . . ŁŁŁWV
4.5.2 **Application de la 2ème heuristique** . . . . . ŁŁŁWW
4.5.3 **Application de la 3ème heuristique** . . . . . ŁŁŁWX
4.6 **Implémentation des heuristiques sur MATLAB** . . . . . ŁŁŁWX
4.7 **Comparaison des résultats obtenus par les trois heuristiques** . . . ŁŁŁXR
4.8 **Conclusion** . . . . . ŁŁŁXS

Conclusion générale **85**

Bibliographie **88**

Liste des tableaux

1.1 Données utilisées pour la réalisation d'un graphe potentiel-tâches 2
1.2 Les données de l'exemple 2
1.3 Les données du problème 2
1.4 Tableau représentatif de la désignation de la complexité algorithmique courante 3
1.5 Tableau représentatif de la complexité de quelques problèmes d'ordonnancement 3

2.1 Données du problème 5
2.2 Table de comparaison entre méthode exacte et méthode approchée 7

3.1 Temps de traitement des tâches de l'exemple 3.3.1 11
3.2 Temps de traitement des tâches de l'exemple 3.4.1 11
3.3 La complexité de quelques problèmes d'ordonnancement 11

4.1 La table des tâches et leurs temps de traitement 14
4.2 La table des machines et la position des tâches pour H1 14
4.3 La table des machines et la position des tâches pour H2 14
4.4 La table des résultats 14

Liste des Figures

1.2.1 Schéma représentant l'exécution d'une tâche . . . . . U
1.2.2 Classification des ressources V
1.2.3 Quelques critères d'évaluation X
1.3.1 Schéma représentant les types de machines . . . . . Y
1.4.1 Modèle à machine unique . . . . . ŁŁŁQR
1.4.2 Modèle à machines parallèles . . . . . ŁŁŁQS
1.4.3 Modèle flow-shop . . . . . ŁŁŁQT
1.4.4 Modèle job-shop . . . . . ŁŁŁQT
1.4.5 Modèle open-shop . . . . . ŁŁŁQU
1.5.1 Diagramme de Gantt d'un ordonnancement . . . . . ŁŁŁQV
1.5.2 Graphe Potentiel-Tâches d'un ordonnancement . . . . . ŁŁŁQW
1.5.3 Réseau PERT . . . . . ŁŁŁQY
1.5.4 Graphe des contraintes de précédence . . . . . ŁŁŁQY
1.5.5 Ordonnancement d'un P2/ prec/C_{max} $< ListeL >$ . . . . . ŁŁŁRP
1.5.6 Ordonnancement d'un P2/ prec/C_{max} $< ListeL' >$ . . . . . ŁŁŁRP
1.5.7 Ordonnancement de même problème avec réduction des temps d'exécution ŁŁŁRP
1.5.8 Ordonnancement de même problème avec augmentation du nombre de machines . . . . . ŁŁŁRP
1.5.9 Graphe des contraintes de précédence affaiblies . . . . . ŁŁŁRQ
1.5.10 Ordonnancement obtenu avec contraintes de précédence affaiblies . . . . ŁŁŁRQ
1.6.1 Schéma représentatif de classes de complexité . . . . . ŁŁŁRT
1.6.2 Réduction des machines . . . . . ŁŁŁRV

1.6.3 Réduction entre les critères . . . . . .ŁŁŁRV
1.6.4 Relation de précédence . . . . . .ŁŁŁRV
1.6.5 Quelques réductions . . . . . .ŁŁŁRV

2.2.1 Arborescence de séparations pour $1//\sum(W_i(C_i - d_i))$ . . . . . .ŁŁŁSP
2.2.2 Ordonnancement optimal de l'exemple 2.2.2 . . . . . .ŁŁŁSS
2.3.1 Classification des méthodes de résolution des problèmes d'ordonnancement .ŁŁŁSV
2.4.1 Organigramme relatif au fonctionnement général du recuit simulé . . . . .ŁŁŁSY
2.5.1 Déplacement des fourmis vers une source de nourriture . . . . . .ŁŁŁTR
2.5.2 Déplacement des fourmis après placement d'un obstacle sur leur chemin . . .ŁŁŁTS
2.5.3 Choix du chemin le plus court par la plupart des fourmis . . . . . .ŁŁŁTS
2.5.4 Classification des méta-heuristiques . . . . . .ŁŁŁTT

3.3.1 Ordonnancement optimal pour les tâches de l'exemple . . . . . .ŁŁŁUR
3.4.1 Ordonnancement au pire des cas de LPT avec $C_{max}(OPT) = 12$ . . . . . .ŁŁŁUW
3.4.2 Ordonnancement au pire des cas de LPT avec $C_{max}(LPT) = 15$ . . . . . .ŁŁŁUW
3.4.3 Correspondance entre 2- partitions et 2- machines . . . . . .ŁŁŁUX
3.4.4 Exemple de $(Pm/split, S_i, r_i/C_{max})$ . . . . . .ŁŁŁUY

4.2.1 L'interface de démarrage du logiciel MATLAB . . . . . .ŁŁŁWS
4.4.1 Schéma représentant l'affectation des tâches selon la 1^{re} heuristique . . . .ŁŁŁWT
4.4.2 Schéma représentant l'affectation des tâches selon la 2^{me} heuristique . . . .ŁŁŁWU
4.5.1 Solution générée de l'exemple par l'heuristique H1 . . . . . .ŁŁŁWV
4.5.2 Solution générée de l'exemple par l'heuristique H2 . . . . . .ŁŁŁWW
4.5.3 Solution générée de l'exemple par l'heuristique H3 . . . . . .ŁŁŁWX
4.6.1 La saisie et le rangement de temps de traitement des tâches . . . . . .ŁŁŁWY
4.6.2 La démarche du diagramme de Gantt . . . . . .ŁŁŁWY
4.6.3 La longueur d'ordonnancement C_{max} . . . . . .ŁŁŁXP
4.6.4 Les positions et les machines qui exécutent chaque tâche . . . . . .ŁŁŁXP
4.6.5 Solution générée par l'heuristique H2 . . . . . .ŁŁŁXQ
4.6.6 Les positions et les machines qui exécutent chaque tâche . . . . . .ŁŁŁXQ
4.6.7 Les valeurs de C_{max} dans chaque étape . . . . . .ŁŁŁXR

Liste des abréviations

SEP : *Séparation et Évaluation Progressive.*

PERT : *Program Evaluation and Review Technique.*

P : *Polynomial.*

NP : *Non déterministe Polynomial.*

PPC : *Programmation par Contraintes.*

PL : *Programmation Linéaire.*

PLNE : *Programmation Linéaire en Nombres Entiers.*

PLM : *Programmation Linéaire Mixte.*

POC : *Problème d'Optimisation Combinatoire.*

FIFO : *First In First Out.*

SPT : *Shortest Processing Time.*

LPT : *Longest Processing Time.*

OPT : *Optimized Production Technology.*

EDD : *Earliest Due Date.*

LLF : *Last Laxity First.*

HPF : *Hight Priority First.*

OCF : *Optimisation par Colonies de Fourmis.*

Introduction générale

Introduction générale

La recherche opérationnelle est la discipline des mathématiques appliquées qui traite des questions d'utilisation optimale des ressources dans l'industrie et dans le secteur public. Depuis une dizaine d'années, le champ d'application de la recherche opérationnelle s'est élargi à des domaines comme l'économétrie, la finance, le marketing et la planification d'entreprise. Plus récemment, elle a été utilisée pour la gestion des systèmes de santé et d'éducation, pour la résolution de problèmes environnementaux et dans d'autres domaines d'intérêt public.

Les problèmes d'ordonnancement d'ateliers constituent sûrement pour les entreprises une des difficultés les plus importantes de leurs systèmes de gestion et de pilotage de la production. En effet, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers.

Dans ce type d'ordonnancement, les ressources sont généralement des machines, et chaque travail à ordonnancer concerne un produit ou un lot de produits à fabriquer en respectant les gammes de fabrication. Nous trouvons une grande variété de problèmes d'ordonnancement qui sont liés à plusieurs paramètres : les tâches ou opérations (préemptives et non préemptives, indépendantes ou non), les paramètres relatifs aux ressources (renouvelables, consommables), types de contraintes sur les tâches (précédence, disjonctions), critères d'optimalité (minimisation de la durée totale d'achèvement de toutes les tâches 'Makespan', minimisation du retard total des tâches…etc).

Dans notre mémoire nous allons s'intéresser à la minimisation de la longueur d'ordonnancement makespan sur des machines parallèles identiques. Où nous donnons quelques problèmes qui sont ***NP***-difficiles et d'autres polynomiaux. Une méthode exacte basée sur la modélisation mathématique du problème et des heuristiques sont aussi présentées avec des résultats expérimentaux, basées sur des instances générées aléatoirement.

Ces expérimentations nous permettent d'apprécier l'efficacité des méthodes proposées. Le présent mémoire est organisé comme suit :

Dans le premier chapitre, les problèmes d'ordonnancement sont introduits, les types d'ateliers ainsi que les paramètres nécessaires pour caractériser un problème d'ordonnancement sont aussi évoqués. Et la complexité de la résolution de ces problèmes, est également abordée.

Le chapitre 2 a pour but d'introduire les différentes techniques de résolution des problèmes d'optimisation en général et en particuliers les problèmes d'ordonnancement, commençant par les méthodes exactes allant aux heuristiques et méta heuristiques, avec la présentation des principes et des algorithmes de certaines techniques.

Dans le troisième chapitre on s'intéresse à des modèles mathématiques d'un problème d'ordonnancement dont le but de minimiser la longueur d'ordonnancement sur des machines parallèles identiques.

Le quatrième chapitre présente notre programme implémenté en MATLAB. On a proposé trois heuristiques afin de traiter le problème d'ordonnancement $\boldsymbol{P3//C_{max}}$ qui est un problème difficile. Enfin, nous terminons par une conclusion générale, quelques remarques et perspectives.

Chapitre 1

Généralités sur l'ordonnancement

Chapitre 1

Généralités sur l'ordonnancement

1.1 Introduction

Dans ce chapitre, on s'intéresse aux problèmes d'ordonnancement dans les entreprises manufacturières. Il constitue un rappel de quelques notions de base relatives aux problèmes d'ordonnancement. Il rappelle aussi quelques concepts sur la théorie de la complexité des problèmes d'ordonnancement.

1.2 Les données d'un problème d'ordonnancement

1.2.1 Formulation d'un problème d'ordonnancement

Les différentes données d'un problème d'ordonnancement sont les tâches, les ressources, les contraintes et les critères. L'ordonnancement permet de programmer l'exécution d'un ensemble de tâches en attribuant des ressources aux tâches et en fixant leurs dates d'exécutions dans le but de satisfaire l'un des critères de performances tout en respectant un ensemble de contraintes [7].

Les tâches

Une tâche $\boldsymbol{T_i}$ est une entité élémentaire localisée dans le temps par la durée de préparation $\boldsymbol{S_i}$ et la date de fin de traitement $\boldsymbol{C_i}$ et le temps de traitement $\boldsymbol{p_i}$. Par ailleurs, les tâches nécessitent l'utilisation de ressources. Les tâches sont, le plus souvent, liées entre elles par des conditions diverses, nous disons alors que les tâches sont dépendantes. Dans le cas contraire, elles sont dites indépendantes. Dans certains problèmes, les tâches peuvent être interruptible 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.

Nous parlons respectivement de problème préemptif et non préemptif.

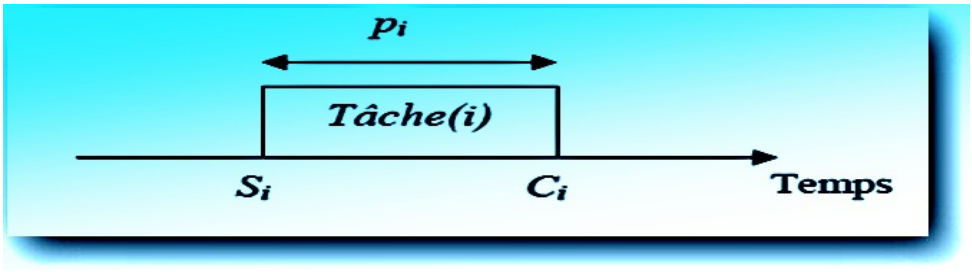


Figure 1.2.1 : Schéma représentant l'exécution d'une tâche

On note en général $T = \{T_1, T_2, \ldots, T_n\}$ l'ensemble des tâches, chaque tâche est caractérisée par :

- Le temps de traitement de la tâche T_i sur la machine M_j : p_i.
- La date de disponibilité de la tâche T_i : r_i.
- Le temps de préparation de la tâche T_i : S_i.
- La date de fin de traitement de la tâche T_i : C_i.
- La date échue de la tâche T_i : d_i.
- La date limite de la tâche T_i : $\widetilde{d_i}$.
- Le poids associé à la tâche T_i : W_i.
- La tardivité de la tâche T_i : $L_i = (C_i - d_i)$.
- Le retard de la tâche T_i : $t_i = max(0, C_i - d_i)$.
- L'indicateur de retard de la tâche T_i : $U_i = \begin{cases} 0 & \text{si } C_i \leq d_i \\ 1 & \text{sinon} \end{cases}$.

Les ressources

Une ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'au moins une tâche est disponible en quantité limité appelé capacité de la ressource. Il existe différent type de ressources :

- **Les ressources renouvelables :** sont disponibles en quantité constantes toute au long de l'exécution des tâches, ce sera le cas de machines (ces machines peuvent être regroupées au centre de production dits souvent ateliers), de processeurs, d'ouvriers spécialisés,…, etc.
- **Les ressources consommables :** il s'agit de ressources dont la disponibilité décroit après avoir été allouées à une tâche, par exemple la matière première.

- **Les ressources disjonctives :** il s'agit de ressources qui ne peuvent exécutées qu'une seule tâche à la fois.
- **Les ressources cumulatives :** il s'agit de ressources qui peuvent être utilisées simultanément par plusieurs tâches.

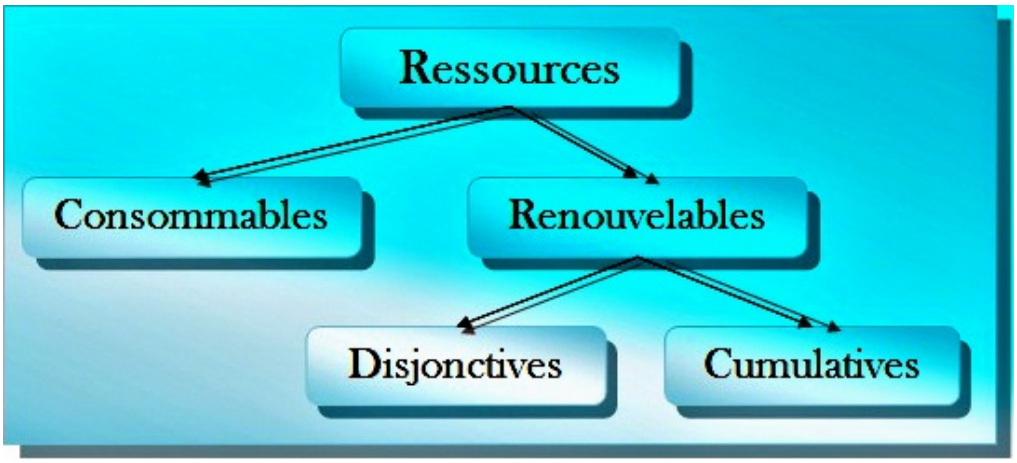


Figure 1.2.2 : Classification des ressources

Les contraintes

Les contraintes sont des conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter lors de la résolution de ces problèmes, en effet la résolution d'un problème d'ordonnancement subit à des principes qu'on doive respecter.

Suivant la disponibilité des ressources et suivant l'évolution temporelle, trois types de contraintes peuvent être distingués : **contraintes de la localisation temporelles**, **contraintes de précédences** et **contraintes de ressources.**

Les contraintes de localisation temporelles : pour la réussite d'un projet ou d'un plan de production, on doit se soumettre aux impératifs de production (réalisation) traduits par le respect d'un échéancier fixé auparavant. Au niveau global on parlera d'une date de lancement d'un projet et d'une date de livraison. A un autre niveau de détail plus fin, pour une tâche ou une opération. Ce dernier niveau se traduit par la définition des dates suivantes :

- Date de disponibilité : $\boldsymbol{r_i}$.
- Date échue : $\boldsymbol{d_i}$.

Les contraintes de précédence : une contrainte qui lie le début d'une activité à la fin d'un autre est appelée contrainte de succession ou de précédence. Les gammes opératoires sont un exemple de ces contraintes.

Les contraintes de ressources : plusieurs types de contraintes peuvent être induits par la nature des ressources. A 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 critères

L'optimisation s'exprime par une fonction de minimisation ou maximisation de ces critères, les critères constituent les objectifs qu'on souhaite optimiser. En fonction du nombre d'objectif on parle d'optimisation mono-objective (d'un seul critère) et d'optimisation multi-objectifs (plusieurs critères).

Les différents critères ne sont pas indépendants ; certains même sont équivalents c'est-à-dire si une solution optimale pour l'un est aussi optimale pour l'autre et inversement.

On cite quelques exemples ci-dessous :

- La minimisation de la longueur d'ordonnancement : $\boldsymbol{C_{max} = \max\{C_i\}, i = \overline{1,n}}$.
- La minimisation du temps de fin de traitement moyen : $\boldsymbol{\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i}$.
- La minimisation du grand retard : $\boldsymbol{T_{max} = \max\{T_i\}, i = \overline{1,n}}$.

Les objectifs

L'objectif principal d'un ordonnancement est de modifier l'ordre des tâches traitées afin d'améliorer la satisfaction des utilisateurs. La satisfaction peut varier suivant les systèmes, ce peut être le délai global de traitement de l'ensemble des tâches (qui conduit à maximiser le facteur d'utilisation des machines), le temps de flot moyen (c'est le temps de réponse moyen, comme par exemple, dans le cas d'un système informatique multi-accès) ou le retard maximum (comme par exemple, le cas d'un système travaillant en temps réel). L'algorithme d'ordonnancement dépend donc de l'objectif souhaité.

On note par exemple :

- La somme des dates de fin de traitement, notée $\boldsymbol{C = \sum_{i=1}^{n} C_i}$, la date de fin de traitement moyenne pondérée notée $\boldsymbol{\overline{C_w}}$.
- La moyenne des retards notée $\boldsymbol{\overline{T}}$ ou la moyenne pondérée des retards notée $\boldsymbol{\overline{T_w}}$.
- La tardivité moyenne notée $\boldsymbol{\overline{L}}$, ou la tardivité moyenne pondérée $\boldsymbol{\overline{L_w}}$.
- Le grand temps de flow $\boldsymbol{F_{max}}$, le temps de flow moyen $\boldsymbol{\overline{F}}$, ou le temps de flow moyen pondéré $\boldsymbol{\overline{F_w}}$.
- Le nombre de tâches en retard $\boldsymbol{N_T = \sum_{i=1}^{n} \mu_i}$.

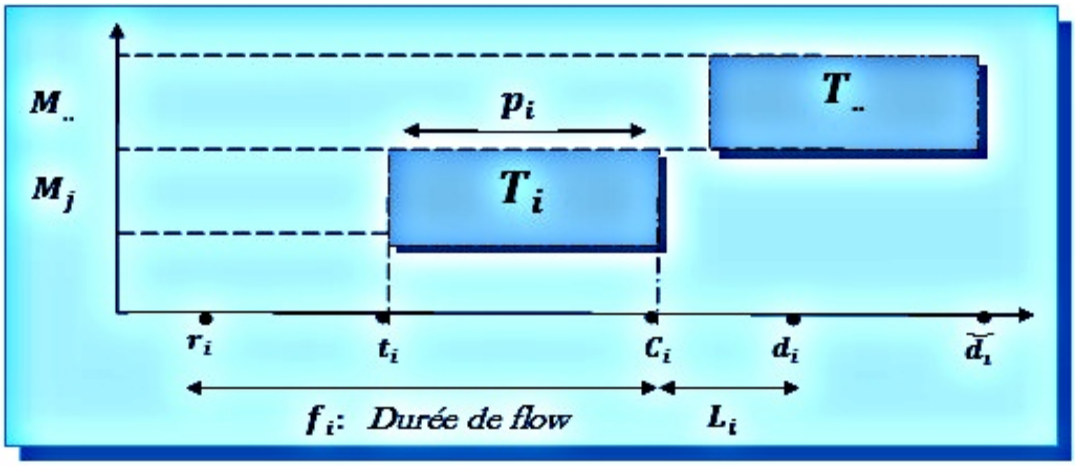


Figure 1.2.3 : Quelques critères d'évaluation

Remarque 1.2.1 *Un ordonnancement pour lequel une certaine mesure d'efficacité est optimisée sera appelé optimal.*

1.3 Caractérisation des machines

1.3.1 Les types de machines

Les machines peuvent être en parallèles, faisant la même fonction, ou spécialisées dans l'exécution de certaines opérations (une tâche est constituée de plusieurs opérations), dans ce cas, les machines sont disposées en séries [32].

Les machines parallèles

Ces machines exécutent les mêmes traitements et sont donc capables de traiter n'importe quelle tâche. En fonction de la vitesse de traitement, nous distinguons trois types de machines parallèles [32]:

Les machines identiques : pour ces machines, les vitesses de traitement des tâches sont toutes égales, sont notées par **P** .

Les machines uniformes : pour ces machines, les vitesses de traitement des tâches sont différentes mais que la vitesse de chaque machine est constante et ne dépend pas des tâches ces machines sont notées par **Q**.

Les machines générales : pour ces machines, la vitesse de traitement d'une machine dépend de la tâche qui est traitée, et notées par **R**.

Les machines spécialisées

Ces machines sont spécialisées à l'exécution de certaines tâches. Dans le cas de machines spécialisées, nous distinguons trois systèmes ou modes de traitement [32]:

Flow shop : Dans un système flow shop, les tâches T_i doivent être traitées par toutes les machines dans un même ordre, notées par **F**, Le même ordre de traitement pour toutes les tâches.
Open shop : Dans un système open shop, les tâches T_i doivent être traitées par toutes les machines mais l'ordre de traitement n'est pas nécessairement le même pour toutes les tâches, sont notées par **O**, l'ordre de traitement est quelconque.
Job shop : Dans un système job shop, le sous-ensemble de machines devant traiter une tâche et l'ordre du traitement sont arbitraires, mais doivent être spécifiés à l'avance, ces machines sont notées par **J**.

Voici le schéma suivant qui représente les types de machines

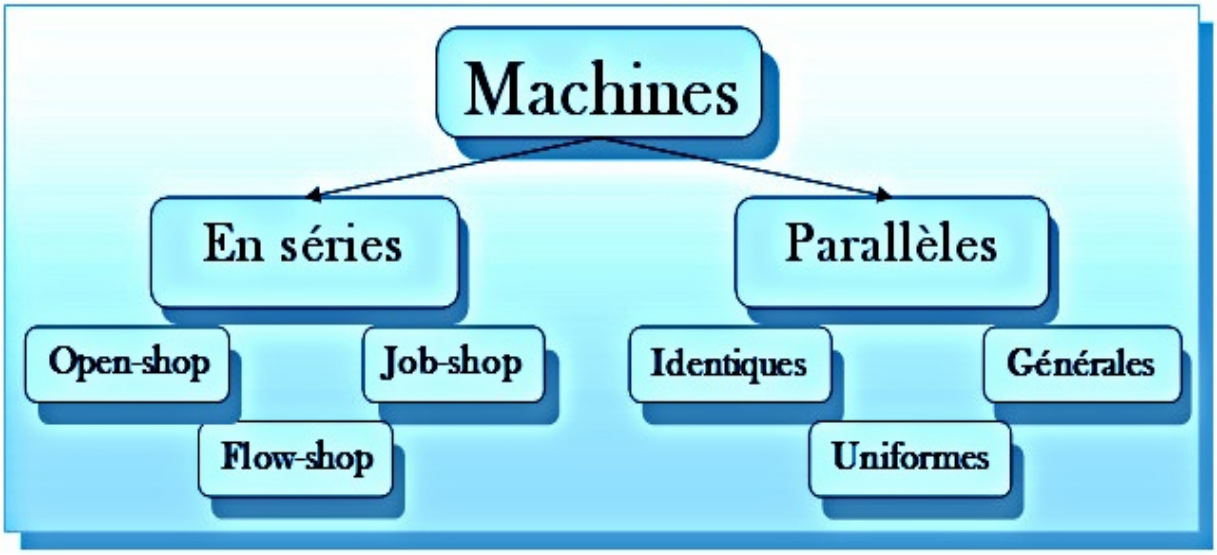


Figure 1.3.1 : Schéma représentant les types de machines

1.3.2 Notation générale $(\alpha/\beta/\gamma)$

Etant donné la diversité des problèmes de l'ordonnancement, un formalisme permettant de distinguer les différents problèmes de l'ordonnancement entre eux et de les classifier est utilisé. Ce formalisme comporte trois champs $\alpha/\beta/\gamma$ permettant de décrire les différentes entités d'un problème d'ordonnancement [32].

Champ α

Ce champ décrit les machines utilisées : type de machines, nombre de machines et type d'atelier. Il est composé de deux sous-champs et désigne par $\boldsymbol{\alpha : \alpha_1}$et $\boldsymbol{\alpha_2}$

- $\alpha_1 \in \{\emptyset, P, Q, R, O, F, J\}$ décrit le type de machine.
- $\alpha_1 = \emptyset$: Une seule machine.
- $\alpha_1 = P$: machines parallèles identiques.
- $\alpha_1 = Q$: machines parallèles uniformes.
- $\alpha_1 = R$: machines parallèles générales.
- $\alpha_1 = O$: machines spécialisée (système open shop).
- $\alpha_1 = F$: machines spécialisées (système flow shop).
- $\alpha_1 = J$: machines spécialisées (système job shop).

- $\alpha_2 \in \{\emptyset, k\}$ dénote le nombre de machines.
- $\alpha_2 = \emptyset$: le nombre de machines est variable.
- $\alpha_2 = k$: le nombre de machines est égal à k (k est un entier positif non nul).

Champ $\boldsymbol{\beta}$

Ce champ $\boldsymbol{\beta}$ décrit les tâches et les contraintes liées à ces dernières. Il est composé de **7** Sous-champs et désigné par $\boldsymbol{\beta : \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6}$ **et** $\boldsymbol{\beta_7}$.

- $\beta_1 \in \{\emptyset, pmtn\}$: indique la possibilité de préemption des tâches.
- $\beta_1 = \emptyset$: la préemption des tâches n'est pas autorisée.
- $\beta_1 = pmtn$: la préemption des tâches est autorisée.

- $\beta_2 \in \{\emptyset, Prec, tree, chains\}$: indique le type de contraintes de précédence.
- $\beta_2 = \emptyset$: pas de contraintes de précédence (les tâches sont indépendantes).
- $\beta_2 = prec$: les contraintes de précédence sont quelconques.
- $\beta_2 = tree$: les contraintes de précédence sont données sous forme d'un arbre.
- $\beta_2 = chains$: les contraintes de précédence sont données sous forme de chaines.

- $\beta_3 \in \{\emptyset, r_i\}$: décrit les dates de disponibilité.
- $\beta_3 = \emptyset$: toutes les dates de disponibilité sont nulles.
- $\beta_3 = r_i$: il existe des dates de disponibilité pour les tâches, non identiques

- $\beta_4 \in \{\emptyset,\ p_i = p, p \leq p_i \leq p\}$.
- $\beta_4 = \emptyset$: les temps de traitement des tâches sont quelconques et non identiques.
- $\beta_4 = p_i = p$: les temps de traitement des tâches sont tous égaux à p (constants).
- $\beta_4 = p \leq p_i \leq p$: le temps de traitement de chaque tâche est compris entre p et p.

- $\beta_5 \in \{\emptyset, d_i\}$.
- $\beta_5 = \emptyset$: il n'y a pas des dates échues pour les tâches ou il existe des dates échues pour les tâches dans le cas ou le critère d'optimisation est en fonction de ces dernières.
- $\beta_5 = d_i$: il existe des dates échues pour les tâches.

- $\beta_6 \in \{\emptyset, splitting\}$: indique la possibilité du découpage des tâches.
- $\beta_6 = \emptyset$: le découpage des tâches n'est pas autorisé.
- $\beta_6 = split$: le découpage des tâches est autorisé.

- $\beta_7 \in \{\emptyset, k\}$: indique la présence des ouvriers spécialisés.
- $\beta_7 = \emptyset$: il n'existe pas d'ouvriers spécialisés.
- $\beta_7 = k$: le nombre d'ouvriers spécialisés est égal à k.

Champ γ

Le troisième champ $\boldsymbol{\gamma}$ décrit les critères d'optimalités qui vont représenter les objectifs à atteindre pour le problème en question, par suit on cherche à minimiser $\boldsymbol{\gamma}$ avec :

$\boldsymbol{\gamma \in \{C_{max}, \overline{C}, \overline{c}_w, T_{max}, \overline{T}, \overline{T}_w, L_{max}, \overline{U}, \overline{U}_W \ldots\}}$

Exemple 1.3.1

$\boldsymbol{P//C_{max}}$ *: Traiter un ensemble de tâches indépendantes sans préemption ayant des temps de traitement arbitraire et des dates de disponibilité nulles sur des machines parallèles identiques l'objectif est de minimiser la date de fin de traitement.*

$\boldsymbol{Q2/tree/T_{max}}$ *: Traiter un ensemble de tâches dépendantes, où le graphe de précédence est donné sous forme d'un arbre, et sans préemption ayant des temps de traitement arbitraires et des dates de disponibilité nulles sur deux machines parallèles uniformes. L'objectif est de minimiser le grand retard.*

P3/p_i=1, Prec / F_{max} *: Traiter un ensemble de tâches dépendantes, sans préemption ayant des temps de traitements égaux à* ***1*** *et des dates de disponibilités nulles sur* ***3*** *machines parallèles identiques l'objectif est de minimiser le grand temps de Flow.*

1.4 Classification des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont généralement classés en deux principaux modèles dépendamment du nombre d'opérations que requièrent les jobs : des modèles à une opération (machine unique et machines parallèles) et des modèles à plusieurs opérations (flow-shop, open shop et job shop) [14].

1.4.1 Modèle à une opération

Modèle à machine unique

Dans un modèle à machine unique, l'ensemble des tâches à réaliser est exécuté par une seule machine. L'une des situations intéressantes où on peut rencontrer ce genre de configurations est le cas où on est devant un système de production comprenant une machine goulot qui influence l'ensemble du processus. Ce modèle est illustré dans la figure 1.4.1

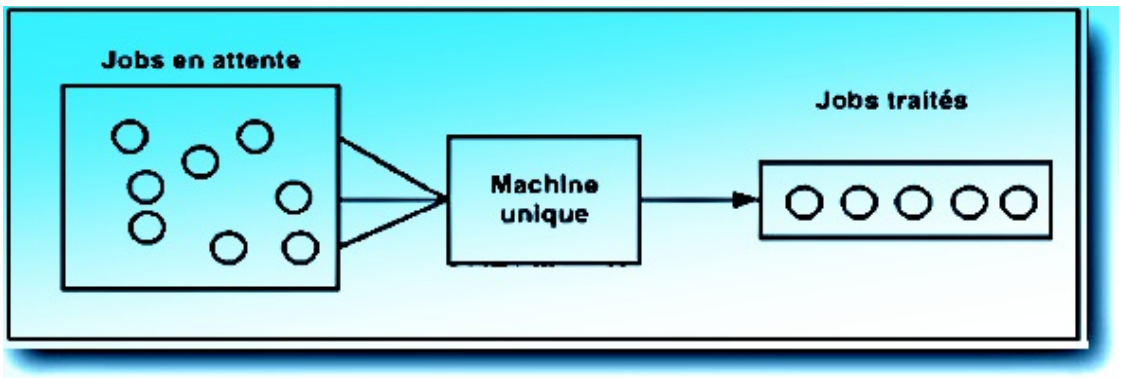


Figure 1.4.1 : Modèle à machine unique

Modèle à machines parallèles

Ce modèle est utilisé surtout dans les secteurs industriels tels que : l'industrie alimentaire, les industries plastiques, les fonderies et en particulier l'industrie textile. Le processus de déroulement de ce système de production est le suivant : à chaque fois qu'une machine ***i*** se libère, ont lui affecte un job ***j*** comme illustré à la figure 1.4.2. Dans le cas d'un processus d'assemblage

industriel, par exemple, si l'une des étapes d'assemblage nécessite beaucoup de temps, il serait très intéressant alors d'avoir plusieurs machines parallèles qui effectuent la même tâche. D'un autre côté, les machines parallèles sont classées suivant leur rapidité. Si toutes les machines de l'ensemble ont la même vitesse de traitement et effectuent les mêmes tâches, elles sont identiques. Si les machines ont des vitesses de traitement différentes mais linéaires alors elles sont dites uniformes. Dans le cas où les vitesses des machines sont indépendantes les unes des autres, on parle alors de modèle de machines parallèles non reliées ou indépendantes.

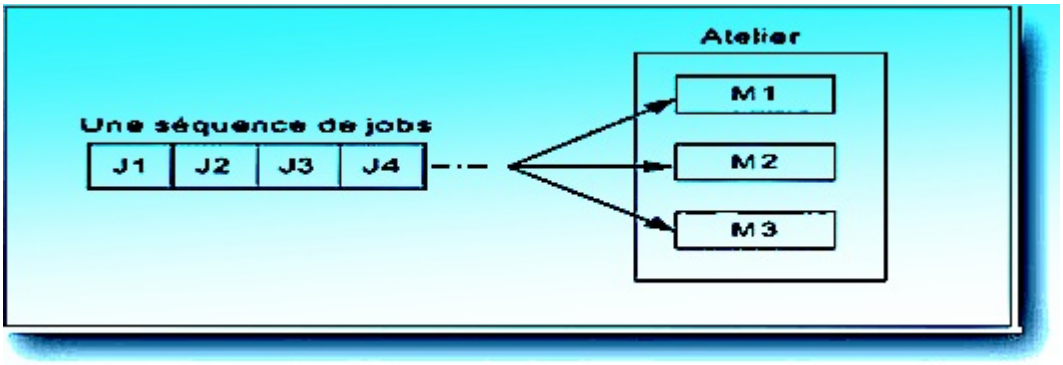


Figure 1.4.2 : Modèle à machines parallèles

1.4.2 Modèle à plusieurs opérations

Le modèle à plusieurs opérations est constitué des cas où un job, pour se réaliser, doit passer par plusieurs machines, chacune de ces machines ayant ses spécificités. On distingue trois modèles selon l'ordre de passage des jobs sur les machines, à savoir les modèles de flow-shop, job-shop et open-shop.

Modèle flow-shop

Dans le modèle de flow-shop, les ordres de fabrication visitent les machines dans le même ordre, avec des durées opératoires pouvant être différentes. Chaque job va être s'exécuter sur les M machines en série et tous les jobs vont suivre le même ordre de passage sur ces machines. Ce type de modèle est aussi appelé modèle linéaire. La figure 1.4.3 illustre le cas d'un flow-shop avec quatre machines et quatre jobs. Les quatre jobs suivent le même ordre de traitement sur les quatre machines.

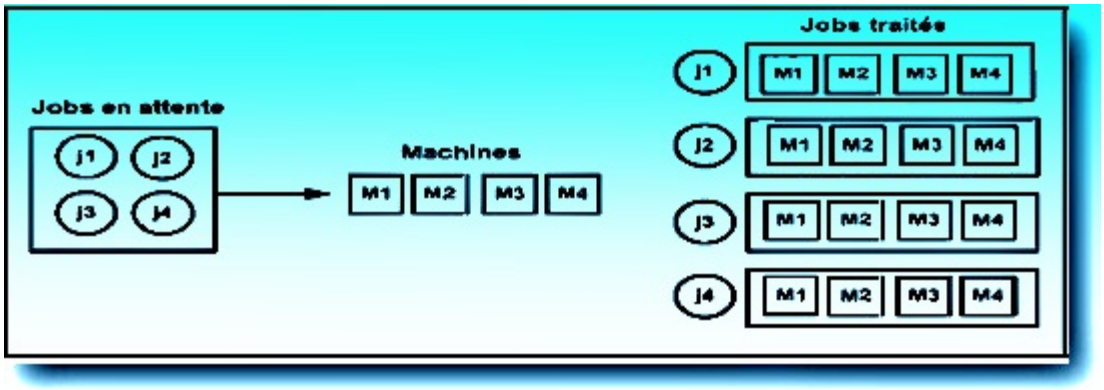


Figure 1.4.3 : Modèle flow-shop

Modèle job-shop

Concernant le modèle de job-shop, chaque job à un ordre à suivre et chacun d'entre eux peut s'exécuter plusieurs fois sur la même machine, ce qui n'est pas le cas du flow-shop. Il s'agit dans ce cas de déterminer les dates de passage sur différentes ressources d'ordres de fabrication ayant des trajets différents dans l'atelier. Ces ordres de fabrication partageant des ressources communes, des conflits sont susceptibles de survenir, résultant des croisements de flux illustrés à la figure 1.4.4.

Dans cette figure, l'ordre de passage de chaque job est indiqué par un chemin de même couleur que celui du job. Par exemple, le job $\boldsymbol{J_1}$ passe par toutes les machines, alors que le job $\boldsymbol{J_3}$ ne passe que par la deuxième et la quatrième machine. Dans son expression la plus simple, le problème consiste à gérer ces conflits tout en respectant les contraintes données, et en optimisant les objectifs poursuivis. Les types de ressources et de contraintes prises en compte peuvent toute fois considérablement compliquer le problème. Plus on intégrera de contraintes, plus on se rapprochera d'un cas réel, mais moins on disposera de méthodes de résolution satisfaisantes.

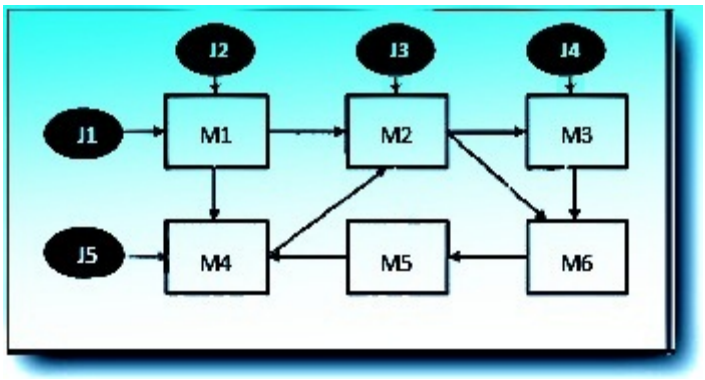


Figure 1.4.4: Modèle job-shop

Modèle open-shop

Dans le modèle d'open shop, l'ordre de passage des n jobs sur les M machines n'est pas connu à l'avance. Cet ordre est déterminé lors de la construction de la solution. Chaque job j peut avoir son propre ordre de passage sur toutes les machines. Le fait qu'il n'y ait pas d'ordre prédéterminé rend la résolution du problème d'ordonnancement de ce type plus complexe, mais offre cependant des degrés de liberté intéressants. À la figure 1.4.5, nous avons un ensemble de quatre jobs et un ensemble de quatre machines. À droite de la figure nous pouvons remarquer que chaque job a suivi un ordre de passage différent sur les quatre machines.

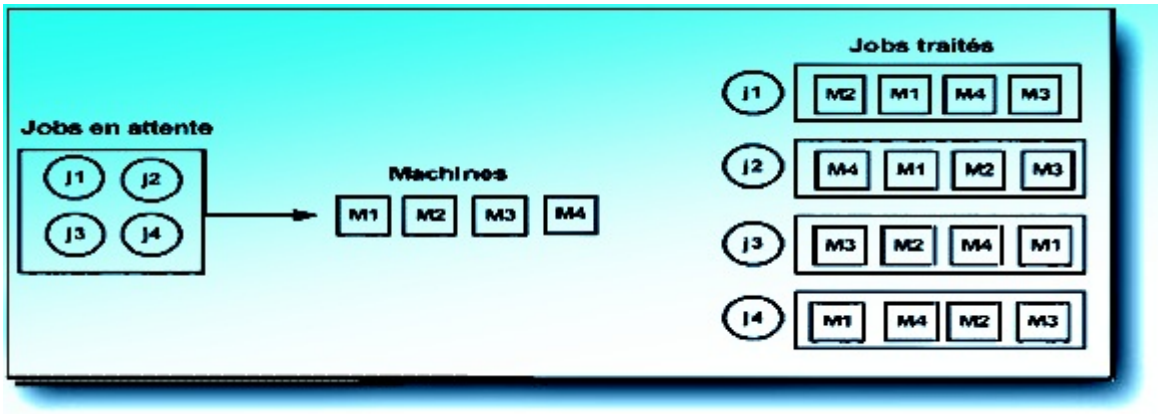


Figure 1.4.5 : Modèle open-shop

1.5 Représentation des problèmes d'ordonnancement sur les machines parallèles

1.5.1 Diagramme de Gantt

Le diagramme de Gantt est un outil permettant de modéliser la planification des tâches nécessaires à la réalisation d'un projet. Il s'agit d'un outil élaboré en 1917 par Henry L. Gantt [7]. Etant donné la facilité relative de lecture des diagrammes de Gantt, cet outil est utilisé par la quasi-totalité des chefs de projet dans tous les secteurs. Il permet de représenter graphiquement l'avancement du projet et constitue également un bon moyen de communication entre les différents acteurs d'un projet. Le diagramme de Gantt présente en ordonnée la liste des tâches, notées T_i à exécuter par les machines notées M_j et en abscisse l'échelle du temps, comme le montre la figure 1.5.1 dans le cas de **5** tâches et **2** machines.

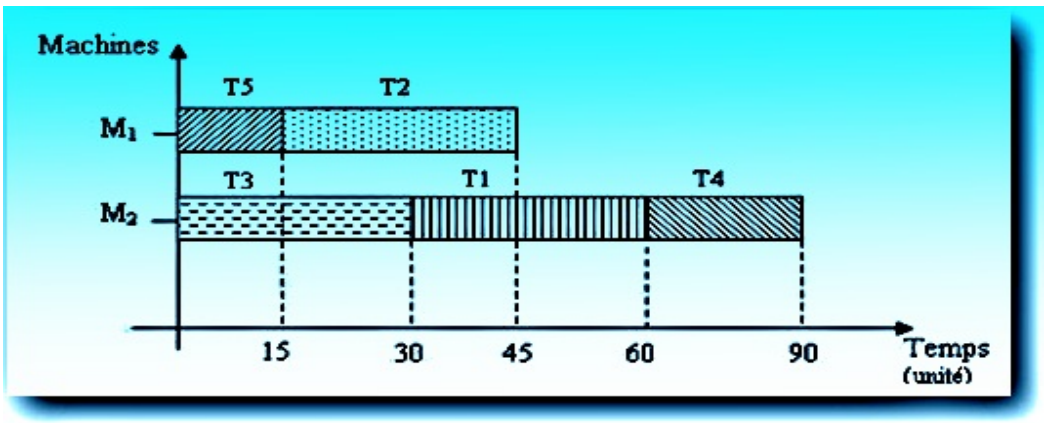


Figure 1.5.1 : Diagramme de Gantt d'un ordonnancement

1.5.2 Graphe potentiel-tâches

Cet outil graphique a été développé grâce à la théorie des réseaux de pétri qui ont surtout servi à modéliser les systèmes dynamiques à évènements discrets [12]. Dans ce genre de modélisation, les tâches sont représentées par des nœuds et les contraintes par des arcs [40], comme le montre la figure 1.5.2. Ainsi, les arcs peuvent être de deux types :

- Les arcs conjonctifs illustrant les contraintes de précédence et indiquant les durées des tâches.
- Les arcs disjonctifs indiquant les contraintes ressources [24], [29].

Exemple 1.5.1

Tâches	*Durées*	*Contraintes*
a	***6** mois*	...
b	***3** mois*	...
c	***6** mois*	...
d	***2** mois*	***b*** achevée
e	***4** mois*	***b*** achevée
f	***3** mois*	***d*** et ***a*** achevée

Tableau 1.1 : Données utilisées pour la réalisation d'un graphe potentiel-tâches

Pour qu'une tâche puisse commencer, il est nécessaire que toutes les tâches qui la relient à la tâche du début ***S*** du projet, soient réalisées. On définit donc : la date de disponibilité r_i de la tâche, qui correspond à la date de début, au plus tôt, de l'exécution de la tâche.

- La tâche ***f*** ne peut s'exécuter que si ***a*** et ***d*** ont été réalisées. Donc, pour exécuter ***a*** il faut ***6*** mois et pour exécuter ***d*** il faut ***2+3*** mois.

- La tâche ***f*** ne pourra commencer au plus tôt que *6* mois après le début du projet : c'est donc le plus long chemin entre ***a*** et *f*.
- La durée du projet, qui correspond au plus long chemin entre ***S*** (tâche de début du projet) et ***S'*** (tâche de fin du projet).

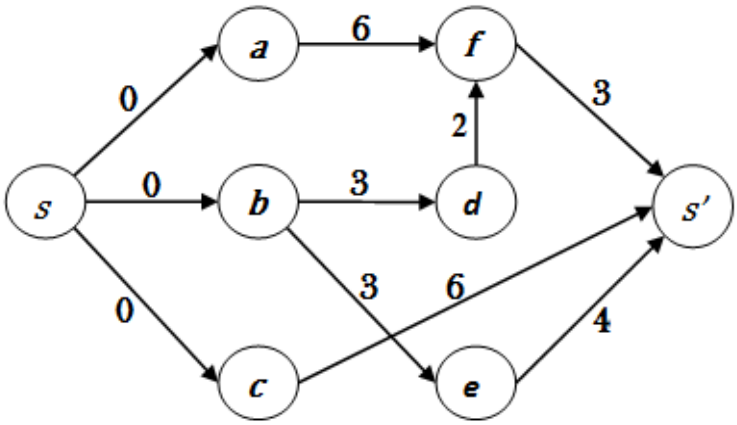


Figure 1.5.2 : Graphe Potentiel-Tâches d'un ordonnancement

1.5.3 Méthode PERT

PERT [47] est une technique permettant de gérer l'ordonnancement dans un projet. Elle consiste à représenter sous forme de graphe, un réseau de tâches dont l'enchaînement permet d'aboutir à l'atteinte des objectifs d'un projet. Elle a été conçue par la marine américaine afin de permettre de coordonner les travaux de plusieurs milliers de personnes pour aboutir à la réalisation de missiles à ogives nucléaires POLARIS.

Ainsi, la méthode ***PERT*** implique au préalable :

- Un découpage précis du projet en tâches.
- L'estimation de la durée de chaque tâche.
- La nomination d'un chef de projet chargé d'assurer le suivi du projet, de rendre compte si nécessaire et de prendre des décisions en cas d'écart par rapport aux prévisions, avec :

- Les sommets représentent l'étape.
- Les arcs représentent les tâches.

La méthode ***PERT*** vise à :

- Réduire le délai de réalisation d'un projet.
- Réduire le cout de réalisation.
- Optimiser les ressources.

Exemple 1.5.2

Soit le projet suivant où les données sont regroupées dans le tableau ci-dessous :

Tâche(s) antérieure(s)	Tâche	Durée
aucune	***A***	***3***
A	***B***	***1***
A	***C***	***5***
B	***D***	***6***
B	***E***	***4***
C, I, D	***F***	***2***
E, F	***G***	***9***
aucune	***H***	***5***
H	***I***	***8***
H	***J***	***2***
I	***K***	***3***
J, K	***L***	***7***

Tableau 1.2 : Les données de l'exemple

La construction d'un réseau ***PERT*** et son exploitation supposent d'effectuer les opérations suivantes :

- Etablir une liste précise des tâches.
- Déterminer les tâches antérieures (ainsi que les tâches postérieures éventuellement).
- Construire les graphes partiels.
- Regrouper les graphes partiels.
- Construire le réseau.

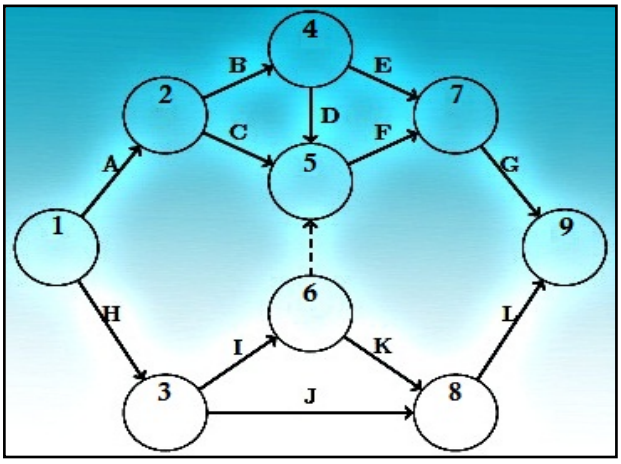


Figure 1.5.3 : Réseau PERT

1.5.4 Les anomalies de GRAHAM

GRAHAM (1966) [18] a met en évidence des anomalies pour déterminer un ordonnancement réalisable suivant l'un des changements suivants [32]:

- La liste de priorité change.
- Les durées d'exécution des tâches diminuent.
- Le nombre de machines augmente.
- Les contraintes de précédence sont affaiblies.

Exemple 1.5.3

Voici les données de problème P2/ prec/ C_{max} *:*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
p_i	3	4	2	4	4	2	13	2

Tableau 1.3 : Les données du problème

Les contraintes de précédence sont données ci-dessous :

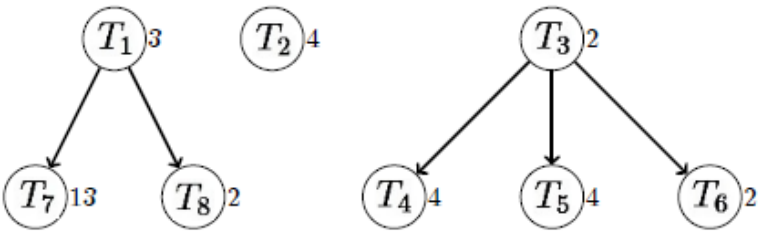


Figure 1.5.4 : Graphe des contraintes de précédence

Sur la figure 1.5.5, la liste de priorité L, $L = T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8$ est utilisée pour déterminer un ordonnancement au problème $P2/prec/C_{max}$.

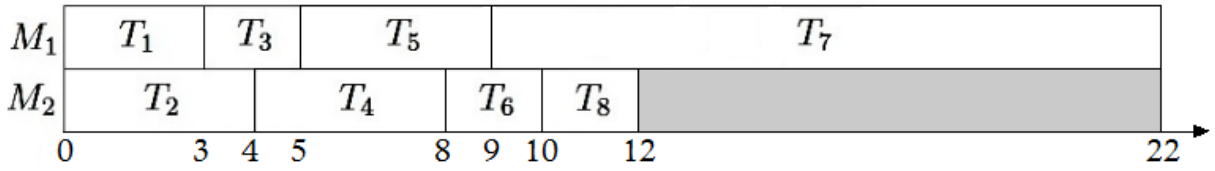


Figure 1.5.5 : Ordonnancement d'un P2/ prec/C_{max} < *Liste L*>

Le temps de fin d'un ordonnancement de problème $P2/prec/C_{max}$ de liste L peut augmenter comme peut diminuer, les figures 1.5.6, 1.5.7, 1.5.8 et 1.5.10 montrent ces anomalies.

Si on change la liste L en L' ,$L' = T_1, T_2, T_3, T_4, T_5, T_6, T_8, T_7$, la longueur d'ordonnancement augmente de ***22*** à ***23*** unités :

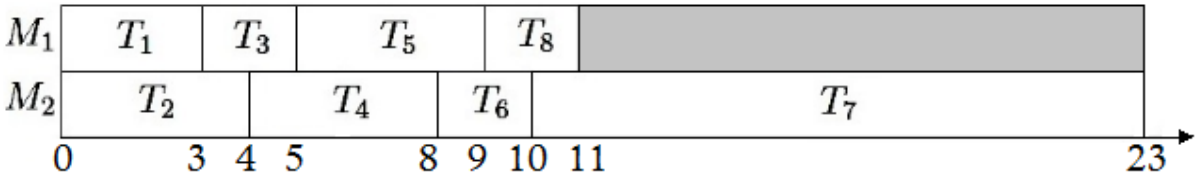


Figure 1.5.6 : Ordonnancement d'un P2/ prec/C_{max} < *Liste L'*>

Même si on réduit les temps d'exécution de chaque tâche d'une unité de temps, $P_{i'} = P_{i'} - 1$ $(i = 1...8)$, la longueur d'ordonnancement devient ***18*** unités :

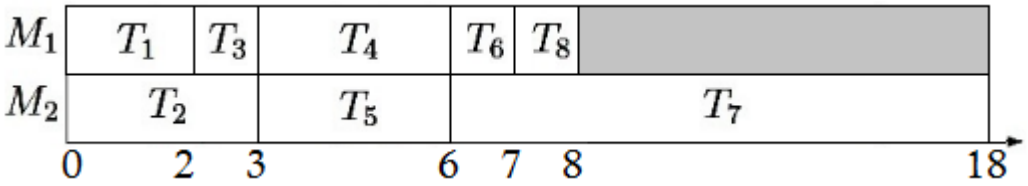


Figure 1.5.7 : Ordonnancement de même problème avec réduction des temps d'exécution

Si on ajoute une machine supplémentaire qu'on affecte à l'atelier, la longueur d'ordonnancement diminue aussi de ***22*** à ***19*** unités :

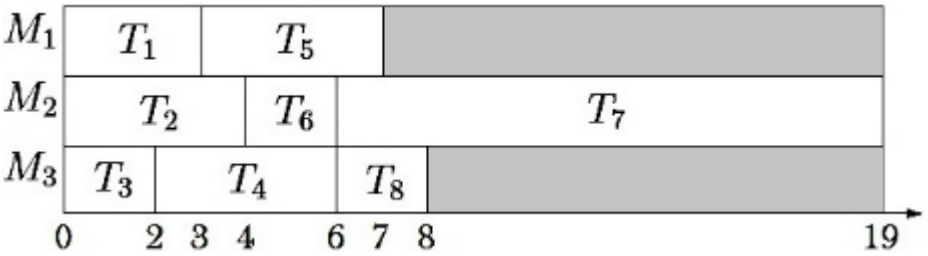


Figure 1.5.8 : Ordonnancement de même problème avec augmentation du nombre de machines

Même si on affaiblit les contraintes de précédence en retirant un arc (T_3, T_4) du graphe de précédence, la longueur d'ordonnancement revient comme elle était ***22*** unités :

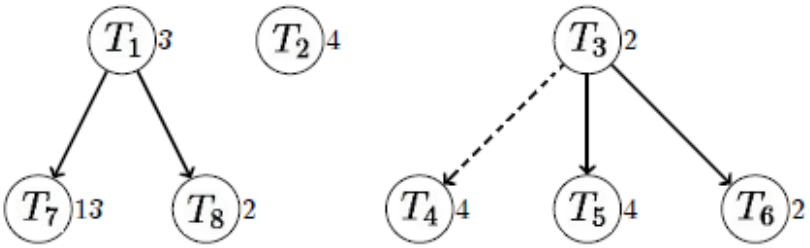


Figure 1.5.9 : Graphe des contraintes de précédence affaiblies

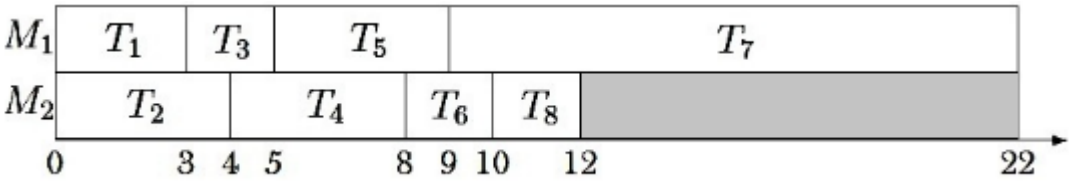


Figure 1.5.10 : Ordonnancement obtenu avec contraintes de précédence affaiblies

Remarque 1.5.1 *Si une liste de priorité est optimale pour un problème donné, on ne peut rien en déduire pour un problème similaire où des paramètres changent. D'où, la difficulté à établir un algorithme de résolution optimale de type liste pour* $P2/prec/C_{max}$.

1.6 La complexité des problèmes d'ordonnancement

La théorie de la complexité cherche à connaître la difficulté d'une réponse algorithmique à un problème. D'une part un problème est une question abstraite à résoudre, c'est-à-dire un énoncé rigoureusement dans un langage des mathématiques. Par exemple, parmi les problèmes de graphe, citons : le plus court chemin entre deux sommets, la connexité, l'ensemble dominant minimum, la coloration minimum…etc. Autrement dit, peut-il y avoir un algorithme efficace pour répondre à la question de décidabilité ou de calculabilité d'autre part un algorithme est une procédure de résolution composée d'une suite finie d'opérations élémentaires. En démarrant d'un état initial, l'exécution des instructions de l'algorithme permet de passer par des états intermédiaires et finir dans un état final. Le passage entre les états peut être algorithme déterministe ou non déterministe. Le mot "algorithme" vient du nom du mathématicien Al Khawarizmi ($9^{éme}$ siècle), qui a fourni le premier algorithmique pour résoudre les équations linéaires et quadratiques [22].

L'efficacité d'un algorithme est mesurée par son temps d'exécution, c'est-à-dire, pour toute instance des données en entrée de l'algorithme, combien d'opérations élémentaires nécessaires

pour atteindre le résultat, cette mesure est aussi dite complexité algorithmique. Quand ce nombre d'opérations est borné par un polynôme, dépendant de la taille des entrées, alors l'algorithme est dit polynomial, par contre, si ce nombre d'opérations est borné inférieurement par une forme exponentielle de la taille des données, alors l'algorithme est dit exponentiel.

1.6.1 Notion de complexité de problèmes

D'une manière générale, les problèmes d'ordonnancement d'ateliers étant des problèmes combinatoires difficiles, il n'existe pas de méthodes universelles permettant de résoudre tous les cas.

La complexité d'un algorithme

On appelle complexité d'un algorithme le nombre asymptotique d'opérations élémentaires $(+, -, \times, /, =$ ***et test***$)$ qu'il doit effectuer en fonction de la taille de l'entrée qu'il a à traiter [6].

Notation	Désignation	Notation	Désignation
$\varphi(1)$	**Constante**	$\varphi(n^2)$	**Quadratique**
$\varphi(log\, n)$	**Logarithmique**	$\varphi(n^3)$	**Cubique**
$\varphi(n)$	**Linéaire**	$\varphi(a^n)$, a >1	**Exponentielle**
$\varphi(n(log\, n))$	**Quasi-linéaire**	$\varphi(n\,!)$	**Factorielle**

Tableau 1.4 : Tableau représentatif de la désignation de la complexité algorithmique courante

On note : $\boldsymbol{n}$ la taille de donnée, $\boldsymbol{T(n)}$: le nombre d'opérations élémentaires.

L'objectif est d'analyser les coûts de résolutions surtout en termes de temps de calcul. Elle vise aussi à classer les problèmes en plusieurs niveaux de difficulté. Une étude a prouvé que la plupart des problèmes d'ordonnancement sont des problèmes difficiles.

1.6.2 Problème de Décision

Un problème de décision est une question dont la réponse est «Oui» ou «Non». Pour un problème ***P***, notons ***O(P)*** l'ensemble des instances de ***P*** dont la réponse est «Oui». Un problème de décision ***D1*** est dit réductible à un autre problème de décision ***D2*** (noté ***D1 α D2***) s'il existe une fonction polynomiale ***f*** qui transforme chaque énoncé de ***D1*** en un autre énoncé ***D2*** de telle manière que la réponse pour ***D1*** et oui si seulement si, la réponse pour ***D2*** est «Oui».

1.6.3 Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire consiste à optimiser (maximiser ou minimiser) une fonction objectif dans un domaine fini. Il est toujours possible de faire correspondre un problème d'optimisation à un problème de décision.

Dans le cas par exemple de la minimisation d'une fonction objectif, le problème peut être posé de la façon suivante : le problème admet-il une solution dont la fonction objective est inférieure ou égale à un seuil défini ? De même dans le cas de maximisation d'une fonction objectif.

Cook [18] a élaboré une théorie de la complexité en mesurant le temps de calcul en fonction de la taille du problème. Sa théorie distingue entre les algorithmes polynomiaux, de complexité de l'ordre d'un polynôme, et les autres dits exponentiels. La puissance de calcul ne résout rien pour des complexités exponentielles, il est faux de croire qu'un ordinateur peut résoudre tous les problèmes combinatoires.
Une représentation standard des données du problème A, appelée codage prend un espace mémoire appelé taille du problème et notée $|A|$. S'il existe un algorithme qui donne une solution optimale en un temps $T \leq p(|A|)$ où p est un polynôme, on dit que notre problème est facile ou polynomial.

1.6.4 Classes de complexité

Pour résoudre un problème d'ordonnancement, il ne suffit pas de prouver l'existence d'une solution, il faut également la construire, il est clair que construire la solution est plus difficile que de prouver son existence ce qui nous conduit donc à classer les problèmes comme étant difficiles ou faciles. Selon son degré de complexité, un problème de décision peut appartenir à l'une des trois classes suivantes [43] :

La classe P

Un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelle que soit la donnée de celui-ci. La classe P est la classe des problèmes résolvables par un algorithme polynomial déterministe.

La classe NP

Est la classe des problèmes de décision pouvant être résolus par un algorithme polynomial non déterministe .Parmi les problèmes décidables, les plus simples à résoudre sont regroupés dans la classe ***NP*** et il est clair que $P \subseteq NP$.

La classe NP-complet

Est la classe des problèmes dont on n'a pas trouvé d'algorithme polynomial pour les résoudre elle regroupe les problèmes les plus difficiles de la classe ***NP***. Cependant, si on pouvait résoudre un problème de ***NP***-complet en un temps polynomial, on obtiendrait automatiquement des algorithmes polynomiaux pour tous les problèmes de ***NP-C***. Quand un problème est ***NP***-complet, il n'est pas raisonnable d'espérer construire un algorithme polynomial le résolvant. Mais dans certains cas, on peut construire des algorithmes très efficaces appelés algorithmes pseudo-polynomiaux, le problème étudié est alors dit ***NP***-complet au sens faible. Dans le cas contraire, il est dit ***NP***-complet au sens fort.

On adapte le schéma suivant :

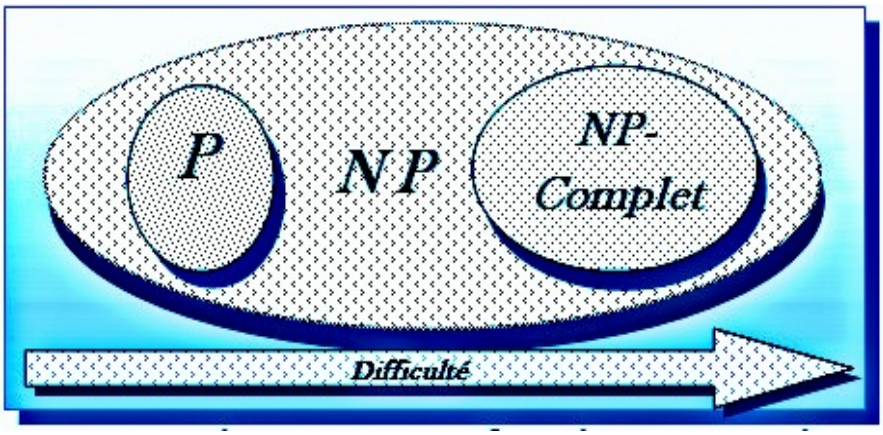


Figure 1.5.1 : Schéma représentatif de classes de complexité

Remarque 1.6.1 *Un problème d'optimisation combinatoire est dit NP- difficile si le problème de décision qui lui correspond est NP-complet et pour les problèmes d'ordonnancements à une machine, certains peuvent être résolus par un algorithme polynomial, certains autres sont démontrés NP -difficiles. Certains ne sont ni démontrés NP-difficiles, ni polynomialement résolvables. Ils restent donc ouverts. Et pour les problèmes d'ateliers, la plupart de ces problèmes ont été démontré NP - difficiles.*

Le tableau suivant donne les complexités de quelques problèmes d'ordonnancement non-préemptif hors-ligne pour multiprocesseur [42]. L'objectif commun de ces algorithmes est de satisfaire la contrainte de précédence, si elle existe, ainsi que de minimiser le temps de fin traitement respectivement le nombre de tâches et le nombre de processeurs :

Nombre de processeurs	**Durée d'exécution**	**Contraintes de précédence**	**Complexité**
m	toutes égales	Arbre	polynomial
2	toutes égales	Quelconques	polynomial
2	sans	Quelconques	*NP*- complet
2	deux valeurs possibles	Quelconques	*NP*- complet
m	toutes égales	Quelconques	*NP*- complet
m	quelconques	Arbre	*NP*- complet

Tableau 1.5 : Tableau représentatif de la complexité de quelques problèmes d'ordonnancement

1.6.5 Réduction entre problèmes d'ordonnancement

La notion de réduction polynomiale a été introduite dans le but de comparer la complexité des problèmes [9]. Un problème ***A*** peut être transformé en un autre problème ***B*** tel que : à partir d'une solution de ***B***, on peut calculer une solution de ***A***. Si on peut faire la transformation et le calcul en un temps polynomial, on dit que ***A*** se réduit polynomialement à ***B*** et on note ***A α B***. Par conséquence, un algorithme polynomial pour résoudre ***B*** induit un algorithme polynomial pour résoudre ***A***.

Nous donnons ci-dessous quelque réduction (transformations polynomiales) entre problème d'ordonnancement. Une flèche de ***X*** à ***Y*** signifie que la résolution d'un modèle avec le paramètre ***Y*** st plus difficile que celle du même modèle avec le paramètre ***X,*** ces relations nous aide à déterminer la complexité des problèmes [32].

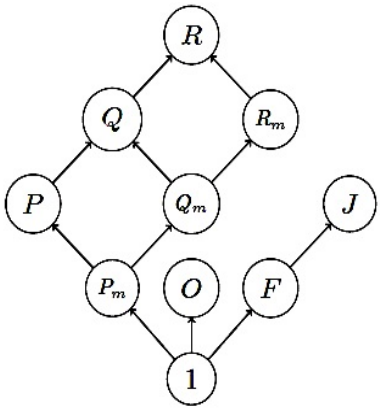


Figure 1.6.2 : Réduction des machines

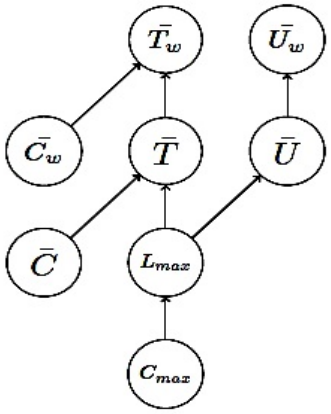


Figure 1.6.3 : Réduction entre les critères

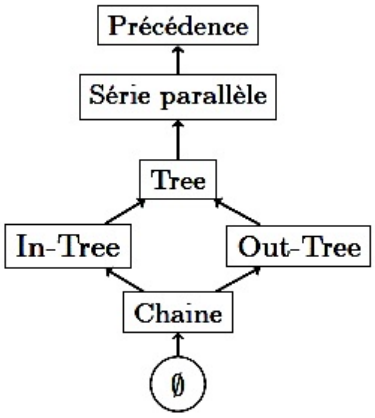


Figure 1.6.4 : Relation de précédence

Remarque 1.6.2 *On remarque que le problème* $\boldsymbol{P//C_{max}}$ *est un cas particulier du problème* $\boldsymbol{Q//C_{max}}$ *où toutes les machines auraient la même vitesse. Tout algorithme développé pour* $\boldsymbol{Q//C_{max}}$ *peut être appliqué au problème* $\boldsymbol{P//C_{max}}$ *.*

Nous donnons ci-dessous quelques transformations polynomiales reliant les dates de disponibilités, les temps de traitement et la préemption des tâches :

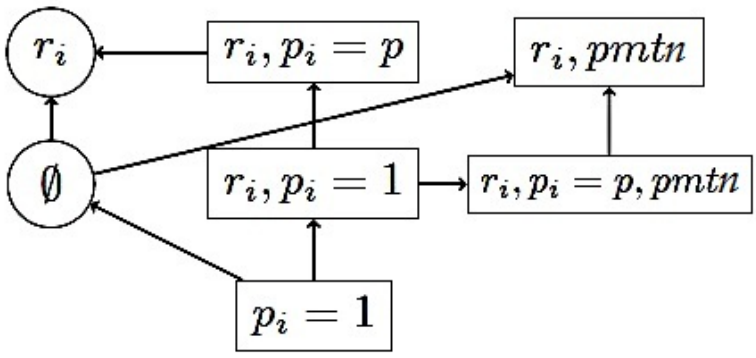


Figure 1.6.5 : Quelques réductions

1.7 Conclusion

À travers ce chapitre nous avons présenté quelques notions de bases sur l'ordonnancement ainsi que les différents types d'ateliers et de machines ont été cité entre autres les problèmes d'ordonnancement étant très variées, ils sont caractérisés par de grands nombres de paramètres relatifs aux tâches, aux ressources aux types de contraintes et aux critères d'optimalités. Enfin nous avons parlé sur la complexité des problèmes d'ordonnancement y'compris leurs classes.

Chapitre 2

Les méthodes de résolution

Chapitre 2

Les méthodes de résolution

2.1 Introduction

La résolution d'un problème est le processus d'identification puis la mise en œuvre d'une solution d'un problème. Pour les problèmes ***NP***-difficiles, on ne connait pas d'algorithmes polynomiaux pour les résoudre et la conjecture $P \neq NP$ fait qu'il est peu probable qu'il en existe [3]. Ainsi, différentes méthodes de résolution sont largement utilisées pour appréhender les problèmes ***NP***-difficiles [9].

Dans ce chapitre nous présentons deux types de méthodes de résolution d'un problème d'ordonnancement : le premier type est exacte et le deuxième type est approché.

2.2 Les méthodes exactes

Une méthode exacte permet de trouver une solution optimale à un problème donné. Ces méthodes peuvent devenir rapidement couteuses en temps d'exécution, notamment pour les problèmes ***NP***-difficiles. Les techniques de résolution exacte sont définies pour les problèmes combinatoires en général. Ce sont des méthodes d'énumération implicite.

L'énumération explicite construit toutes les solutions réalisables et retient une parmi les meilleures. L'énumération implicite consiste à explorer l'ensemble de toutes les solutions réalisables en éliminant des sous-ensembles de solutions moins intéressant sans avoir à les construire. Nous pouvons citer quelques méthodes exactes particulièrement célèbres :

la programmation dynamique introduite par Bellman [12] dans les années ***50***, la méthode par séparation et évaluation Branch and Bound (*B&B*) et l'algorithme de retour arrière d'après Backtracking.

2.2.1 La méthode de Branch and Bound

La méthode de Branch and Bound ou la méthode ***SEP*** consiste à placer progressivement les tâches sur les ressources en explorant un arbre de recherche décrivant toutes les combinaisons possibles. Il s'agit de trouver la meilleure configuration donnée de manière à élaguer les branches de l'arbre qui conduisent à de mauvaises solutions.

En effet la séparation procède à une décomposition de l'ensemble global des solutions en plusieurs sous-ensembles, telle que tous les sous-ensembles forment un recouvrement ou une partition de l'ensemble initial. Ainsi, en calculant la solution de chaque sous-ensemble puis en choisissant la meilleure on est assuré d'avoir résolu le problème initial. Si une solution d'un sous ensemble est difficile à trouver, on procède alors à la séparation de cet sous-ensemble ainsi de manière récursive, jusqu'à aboutir [1] :

- Soit à un sous-ensemble dont la solution optimale est facile à calculer.
- Soit à un sous-ensemble de solutions non intéressant à explorer.

Quant à l'évaluation, l'objectif principal est d'éviter l'énumération systématique de toutes les solutions, par l'établissement de l'optimum d'un sous-ensemble de solutions ou de montrer que cet ensemble ne contient pas de solution intéressante [1].

Dans les deux cas il est inutile d'effectuer la séparation de l'espace des solutions. Cet algorithme accomplit une recherche complète de l'espace des solutions d'un problème donné, pour découvrir la meilleure solution.

La démarche de l'algorithme Branch and Bound consiste à :

- Diviser l'espace de recherche en sous espace.
- Chercher une borne minimale en termes de fonction objectif associée à chaque sous espace de recherche.
- Eliminer les mauvais sous-espaces.
- Reproduire les étapes précédentes jusqu'à l'obtention de l'optimum global.

Exemple 2.2.1

Soit le problème $1//\sum w_i(c_i - d_i)$ *qui consiste à exécuter* n *tâches sur une machine [9]. Une tâche de durée* p_i *doit terminer son exécution avant l'instant* d_i *sinon il faut payer une pénalité de retard* w_i *par unité de temps. Le critère à minimiser est la somme des pénalités de retard. L'ensemble des solutions réalisables* S_0 *est formé des* $n!$ *permutations possibles des* n *tâches.*

La séparation se fera sur la tâche que l'on fixe en dernier, puis en avant dernier et ainsi de suite. Pour le calcul des bornes ou minorants, on prend en considération la somme des pénalités des tâches placées en queue. L'exploration est en profondeur en suivant le sommet ayant le plus petit minorant. Le tableau suivant contient les données pour 3 *tâches.*

	T_1	T_2	T_3
p_i	2	4	1
d_i	3	5	5
w_i	4	2	3

Tableau 2.1 : Données du problème

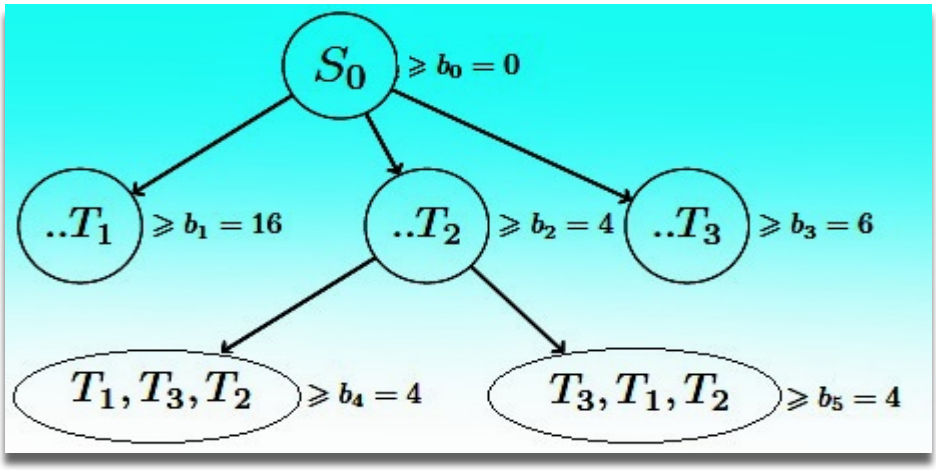


Figure 2.2.1 : Arborescence de séparations pour $1//\sum w_i(c_i - d_i)$

Remarque 2.2.1 *Au pire des cas, aucune tâche n'est en retard, d'où* $\boldsymbol{b_0 = 0}$*. On a séparé par rapport à une tâche placée en dernière position. Pour calculer* $\boldsymbol{b_1}$*, quel que soit la séquence* $(\boldsymbol{T_2},\boldsymbol{T_3},\boldsymbol{T_1})$ *ou* $(\boldsymbol{T_3},\boldsymbol{T_2},\boldsymbol{T_1})$*, la tâche* $\boldsymbol{1}$ *ne peut commencer son exécution qu'à la date* $\boldsymbol{p_2 + p_3 = 5}$ *. Elle sera en retard de* $\boldsymbol{4}$ *unités. Le coût de retard sera d'au moins* $\boldsymbol{4 \times 4 = 16}$ *d'où* $\boldsymbol{b_1 = 16}$*. Ainsi de suite* $\boldsymbol{b_2 = 2 \times 2}$ *car la tâche* $\boldsymbol{2}$ *sera en retard si elle est placée la dernière. Si la tâche* $\boldsymbol{3}$ *est placée la dernière, elle sera en retard de deux unités,* $\boldsymbol{b_3 = 3 \times 2 = 6.}$ *On explore suivant le sommet de* $\boldsymbol{b_2}$*. Les séquences* $(\boldsymbol{T_1},\boldsymbol{T_3},\boldsymbol{T_2})$ *et* $(\boldsymbol{T_3},\boldsymbol{T_1},\boldsymbol{T_2})$ *sont optimales et ont des pénalités de* $\boldsymbol{4}$ *unités. Les sommets de* $\boldsymbol{b_1 = 16}$ *et* $\boldsymbol{b_3 = 6}$ *sont alors éliminés.*

2.2.2 La programmation dynamique

La programmation dynamique est une technique mathématique qui aide à prendre des décisions séquentielles indépendantes les unes des autres. Il n'ya pas de formalisme mathématique standard, c'est une approche de résolution où les équations doivent être spécifiées selon le problème à résoudre. Elle se base sur le principe de Bellman [4] : « Si $\boldsymbol{C}$ est un point qui appartient au chemin optimal entre $\boldsymbol{A}$ et $\boldsymbol{B}$, alors la portion de ce même chemin allant de $\boldsymbol{A}$ à $\boldsymbol{C}$ est le chemin optimal entre $\boldsymbol{A}$ et $\boldsymbol{C}$ ».

C'est une méthode qui consiste donc à construire d'abord les sous-chemins optimaux et ensuite par récurrence le chemin optimal pour le problème entier, elle est utile aussi pour obtenir une solution exacte à un problème algorithmique, là où une solution « classique » se trouve être trop complexe c'est-à-dire trop peu efficace. On parle alors d'optimisation combinatoire.

On peut construire un algorithme pseudo-polynomial pour le problème $\boldsymbol{Pm\ //C_{max}}$, voici les variables booléennes suivantes pour l'algorithme [32]:

$$X_i(t_1, t_2, \dots t_m) = \begin{cases} V & \text{si les tâches } T_1, T_2, \dots, T_n \text{ peuvent être ordonnancées sur les machines } M_1, \dots, M_m \text{ telle que la machine } M_j \text{ est occupée pendant l'intervalle de temps } [0, t_j] \text{ (Sans temps morts).} \\ F & \text{sinon} \end{cases}$$

Ou $\boldsymbol{t_j} \in \{\boldsymbol{0}, \boldsymbol{1}, \dots, \boldsymbol{C}\}$ et $\boldsymbol{C}$ est une borne supérieure de $\boldsymbol{C_{max}}$, on peut prendre $\boldsymbol{C} = \sum_{i=1}^{n} \boldsymbol{p_i}$

Algorithme 2.2.1 *programmation dynamique*

Début

pour tout $(t_1, t_2, \dots, t_m) \in \{0,1, \dots, C\}^m$ *faire*

$X_i(t_1, t_2, \dots, t_m) := F$;

$X_0(0, 0, \dots, 0) := V$;

pour i := 1 haut n faire

pour tout $(t_1, t_2, \dots, t_m) \in \{0,1, \dots, C\}^m$ *faire*

$X_i(t_1, t_2, \dots, t_m) := \bigvee_{j=1}^{m} X_{i-1}(t_1, t_2, \dots, t_j - p_i, t_{j-1}, \dots, t_m)$;

$C_{max} := min(max(t_1, t_2, \dots, t_m) / X_n(t_1, t_2, \dots, t_m) = V)$;

Fin.

Exemple 2.2.2 *On considère l'ordonnancement de* **3** *tâches* T_1, T_2, T_3 *de temps de traitement respectifs* **2**, **1** *et* **2** *, sur* **2** *machines* M_1 *et* M_2*. Nous avons* **C=5** *le déroulement de l'algorithme nous donne :*

$i = 0, X_0(t_1, t_2) =$

t_2/t_1	0	1	2	3	4	5
0	v					
1						
2						
3						
4						
5						

$i = 1, X_1(t_1, t_2) =$

t_2/t_1	0	1	2	3	4	5
0			v			
1						
2	v					
3						
4						
5						

$i = 2, X_2(t_1, t_2) =$

t_2/t_1	0	1	2	3	4	5
0				v		
1			v			
2		v				
3	v					
4						
5						

$i = 3, X_3(t_1, t_2) =$

t_2/t_1	0	1	2	3	4	5
0						v
1					v	
2				v		
3			v			
4		v				
5	v					

Nous avons $X_3(3,2) = V$ et $C_{max} = \max\{3,2\} = 3$.

$X_3(3,2) = X_2(1,2) \vee X_2(3,0)$; $(X_2(1,2) = V \text{ et } X_2(3,0) = V)$
Prenons $X_2(3,0) = V$ (La tâche T_3 est affectée à la machine M_2)

$X_2(3,0) = X_1(2,0) \vee X_1(3,-2)$; $(X_1(2,0) = V \text{ et } X_1(3,-2) = F)$
Prenons $X_1(2,0) = V$ (La tâche T_2 est affectée à la machine M_1)

$X_1(2,0) = X_0(0,0) \vee X_0(2,-2)$; $(X_0(0,0) = V \text{ et } X_0(2,-2) = F)$
Prenons $X_0(0,0) = V$ (La tâche T_1 est affectée à la machine M_1)

L'ordonnancement optimal est donc:

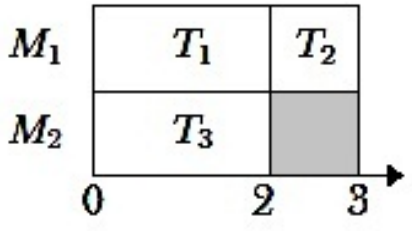


Figure 2.2.2 : Ordonnancement optimal de l'exemple 2.2.2

2.2.3 La programmation par contrainte

La programmation par contrainte [5], [27] est utilisée pour résoudre plusieurs types de problèmes tels que la planification, l'ordonnancement,..,etc. Dans la ***PPC***, le problème est modélisé à l'aide de variables de décisions et de contraintes, où une contrainte est une relation entre une ou plusieurs variables qui limite les valeurs que peuvent prendre simultanément chacune des variables qui est liée à cette contrainte. Un domaine est associé à chaque variable et celui ci est constitué de l'ensemble des valeurs pouvant être affectées à cette variable. Lorsque le

domaine d'une variable est réduit à une valeur, on dit que la variable est instanciée. Un ensemble de variables instanciées est une instanciation. Une solution au problème est alors définie comme une instanciation des variables qui satisfait la conjonction de toutes les contraintes.

Les algorithmes de recherche de solutions en ***PPC*** s'appuient généralement sur la propagation de contraintes pour réduire l'espace de recherche de solutions. La propagation de contraintes est le mécanisme qui consiste à déduire de nouvelles contraintes à chaque fois qu'un domaine de variable est modifié. Les algorithmes de recherche de solutions garantissent de trouver une solution, si toutefois elle existe, sinon ils peuvent prouver qu'il n'existe pas une solution qui satisfait toutes les contraintes.

Cette technique de programmation par contraintes a été utilisée dans plusieurs travaux [27], [38], [20], [15] pour résoudre le problème d'ordonnancement temps réel multiprocesseur.

2.2.4 La programmation linéaire

La programmation linéaire [45], [19] permet de résoudre les problèmes d'optimisation dont la fonction objective et les contraintes sont linéaires.
C'est un domaine central dans l'optimisation car les problèmes de programmation linéaire sont les problèmes d'optimisation les plus faciles (toutes les contraintes ***y*** sont linéaires). Dans un problème de ***PL***, il est toujours question de minimiser ou de maximiser une fonction objective exprimée à l'aide de variables de décisions.

Les solutions à trouver doivent être représentées par des variables réelles. S'il est nécessaire d'utiliser des variables discrètes dans la modélisation, on parle de programmation linéaire en nombre entiers. Un problème de ***PLNE*** est un problème de ***PL*** dans lequel on ajoute la contrainte supplémentaire que certaines variables ne peuvent prendre que des valeurs entières. La programmation linéaire mixte ***PLM***, permet de traiter des problèmes dans lesquels les variables sont mixtes (entières et réelles) [2].

Une formulation sous la forme de ***PLNE*** est présentée dans l'article [44] pour le problème d'ordonnancement des tâches temps réel dépendantes dans un système multiprocesseur homogène. Khachiyan c'est lui qui a permis de démontrer que le problème de la ***PL*** était polynomial.

2.3 Les méthodes approchées

L'objectif d'une méthode approchée est d'obtenir une solution exécutable, prenant en considération la fonction objectif mais sans garantie d'optimalité. Son utilisation offre de multiples avantages par rapport à une méthode exacte [33].

2.3.1 Les heuristiques

Les heuristiques sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables. Elles se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de cette catégorie de méthodes est d'intégrer des stratégies de décision pour construire une solution proche de celle optimale tout en cherchant à avoir un temps de calcul raisonnable [30]. Parmi ces stratégies, nous distinguons :

- **FIFO** : où la première tâche arrivée est la première à être ordonnancée.
- **SPT** : où la tâche ayant le temps de traitement le plus court est traitée en premier [10].
- **LPT** : où la tâche ayant le temps de traitement le plus important est traitée en premier [10].
- **EDD** : où la tâche ayant la date échue la plus petite est la plus prioritaire [10].
- **LLF** : cet algorithme choisit parmi les tâches exécutables celle dont la distance entre la fin de son exécution et son délai est le plus court. Si aucune tâche n'est disponible, alors un temps libre est généré c'est-à-dire est un algorithme de planification dynamique des priorités au niveau des tâches. Cela signifie que chaque instant est événement de planification, car la laxité de chaque tâche change à chaque instant. Une tâche qui à le moins de laxisme à un instant, aura une priorité plus élevée que les autres à cet instant plus formellement, la priorité d'une tâche est inversement proportionnelle à la laxité de son exécution. Comme le laxisme d'une tâche est défini comme son urgence à exécuter [10].
- **HPF** : cet algorithme choisit parmi les tâches exécutables celle dont la priorité est la plus grande, c'est-à-dire l'attribution de priorité à chaque processus, l'ordonnanceur lance le processus prêt de la priorité la plus élevée. En cas de priorités égales on utilise la règle d'algorithme ***FIFO***, et si aucune tâche n'est disponible, alors un temps libre est généré [10].

2.3.2 Les méta-heuristiques

Le mot métaheuristique est dérivé de la composition de deux mots grecs [25]:

- Méta : qui est un suffixe signifiant ''dans un niveau supérieur''.
- Heuristique : qui signifie ''trouver''.

Une méta-heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème considéré, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible [37].
Les méta-heuristiques sont ainsi des méthodes de recherche générales, dédiées aux problèmes d'ordonnancement difficile. Elles sont, en général, présentées sous forme de concepts, Le but visé par les méta-heuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.

Afin de mieux clarifier les types des méthodes de résolution, on propose le schéma suivant :

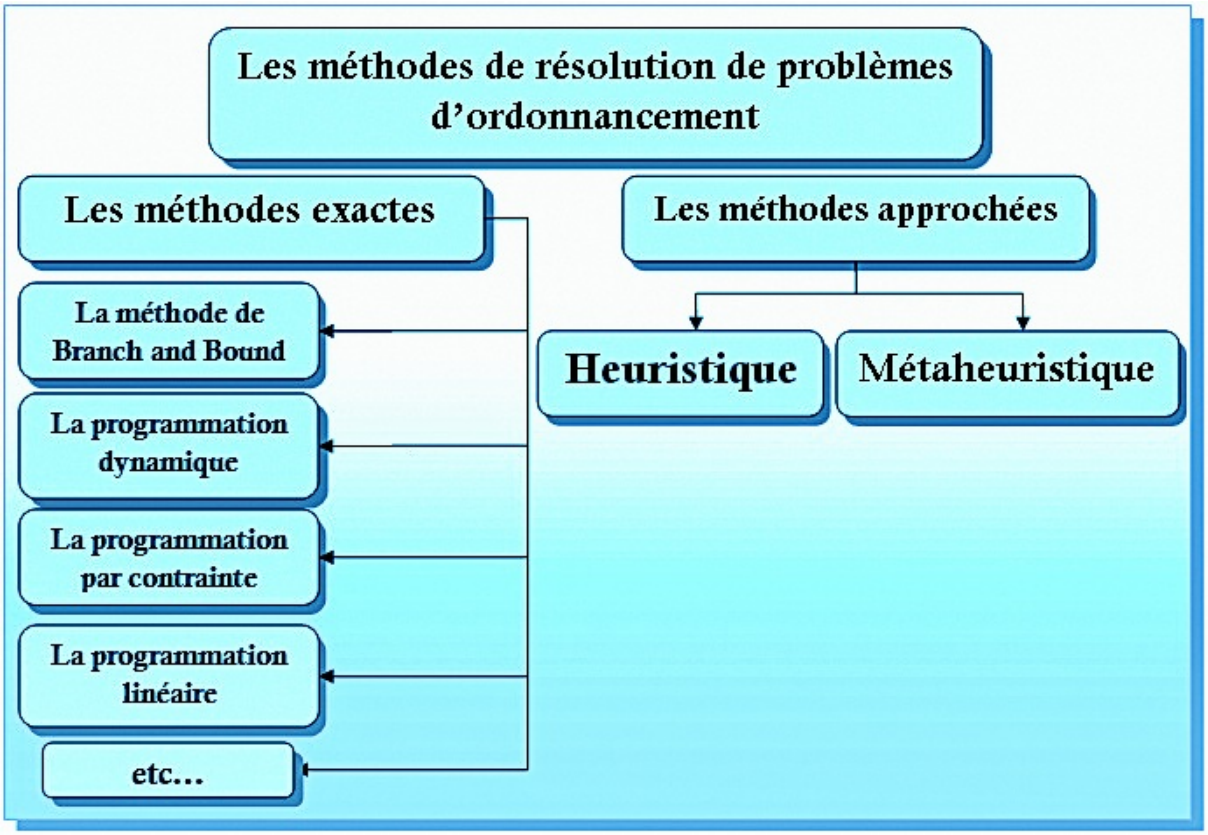


Figure 2.3.1 : Classification des méthodes de résolution des problèmes d'ordonnancement

Remarque 2.3.1 *On peut regrouper les méta-heuristiques en deux grandes classes : les méta-heuristiques à solution unique (c'est-à-dire évoluant avec une seule solution) et celles à solutions multiples ou population de solutions. L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité.*

2.4 Les méta-heuristiques à solution unique

Dans cette section, nous présentons les méta-heuristiques à base de solution unique, aussi appelées méthodes de trajectoire. Contrairement à la méta-heuristique à base de population, les méta-heuristiques à solution unique commencent avec une seule solution initiale et s'en éloignent progressivement, en construisant une trajectoire dans l'espace de recherche. Les méthodes de trajectoire englobent essentiellement **la méthode de descente**, **le recuit simulé** et **la recherche TABOU**.

2.4.1 La méthode de descente

Cette méthode s'articule toute autour d'un principe simple [14]. A partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante. L'algorithme suivant présente le squelette d'une méthode de descente simple.

A partir d'une solution initiale $\boldsymbol{x}$, on choisit une solution $\boldsymbol{x'}$ dans le voisinage $\boldsymbol{N(x)}$ de $\boldsymbol{x}$. Si cette solution est meilleure que $\boldsymbol{x}$, $\boldsymbol{(f(x') < f(x))}$ alors on accepte cette solution comme nouvelle solution $\boldsymbol{x}$ et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de $\boldsymbol{x}$.

Algorithme 2.4.1 *Méthode de descente simple*

Début
  Initialisation : trouver une solution initiale x
  Répéter
    Recherche d'un voisinage : trouver une solution $x' \in N(x)$
    Si $f(x') < f(x)$ *alors*
      $x' \leftarrow x$
    Fin si
  Jusqu'à $f(y) \geq f(x), \forall\, y \in N(x)$
Fin.

2.4.2 Le recuit simulé

Inspiré du recuit physique, ce processus est utilisé en métallurgie pour améliorer la qualité d'un solide et cherche un état d'énergie minimale qui correspond à une structure stable du solide. Ainsi, pour qu'un métal retrouve une structure proche du cristal parfait, on porte celui-ci à une

température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement.

L'algorithme du recuit simulé [31], permet de résoudre les problèmes de minimaux locaux. En effet, une nouvelle solution de coût supérieur à celui de la solution courante ne sera pas forcément rejetée, son acceptation sera déterminée aléatoirement en tenant compte de la différence entre les coûts ainsi que du facteur température T.

Ce paramètre, sert à prendre en compte le fait que plus le processus d'optimisation est avancé, moins on est prêt à accepter une solution plus coûteuse. Par contre, l'acceptation de solutions fortement coûteuses permet d'explorer l'espace des solutions possibles et ainsi, d'accroître les chances d'approcher le minimum global.

Kirkpatrick [31] et Cenry [13] se sont inspirés d'une telle technique pour résoudre des problèmes d'optimisation combinatoire. Le voisinage $N(s)$ d'une solution, s'apparente à l'ensemble des états atteignables depuis l'état courant, en faisant subir des déplacements aux atomes du système physique.

A chaque itération, une seule solution voisine est générée. Celle-ci est acceptée si elle est meilleure que la solution courante.

Dans le cas contraire, la nouvelle solution est acceptée avec une certaine probabilité qui dépend de l'importance de la détérioration et du paramètre T correspondant à la température. En règle générale, la température est diminuée par paliers, à chaque fois qu'un certain nombre d'itérations est effectué. La meilleure solution trouvée est mémorisée.

L'algorithme est interrompu lorsqu'aucune solution voisine n'a été acceptée pendant un cycle complet d'itérations à température constante [46].

Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coûts plus faibles tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût [17], [26]. Ainsi, si une amélioration du critère est constatée, le nouvel état est retenu, sinon une diminution ΔE du critère est calculée et le nouvel état est retenu si p étant un nombre tiré de façon aléatoire entre 0 et 1, on a $e^{(\Delta E/T)} > p$.

Cet algorithme est présenté dans l'organigramme suivant :

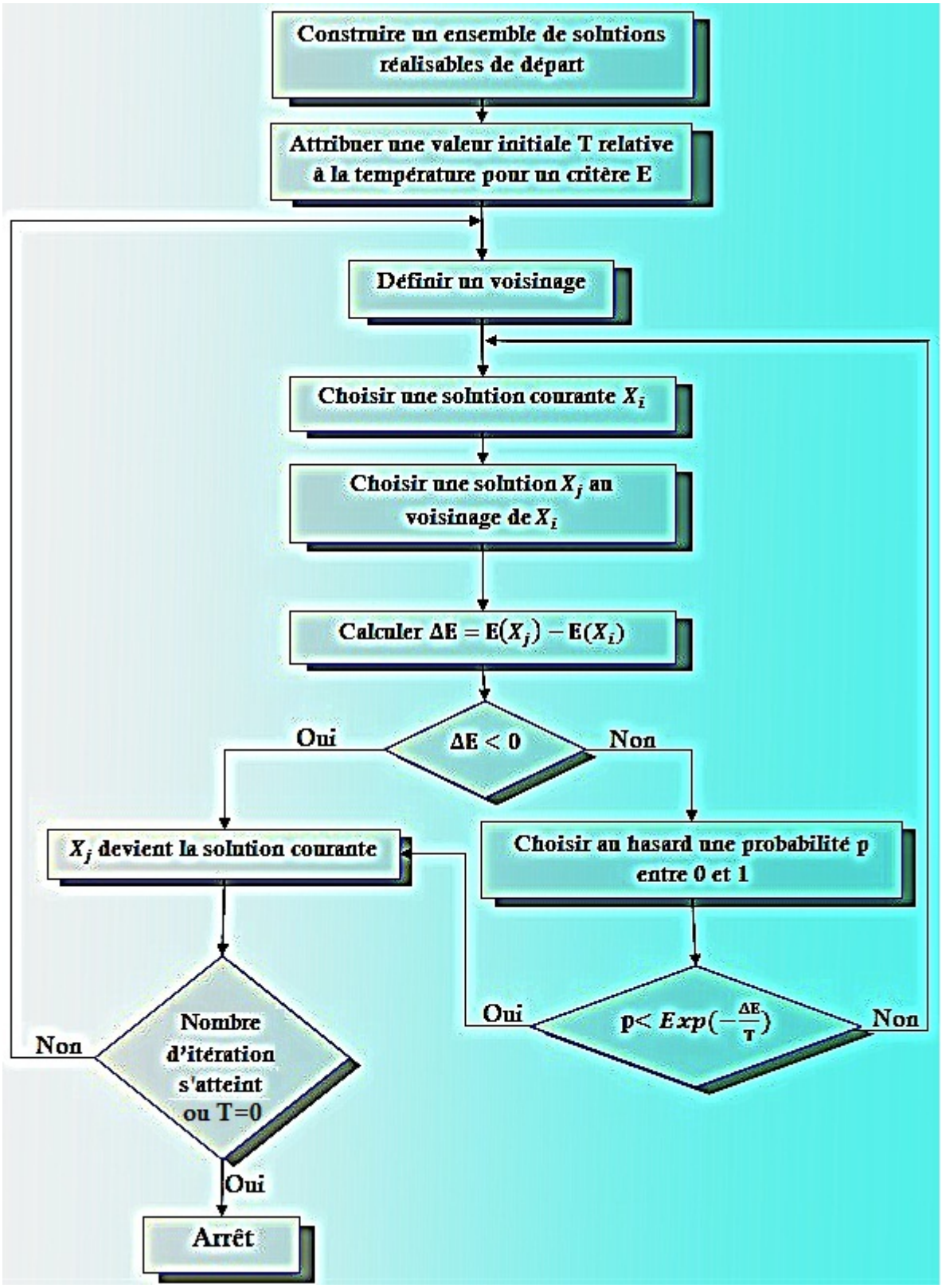


Figure 2.4.1 : Organigramme relatif au fonctionnement général du recuit simulé

2.4.3 La recherche TABOU

La recherche TABOU est une méta-heuristique originalement développée par Glover 1986 et indépendamment par Hansen 1986 [23]. Elle est basée sur des idées simples, mais elle est néanmoins très efficace.

Cette méthode combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes permettant à celle-ci de surmonter l'obstacle des optimums locaux, toute n évitant de cycler. Elle a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire : problèmes de routage de véhicule, problèmes d'ordonnancement, problèmes de coloration de graphes, etc.

Dans une première phase, la méthode de recherche Tabou peut être vue comme une généralisation des méthodes d'amélioration locales. En effet, en partant d'une solution quelconque $\boldsymbol{s_0}$ appartenant à l'ensemble de solutions $\boldsymbol{S}$, on se déplace vers une solution $\boldsymbol{s}'$ située dans le voisinage $\boldsymbol{N(s)}$. Donc l'algorithme explore itérativement l'espace de solutions $\boldsymbol{S}$.

Principe 2.4.1 *Le principe de l'algorithme de la recherche TABOU se compose des étapes suivantes :*

- *Concevoir une méthode de recherche intelligente.*
- *Une position donnée.*
- *Explorer l'ensemble des voisins.*
- *Choisir le meilleur voisin.*
- *Minimiser la fonction objective.*

Algorithme 2.4.2 *La recherche TABOU*

Début

1. *Initialisation*

s_0 *une solution initiale.*

$s \leftarrow s_0$; $s^* \leftarrow s_0$; $c^* \leftarrow f(s_0)$;

$T = \emptyset$

2. *Générer un sous-ensemble de solution au voisinage de* s

$s' \in N(s)$ *tel que* $\forall\, x \in N(S),\ f(x) \geq f(s')$ *et* $s' \notin T$

Si $f(s') < C^*$ *alors* $s^* \leftarrow s'$ *et* $c^* \leftarrow f(s')$;

Mise-à-jour de T ;

3. *Si la condition d'arrêt n'est pas satisfaite retour à l'étape* ***2***

Fin.

2.5 Les méta-heuristiques à population de solutions

Les méthodes d'optimisation à population de solutions améliorent, Avec répétition continue, un ensemble de solutions.

2.5.1 Les algorithmes génétiques

Les Algorithmes génétiques sont fondés sur les principes de l'évolution biologique des espèces. Ils permettent de trouver au bout d'un temps raisonnable des solutions pour des problèmes de complexité élevée.

Les algorithmes génétiques sont basés sur une notion étendue de voisinage. Plusieurs solutions sont maintenues simultanément et évoluent par des techniques de recombinaison et de mutation. Une des difficultés rencontrée est le codage des solutions.

En effet ces algorithmes sont des algorithmes itératifs [28] dont le but est d'optimiser une fonction prédéfinie, appelée « fitness ». Pour réaliser cet objectif, l'algorithme travaille sur un ensemble de points, appelés population d'individus. Chaque individu ou chromosome (chaîne binaire de longueur finie) représente une solution possible du problème donné. Il est constitué d'éléments appelés gènes, pouvant prendre plusieurs valeurs, appelées allèles [35].

Une fonction de coût est utilisée afin d'évaluer les individus et de sélectionner les meilleurs pour constituer une autre population, à laquelle on appliquera de nouveau la sélection et la reproduction. Ce processus est réitéré jusqu'à ce qu'un critère d'arrêt soit atteint. La reproduction des individus est réalisée avec le croisement et la mutation. Le croisement consiste à couper deux individus en un point qui peut être choisi aléatoirement et à échanger les parties coupées. La mutation, quand à elle, modifie les gènes des individus afin de produire d'autres individus (par exemple en ordonnancement on crée d'autres solutions à partir d'une solution existante en modifiant l'allocation des tâches dans cette solution).

Les algorithmes génétiques sont utilisés pour résoudre le problème d'ordonnancement temps réel multiprocesseur avec des contraintes de précédences.

Principe 2.5.1 *Le principe des algorithmes génétiques est basé sur :*

- *Calcul de la population initiale et construire une population de solutions réalisables.*
- *Calcul des nouveaux individus, Accroître la population en ajoutant des individus issus de mutations, ou de croisements d'individus de la génération précédente.*

- *Sélection des individus, Sélectionner les meilleurs individus pour revenir à la taille de la population initiale.*
- *Condition d'arrêt, Recommencer les croisements et les mutations jusqu'à vérifier une condition d'arrêt.*

2.5.2 L'optimisation par colonie de fourmis

Les algorithmes d'optimisation par colonie de fourmis sont nés à la suite de constatations faites sur le comportement des fourmis qui sont capables de trouver le chemin le plus court, du nid à une source de nourriture, et de s'adapter aux changements de l'environnement et ceci grâce à la phéromone [48].

L'optimisation par colonie de fourmis est un algorithme récent, introduit et utilisé pour résoudre le problème d'ordonnancement dans des ateliers de type job shop, Cet algorithme est dérivé du comportement de fourmis, il est proposé pour résoudre des problèmes ***NP***-difficiles. Il a fait preuve d'une rapidité de convergence, lui permettant d'être appliqué à plusieurs classes de problèmes d'optimisation. Dans une telle démarche algorithmique, les paramètres de réglage jouent un rôle crucial dans l'efficacité et influencent sur la qualité des solutions. ***OCF*** présente une comparaison de la qualité de chacune des solutions obtenues en fonction du réglage de la quantité de phéromone déposée et de la visibilité pour les problèmes d'ordonnancements.

Ainsi, les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation des sources de nourriture.

L'exemple illustré dans les figures 2.5.1, 2.5.2 et 2.5.3 montre la capacité de cet algorithme à se rapprocher de la solution optimal.

Au départ, les fourmis disposent d'un chemin pour aller de leur nid, la fourmilière, à la source de nourriture figure 2.5.1 :

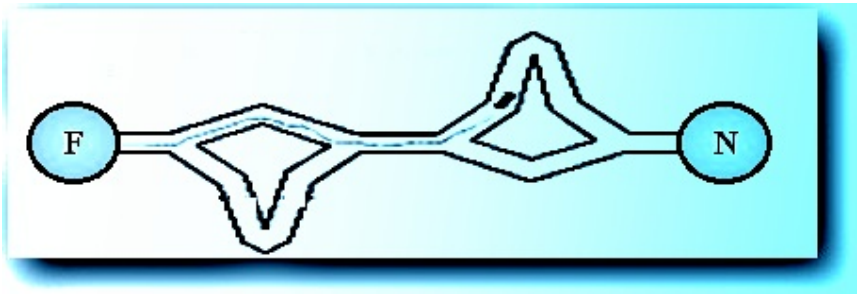


Figure 2.5.1 : Déplacement des fourmis vers une source de nourriture

Un obstacle est placé entre le nid et la source de nourriture, les fourmis vont alors commencer par se séparer de part et d'autre de l'obstacle en déposant de la phéromone sur leur passage. La figure 2.5.2. Les fourmis les plus rapidement arrivées au nid, après avoir visité la source de nourriture, sont celles qui ont emprunté les deux branches les plus courtes à l'aller et au retour.

Ainsi la quantité de phéromone présente sur le plus court trajet est plus importante que celle présente sur le chemin le plus long.

Une piste présentant une plus grande concentration en phéromone, étant plus attirante pour les fourmis, a donc une probabilité plus grande d'être empruntée.

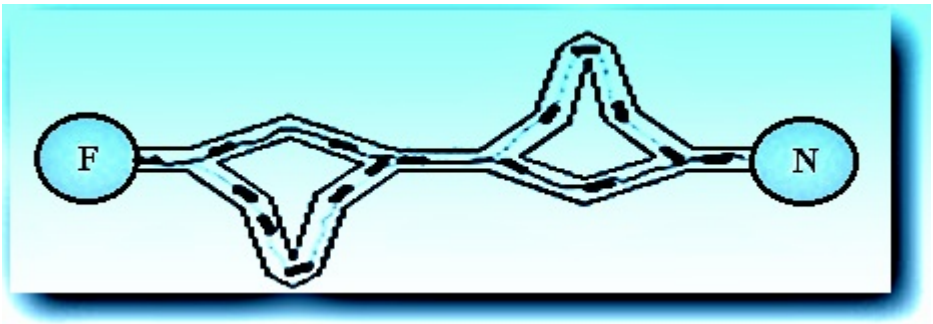


Figure 2.5.2 : Déplacement des fourmis après placement d'un obstacle sur leur chemin

Finalement la quantité de phéromone sur le chemin le plus court fait que ce chemin finit par être emprunté par la plupart des fourmis, voir la figure suivant :

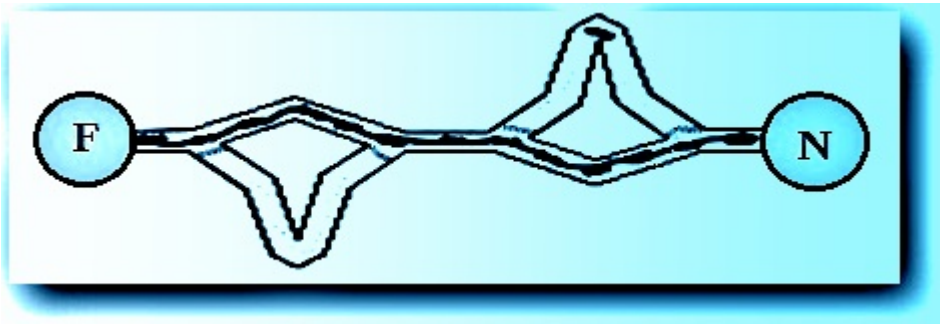


Figure 2.5.3 : Choix du chemin le plus court par la plupart des fourmis

Algorithme 2.5.1 *L'algorithme d'OCF :*

```
Début
    Initialiser une population de m fourmis ;
    Evaluer les m fourmis ;
      Tant que  la condition d'arrêt n'est pas satisfaite faire
          Pour  i=1 à m  faire
              Construire le trajet de la fourmi i ;
              Déposer des phéromones sur le trajet de la fourmi i ;
          Fin pour
        Evaluer les m fourmis;
       Evaporer les pistes de phéromones ;
      Fin Tant que
    Retourner la ou les meilleures solutions ;
Fin.
```

Afin de mieux comprendre, voici ce schéma qui résume la classification des méta-heuristiques :

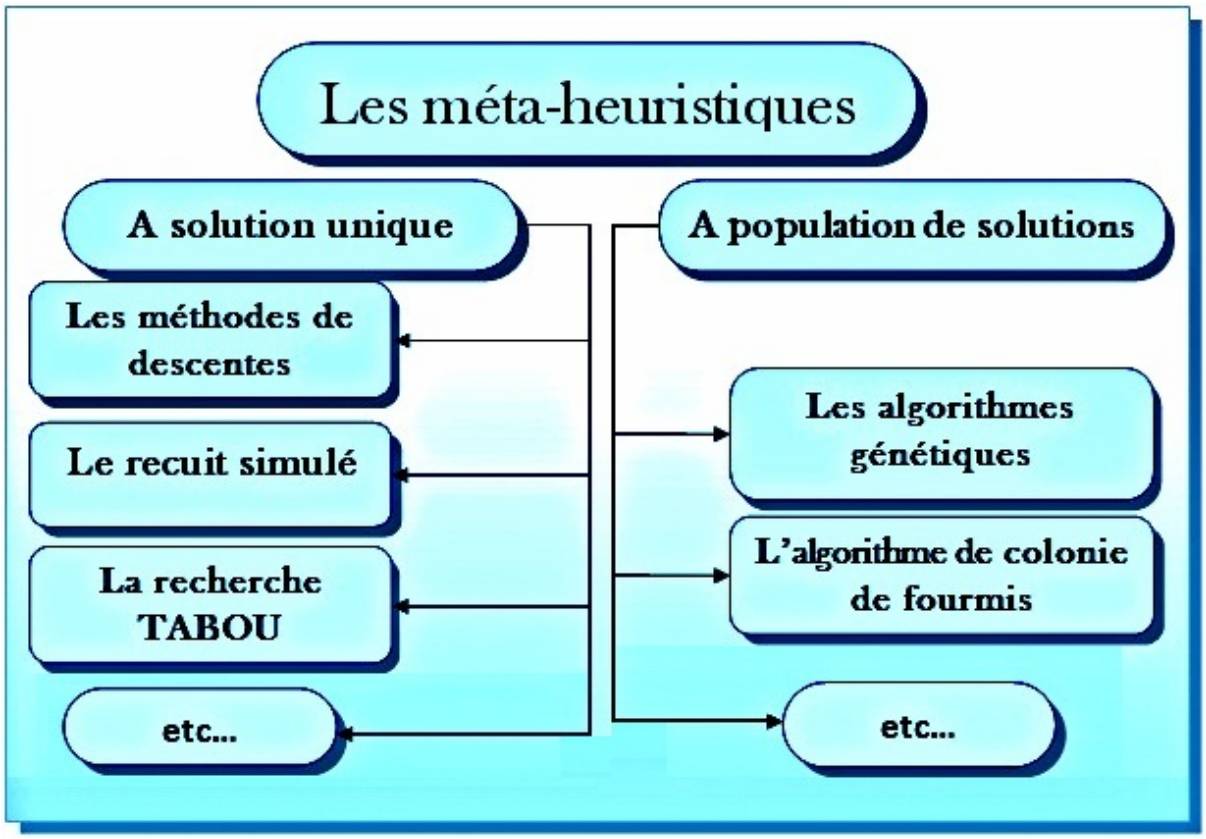


Figure 2.5.4 : Classification des méta-heuristiques

2.6 Les avantages et les inconvénients des heuristiques et méta-heuristiques

2.6.1 Les avantages des heuristiques

- La probabilité d'obtenir une solution de mauvaise qualité est faible.
- Robustesse : elle fournit le plus souvent une solution proche de l'optimum.
- Elles sont plus souples dans la résolution des problèmes réels.
- Décrire la qualité d'une solution au problème.
- Elles sont simples à mettre en œuvre.

2.6.2 Les inconvénients des heuristiques

- Elles ne garantissent pas la qualité du point obtenu.
- Sont des méthodes approchées, en tout cas très proches de l'optimum dans un temps raisonnable.
- Sont des méthodes satisfaites pour des cas particuliers (si on change l'exemple, il pourra de perdre la qualité de solution obtenue).

2.6.3 Les avantages des méta-heuristiques

- Progressent vers un optimum global.
- Décrire la qualité d'une solution au problème.
- Faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.
- Explorer l'espace de recherche efficacement et de déterminer des points (presque) optimaux.

2.6.4 Les inconvénients des méta-heuristiques

- Simples mais performance médiocres.
- Sont en général non déterministes et ne donnent aucune garantie d'optimalité.
- Elles ne garantissent pas non plus la découverte de l'optimum global en un temps fini.

2.7 La comparaison entre les méthodes exactes et les méthodes approchées

Exploration de S	***Systématique (intelligente)***	***Partielle (astucieuse)***
Méthode/solution	Exacte	approchée
caractéristique	Optimalité	approximation
aspect	Déterministe	Stochastique (guidée)
complexité	$\varphi(2^n)$ souvent	$\varphi(n^k)$
Handicap	Explosion combinatoire	Convergence précoce
Adaptée aux POCs	De petites tailles	*NP*-difficiles de grandes tailles

Tableau 2.2 : Table de comparaison entre méthode exacte et méthode approchée [1]

2.8 Conclusion

Dans ce chapitre nous avons présenté les principes et les algorithmes de différentes méthodes de résolution de problèmes d'optimisation commençant par les méthodes exactes ensuite les méthodes approchées. Parmi les méthodes exactes, nous avons distingué la programmation linéaire, la programmation dynamique, la méthode Branch and Bound et la programmation par contrainte.

Parmi les méthodes approchées, nous avons présenté les heuristiques et les méta-heuristiques, tel que la méthode de descente, le recuit simulé, la recherche TABOU et la recherche à voisinage variables qui sont des méta-heuristiques à solution unique tandis que les algorithmes génétiques et les algorithmes de colonies de fourmis sont des méta-heuristiques à population de solution.

Chapitre 3

La modélisation des problèmes d'ordonnancement sur les machines parallèles identiques

Chapitre 3

La modélisation des problèmes d'ordonnancement sur les machines parallèles identiques

3.1 Introduction

Les modèles mathématiques visent à trouver des équations mathématiques pour décrire les données, les contraintes ainsi que les critères utilisés pour l'optimisation des solutions.

L'avantage de ce type de modélisation est d'obtenir des expressions mathématiques simples et robustes, qui peuvent être exploitées facilement en programmation. Dans ce chapitre, nous allons modéliser quelques problèmes d'ordonnancement sur des machines parallèles identiques pour minimiser le C_{max} .

3.2 Les problèmes d'ordonnancement sur des machines parallèles identiques

3.2.1 Le Makespan C_{max}

Le Makespan représente le temps de fin de traitement de la dernière tâche dans une séquence. Il est l'un des critères les plus utilisés pour évaluer le coût d'un ordonnancement. En minimisant ce critère, on peut améliorer le rendement et réduire le temps moyen d'inactivité des machines [21].

3.2.2 La modélisation

La modélisation, est en général, une étape très importante dans la résolution d'un problème d'ordonnancement. C'est une écriture simplifiée de toutes les données du problème permettant d'en traduire tous les détails pour mieux représenter la réalité des choses.

3.2.3 La formulation

La formulation mathématique consiste à déterminer la fonction objectif : c'est la fonction de coût, notée f qu'il s'agit d'optimiser, les variables de décision : elles sont regroupées dans un vecteur, c'est en faisant varier ce vecteur que l'on recherche un optimum de la fonction f [17], et la formulation consiste aussi sur les contraintes.

3.2.4 Les problèmes mono-objectifs

D'un point de vue mathématique, un problème d'optimisation mono-objectif est formulé de la façon suivante [8] :

$$\begin{cases} \textit{Minimiser } f(x) & \textit{fonction à optimiser} & x \in \mathbb{R}^n \text{ , } f(x) \in \mathbb{R} \\ \textit{avec } g(x) \leq 0 & \textit{m contraintes d'inégalités} & g(x) \in \mathbb{R}^m \\ \textit{et } \quad h(x) = 0 & \textit{n contraintes d'égalités} & h(x) \in \mathbb{R}^n \end{cases}$$

3.3 Minimisation du Makespan avec l'ordonnancement de type préemptif

3.3.1 Le problème $(P/pmtn/C_{max})$

Définition du problème

Est un problème qui traite un ensemble de tâches indépendantes avec préemption ayant des temps de traitement arbitraire et des dates de disponibilité nulle sur des machines parallèles identiques [32]. L'objectif de ce problème est de minimiser la date de fin de traitement.

La modélisation du problème

$$\begin{cases} \textbf{Min } y \\ \sum_{j=1}^{m} x_{ij} \leq p_i \qquad i = 1, \dots, n & (1) \\ \sum_{j=1}^{m} x_{ij} \leq y \qquad i = 1, \dots, n & (2) \\ \sum_{i=1}^{n} x_{ij} \leq y \qquad j = 1, \dots, m & (3) \\ x_{ij} \geq 0 \quad y \geq 0 \qquad i = 1, \dots, n \quad j = 1, \dots, m \\ \text{ou } y \text{ est la date de fin de traitement} \end{cases}$$

- La variable x_{ij} représente le temps total que la tâche i passe sur la machine j.
- Le premier ensemble de contraintes garantit que chaque tâche reçoit la quantité de traitement requise.
- Le deuxième ensemble de contraintes garantit que le nombre total de traitements reçus par chaque tâche est inférieur ou égal à la valeur de la période.
- Le troisième ensemble veille à ce que la quantité totale de traitement sur chaque machine soit inférieure à la durée de fabrication, puisque C_{max} est fondamentalement une variable de décision et non un élément du vecteur de ressources du programme linéaire.

L'algorithme donné par Mc Naughton [34] et présenté ci-dessous construit un ordonnancement optimal de longueur égale à C^*_{max}. Sa complexité est en ***O(n).***

Avec :

$$C^*_{max} = max\left\{ \max_{1 \leq i \leq n} \{P_i\}, \left\lceil \frac{1}{m} \sum_{i=1}^{n} p_i \right\rceil \right\}.$$

Algorithme 3.3.1 *Pour le problème* $(P/\ pmtn/C_{max}\ )$*:*

Début
t :=0; i :=0; j :=0;
Répéter
Si $t+p_i < C^*_{max}$ *alors*
Affecter la tâche T_i *à la machine* M_j *à l'instant t;*
$t := t+p_i$ *;* $i := i+1$*;*
(La tâche T_i *est entièrement affectée à la machine* M_j *et l'affectation de la tâche suivante se poursuit sur la même machine à l'instant* $t+p_i$ *).*
Sinon
Si $t+p_i = C^*_{max}$ *alors*
Affecter la tâche T_i *à la machine* M_j *à l'instant t;*
t :=0; i :=i+1; j :=j+1;
(La tâche T_i *est entièrement affectée à la machine* M_j *et l'affectation de la tâche suivante se poursuit sur la machine suivante à l'instant 0).*
Sinon
A l'instant t, Affecter la tâche T_i *à la machine* M_j *pour* $C^*_{max} - t$ *unités de temps ;* $p_i := p_i - (C^*_{max} - t)$ *; t :=0 ; j :=j+1 ;*
(La tâche T_i *est préemptée et son affectation se poursuit sur la machine suivante à l'instant 0).*
Fin si
Fin si
Jusqu'à i=n (toute les tâches ont été affectées)
Fin.

Exemple 3.3.1 *Soit à traiter* $\boldsymbol{8}$ *tâches indépendantes* $\boldsymbol{T_1, \ldots, T_8}$ *sur trois machines parallèles identiques.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
p_i	1	2	3	4	5	6	7	8

Tableau 3.1 : Temps de traitement des tâches de l'exemple 3.3.1

L'application de l'algorithme de Mc Naughton [34] fournit l'ordonnancement optimal représenté par la figure suivante avec $\boldsymbol{C_{max} = 12}$

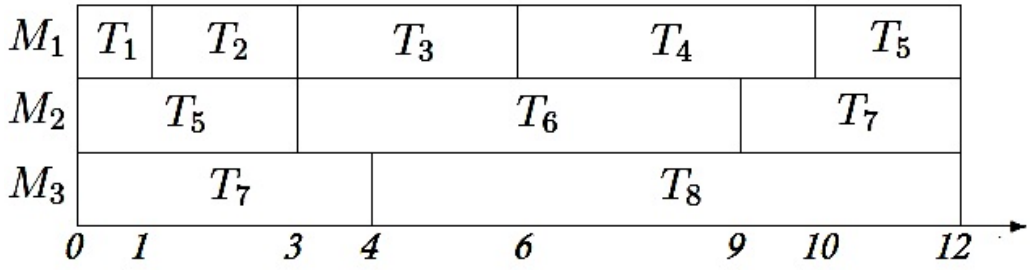


Figure 3.3.1 : Ordonnancement optimal pour les tâches de l'exemple

La tâche T_5 commence son traitement sur la machine M_2 à l'instant **0** et est interrompue à l'instant **3** pour terminer son traitement sur la machine M_1 à l'instant **10**. De même, la tâche T_7 commence son traitement sur la machine M_3 à l'instant ***0*** et est interrompue à l'instant ***4*** pour terminer son traitement sur la machine M_2 à l'instant **9**.

Les autres tâches sont traitées sans interruption.

'MacNaughton [34]' construit un ordonnancement optimal en affectant les tâches aux machines, dans un ordre quelconque, une tâche étant éclatée chaque fois que cette borne est dépassée .On réalise ainsi également un nombre de préemption au plus égal à $m - 1$.

3.4 Minimisation du Makespan avec l'ordonnancement de type non-préemptif

3.4.1 Le problème ($P//C_{max}$)

Définition du problème

Ce problème consiste à placer des tâches sur des machines parallèles identiques de sortes à minimiser la date de fin de traitement de l'ordonnancement où la préemption n'est pas autorisée [32]. Malgré son apparente simplicité, ce problème est ***NP***-difficile.

Notons que :

$$C_{max} \geq max \left\{ \max_{1 \leq i \leq n} \{P_i\}, \left\lceil \frac{1}{m} \sum_{i=1}^{n} p_i \right\rceil \right\}.$$

La modélisation du problème

Ce problème se formule comme un programme linéaire en variables bivalentes de la manière suivante :

$$\begin{cases} \boldsymbol{Min\ y} \\ \sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1, \dots, n \qquad (1) \\ \sum_{i=1}^{m} p_i x_{ij} \leq y \qquad j = 1, \dots, m \qquad (2) \\ x_{ij} \in \{0, 1\} \\ y \geq 0 \end{cases}$$

Avec :

$$x_{ij} = \begin{cases} 1 & si \quad la\ tâche\ T_i\ est\ traitée\ sur \\ & \qquad la\ machine\ M_j \\ 0 & sinon \end{cases}$$

- La première contrainte indique qu'une tâche ne peut être traitée que sur une seule machine.
- La deuxième contrainte indique que le temps de fin de traitement sur chaque machine doit être inférieur ou égal à y.

Théorème 3.4.1 *Pour le problème $P//C_{max}$ l'analyse de complexité de ce problème montre qu'il est difficile à résoudre [39].*

Preuve. Pour prouver que $P//C_{max}$ est *NP-* dur dans le sens fort, on réduit tout simplement du problème de *3*-partition. Dans ce dernier on dispose de n nombres $a_1, \dots, a_n$ et d'un entier B tel que pour tout i, $B/4 < a_i < B/2$. Le but est de partitionner $\{1,\dots, n\}$ en $m = n/3$ ensembles $S_1,\dots, S_m$, tel que pour tout j, $\sum_{k \in s_j} = B$.

On suppose que n est un multiple de **3**. Pour la réduction on construit tout simplement une instance de $P//C_{max}$ de n tâches de durées $p_i = a_i$ et on demande s'il existe un ordonnancement sur m machines de Makespan B. Clairement une tout solution à l'instance du problème d'ordonnancement correspond à une solution pour l'instance du problème de *3*-partition. Et bien sur toute solution de la *3*-partition correspond à la solution à l'instance du problème d'ordonnancement [39].

Remarque 3.4.1 *Il est facile de voir que* $\boldsymbol{P2//C_{max}}$ *est NP-difficile au sens ordinaire tel qu'il est équivalent à PARTITION au cours des deux dernières décennies, de nombreuses heuristiques ont été développées pour* $\boldsymbol{Pm//C_{max}}$ *Une de ces heuristiques est décrite ci-dessous. La règle du temps de traitement le plus long en premier (LPT) assigne à* $\boldsymbol{(t = 0)}$ *le plus long* $\boldsymbol{m}$ *travaux aux* $\boldsymbol{m}$ *machines. Après cela, chaque fois qu'une machine est libérée le plus long temps travail parmi ceux qui n'ont pas encore été traités est mis sur la machine [49].*

3.4.2 Le sous problème ($Pm//C_{max}$) et sa difficulté

On souhaite programmer l'exécution d'un ensemble de $\boldsymbol{n}$ tâches $\boldsymbol{T_1, T_2, .. ,T_n}$ sur $\boldsymbol{m}$ machines parallèles identiques (la vitesse de traitement des machines est la même). Les tâches ont des temps de traitement entiers $\boldsymbol{P_1 , P_2 , .., P_m}$. Chaque tâche doit être affectée à une seule machine et ne peut être interrompue une fois commencée. L'objectif est de minimiser le temps $\boldsymbol{C_{max}}$ nécessaire pour le traitement de toutes les tâches sur les machines. Le problème consiste à affecter les tâches aux différentes machines de manière à finir le traitement de toutes les tâches au plus vite.

Remarque 3.4.2 *Une limite supérieure est présentée par :*

$$\frac{C_{max}\,(LPT)}{C_{max}\,(OPT)}$$

Où $\boldsymbol{C_{max}(LPT)}$ *désigne le Makespan de la planification* $\boldsymbol{LPT}$ *et* $\boldsymbol{C_{max}(OPT)}$ *désigne le makespan de la planification optimale (éventuellement inconnue). Ce type d'analyse du cas le plus défavorable est intéressant car il donne une indication de la qualité de réalisation de l'heuristique ainsi que du type d'instances pour lesquelles l'heuristique se comporte mal [39].*

Théorème 3.4.2 *Pour* $\boldsymbol{(Pm\ //C_{max})}$ *on a la borne supérieure comme suite :*

$$\frac{C_{max}\,(LPT)}{C_{max}\,(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$

Preuve. Par contradiction, supposons qu'il existe un ou plusieurs contre-exemples avec le rapport strictement supérieur à $\boldsymbol{\frac{4}{3} - \frac{1}{3m}}$. S'il existe plusieurs contre-exemples de ce type, il doit exister un exemple avec le plus petit nombre de travaux. Considérez ce «plus petit» contre-exemple et supposez qu'il ait n emplois. Ce plus petit contre-exemple a une propriété utile: sous

LPT, le travail le plus court est le dernier travail à démarrer, ainsi que le dernier à terminer son traitement. Cela est vrai comme suit: premièrement, sous ***LPT***, par définition, le travail le plus court est le dernier à commencer son traitement. De plus, si ce travail n'est pas le dernier à terminer son traitement, sa suppression entraînera un contre-exemple avec moins de travaux (le C_{max} ***(LPT)*** reste la même alors que le $C_{max}(OPT)$ peut rester la même ou peut diminuer). Ainsi, pour le plus petit contre-exemple, l'heure de début du travail le plus court sous ***LPT*** est $C_{max}(LPT) - p_n$. Comme à ce moment toutes les autres machines sont encore occupées, il en résulte que :

$$C_{max}\,(LPT) \;-\; p_n \;\leq\; \sum_{j=1}^{n-1} p_j \Big/ m.$$

Le côté droit est la limite supérieure de l'heure de début du travail le plus court. Cette limite supérieure est atteinte lors de la planification des $n-1$ premiers travaux en fonction du ***LPT***, chaque machine ayant exactement le même temps de traitement. À présent

$$C_{max}\,(LPT) \;\leq\; p_n + \sum_{j=1}^{n-1} p_j \Big/ m \;\;=\;\; p_n(1-1/m) + \sum_{j=1}^{n} p_j/m\,.$$

Puisque :

$$C_{max}\,(OPT) \;\geq\; \sum_{j=1}^{n} p_j/m$$

la série d'inégalités suivante est valable pour le contre-exemple :

$$\frac{4}{3} - \frac{1}{3m} \;<\; \frac{C_{max}\,(LPT)}{C_{max}\,(OPT)} \;\leq\; \frac{p_n(1-1/m) + \sum_{j=1}^{n} p_j/m}{C_{max}\,(OPT)}$$

$$= \frac{p_n(1-1/m) + \sum_{j=1}^{n} p_j/m}{C_{max}\,(OPT)} \;\leq\; \frac{p_n(1-1/m)}{C_{max}\,(OPT)} + 1$$

Ainsi

$$\frac{4}{3} - \frac{1}{3m} \;<\; \frac{p_n(1-1/m)}{C_{max}\,(OPT)} + 1$$

Et

$$C_{max}\,(OPT) \;< 3p_n.$$

Notez que cette dernière inégalité est une inégalité stricte. Cela implique que pour le plus petit contre-exemple du planning optimal peut entraîner au plus deux tâches sur chaque machine. On peut montrer que si une planification optimale est une planification avec au plus deux tâches sur chaque machine, la planification ***LPT*** est optimale et le rapport des deux durées de fabrication est égal à un, cette contradiction complète la preuve du théorème [39].

Exemple 3.4.1

Un exemple du pire des cas de ***LPT****, Considérez* ***4*** *machines parallèles et* ***9*** *tâches dont les temps de traitement sont indiqués dans le tableau ci-dessous :*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
p_i	7	7	6	6	5	5	4	4	4

Tableau 3.2 : Temps de traitement des tâches de l'exemple 3.4.1

La planification des travaux en fonction du ***LPT*** génère une durée de travail de ***15***. Elle peut être montré facilement que pour cet ensemble d'emplois un calendrier peut être trouvé avec le makespan de ***12*** (voir la figure 3.4.1 et 3.4.2). Ce cas particulier est donc le pire des cas quand il y'a ***4*** machines en parallèle. Quel serait le pire des cas si au lieu de ***LPT***, une règle de priorité arbitraire est utilisée ?

Prenons le cas où, à l'instant $t = 0$, les travaux sont placés dans une liste arbitraire. À chaque fois qu'une machine est libérée, le travail le plus élevé de la liste figure parmi les travaux restants. On peut montrer que le pire cas de cette règle de liste arbitraire corrige l'inégalité :

$$\frac{C_{max}\,(LIST)}{C_{max}\,(OPT)} \leq 2 - \frac{1}{m}.$$

Cependant, il existe également plusieurs autres heuristiques pour la classe problème $(Pm\,//C_{max})$ qui sont plus sophistiqués que ***LPT*** et qui ont des limites plus strictes pire.

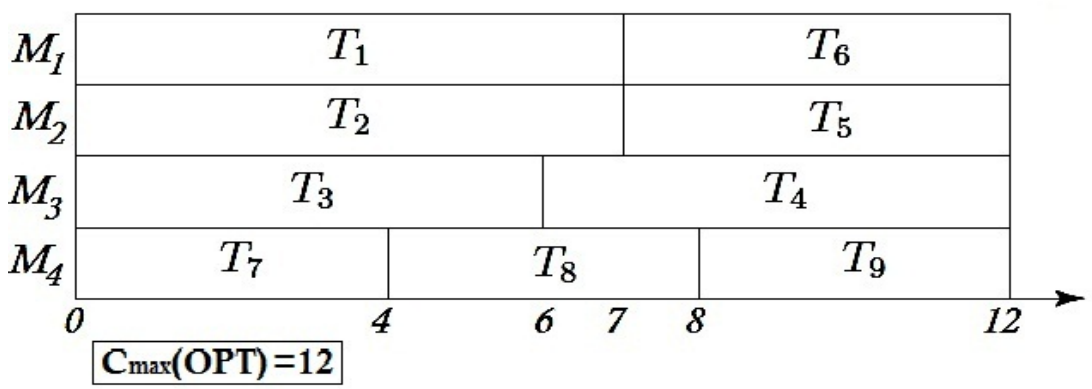


Figure 3.4.1 : Ordonnancement au pire des cas de ***LPT*** avec $C_{max}\,(OPT) = 12$

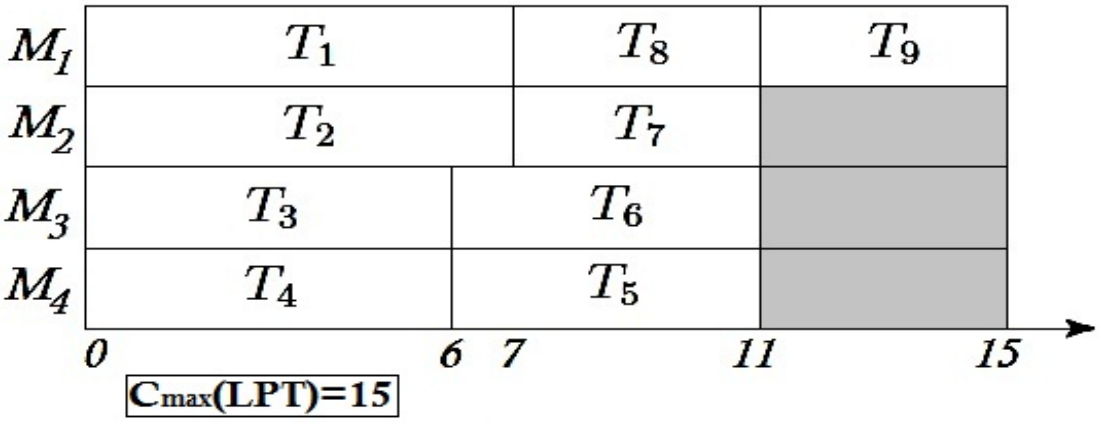


Figure 3.4.2 : Ordonnancement au pire des cas de ***LPT*** avec $C_{max}\,(LPT) = 15$

3.4.3 Le sous problème $(P2//C_{max})$ et sa difficulté

Le problème $\boldsymbol{(P2//C_{max})}$ consiste à traiter un ensemble de tâches qui sont indépendantes, sans préemption ayant des temps de traitement arbitraire et des dates de disponibilité nulle sur deux machines parallèles identiques l'objectif est de minimiser la date de fin de traitement [32].

Théorème 3.4.3 *Le problème* $\boldsymbol{(P2//C_{max})}$ *est NP- difficile.*

Preuve. Considérons le problème de décision suivant connu sous le nom de ***2***-partitions, étant donnés $\boldsymbol{n}$ entier positifs $\boldsymbol{a_1, \ldots, a_n}$. Existe-t-il un sous ensemble $\boldsymbol{J \subseteq I = \{1, 2, \ldots, n\}}$ tel que $\sum_{i\in J} a_i = \sum_{i\epsilon I\setminus J} a_i$? Ce problème est $\boldsymbol{NP-}$complet.

Montrons que ce problème se réduit polynomialment au problème de décision ***2***-machines étant donnés $\boldsymbol{n}$ tâches indépendantes et non morcelables. chaque tâche $\boldsymbol{T_i}$ a un temps de traitement $\boldsymbol{p_i}$ égal à $\boldsymbol{a_i}$, et un nombre $\boldsymbol{k}$ tel que :

$$\boldsymbol{k = \frac{1}{2}\sum_{i=1}^{n} a_i}\ .$$

Existe –t-il un ordonnancement de ces tâches sur deux machines de durée inférieure ou égale à $\boldsymbol{k}$? Il est clair qu'il existe un sous ensemble $\boldsymbol{J}$ avec la propriété désirée pour ***2***-partitions si et seulement si il existe un ordonnancement de longueur inférieur ou égale à $\boldsymbol{k}$ pour ***2*** machines.

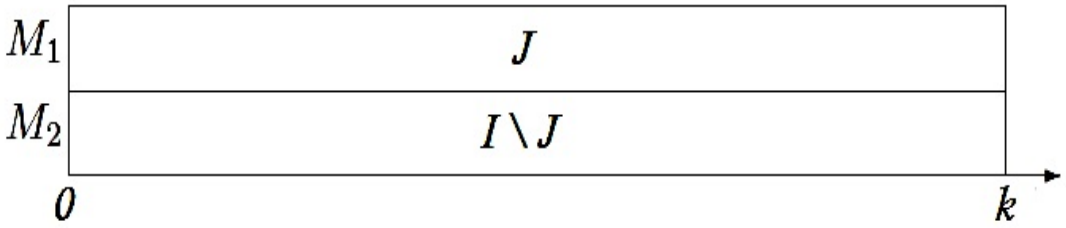


Figure 3.4.3 : Correspondance entre ***2***-partitions et ***2***-machines

Comme le problème de décision ***2***- partitions est ***NP***- complet, alors le problème de décision ***2***-machines est aussi ***NP***- complet.

3.4.4 Le problème $(Pm/Split, S_i, r_i/C_{max})$

Définition du problème

Le problème auquel nous nous intéressons est un problème d'ordonnancement à machines parallèles identiques avec temps de préparation des machines dépendant de la séquence et découpage des opérations. Chacune des opérations peut-être découpée en fractions qui peuvent être exécutées en parallèle sur plusieurs machines. La différence entre le découpage d'une opération *(splitting)* et la préemption est que le premier autorise plusieurs fractions d'une même opération à être exécutées simultanément sur différentes machines alors que dans le cas de la préemption, une fraction d'une opération ne peut débuter que si aucune autre fraction de cette opération n'est en cours sur une autre machine. Des temps de préparation sont nécessaires pour passer de l'exécution d'une fraction d'une opération à l'exécution d'une fraction d'une autre opération. L'objectif est de minimiser la date de fin de l'ordonnancement. Ce problème est noté : $(Pm/Split,S_i,r_i/C_{max})$. Nous renvoyons à [49] pour une description des applications possibles de ce problème.

Le problème d'ordonnancement $(Pm\ /split, S_i, r_i/C_{max})$ consiste à traiter $\boldsymbol{n}$ tâches qui doivent être ordonnancées sur $\boldsymbol{m}$ machines parallèles identiques.

Chacune des $\boldsymbol{n}$ tâches définie par une durée $\boldsymbol{p_i}$. Elles peuvent être découpées en fractions qui peuvent être exécutées indifféremment sur n'importe quelle machine. Cependant, lorsqu'une machine bascule la production d'une fraction d'une tâche $\boldsymbol{i}$ $\forall\, \boldsymbol{i} \in [\,\boldsymbol{1} \ldots \boldsymbol{n}\,]$ vers la fraction d'une tâche $\boldsymbol{j}$, $\forall\, \boldsymbol{j} \in [\,\boldsymbol{1} \ldots \boldsymbol{n}\,]$ et $\boldsymbol{j} \neq \boldsymbol{i}$ un temps de préparation $\boldsymbol{S_{i,j}}$ est nécessaire. Par hypothèse, lorsque deux fractions d'une même opération $\boldsymbol{i}$ sont exécutées consécutivement sur une même machine, alors aucun temps de montage n'est nécessaire $\boldsymbol{S_{i,i} = 0}$, $\forall\, \boldsymbol{i} \in [\,\boldsymbol{1} \ldots \boldsymbol{n}\,]$. De plus nous considèrerons que si une opération est la première à s'exécuter sur une machine alors aucun temps de préparation n'est nécessaire. Enfin les temps de préparation vérifient l'inégalité triangulaire :

Soient $\boldsymbol{i, j}$ et $\boldsymbol{k} \in [\,\boldsymbol{1} \ldots \boldsymbol{n}\,]$.

$$\boldsymbol{S_{i,i} + S_{j,k} \geq S_{i,k}}\,.$$

La figure 3.4.4 présente un exemple à $\boldsymbol{5}$ machines et $\boldsymbol{8}$ tâches.

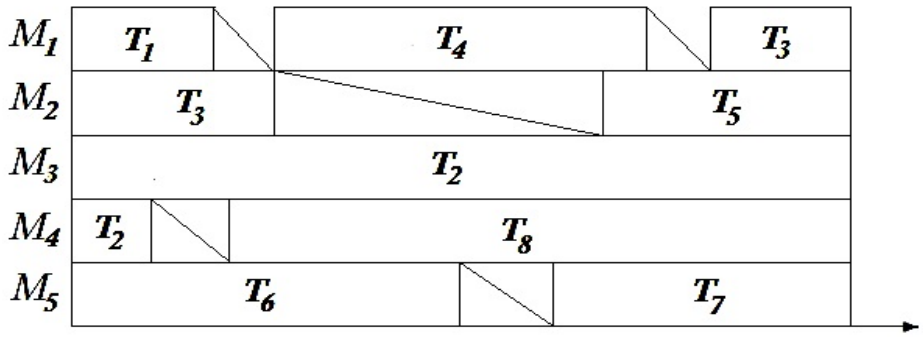


Figure 3.4.4 : Exemple de ($Pm\ /split, S_i, r_i\ / C_{max}$)

Notons $\boldsymbol{C_i}$ la date de fin de l'opération $\boldsymbol{i}$, c'est-à-dire la date de fin de sa fraction qui finit le plus tard, nous cherchons à obtenir un ordonnancement de durée minimale :

$$\boldsymbol{min\, c_{max} = min\, \{ \max_{i \in [\,1 \ldots n\,]} C_i \}.}$$

Soit $\boldsymbol{D_m}$ le temps qui correspond à la fin d'exécution de toutes les fractions sur la machine $\boldsymbol{M}$, $\boldsymbol{D_m}$ est la somme de toutes les fractions assignées à la machine $\boldsymbol{M}$ et des temps de préparation correspondant à leur séquence d'exécution. Minimiser $\boldsymbol{C_{max}}$ revient à trouver :

$$\boldsymbol{min\, \{ \max_{M \in [\,1 \ldots m\,]} D_m \}.}$$

Une solution du problème consiste en la spécification d'un découpage des $\boldsymbol{n}$ tâches et la répartition de leurs fractions sur les $\boldsymbol{m}$ machines. Soit $\boldsymbol{Q_{i,m}}$ la fraction de la tâche $\boldsymbol{i,}$ $\forall\, \boldsymbol{i} \in [\,\boldsymbol{1} \ldots \boldsymbol{n}\,]$ qui doit être exécutée sur la machine $\boldsymbol{M,}$ $\forall\, \boldsymbol{M} \in [\,\boldsymbol{1} \ldots \boldsymbol{m}\,]$ alors :

$$\sum_{M=1..m} Q_{i,m} = P_i, \forall\, i \in [\,1 \ldots n\,].$$

La modélisation du problème

Les variables de notre modèle sont :

- C_{max} : La date de fin de la dernière machine.
- D_m : La date de fin de la dernière tâche exécutée sur la machine ***M***.
- $Q_{i,m}$: La fraction de la tâche i exécutée sur la machine ***M***.
- $y_{i,m}$: La variable de décision qui vaut ***1*** si une partie de la tâche i est exécutée sur la machine ***M***.
- $X_{i,j,m} = \begin{cases} 1 & si \quad la\ tâche\ j\ suit\ immédiatement\ la\ tâche\ i \\ & \qquad sur\ la\ machine\ M \\ 0 & sinon \end{cases}$
- $u_{i,m}$: Les variables artificielles introduites pour éviter les circuits. Nous reviendrons par la suite sur le rôle exact de ces variables.

Nous ajoutons également au problème deux opérations fictives, c'est-à-dire. de durée nulle, d'indice 0 et $n+1$. Nous considérons que les temps de préparation entre ces tâches et toutes les autres sont nuls. De plus, nous imposons que chaque machine commence par l'exécution de l'opération 0 et finisse par l'exécution de l'opération $n+1$, dans le but de modéliser le fait qu'aucun temps de montage n'est utile en début, (resp enfin), d'exécution de la première (resp la dernière) opération sur chaque machine.

Voici le modèle mathématique en nombres entiers pour la résolution de ce problème [49] :

$$\textbf{Min } C_{max} \tag{1}$$

$$\sum_{M=1}^{m} Q_{i,m} = p_i \qquad \forall\, i \in [\,1 \dots n\,]. \tag{2}$$

$$\sum_{i=1}^{n} Q_{i,m} + \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j,m}.S_{i,j} = D_m \qquad \forall\, M \in [\,1 \dots m\,]. \tag{3}$$

$$C_{max} \geq D_m \qquad \forall\, M \in [\,1 \dots m\,]. \tag{4}$$

$$y_{i,m} \geq Q_{i,m}\,/p_i \qquad \forall\, i \in [\,1 \dots n\,], \quad \forall\, M \in [\,1 \dots m\,]. \tag{5}$$

$$y_{i,m} \leq Q_{i,m} \qquad \forall\, i \in [\,1 \dots n\,], \quad \forall\, M \in [\,1 \dots m\,]. \tag{6}$$

$$\sum_{j=1}^{n+1} x_{i,j,m} = y_{i,m} \qquad \forall\, i \in [\,1 \dots n\,], \quad \forall\, M \in [\,1 \dots m\,]. \tag{7}$$

$$\sum_{i=0}^{n} x_{i,j,m} = y_{i,m} \qquad \forall\, i \in [\,1 \dots n\,], \quad \forall\, M \in [\,1 \dots m\,]. \tag{8}$$

$$\sum_{i=1}^{n} x_{0,i,m} = 1 \qquad \forall\, M \in [\,1 \dots m\,]. \tag{9}$$

$$\sum_{i=1}^{n} x_{i,n+1,m} = 1 \qquad \forall\, M \in [\,1 \dots m\,]. \tag{10}$$

$$x_{i,j,m} = 0 \qquad \forall\, M \in [\,1 \dots m\,], \quad \forall\, i \in [\,0 \dots n+1\,]. \tag{11}$$

$$u_{i,m} - u_{j,m} + n \times x_{i,j,m} \leq n - 1 \qquad \forall\, i,j \in [\,1 \dots n\,],\ i \neq j \ \textit{et}\ \forall\, M \in [\,1 \dots m\,]. \tag{12}$$

Avec :

- (1) est la fonction objectif de notre modèle.
- La contrainte (2) impose que la totalité des tâches soit exécutée.
- La contrainte (3) fixe la date de fin de l'ordonnancement obtenu à la date de fin de la dernière machine.
- La contrainte (4) calcule la date de fin associée à une machine en fonction des parties de tâches qu'elle exécute.
- Les contraintes (5) et (6) imposent que la variable de décision opération-machine $y_{i,m}$ prenne la valeur **1** si une partie de la tâche ***i*** est exécutée sur la machine m.

- Les contraintes (7) et (8) indiquent que toute partie d'une tâche non fictive exécutée sur une machine est précédée exactement par une autre fraction d d'une tâche et est suivie exactement par une autre fraction d'une tâche.

Les fractions suivantes et précédentes peuvent être des fractions des tâches fictives.

- Les contraintes (9) et (10) indiquent que sur chaque machine la partie de la tâche fictive de tête est suivie par exactement une partie de la tâche non fictive, et que la partie de la tâche fictive de fin est précédée par une partie de la tâche non fictive.
- La contrainte (11) empêche la succession sur une même machine de deux parties d'une même tâche.
- La contrainte (12) empêche l'existence de sous-circuit.

3.4.5 Le problème $(Pm/S_i, k/C_{max})$

Définition du problème

On s'intéresse au problème d'ordonnancement qu'il s'agit d'ordonnancer un ensemble de n tâches non préemptibles et indépendantes $T_1, T_2, \ldots, T_n$ sur m machines parallèles identiques $M_1, M_2, \ldots, M_m$ pour optimiser le makespan. Chaque tâche T_i nécessite un temps de traitement p_i et un temps de préparation S_i pour cette préparation, nous disposons de k ouvriers spécialisés, il est donc impossible de faire plus de k préparations en même temps. Ce problème sera noté : $(Pm/S_i, k/C_{max})$. En présence de k ouvriers spécialisés. Chaque tâche doit subir avant d'être exécuté sur une machine, un traitement particulier par un ouvrier spécialisé. Les machines identiques ainsi que les ouvriers spécialisés ne peuvent traiter qu'une seule tâche à la fois [50].

La modélisation du problème

Pour ce faire considérons les variables bivalentes x_{ijt} telles que :

- $x_{ijt} = \begin{cases} 1 & si \quad la\ tâche\ T_i\ débute\ sa\ préparation\ sur\ la\ machine\ M_j\ à\ l'instant\ t \\ 0 & sinon \end{cases}$

Pour $i = 1,\ldots, n$; $j = 1,\ldots, m$ et $t = 0,\ldots, H-1$ avec H un temps limite qu'on peut estimer à la borne supérieure.

On veut minimiser la date de fin de traitement de l'ensemble des tâches notée μ.

Le modèle mathématique ainsi développé s'écrit [50] :

$$\begin{cases} \textbf{Min } \boldsymbol{\mu} & \boldsymbol{\mu} \in \mathbb{R} & (1) \\ \sum_{j=1}^{m} \sum_{t=0}^{H-1} \boldsymbol{x_{ijt}} = \mathbf{1} & \boldsymbol{i} = \overline{\mathbf{1,n}} & (2) \\ \sum_{y=t}^{t+S_i+p_i-1} \boldsymbol{x_{ijy}} \leq \mathbf{1} & \boldsymbol{j} = \overline{\mathbf{1,m}}, \boldsymbol{i} = \overline{\mathbf{1,n}} \ \textit{et} \ \boldsymbol{t} = \overline{\mathbf{0,H-1}} & (3) \\ \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{y=max\{0,t-S_i+1\}}^{t} \boldsymbol{x_{i,j,y}} \leq \boldsymbol{k} & \boldsymbol{t} = \overline{\mathbf{0,H-1}} & (4) \\ (\sum_{j=1}^{m} \sum_{t=0}^{H-1} \boldsymbol{t.x_{ijt}}) + \boldsymbol{S_i} + \boldsymbol{p_i} \leq \boldsymbol{\mu} & \boldsymbol{i} = \overline{\mathbf{1,n}} & (5) \\ \boldsymbol{x_{ijy}} \in \{\mathbf{0,1}\} & \boldsymbol{j} = \overline{\mathbf{1,m}}, \boldsymbol{i} = \overline{\mathbf{1,n}} \ \textit{et} \ \boldsymbol{t} = \overline{\mathbf{0,H-1}} & (6) \end{cases}$$

- La (1)^{ère} minimise le temps de fin de traitement.
- La (2)^{ème} contrainte assure que l'on assigne une tâche à une machine et une seule.
- La (3)^{ème} contrainte indique que dans l'intervalle du temps $[\boldsymbol{t}, \boldsymbol{t} + \boldsymbol{S_i} + \boldsymbol{p_i} - \mathbf{1}]$ on ne peut traiter qu'au plus une seule tâche.
- La (4)^{ème} contrainte signifie qu'on ne peut pas faire plus de $\boldsymbol{k}$ préparation à un instant $\boldsymbol{t}$ donné.
- La (5)^{ème} contrainte indique que la date de fin de traitement de chaque tâche doit être inférieure ou égale à $\boldsymbol{\mu}$.
- La (6)^{ème} est pour les variables bivalentes $\boldsymbol{x_{ijt}}$.

Remarque 3.4.3 *Le problème à deux machines* $\boldsymbol{P2/S_i, k/C_{max}}$ *est NP-difficile, car le problème* $\boldsymbol{P2\ //C_{max}}$ *est un cas particulier en prenant* $\boldsymbol{S_i = 0}$.

3.4.6 Le sous problème $(P2/p_i = p,\ S_i, k = 2/C_{max})$ et sa difficulté

Le problème $\boldsymbol{(P2/p_i = p,\ S_i, k = 2/C_{max})}$ est un problème d'ordonnancement il s'agit ordonnancer un ensemble de $\boldsymbol{n}$ tâches non préemptibles et indépendantes avec des temps de traitement identiques et un temps de préparation $\boldsymbol{S_i}$, en présence de deux ouvriers spécialisés pour minimiser la date de fin de traitement.

Théorème 3.4.4 *Le problème* $\boldsymbol{(P2/p_i = p,\ S_i, k = 2/C_{max})}$ *est NP- difficile.*

Preuve. Montrons que le problème de ***2***-partition se réduit polynomialement au problème de décision suivant [50] : Etant données $\boldsymbol{n}$ tâches $\boldsymbol{T_1, T_2, \ldots, T_n}$ indépendantes et non morcelables avec des temps de traitement $\boldsymbol{p_i = p = 0}$, le temps nécessaire pour la préparation d'une tâche est $\boldsymbol{S_i = a_i}$, $\boldsymbol{k = 2}$ et :

$$\boldsymbol{y = \frac{1}{2}\sum_{i=1}^{n} a_i} .$$

Existe-t-il un ordonnancement de ces $\boldsymbol{n}$ tâches sur deux machines de durée inférieure ou égale à $\boldsymbol{y}$?

Ce problème appartient à la classe ***NP***, car on peut vérifier en un temps polynomial qu'une affectation des tâches aux machines vérifie toutes les contraintes. Il est clair que la réduction précédente est polynomiale. Nous prouvons que le problème ***2***-partition a une solution si et seulement si le problème d'ordonnancement a une solution.

Si le problème ***2***-partition a une solution, alors il existe un sous-ensemble $\boldsymbol{A_1 \subseteq A}$ avec la propriété désirée :

$$\sum_{a_i \epsilon A_1} \boldsymbol{a_i} = \sum_{a_i \epsilon A/A_1} \boldsymbol{a_i} .$$

Nous construisons une solution pour le problème d'ordonnancement comme suit : on prépare les tâches qui appartiennent à l'ensemble $\boldsymbol{A_1}$ sur la machine $\boldsymbol{M_1}$ et les tâches restantes, on les prépare sur la machine $\boldsymbol{M_2}$. Donc, la durée de l'ordonnancement est égale à :

$$\boldsymbol{y = \frac{1}{2}\sum_{i=1}^{n} a_i} .$$

Si le problème d'ordonnancement a une solution de durée inférieure ou égale à $\boldsymbol{y}$, alors chaque tâche n'est préparée que sur l'une des deux machines. Donc le problème ***2-*** partition a une solution, en posant $\boldsymbol{A_1}$ comme étant l'ensemble de tâches préparées sur la machine $\boldsymbol{M_1}$ et $\boldsymbol{A/A_1}$ l'ensemble de tâches préparées sur la machine $\boldsymbol{M_2}$.

3.4.7 Le sous problème $\boldsymbol{(P2/S_i = s, k = 2/C_{max})}$ et sa difficulté

Ce problème est un problème d'ordonnancement où s'agit d'ordonnancer un ensemble de $\boldsymbol{n}$ tâches non préemptibles et indépendantes avec des temps de préparation $\boldsymbol{S_i}$ qui sont identiques et

des temps de traitements p_i et des dates échues arbitraires, en présence de deux ouvriers spécialisés pour minimiser le Makespan.

Théorème 3.4.5 *Le problème* $(P2/S_i = s, k = 2/C_{max})$ *est NP- difficile.*

Preuve. Montrons que le problème de *2*-partition se réduit polynomialement au problème à deux machines parallèles identiques, étant données n tâches $T_1, T_2, \ldots, T_n$ indépendantes et non morcelables avec des temps de préparation $p_i = p = 0$ et $k = 2$, chaque tâche T_i a un temps de préparation $S_i = a_i$ et un nombre :

$$y = \frac{1}{2}\sum_{i=1}^{n} a_i .$$

Existe-t il un ordonnancement de ces n tâches sur deux machines de durée inférieure ou égale à y ?

Ce problème appartient à la classe *NP*, car on peut vérifier en un temps polynomial qu'une affectation des tâches aux machines vérifie toutes les contraintes. Il est clair que la réduction précédente est polynomiale. Nous prouvons que le problème *2*-partition a une solution si et seulement si le problème d'ordonnancement a une solution.

Supposons que le problème *2*-partition à une solution, Ainsi, on peut construire un ordonnancement pour le problème $(P2/S_i = s, k = 2/C_{max})$ en traitant les $|A_1|$ tâches sur la machine M_1 et les $|A/A_1|$ tâches restantes sur la machine M_2. Donc, la durée de l'ordonnancement est égale à y.

Supposons maintenant que le problème d'ordonnancement a une solution de durée inférieure ou égale à y , alors chaque tâche n'est affectée qu'à l'une des deux machines de telle sorte que A_1 soit l'ensemble des tâches affectées à la machine M_1 et A/A_1 l'ensemble des tâches affectées à la machine M_2. Donc, le problème *2*-partition a une solution.

3.4.8 Le sous problème $(P2/p_i = p, S_i = s, k \leq 2/C_{max})$ et sa difficulté

On s'intéresse maintenant au problème d'ordonnancement $(P2/p_i = p, S_i = s,\ k \leq 2/C_{max})$ qu'il s'agit de traiter un ensemble de n tâches non préemptibles et indépendantes $T_1, T_2, \ldots, T_n$ avec temps de traitements p_i, temps de préparation S_i qu'ils sont identiques et dates de disponibilités qu'ils sont arbitraires en présence d'au plus de deux ouvriers spécialisés sur m machines parallèles identiques $M_1, M_2, \ldots, M_m$ pour minimiser le Makespan [50].

Proposition 3.4.1 $\bar{M}$ *est une borne inférieure de* C_{max} *tel que :*

$$\bar{M} = max\left\{\left\lceil \frac{1}{m}\sum_{i=1}^{n}(p_i + S_i)\right\rceil , max_{i=1}^{n}(p_i + S_i)\right\}.$$

Preuve. Par l'absurde $\bar{M} < max_{i=1}^{n}(p_i + S_i)$ la tâche ayant le plus grand temps nécessaire pour son traitement $(p_i + S_i)$ sera traitée sur au moins deux machines simultanément, contradiction avec les hypothèses qu'à chaque instant, une tâche ne peut être exécutée que par une seule machine au plus, et à chaque instant, une machine ne peut exécuter qu'une seule tâche à la fois. D'où $\bar{M}$ est une borne inférieure pour le C_{max} .

- Si $\bar{M} < max\left\{\left\lceil \frac{1}{m}\sum_{i=1}^{n}(p_i + S_i)\right\rceil\right\}$ contradiction du fait que le C_{max} ne peut pas prendre une valeur inférieure à la somme des temps de traitement et les temps de préparation des tâches divisée par le nombre de machines.
- Si toutes les tâches sont traitées sur la même machine, on aura comme borne supérieure $\bar{S}$ tel que : $\bar{S} = \sum_{i=1}^{n}(p_i + S_i)$.

Théorème 3.4.6 *Le problème* $(P2/p_i = p, S_i = s, k \leq 2/C_{max})$ *est de classe* P *avec* $\overline{M'}$ *une borne inférieur qui peut donne la solution optimale [51] tel que :*

$$\overline{M'} = \begin{cases} \frac{n(p+s)}{2} & Si\ \ n\ \ est\ pair\ \ et\ \ k=2\ ; \\ \frac{n(p+s)}{2} + s & Si\ \ n\ \ est\ pair\ \ et\ \ k=1\ \ \ avec\ \ \ s \leq p; \\ n.s + p & Si\ \ n\ \ est\ pair\ \ et\ \ k=1\ \ \ avec\ \ \ s > p; \\ \frac{(n+1)(p+s)}{2} & Si\ \ n\ \ est\ impair\ et\ \ k=2\ ; \\ \frac{(n+1)(p+s)}{2} & Si\ \ n\ \ est\ impair\ et\ \ k=1\ \ \ avec\ \ \ s \leq p; \\ n.s + p & Si\ \ n\ \ est\ impair\ et\ \ k=1\ \ \ avec\ \ \ s > p; \end{cases}$$

Preuve. La valeur $\overline{M'}$ constitue une borne inférieure pour la solution optimale. Comme les temps de traitement et les temps de préparation sont constants et égaux à p et s respectivement, on a :

- Pour $k = 2$ et d'après la borne inférieure énoncée dans la proposition 3.5.1, nous avons [50]: si n est pair : $\bar{M} = max\left\{\left\lceil \frac{1}{2}n(p+s)\right\rceil , p+s\right\} = \left\lceil \frac{1}{2}n(p+s)\right\rceil = n(p+s)/2$

si $\boldsymbol{n}$ est impair alors : $\boldsymbol{C_{max} = (n+1)(p+s)/2 + (p+s) = (n+1)(p+s)/2}$ qui vient du fait que la préemption des tâches n'est pas autorisée. Il suffit donc de considérer le cas pair et d'ajouter le traitement de la $\boldsymbol{n^{ième}}$ tâche à la fin de l'ordonnancement.

- Pour $\boldsymbol{k = 1}$, nous avons un seul ouvrier. Donc la préparation simultanée de deux tâches n'est pas possible. Ainsi, la seconde machine sera libre pendant $\boldsymbol{s}$ unités de temps au début de l'ordonnancement.
- Si le temps de préparation est supérieur au temps de traitement ($\boldsymbol{s \geq p}$), alors la préparation des différentes tâches se fera en alterné sur les deux machines et chaque tâche sera traitée entre deux préparations successives. On obtient donc ($\boldsymbol{n.s}$) comme temps totale de préparation de l'ensemble des tâches. Finalement, on obtient $\boldsymbol{C_{max} = n.s + p}$.
- Si le temps de préparation est inférieur au temps de traitement ($\boldsymbol{s < p}$), alors les tâches seront traitées comme dans le cas où $\boldsymbol{k = 1}$ avec un décalage de $\boldsymbol{s}$ unités de temps sur la deuxième machine (la préparation de la deuxième tâche commence à $\boldsymbol{t = s}$ sur la deuxième machine). On obtient donc :

  dans le cas ou $\boldsymbol{n}$ est pair : $\boldsymbol{C_{max} = n(p+s)/2 + s}$ et dans le cas où $\boldsymbol{n}$ est impair $\boldsymbol{C_{max} = (n+1)(p+s)/2}$ (la deuxième machine sera libre pendant $\boldsymbol{p}$ unités de temps à la fin de l'ordonnancement).

3.5 Minimisation le Makespan dans l'ordonnancement du graphe de précédence

3.5.1 Le problème ($\boldsymbol{Pm/Prec/C_{max}}$)

Définition du problème

C'est un problème qui traite un ensemble des tâches dépendantes sans préemption ayant des temps de traitements arbitraires, des dates de disponibilité nulle sur des machines parallèles identiques et l'objectif de ce problème est de minimiser la date de fin de traitement. D'un point de vue de complexité, ce problème doit être au moins aussi dur que le problème sans contraintes de précédence. Pour obtenir un aperçu des effets des contraintes de priorité, un certain nombre de cas spéciaux doivent être pris en compte. Le cas particulier avec une seule machine [32].

La modélisation du problème

$$\begin{cases} \textbf{Min Z} & & (1) \\ \sum_{j=1}^{m} x_{ij} = 1 & i = \overline{1..n}. & (2) \\ Z_{ik} + Z_{ki} = 1 & i < k \;\; et \;\; i,k = \overline{1..n}. & (3) \\ t_k - t_i \leq M_{Z_{ik}} & i \neq k \;\; et \;\; i,k = \overline{1..n}. & (4) \\ t_i + p_i - t_k \leq M\left(1 - Z_{ik} + 2 - x_{ij} - x_{kj}\right) & i \neq k \;\; i,k = \overline{1..n} \;\; j = \overline{1..m}. & (5) \\ Z_{ik} = 1 \;\; et \;\; t_i + p_i - t_k \leq 0 & T_i < T_k. & (6) \\ t_i + p_i \leq Z & i = \overline{1..n}. & (7) \\ x_{ij}\,, Z_{ik} \in \{1,0\} \;\; et \;\; t_i\,, Z \geq 0 & i = \overline{1..n} \;\; j = \overline{1..m}. & (8) \end{cases}$$

- La (1)^{ère} pour minimiser le temps de fin de traitement.
- La (2) ^{ème} contrainte signifie que chaque tâche doit être affectée à une seule machine et donc toutes les tâches seront traitées.

$$\sum_{i=1}^{n} p_i = M, \quad \textit{Une borne supérieure}$$

- La (3) ^{ème} contrainte signifie que la préemption est interdite tel que :

$$\begin{cases} Si \qquad\qquad Z_{ik} = 0 \;:\; t_k - t_i \leq M_{Z_{ik}} = 0 \;\; et \;\; t_k \leq t_i \\ Si \;\; Z_{ik} = 1 \;\; alors \;\; Z_{ki} = 0 \;\; t_i - t_k \leq M_{Z_{ik}} = 0 \;\; et \;\; t_i \leq t_k \end{cases}.$$

- La (4) ^{ème} contrainte signifie que la différence entre les dates de début de traitement entre la tâche T_i et T_k respectivement ne doit pas dépassé la borne supérieure.
- La (5) ^{ème} contrainte est un groupe de contraintes de non chevauchement elles sont obtenues à partir :

$$x_{ij} = 1, \;\; x_{kj}=1 \;\; \textbf{et} \;\; Z_{ik} = 1 \;\; \textbf{alors} \;\; t_i + p_i - t_k \leq 0 \; donc \;\; t_i + p_i \leq t_k$$

$$(T_i \;\; termine \; avant \; le \; début \; de \;\; T_k).$$

- La (6) ^{ème} contrainte est pour les contraintes de précédence.
- La (7) ^{ème} contrainte signifie que toutes les tâches doivent être terminées avant la date Z.
- La (8)^{ème} contrainte, on définit les variables tels que :

$$x_{ij} = \begin{cases} 1 \;\; si \; la \; tâche \;\; T_i \; traité \; sur \\ \qquad la \; machine \; M_j \\ 0 \;\; sinon \; i = \overline{1..n} \;\; j = \overline{1..m} \end{cases} \qquad et \qquad Z_{ik} = \begin{cases} 1 \;\; si \;\; t_i \leq t_k \;\; T_i < T_k \\ 0 \;\; sinon \;\; i = \overline{1..n} \;\; k = \overline{1..n} \end{cases}.$$

3.5.2 Le problème ($Pm/Prec, \widetilde{d}_i/C_{max}$)

Définition du problème

C'est un problème qui traite un ensemble de tâches dépendantes avec préemption non autorisée ayant des p_i, des dates de disponibilité nulles et dates limites arbitraires sur des machines parallèles identiques. L'objectif de ce problème est de minimiser le Makespan [32].

La modélisation du problème

$$
\begin{cases}
\textbf{Min Z} & & (1)\\
\sum_{j=1}^{m} x_{ij} = 1 & i = \overline{1..n}. & (2)\\
Z_{ik} + Z_{ki} = 1 & i < k \;\; et \;\; i,k = \overline{1..n}. & (3)\\
t_k - t_i \leq M_{Z_{ik}} & i \neq k \;\; et \;\; i,k = \overline{1..n}. & (4)\\
t_i + p_i - t_k \leq M(1 - Z_{ik} + 2 - x_{ij} - x_{kj}) & i \neq k \;\; i,k = \overline{1..n} \;\; j = \overline{1..m}. & (5)\\
Z_{ik} = 1 \;\; et \;\; t_i + p_i - t_k \leq 0 & T_i < T_k. & (6)\\
t_i + p_i \leq \widetilde{d_i} & i = \overline{1..n}. & (7)\\
t_i + p_i \leq Z & i = \overline{1..n}. & (8)\\
x_{ij}, Z_{ik} \in \{1,0\} \;\; et \;\; t_i, Z \geq 0 & i = \overline{1..n} \;\; j = \overline{1..m}. & (9)
\end{cases}
$$

Tout les contraintes de problème $(Pm \,/ Prec \,/ C_{max})$ restent les mêmes sauf :

- La (7) ème contrainte signifie que les tâches doivent être terminées avant la date limite.

Remarque 3.5.1 *Voici quelques résultats de complexité :*

Polynomial	*NP*-difficile
$(P/Pmtn/C_{max})$	$(P2//C_{max})$
$(P/chains, P_j = 1/C_{max})$	$(P//C_{max})$
$(P/tree, P_j = 1/C_{max})$	$(Pm//C_{max})$
$(P/Pmtn, tree/C_{max})$	$(P/Prec, P_j = 1/C_{max})$
$(P2/\boldsymbol{p_i} = \boldsymbol{p}, \boldsymbol{S_i} = \boldsymbol{s}, k \leq 2/\boldsymbol{C_{max}})$	$(P2/S_i = s, k=2/C_{max})$
. . .	$(P2/S_i, k/C_{max})$
. . .	$(Pm/Split, S_i, r_i/C_{max})$

Tableau 3.3 : La complexité de quelques problèmes d'ordonnancement [39]

3.6 Conclusion

Dans ce chapitre nous avons donné une définition de Makespan et les modèles mathématiques des problèmes d'ordonnancement, nous avons traité l'ordonnancement d'opérations sur des machines parallèles identiques en vue de minimiser la date d'achèvement ainsi on a considéré les ordonnancements non-préemptifs et préemptifs aussi les problèmes d'ordonnancement à contraintes de précédence sur des machines parallèles qui sont souvent difficiles ce qui implique qu'une résolution exacte n'est pas efficace.

Chapitre 4

Implémentation du problème $P3//C_{max}$

Chapitre 4

Implémentation du problème $P3//C_{max}$

4.1 Introduction

Dans ce chapitre, nous allons proposer trois heuristiques pour résoudre le problème d'ordonnancement $\boldsymbol{P3//C_{max}}$ qui est ***NP***-difficile, ensuite implémenter ces heuristiques en utilisant le langage de programmation MATLAB.

4.2 MATLAB

Est un langage de programmation de quatrième génération émulé par un environnement de développement du même nom, il est utilisé à des fins de calcul numérique. Développé par la société 'The Math Works'.

MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le **C**, **C++**, **Java**, et **Fortran**. Les utilisateurs de MATLAB (environ **2** millions en **2019**) sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche. Matlab peut s'utiliser seul ou bien avec la boîte à outils.

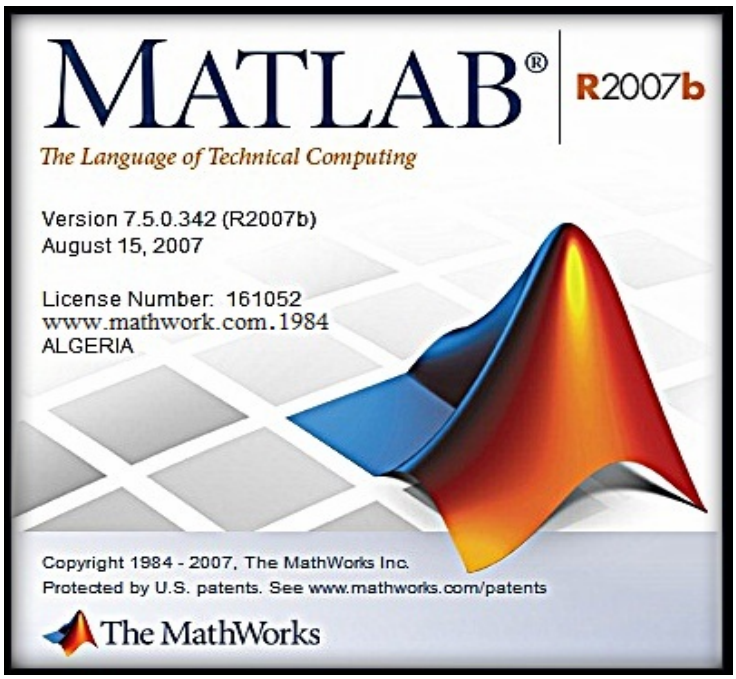


Figure 4.2.1 : L'interface de démarrage du logiciel MATLAB

4.3 Définition du problème $P3//C_{max}$

Le problème $\boldsymbol{P3//C_{max}}$ consiste à traiter un ensemble de tâches indépendantes sans préemption ayant des temps de traitement arbitraires des dates de disponibilité nulles. L'objectif de ce problème est de minimiser la date de fin de traitement.

4.4 Les heuristiques proposées

Dans cette session on va proposer des heuristiques lesquelles fournissent rapidement une solution réalisable proche de l'optimum pour le problème d'ordonnancement $\boldsymbol{P3//C_{max}}$, où notre but est de trouver des heuristiques de bonnes qualité pour cela on va calculer le ratio d'approximation $\boldsymbol{R}$ avec :

$$\boldsymbol{R = \frac{C_{max}(H)}{Q}}$$

Où :

- $\boldsymbol{C_{max}(H)}$: Est la longueur d'ordonnancement obtenue par l'heuristique proposée.
- $\boldsymbol{Q}$: Est la borne inférieur du problème $\boldsymbol{P3//C_{max}}$.

$$\boldsymbol{Q = max\left\{\max_{1\le i\le n}\{p_i\}, \left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil\right\}}.$$

Lorsque $\boldsymbol{R}$ tend vers **1** alors l'approximation est de bonne qualité.

Chaque heuristique proposée a un principe bien déterminer qui se base sur l'ordonnancement par liste donc la longueur d'ordonnancement de notre problème dépend de la liste de priorité, du nombre de machines et des temps de traitement des tâches.

4.4.1 Principe de la $1^{ère}$ heuristique

La première phase de cette heuristique consiste à ordonner les tâches selon la règle ***LPT*** c'est-à-dire ranger les tâches par ordre décroissant des temps de traitement $\boldsymbol{p_i}$ dans une liste $\boldsymbol{L}$ ensuite ordonnancer les tâches aux machines selon la procédure présentée dans la figure 4.4.1.

Algorithme H1

Début
  Appliquer la règle LPT.
    Tant que $L \neq \emptyset$ *faire*
      Affecter les tâches de la manière suivante : de la $1^{ère}$ *machine jusqu'à la dernière machine et de la dernière jusqu'à la première ainsi de suite jusqu'à la fin de la liste L.*
      (voir le schéma d'affectation 4.4.1)
      $L=L-\{T_i\}$.
    fait
  Calculer la longueur d'ordonnancement C_{max} .
Fin.

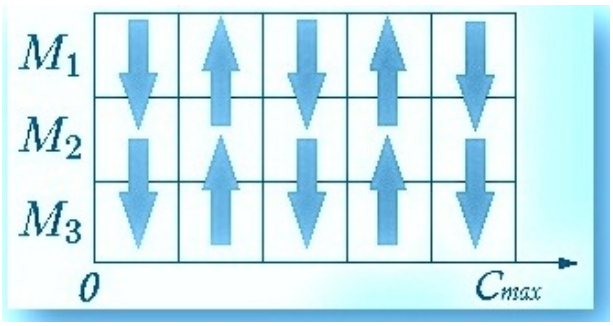


Figure 4.4.1 : Schéma représentant l'affectation des tâches selon la $1^{ère}$ heuristique

4.4.2 Principe de la $2^{ème}$ heuristique

L'heuristique suivante ressemble à la première heuristique la seule différence entre les deux se trouve dans l'affectation des tâches.

Algorithme H2

Début

Ranger les tâches suivant la règle LPT

Tant que $\boldsymbol{L} \neq \emptyset$ *faire*

Affecter les tâches de la manière suivant : de la $1^{ère}$ *machine jusqu'à la dernière machine et ainsi de suite jusqu'à la fin de la liste L.*

(voir le schéma d'affectation 4.4.2)

$L=L-\{T_i\}$.

fait

Calculer la longueur d'ordonnancement C_{max} .

Fin.

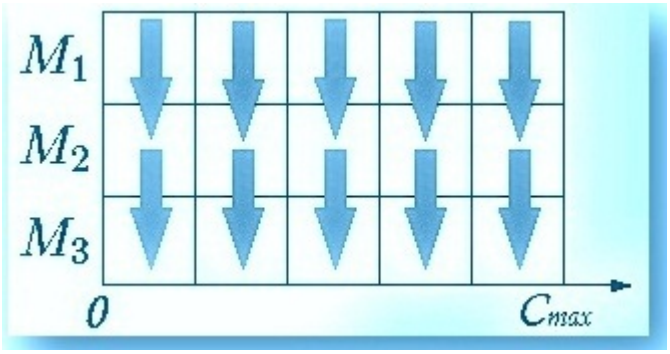


Figure 4.4.2 : Schéma représentant l'affectation des tâches selon la $2^{ème}$ heuristique

4.4.3 Principe de la $3^{ème}$ heuristique

Le principe de la troisième heuristique est un peu diffèrent des heuristiques précédentes, le problème d'ordonnancement reste le même, l'ordre des tâches suit la règle ***LPT***, tandis que l'ordonnancement concerne d'affecter la première tâche de la liste à la première machine libre.

Algorithme H3

Début

Ranger les tâches suivant la règle LPT

Tant que $L \neq \emptyset$ *faire*

Traiter la première tâche de la liste L sur la $1^{ère}$ *machine libre et compléter l'affectation des tâches de cette manière jusqu'à la fin de la liste L.*

$L=L-\{T_i\}$.

fait

Calculer la longueur d'ordonnancement C_{max} .

Fin.

4.5 Exemple d'application

Dans cette session on désire traiter ***12*** tâches indépendantes sur ***3*** machines parallèles identiques dont les temps de traitements sont données dans le tableau suivant et appliquer ce exemple sur les trois heuristiques présentées précédemment ensuite on va calculer le ratio d'approximation $\boldsymbol{R}$ pour chaque heuristique pour voir sa qualité.

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
p_i	8	9	5	3	1	1	2	4	3	6	2	5

Tableau 4.1 : La table des tâches et leurs temps de traitement

On trie les tâches selon l'ordre décroissant de $\boldsymbol{p_i}$, on obtient la liste de tâches suivante :

$$\boldsymbol{L = \{T_2, T_1, T_{10}, T_3, T_{12}, T_8, T_9, T_4, T_7, T_{11}, T_5, T_6\}.}$$

4.5.1 Application de la $1^{ère}$ heuristique

En appliquant notre heuristique H1, nous avons trouvé la solution présentée par la figure 4.5.1 avec $\boldsymbol{C_{max}(H1) = 17}$.

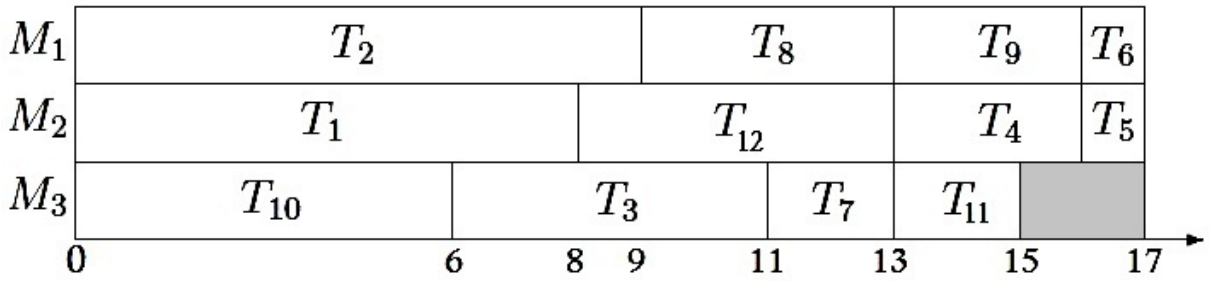


Figure 4.5.1 : Solution générée de l'exemple par l'heuristique H1

Et l'heuristique affiche la position et la machine qui exécute chaque tâche comme suit :

T_i	T_2	T_1	T_{10}	T_8	T_{12}	T_3	T_9	T_4	T_7	T_6	T_5	T_{11}
***J* :Machine**	1	2	3	1	2	3	1	2	3	1	2	3
***k* :Position**	1	1	1	2	2	2	3	3	3	4	4	4

Tableau 4.2 : La table des machines et la position des tâches pour H1

Pour calculer le ratio d'approximation $\boldsymbol{R}$ de cette heuristique on doit d'abord calculer sa borne inférieur :

$$\boldsymbol{Q = max\left\{\max_{1\le i\le n}\{p_i\}, \left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil\right\} = max\left\{9, \left\lceil \frac{1}{3}\sum_{i=1}^{n} p_i \right\rceil\right\} = max\{9, 17\} = 17}$$

Et donc :

$$\boldsymbol{R = \frac{C_{max}(H1)}{Q} = \frac{17}{17} = 1.}$$

Alors cette heuristique est de bonne qualité et elle a donné un ordonnancement optimale car $\boldsymbol{R = 1}$.

4.5.2 Application de la $2^{ème}$ heuristique

Reprenons l'exemple précédent, en appliquant l'heuristique ***2*** nous avons trouvé la solution présentée par la figure 4.5.2 avec $\boldsymbol{C_{max}(H2) = 19}$.

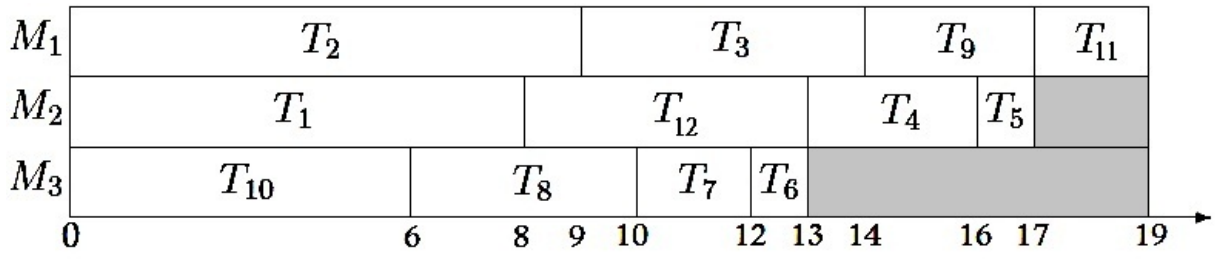


Figure 4.5.2 : Solution générée de l'exemple par l'heuristique H2

Et l'heuristique affiche la position et la machine qui exécute chaque tâche comme suit :

T_i	T_2	T_1	T_{10}	T_3	T_{12}	T_8	T_9	T_4	T_7	T_{11}	T_5	T_6
***J* :Machine**	1	2	3	1	2	3	1	2	3	1	2	3
***k* :Position**	1	1	1	2	2	2	3	3	3	4	4	4

Tableau 4.3 : La table des machines et la position des tâches pour H2

Si on veut calculer le ratio d'approximation $\boldsymbol{R}$ de cette heuristique on trouve :

$$\boldsymbol{R = \frac{C_{max}(H2)}{Q} = \frac{19}{17} = 1,12.}$$

Alors cette heuristique est de bonne qualité car $\boldsymbol{R = 1,12}$ proche de $\boldsymbol{1}$.

4.5.3 Application de la $3^{ème}$ heuristique

Sur le même exemple en appliquant notre heuristique, nous avons trouvé la solution présentée par la figure 4.5.3 avec $\boldsymbol{C_{max}(H3) = 17}$.

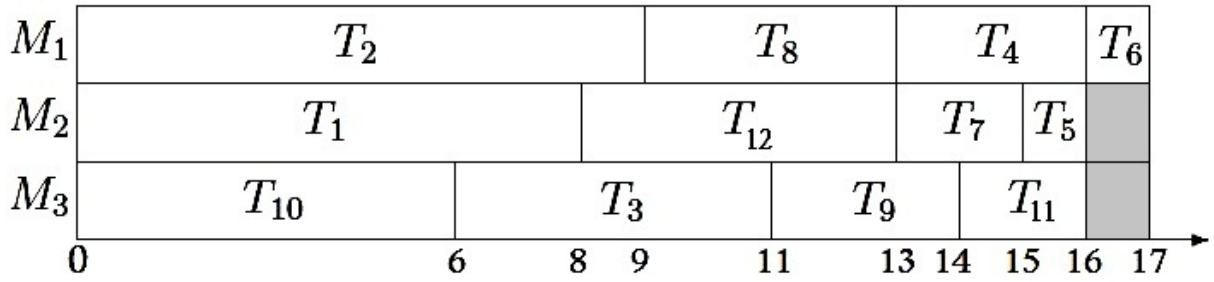


Figure 4.5.3 : Solution générée de l'exemple par l'heuristique H3

Le ratio d'approximation $\boldsymbol{R}$ est :

$$\boldsymbol{R = \frac{C_{max}(H3)}{Q} = \frac{17}{17} = 1.}$$

Alors cette heuristique est de bonne qualité et l'ordonnancement est optimal.

4.6 Implémentation des heuristiques sur MATLAB

Le temps d'exécution d'un programme dépend de plusieurs facteurs comme la représentation des données en mémoire, proprement dit de la structure de données représentant le problème, il dépend aussi de la puissance du processeur exécutant les instructions du programme. Un autre facteur d'importance décisif est la complexité algorithmique du programme qui mesure l'efficacité en termes de nombre d'itérations effectuées pour résoudre le problème.

L'efficacité de l'exécution mesurée en fonction de l'utilisation des ressources de l'ordinateur :

- Le temps de calcul pour exécuter les opérations composants les instructions.
- La taille mémoire nécessaire pour contenir et manipuler les programmes ainsi que leurs structures de données

Pour la première heuristique

Dans la figure 4.6.1 on va saisir les temps de traitement des tâches arbitrairement et en utilisant des commandes sur MATLAB pour arranger les tâches par ordre décroissant de leurs temps de traitement suivant la règle ***LPT***, ainsi notre programme indique qu'on a ***18*** tâches

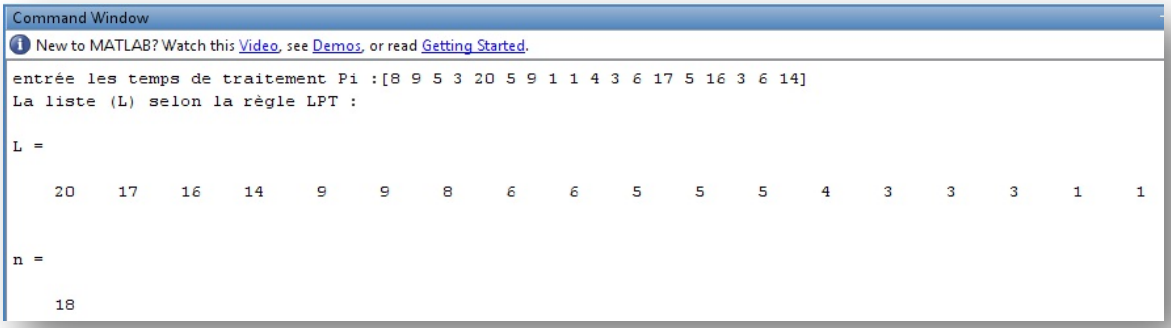


Figure 4.6.1 : La saisie et le rangement de temps de traitement des tâches

A partir de la liste ***L***, l'affectation des ***18*** tâches est faite sur la matrice qui représente le diagramme de Gantt illustré sur la figure 4.6.2

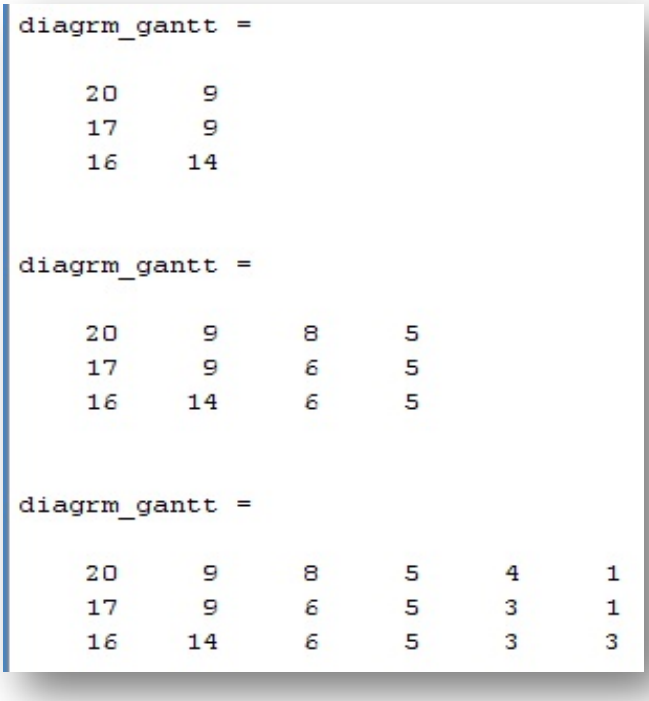


Figure 4.6.2 : La démarche du diagramme de Gantt

Notre programme a évalué le temps de fin de traitement : $\boldsymbol{C_{max} = 47}$.

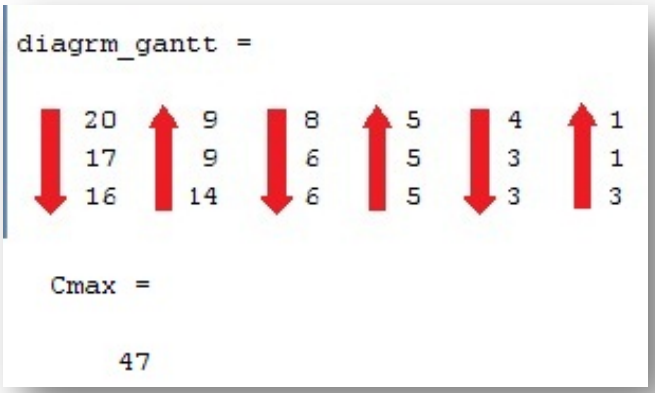


Figure 4.6.3 : La longueur d'ordonnancement C_{max}

A la fin on a affiché la position et la machine qui exécute chaque tâche sur la figure 4.6.4

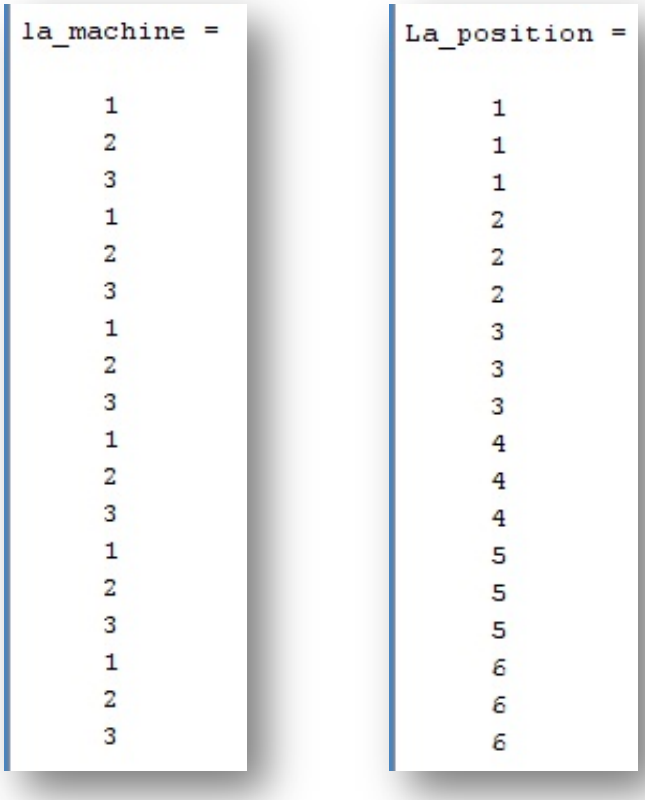


Figure 4.6.4 : Les positions et les machines qui exécutent chaque tâche

Pour la deuxième heuristique

Après la saisie des temps de traitement des ***18*** tâches et le tri des tâches avec la règle ***LPT*** dans la liste $\boldsymbol{L}$ **,** et en appliquant notre implémentation on obtient une matrice qui représente le diagramme de Gantt illustré sur la figure 4.6.5 et la longueur d'ordonnancement $\boldsymbol{C_{max}}$.

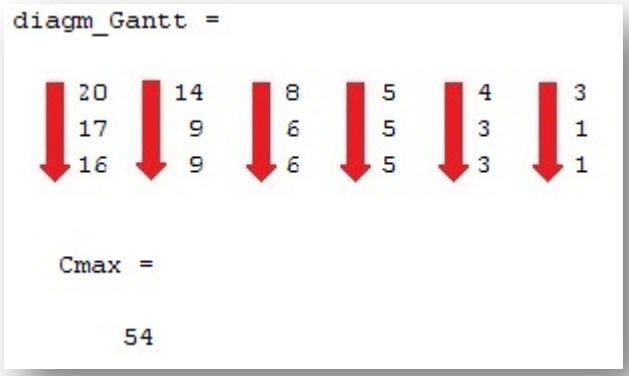


Figure 4.6.5 : Solution générée par l'heuristique H2

On a affiché la position et la machine qui exécute chaque tâche sur la figure 4.6.6

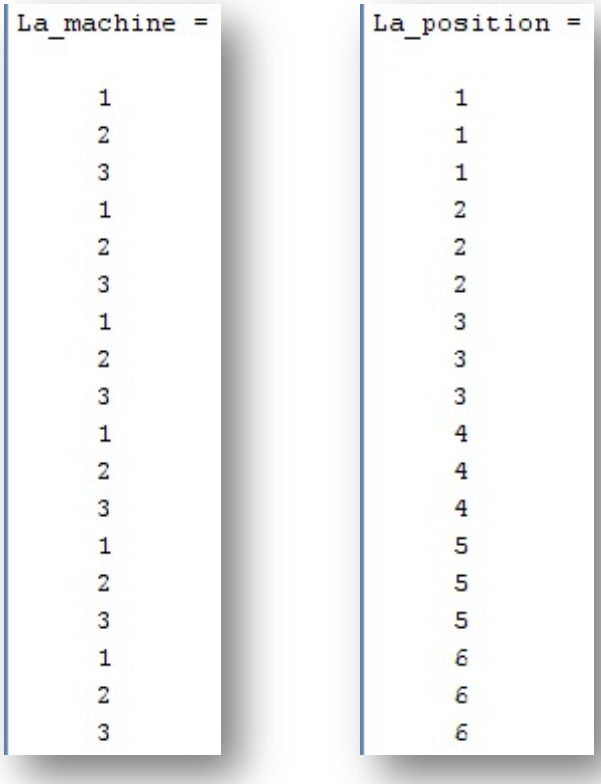


Figure 4.6.6 : Les positions et les machines qui exécutent chaque tâche

Pour la troisième heuristique

Avec notre implémentation on trouve les valeurs de $\boldsymbol{C_{max}}$ dans chaque étape sur les trois machines qu'ils sont illustrées sur la figure 4.6.7

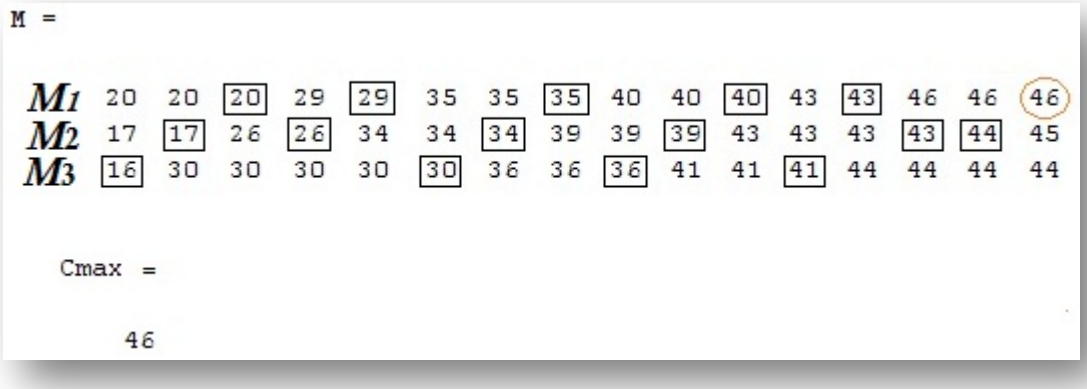


Figure 4.6.7 : Les valeurs de C_{max} dans chaque étape

4.7 Comparaison des résultats obtenus par les trois heuristiques

On a traité plusieurs exemples pour les trois heuristiques où le nombre de tâches est compris entre ***6*** et ***600*** avec des temps de traitement arbitraires.

Les résultats sont données dans le tableau 4.4, la $\mathbf{1}^{\grave{e}re}$ première ligne du tableau fournit le nombre de tâches, la deuxième ligne indique la borne inférieure du problème et les trois autres lignes successives indiquent la valeur de la fonction objective $\boldsymbol{C_{max}}$ respectivement les trois heuristiques.

C_{max} \ n	6	12	21	30	60	120	300	600
$Q\ (Borne\ inf)$	9	17	75	100	185	687	2217	5000
$C_{max}(Heuri : 1)$	10	17	78	100	191	690	2218	5007
$C_{max}(Heuri : 2)$	11	19	82	114	196	707	2230	5020
$C_{max}(Heuri : 3)$	9	17	75	100	185	687	2217	5000

Tableau 4.4 : La table des résultats

A partir des résultats obtenues, on remarque que la $\mathbf{1}^{\grave{\mathbf{e}}\boldsymbol{re}}$ et la $\mathbf{3}^{\grave{\mathbf{e}}\boldsymbol{me}}$ heuristique donnent les petites valeurs de la longueur d'ordonnancement car les $\boldsymbol{C_{max}}$ obtenues sont proches aux valeurs de la borne, ainsi on a constaté que l'heuristique **3** est de très bonne qualité elle nous a donné a chaque fois la solution optimal, mais elle reste une heuristique qui donne des solutions approchées (pour d'autres exemples, elle peut donner des solutions non optimaux). Par contre la $\mathbf{2}^{\grave{\mathbf{e}}\boldsymbol{me}}$ heuristique donne de grandes valeurs par rapport à la borne $\boldsymbol{Q}$, mais avec bonne qualité car son ratio d'approximation $\boldsymbol{R}$ reste toujours proche de **1**.

4.8 Conclusion

Dans ce chapitre on a présenté trois heuristiques adapté au problème $\boldsymbol{P3//C_{max}}$ en utilisant MATLAB comme outil d'implémentation et après l'exécution des travaux et la comparaison des résultats implémentés, on constate que les heuristiques choisies fournissent de bons résultats.

Conclusion générale

Conclusion générale

Les problèmes d'ordonnancement consistent à rechercher, dans un ensemble de solutions possibles, une solution particulièrement intéressante au regard d'un ou de plusieurs critères. La plupart de ces problèmes sont ***NP***-complets.

Nous nous sommes, tout d'abord, intéressés à l'étude de l'ordonnancement dans sa globalité, à savoir ses différentes composantes (tâches, ressources, contraintes, critères), ses différents types d'ateliers (job-shop, flow-shop et open-shop) et les méthodes d'optimisation les plus souvent utilisées pour sa résolution.

Des méthodes exactes (programmation linéaire, programmation dynamique) et aussi des méthodes approchées tel que (recuit simulé, recherche TABOU, Algorithmes génétiques et algorithmes de colonies de fourmis).

Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace mémoire requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution.

Par la suite, notre intérêt s'est porté sur l'étude de la minimisation du makespan sur des machines parallèles identiques ou nous avons traité plusieurs problème d'ordonnancement (avec et sans préemption, avec et sans précédence…) en donnant leurs modèles mathématiques ou l'objectif est de minimiser le $\boldsymbol{C_{max}}$.

Enfin on a traité l'implémentation du problème $\boldsymbol{P3//C_{max}}$ en utilisant trois heuristiques, une simulation sous le logiciel Matlab du programme proposé nous a permet de traiter plusieurs exemples par lesquels nous avons fait une comparaison entre l'utilisation des trois heuristiques.

Nous concluons que l'heuristique H3 basée sur le rangement des tâches selon la règle ***LPT*** est assez performante, et elle donne des résultats meilleurs que les deux heuristiques H1 et H2. Les principales perspectives de recherche qui apparaissent à l'issue de ce mémoire sont :

- Il est intéressent non seulement d'améliorer les heuristiques proposées pour le problème $\boldsymbol{P3//C_{max}}$ mais également la recherche de nouvelle méthode de résolution et de comparer ces résultats avec des méthodes exactes.
- Tenir compte des différents problèmes d'ordonnancement, et étudier les autres problèmes avec d'autres critères.
- Etudier l'hybridation des heuristiques avec d'autres méthodes approchées.
- La plupart des problèmes d'ordonnancement restent des problèmes difficiles qui offrent un large éventail de recherche.

Bibliographie

Bibliographie

[1] H.Adjiri, "*Les problèmes d'ordonnancement d'atelier : M-Machine identique en parallèle*", Mémoire de Master, Université Mohammed Boudiaf, M'sila, 2018.

[2] B.Ahmad Al Sheikh, "*Resource allocation in hard real-time avionic systems, Scheduling and routing problems. PhD thesis*", Institut National des Sciences Appliquées de Toulouse, Département Systèmes Informatiques et Systèmes Embarqués, 2011.

[3] V.Barichard, "*Approches hybrides pour les problèmes multiobjectifs*", Thèse de Doctorat, Université d'Angers, 2003.

[4] R.Bellman, "*Dynamic Programming*", Operations research, Mineola, New-York, vol 50, n° 1, pp 48 − 51, Jan-fév 2002.

[5] F.Benhamou, "*volume* 4204 *of Lecture Notes in Computer Science*". Springer, 2006.

[6] A.Benkaci, Notes de cours "*Optimisation combinatoire*", pour M2 RO, Univesité de Médéa 2016.

[7] H.Boukef ben othman, "*L'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques, optimisation par algorithmes génétiques et essaims particulaires*", Thèse de doctorat, Université de TUNIS EL MANAR, juillet 2009.

[8] H.Boukef, F.Tangour, M.Benrejeb et P.Borne, "*Sur la formulation d'un problème d'ordonnancement de type flow-shop d'ateliers de production en industries pharmaceutiques*", JTEA'06, Hammamet 12 − 14, mai 2006.

[9] M.H.Boumediene, "*Les problèmes d'ordonnancement a machines parallèles, de tachês dependantes*", Mémoire de magister, Univesité SAAD DAHLAB de Blida, juillet 2006.

[10] M.Bourzik, "*Problèmes d'Ordonnancement d'Atelier*", Mémoire de Licence, Université SIDI MOHAMED BEN ABDELLAH, juin 2016.

[11] J.Carlier, P.Carlier, P.Chrétienne et C.Girault, "*Modelling scheduling problems with Petri nets. Advanced studies in Petri nets*" ,"*Lecture notes in Computer Science*", Springer Verlag, Paris, 1984.

[12] J.Carlier, P.Chrétienne, "*Problèmes d'ordonnancement, Modélisation, Complexité, Algorithmes*", Masson, Paris, 1988.

[13] J.Carlier, "*Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité*", Thèse de Doctorat, Université Paris VI, 1984.

[14] A.Chergui, A.Dahmani et A.Adda.A, "*Ordonnancement d'un flow-shop par métaheuristique hybride*", Mémoire de master, Université Abou Bekr Belkaid,Tlemcen, Juin 2017.

[15] S.Cheng et A.K.Agrawala, "*Allocation and scheduling of real-time periodic tasks with relative timing constraints. In Second International Workshop on Real-Time Computing Systems and Applications*" (RTCSA), IEEE, pp $25-27$, 1995.

[16] P.Chrétienne, "*Les réseaux de pétri temporises*", Thèse de Doctorat, Université de Paris VI, Paris, 1983.

[17] Y.Collette, P.Siarry, "*Optimisation Multiobjectif*", Eyrolles, Paris, 2002.

[18] S.A.Cook, "*The complexity of theorem-proving procedures*", "Proceedings of the $3rd$ Annual ACM Symposium on the Theory of Computing", STOC, pp $151-158$, 1971.

[19] G.B.Dantzig et M.N.Thapa, "*Linear programming 2 : Theory and. Springer Series in Operations Research*", 2003.

[20] C.Ekelin et J.Jonsson, "*Solving embedded system scheduling problems using constraint programming, Technical report, Dept. of Computer Engineering*", Chalmers University of Technology, 2000.

[21] S.El Bahloul, "*Flow-shop à deux machines avec des temps de latence : approche exacte et heuristique*", Mémoire présenté comme exigence partielle, Université du Québec partielle de la maitrise en informatique, 2008.

[22] K.Fekhar, "*Evolution de la cryptographie et robustesse des nouveaux cryptosystèmes*", Mémoire de master, Univesité de Médéa, Septembre 2018.

[23] F.Glover, M.Laguna, "*Tabu search*", KluwerAcademic Publishers, 1997.

[24] G.Gotha, "*Les problèmes d'ordonnancement*", Recherche Opérationnelle, RAIRO, vol. 27, n°1, pp. 77 – 150, 1993.

[25] C.Hamzacebi, F.Kutay, "*A heuristic approach for finding theglobal minimum : Adaptive random search technique*", Applied Mathematicsand Computation, 173 : 1323 – 1333, 2006.

[26] J.K.Hao, P.Galinier et M.Habib, "*Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes*", Revue d'Intelligence Artificielle, vol (1999), pp 2 – 39, 1999.

[27] P.E.Hladik, "*Ordonnançabilité et placement des systèmes temps réel ditribuès préemtifs et à priorités fixes*", PhD thesis, Université de Nantes, Spécialité Automatique et informatique Appliquée, 17/12/2004.

[28] J.H.Holland, "*Adaptation in natural and artificial systems*", PhD, Thesis Michigan Press University, Ann Arbor, Michigan, 1975.

[29] A.S.Jain, S.Meeran, "*Deterministic job-shop scheduling : past, present and future*", European Journal of Operational Research, vol. 113, pp. 390 – 434, 1999.

[30] I.Kacem , "*Ordonnancement multicritère des job-shops flexibles : formulation, bornes inférieures et approche évolutionniste coopérative*", Thèse de Doctorat, Université de Lille 1, 2003.

[31] S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi, "*Optimisation by simulated annealing*", Science, Vol 220, pp 671 – 680, 1983.

[32] S.Kouider Aïad, Notes de cours "*Formulation des problèmes d'ordonnancemnt*", pour M2 RO, Univesité de Médéa 2008.

[33] N.Liouane, "*Contribution à l'élaboration d'un outil d'aide à la décision pour l'ordonnancement de production en environnement incertain*", Thèse de Doctorat, Université des Sciences et Technologies de Lille1, 1998.

[34] R.McNaughton, "*Scheduling with deadlines and loss functions*", Management Sciences, $6 : 1 - 12$, 1959.

[35] K.Mesghouni , "*Application des algorithmes évolutionnistes dans les problèmes d'optimisation en ordonnancement de la production*", Thèse de Doctorat, Université des Sciences et Technologies de Lille 1, 1999.

[36] M.E.A.Meziane , "*Optimisation par phases pour les problèmes d'ordonnancement des ateliers de type job-shop totalement flexibles*", Mémoire de magister en des sciences, Université d'Oran, Algérie 2011.

[37] T.Nicholson, "*Optimization in industry*", Editions Longmann Press, Londres, 1971.

[38] H. Pierre-Emmanuel, C.Hadrien, A.M.Deplanche, et J.Narendra, "*Solving a real-time allocation problem with constraint programming*" J.S.Syst, $81 : 132 - 149$, Jan 2008.

[39] L.Pinedo, M.Michael, "*Scheduling : Theory, Algorithms, and Systems*", Springer, NewYork,Ed 3 pp $111 - 129$, 2008.

[40] B.Roy, "*Algèbre moderne et théorie des graphes, volume* 2", Dunod, Paris, 1970.

[41] M.H.Rothkopt, "*scheduling independant tasks on parallel processors*", Management Science, vol. 12, No.5, Jan1966.

[42] M.Sacarovich, "*Graphes et Programmation Linéaire*", Hermann, Paris, 1984.

[43] M.Sacarovich "*Optimisation combinatoire*", Hermann, Paris, 1984.

[44] K.Subramani, "*Periodic linear programming with applications to real-time scheduling*", Mathematical. Structures in Comp. Sci, $15 : 383 - 406$, April 2005.

[45] R.J. Vanderbei, "*Linear programming: Foundations and extensions*", 1996.

[46] M.Widmer, "*Les Métaheuristiques : Des outils performants pour les problèmes industriels*", 3*ème* Conférence Francophone de MO délisation et SIMulation MOSIM'01,Troyes, 25 − 27, avril 2001.

[47] Wikipédia, www.wikipédia "*La méthode de PERT*", http://back.ac-rennes.fr/ pedagogie/ espaceeg/ prodpeda/ spicialites/ pert/ pert.html, consulté le 28 mars 2019.

[48] Wikipédia, www.wikipédia "algorithme_de_colonies_de_fourmis.com", consulté le 4 avril 2019.

[49] F. Yalaoui, C.Chu, "*Parallel machine scheduling with job splitting and setup times : a heuristic approach and performance analysis*", Proceedings of the International Conference on Industrial Engineering and Production Management, Glasgow, vol. 2, p. 297 − 305, 1999.

[50] M.Zouba, P.Baptiste et D.Rebaine, "*Ordonnancement de jobs sur deux machines par-all'eles en presence d'un seul opérateur, 6ième conférence froncophone de Modélisation et Simulation*" , MOSIM'06 2006.