Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Université Dr.Yahia Farès de Médéa

Faculté des Sciences -Département de Mathématiques & Informatique



MÉMOIRE

présenté en vue de l'obtention

du

Diplôme de Master en Mathématiques
Spécialité : Recherche Opérationnelle

Par

ZEMOUR Kheira et TOUAL Bouchra

Thème

Minimisation du temps de fin de traitement moyen sur une seule machine

Soutenu devant le jury composé de :

M_r	N. IKHLEF ECHOUF	Professeur	UYF Médéa	Président
M_r	A. BENKACI	Maitre assistant	UYF Médéa	Examinateur
M_{me}	S. KOUIDER AIAD	Maitre assistant	UYF Médéa	Promotrice

17 juin 2023

Remerciements

Tous d'abord, nous tiens à remercier **Allah** clément et miséricordieux de notre avoir donné la force et le courage de mener à terme ce modeste travail. Comme, c'est un plaisir de remercier tous ce qui ont contribué à la réalisation de ce mémoire :

En premier lieu, nous voudrons exprimer nos remerciements les plus sincères à notre promotrice **Mme S. KOUIDER AIAD**, pour notre avoir guidé pour la réalisation de cette étude et le soutient scientifique et moral.

Monsieur **N. IKHLEF ECHOUF** Professeur à l'Université Dr.Yahia Farés de Médéa, de nous avoir fait l'honneur de présider le jury de ce mémoire.

Nous tenons à remercier Monsieur **A. BENKACI** Maitre assistant à l'Université Dr.Yahia Farés de Médéa parce qu'il nous a fait l'honneur d'être membres du jury. Ainsi que pour consacrer une partie de son temps précieux pour lire et corriger ce mémoire.

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.

Enfin nous remercions toutes les personnes qui ont aidé de prés ou de loin à la réalisation de ce projet.

Dédicaces

Les sons de la joie sont venus, et ce qui était hier un rêve s'est approché. Mon rêve tant attendu s'est approché, et les phrases se sont mélangées pour garder des souvenirs dans le cur. Dieu soit loué d'abord et avant tout.

Je dédie cette cérémonie de remise des diplômes à mon gentil père "Youcef" mon modèle et mon idéal dans la vie ; Il m'a appris à vivre avec dignité et dignité.

À ma mère bien-aimée "Zahra" Je ne trouve pas de mots qui puissent lui rendre hommage, car elle est l'épopée de l'amour et de la joie de vivre, et l'exemple du dévouement et du don.

À mes frères "Jamila", "Omar", "Hisham" mon soutien et ma force et le partage de mes joies et de mes peines.

À mes deuxièmes parents "mohamed" et "Fatom".

À tous ma famille "ZEMOUR" et la famille "Bekkaye".

À ma binôme "Bouchra", mon intime amie et ma sur.

À tous les amis qui m'ont accompagné et soutenu.

À tout mes collègues de la promo RO.

pour tous ceux qui me connaissent et qui m'aiment.

KHEIRA ZEMOUR

Dédicaces

Le voyage n'a pas été court et le rêve n'était pas proche, mais je l'ai fait, Dieu merci d'abord et avant tout.

Je dédie cette graduation à celui dont je porte le nom avec fierté, à celui qui m'a ouvert le chemin de la connaissance, "mon cher père". Après la grâce de Dieu, ce sur quoi je suis vous reviendra, mon cher père.

À la propriétaire de l'odeur du ciel, ma chère mère, qui était et sera toujours une lumière sur mon chemin, que Dieu la protège pour moi.

Et à tous mes chers frères.

À tous ceux qui m'aiment sincèrement et à tous ceux qui m'ont tendu la main et m'ont aidé dans mon parcours d'études.

À tous les amis qui m'ont accompagné et soutenu.

Heureusement, d'abord et avant tout, le rêve tant attendu s'est réalisé.

TOUAL BOUCHRA

Résumé

Un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des tâches et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs.

Pour l'ordonnancement à une seule machine, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine.

Dans ce travail, nous intéressons à étudier quelques problèmes d'ordonnancement avec temps de fin de traitement moyen sur une seule machine à savoir des problèmes faciles avec sa résolution et des problèmes difficiles avec la preuve de son difficulté. Nous représentons problème NP-difficile et nous proposons quelques heuristiques pour le résoudre et nous implémentons ces heuristiques sur Matlab et nous les comparons utilisant des exemples différents.

Mots clés : Ordonnancement, temps de fin de traitement, complexité, NP-difficile, heuristique.

Abstract

Scheduling is a solution to the scheduling problem. It is defined by the task execution schedule and resource allocation, aiming to achieve one or multiple objectives.

For single-machine scheduling, the set of tasks to be performed is handled by a single machine. The tasks consist of a single operation each, which requires the same machine.

In this work, we are interested in studying some scheduling problems with total completion time on a single machine, namely easy problems with its resolution and difficult problems with proof of its diffculty. We represent NP-hard problem and we propose some heuristics to solve it and we implement these heuristics on Matlab and we compare them using different examples.

Keywords : Scheduling, total completion time, complexity, NP-hard, heuristic.

Table des matières

Résumé **i**

Abstract **ii**

Introduction Générale **2**

1 Notions de base **4**

1.1 Introduction 4

1.2 Présentation d'un problème d'ordonnancement 4

1.3 Formulation d'un problème d'ordonnancement 4

1.3.1 Les tâches 5

1.3.2 Les ressources 5

1.3.3 Les contraintes 6

1.3.4 Les objectifs 6

1.4 Caractérisation des machines d'un atelier 7

1.4.1 Les machines parallèles 7

1.4.2 Les machines spécialisées 8

1.5 Classification d'un problème d'ordonnancement d'atelier 10

1.5.1 Champ α : Ressources 10

1.5.2 Champ β : Contraintes 11

1.5.3 Champ γ : Critère d'optimisation 11

1.6 Représentation des problèmes d'ordonnancement par un diagramme de Gantt 12

1.7 Notions de complexité 13

1.7.1 Les principes fondamentaux de complexité 13

1.7.2 Les classes de complexité 15

1.7.3 Réduction entre problèmes d'ordonnancement 16

1.8 Conclusion 18

2 Quelques problèmes avec temps de fin de traitement moyen 19

2.1 Introduction 19

2.2 Problème $1//\overline{C}$ 19

2.2.1 Description du problème 19

2.2.2 Complexité du problème 19

2.2.3 Résolution de problème 19

2.3 Problème $1/r_i, pmtn/\overline{C}$ 21

2.3.1 Description du problème 21

2.3.2 Complexité du problème 21

2.3.3 Résolution de problème 21

2.4 Problème $1/r_i/\overline{C}$ 23

2.4.1 Description du problème 23

2.4.2 Complexité du problème 23

2.5 Conclusion 25

3 Implémentation et heuristiques proposées pour résoudre le problème $1/r_i/\overline{C}$ 26

3.1 Introduction 26

3.2 Modèle mathématique du problème $1/r_i/\overline{C}$ 26

3.3 Heuristiques pour résoudre le problème $1/r_i/\overline{C}$ 27

3.3.1 Définition 27

3.3.2 Heuristiques proposés 28

3.3.3 Exemple d'application 29

3.4 Résolution du problème et implémentation 33

3.4.1 Choix du langage 33

3.4.2 Généralités sur le langage Matlab 33

3.4.3 Exemple illustratif 34

3.4.4 Résultats numériques 34

3.4.5 Comparaison entre les heuristiques proposées 37

3.5 Conclusion 38

Conclusion Générale 36

BIBLIOGRAPHIE 37

Table des figures

1.1 Problème à machine parallèles 8
1.2 Flow shop 9
1.3 Open shop 9
1.4 Jop shop 9
1.5 Diagramme de Gantt d'un ordonnancement de l'exemple 1.2 12
1.6 Schéma représentatife décision de classes de complexité 15
1.7 Hiérarchie de complexité entre différents problèmes 17
1.8 Hiérarchie de complexité des relaxations de problèmes 18

2.1 Solution optimale pour l'exemple 2.1 21
2.2 Solution optimale de l'exemple 2.2 22
2.3 Distribution de tâches dans un ordonnancement basé le problème de 3-partition 24

3.1 Solution de l'exemple par l'heuristique H_1 30
3.2 Solution de l'exemple par l'heuristique H_2 31
3.3 Solution de l'exemple par l'heuristique H_3 33
3.4 Programme Matlab de l'algorithme H1 35
3.5 Programme Matlab de l'algorithme H2 35
3.6 Programme Matlab de l'algorithme H3 36
3.7 Résultats de l'exemple 6 pour $n = 6$ 36
3.8 Résultats de l'exemple 2 pour $n = 15$ 37

Liste des tableaux

1.1 Les données de l'exemple 1.2 12

2.1 Les temps de traitement de l'exemple 2.1 20
2.2 Les données de l'exemple 2.2 22
2.3 Résolution de l'exemple 2.2 22

3.1 Les données de l'exemple d'application 30
3.2 Les données de l'exemple 1 34
3.3 Les données de l'exemple 2 34
3.4 Les résultats des trois heuristiques 37
3.5 Comparaison entre les trois heuristiques 38

Liste des algorithmes

1 Algorithme de H_1 28
2 Algorithme de H_2 29
3 Algorithme de H_3 29

Notations et Abréviations

n	Nombre de tâches (jobs, traveaux).
m	Nombre de machines.
i	Indice des tâches.
j	Indice des machines.
$T = \{T_1, ..., T_n\}$	Ensemble des tâches.
$M = \{M_1, ..., M_m\}$	Ensemble des machines.
P	Polynomial.
NP	Non polynomial.
pmtn	Préemption.
S_i	Temps de préparation.
r_i	Date de disponibilité de la tâche T_i.
p_i	Temps de traitement de la tâche T_i.
C_i	Date de fin de traitement de la tâche T_i.
d_i	Date échue de la tâche T_i.
$\tilde{d}_i$	Date limite de la tâche T_i.
C_{max}(makespan)	Date de fin de traitement de toutes les tâches.
L_{max}	La grande tardivité.
F_{max}	Le grand temps de flow.
T_{max}	Le grand retard.
$\overline{C}$	Le temps de fin de traitement moyen.
$\overline{C_w}$	Le temps de fin de traitement moyen pondéré.
$\overline{F}$	Le temps de flow moyen.
$\overline{F_w}$	Le temps de flow moyen pondéré.
$\overline{L}$	La tardivité moyenne.
$\overline{L_w}$	La tardivité moyenne pondérée.
$\overline{T}$	Le retard moyen.
$\overline{T_w}$	Le retard moyen pondéré.

Introduction Générale

La recherche opérationnelle peut être définie comme l'ensemble des méthodes et techniques rationnelles orientées vers la recherche du meilleur choix dans la façon d'opérer en vue d'aboutir au résultat visé ou au meilleur résultat possible ou encore au résultat optimal. Elle fait partie des "aides à la décision" dans la mesure où elle propose des modèles conceptuels en vue d'analyser et de maitriser des situations complexes pour permettre aux décideurs de comprendre, d'évaluer les enjeux et d'arbitrer ou de faire les choix les plus efficaces.

La théorie de l'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales de tâches. Pour cela, il est très souvent nécessaire d'affecter en même temps les ressources nécessaires à l'exécution de ces tâches. Un problème d'ordonnancement peut être considéré comme un sous-problème de planification dans lequel il s'agit de décider de l'exécution opérationnelle des tâches planifiées.

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises. En production (manufacturière, de biens, de service), on peut le présenter comme un problème où il faut réaliser le déclenchement et le contrôle de l'avancement d'un ensemble de commandes à travers les différents centres composant le système. Un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des tâches ("ordre" et "calendrier") et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs. Un ordonnancement est très souvent représenté par un diagramme de Gantt.

Dans notre travail, nous nous intéressons à l'ordonnancement avec temps de fin de traitement moyen sur une seule machine. Pour cela nous divisons ce travail en trois chapitres, ils sont organisé comme suit :

Dans le premier chapitre nous traitons les notions de base de l'ordonnancement, présentation, formulation, classification d'un problème d'ordonnancement et la théorie

de la complexité. Le deuxième chapitre est consacré à étudier quelques problèmes avec temps de fin de traitement moyen à savoir des problèmes faciles avec sa résolution et des problèmes difficiles avec la preuve de son difficulté.

Dans le troisième chapitre nous représentons le problème $1/r_i/\overline{C}$ et nous proposons trois heuristiques pour le résoudre, nous appliquons ces heuristiques à un exemple donné.

Enfin, nous implémentons ces trois heuristique sur Matlab et nous les comparons utilisant des exemples différents.

Chapitre 1

Notions de base

1.1 Introduction

La théorie de l'ordonnancement est une branche de la recherche opérationnelle, dans ce chapitre nous présentons les notions de base nécessaires de la théorie de l'ordonnancement, notamment la présentation, la formulation et la classification d'un problème d'ordonnancement, ainsi que la théorie de la complexité.

1.2 Présentation d'un problème d'ordonnancement

Puisqu'il existe un grand nombre de définition de l'ordonnancement dans la littérature, nous présentons deux définitions courtes mais typiques comme illustré ci-dessous :

Définition 1.1. *Le problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchainement,.....) [15].*

Définition 1.2. *Ordonnancer un ensemble de tâches, c'est programmer leurs exécution en leurs allouant les ressources requises et en fixant leurs date de début, comme l'ordonnancement concerne l'allocation de ressources limitées aux activités avec l'objectif d'optimiser un ou plusieurs performance [16].*

1.3 Formulation d'un problème d'ordonnancement

Les éléments nécessaires pour formuler un problème d'ordonnancement sont :

1.3.1 Les tâches

Une tâche est une entité élémentaire de travail localisé dans le temps par une date de début et une date de fin, dont la réalisation est caractérisée par une durée positive [13].
Une tâche notée T_i est définie par ses caractéristiques temporelles :
p_i : temps de traitement de la tâche T_i.
S_i : temps de préparation de la tâche T_i.
r_i : date de disponibilité de la tâche T_i.
d_i : date échue de la tâche T_i.
d_i : date limite ou absolue de la tâche T_i.
C_i : date de fin de traitement de la tâche T_i.
W_i : poids associé à la tâche T_i.
$l_i = C_i - d_i$: tardiveté de la tâche T_i.
$t_i = \max\{l_i, 0\}$: *retard* de la tâche T_i.
U_i : indicateur de retard de la tâche $T_i = \begin{cases} 1 & \text{si } C_i > d_i \\ 0 & \text{sinon} \end{cases}$
$f_i = C_i - r_i$: durée de flow.
Dans le reste de ce mémoire, nous avons :
L'ensemble des n tâches est noté $T = \{T_1, T_2, ..., T_n\}$.
L'ensemble des m machines est noté $M = \{M_1, M_2, ..., M_n\}$.
Une tâche est constituée d'un ensemble d'opérations qui requiert, pour son exécution, certaines ressources et qu'il est nécessaire de programmer de façon à optimiser un certain objectif. On distingue deux types de tâches :

Les tâches morcelables : (préemptibles) qui peuvent être exécutées en plusieurs fois, facilitant ainsi la résolution de certains problèmes.

Les tâches non morcelables : (indivisibles) qui doivent être exécutées en une seule fois et ne sont interrompues qu'une fois terminées.

1.3.2 Les ressources

Une ressource est un moyen technique ou humain qui sert à l'exécution d'une tâche. On trouve plusieurs types de ressources [8] :

Les ressources renouvelables : qui, redevient disponible après avoir attribué une tâche(les machines, le personnel, etc).

Les ressources consommables : qui, n'est plus disponible après utilisation (l'argent, la

matière première, etc).

Les ressources doublement contraintes : qui, est utilisé instantanée et sa consommation globale sont toutes les deux limitées (les sources d'énergies, de financement, etc).

Les ressources disjonctives : qui, ne peut exécuté qu'une seule tâche à la fois (les machines, rebots, etc).

Les ressources cumulatives : qui, est partageable, peut être utilisées par plusieurs tâches en même temps.

1.3.3 Les contraintes

Les contraintes représentent les restrictions imposées d'une part sur les relations reliant les tâches, et d'autre part sur les ressources. On distingue donc deux types de contraintes [5] :

Les contraintes de ressources : Ce type de contrainte s'exprime à travers la nature des ressources, qu'elles soient disjonctives ou bien cumulatives. Dans le premier cas, deux tâches ne peuvent s'exécuter en même temps sur la même ressource. Dans le deuxième cas, la contrainte exprime le fait que la somme des besoins simultanés des tâches doit être inférieure à la capacité de la ressource.

Les contraintes temporelles : Les contraintes temporelles peuvent être soit des contraintes d'antériorité ou bien des contraintes de temps alloué. Le premier type est une contrainte de succession entre deux tâches d'une même gamme de production. Le deuxième type est relatif aux dates limites des tâches.

1.3.4 Les objectifs

Généralement l'objectif d'un problème d'ordonnancement consiste à optimiser un ou plusieurs critères ou à trouver une solution admissible qui respect les contraintes du problème [3].
Les critères liés aux paramètres temporels caractérisant les tâches, les plus utilisés sont [12] :

— Longueur d'ordonnancement : $C_{max} = \max_{1 \leqslant i \leqslant n} \{C_i\}$.

— Le temps de fin de traitement moyen : $\overline{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$.

— Le temps de fin de traitement moyen pondéré : $\overline{C_w} = \dfrac{\sum_{i=1}^{n} w_i C_i}{\sum_{i=1}^{n} w_i}$.

— La grande tardivité : $L_{max} = \max_{1 \leqslant i \leqslant n} \{l_i\}$.

— La tardivité moyenne : $\overline{L} = \dfrac{1}{n} \sum_{i=1}^{n} l_i$.

— La tardivité moyenne pondérée : $\overline{L_w} = \dfrac{\sum_{i=1}^{n} w_i l_i}{\sum_{i=1}^{n} w_i}$.

— Le grand temps de flow : $F_{max} = \max_{1 \leqslant i \leqslant n} \{f_i\}$.

— Le temps de flow moyen : $\overline{F} = \dfrac{1}{n} \sum_{i=1}^{n} f_i$.

— Le temps de flow moyen pondéré : $\overline{F_w} = \dfrac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$.

— Le grand retard : $T_{max} = \max_{1 \leqslant i \leqslant n} \{t_i\}$.

— Le retard moyen : $\overline{T} = \dfrac{1}{n} \sum_{i=1}^{n} t_i$.

— Le retard moyen pondéré : $\overline{T_w} = \dfrac{\sum_{i=1}^{n} w_i t_i}{\sum_{i=1}^{n} w_i}$.

— Le nombre de tâches en retard : $N_T = \sum_{i=1}^{n} u_i$.

1.4 Caractérisation des machines d'un atelier

Il existe en général 2 types de machines [12] :

1.4.1 Les machines parallèles

Ces machines exécutent les mêmes traitement et sont donc capable de traiter n'importe quelle tâche.

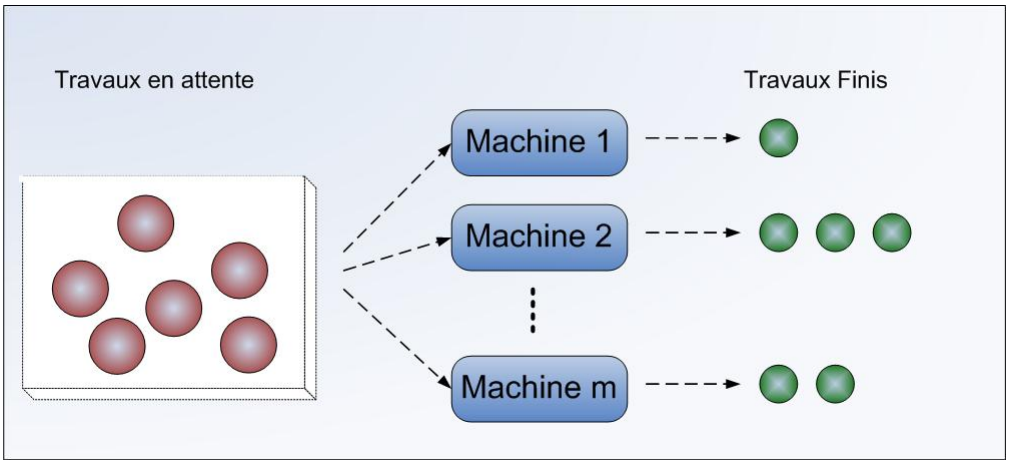


FIGURE 1.1 – Problème à machine parallèles

En fonction de la vitesse de traitement, nous distinguons 3 types de machines parallèles :

Machines identiques :

Pour ces machines, les vitesses de traitement des tâches sont toutes égales.

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 de la tâche T_i.

Par example si une opération est exécutée deux fois plus rapidement sur une machine M_1 que sur une machine M_2, cela reste vraie pour toute autre opération exécutable par ces machines parallèles.

Machines générales :

Contrairement au cas précédent, pour ces machines, la vitesse de traitement d'une machine dépend de la tâche qui est traitée.

1.4.2 Les machines spécialisées

Ces machines sont spécialisées à l'exécution de certaines tâches. Dans le cas de machines spécialisées, nous distinguons 3 systèmes ou mode de traitement :

Flow shop :

Dans un système Flow shop, les tâches doivent être traitées par toutes les machines, et l'ordre des opérations est fixé et est le même pour toutes les tâches.

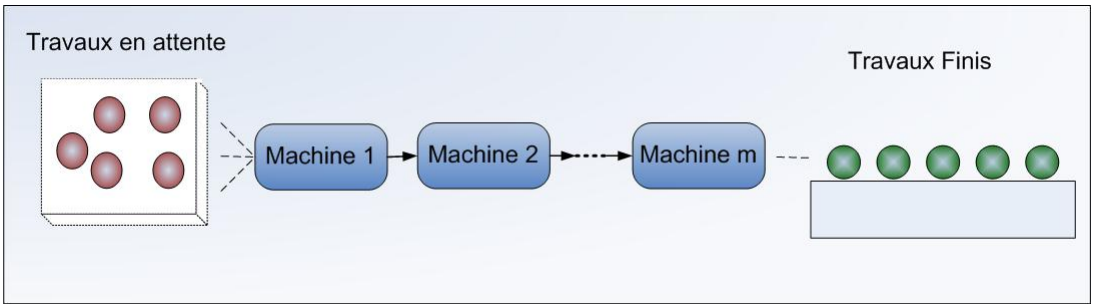


FIGURE 1.2 – Flow shop

Open shop :

Dans un système Open shop, les tâches doivent être traitées par toutes les machines mais l'ordre de traitement n'est pas nécessairement le même pour toutes les tâches. L'ordre de traitement est quelconque.

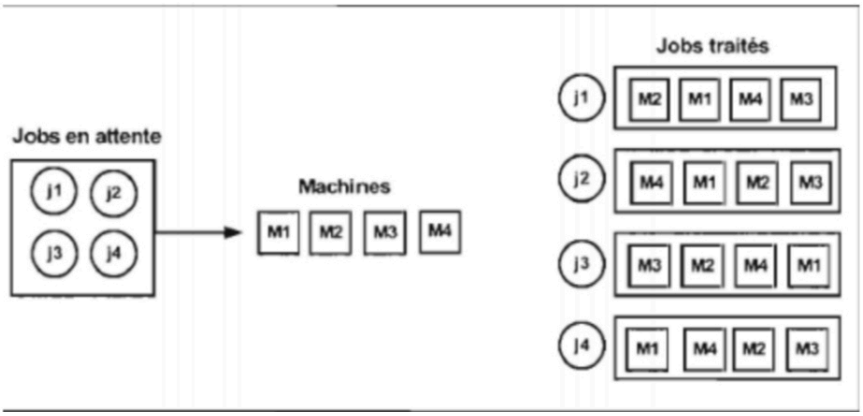


FIGURE 1.3 – Open shop

Jop shop :

Dans un système Job shop, un sous-ensemble de machines devant traiter une tâche et l'ordre du traitement est arbitraire, mais doit être spécifié à l'avance.

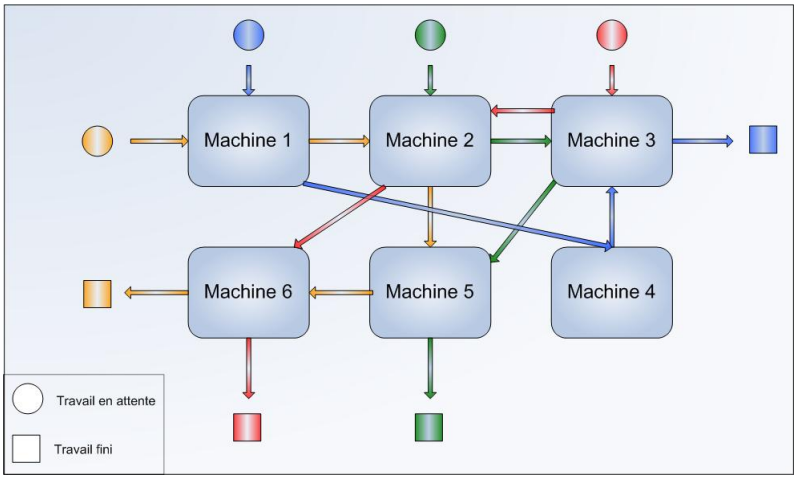


FIGURE 1.4 – Jop shop

1.5 Classification d'un problème d'ordonnancement d'atelier

Pour des raisons de compréhension et que les choses soit claires. Il est important d'adopter un descripteur qui soit basé sur l'existant et homogénéisant les notions actuelles [9].
La notation $\alpha/\beta/\gamma$ introduite par Graham et al [15]. est couramment utilisée pour distinguer les différents problèmes d'ordonnancement entre eux et les classifier.

1.5.1 Champ α : Ressources

Renseigne les informations relatives aux ressources à savoir le nombre de machine, type de machine et les types d'ateliers. Il est composé de deux sous champs et désigné par α_1, α_2 :
$\alpha_1 \in \{\emptyset, P, Q, R, F, O, J\}$: Type de machine.
$\emptyset$: Une seule machine.
P : Machine parallèle identique : la vitesse d'exécution est la même pour toutes les machines M_j et tous les tâches T_i.
Q : Machine parallèle uniforme : chaque machine M_j a une vitesse d'exécution propre et constante et la vitesse d'exécution est la même pour tous les tâches T_i d'une même machine M_j.
R : Machine parallèle générale : la vitesse d'exécution est différente pour chaque machines M_j et pour chaque tâche T_i.
F : Machine spécialisée (système flow shop) : chaque tâche doit être exécutée sur l'ensemble des machines disposées en série, selon un même routage.
O : Machine spécialisée (système open shop) : les tâches doivent être exécutées sur certaines machines sans restriction sur le routage des tâches.
J : Machine spécialisée (système job shop) : les tâches doivent être exécutées sur l'ensemble des machines disposées en série, mais peuvent avoir des routages différents.
$\alpha_2 \in \{\emptyset, m\}$: Nombre de machine.
$\alpha_2 = \emptyset$: Le nombre de machine est variable.
$\alpha_2 = m$: Le nombre de machine est constant et éga l à m.

1.5.2 Champ β : Contraintes

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

$\beta_1 \in \{\emptyset, pmtn\}$: Indique l'autorisation ou non de la préemption.

$\beta_1 = \emptyset$: La préemption n'est pas autorisée (lorsqu'une opération est commencée, elle doit être terminée avant de pouvoir exécuter une autre opération sur la même machine).

$\beta_1 = pmtn$: La préemption est autorisée (l'exécution d'une opération peut être interrompue puis reprise sur la même machine ou sur une autre).

$\beta_2 \in \{\emptyset, prec, tree, chain, etc...\}$: Décrit les contraintes de précédence.

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

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

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

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

$\beta_3 \in \{\emptyset, r_i\}$ Décrit les dates de disponibilité des différentes tâches.

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

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

$\beta_4 \in \{\emptyset, p_i = p, \underline{P} \leqslant p_i \leqslant \overline{P}\}$: Définit les particularités des durées de traitement.

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

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

$\beta_4 = \underline{P} \leqslant p_i \leqslant \overline{P}$: Les temps de traitement de chaque tâches est compris entre $\underline{P}$ et $\overline{P}$.

$\beta_5 \in \{\emptyset, d_i, \widetilde{d_i}\}$: Définit les dates échues et les dates limites.

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

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

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

β_6 $\{\emptyset, S_i\}$: Définit le temps de préparation.

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

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

1.5.3 Champ γ : Critère d'optimisation

Le troisième champ γ décrit le critère d'optimalité. On cherche à minimiser Y avec $Y \in \{C_{max}, \overline{C}, \overline{C_w},\ T_{max}, \overline{T}, \overline{T_w}, L_{max}, \overline{L}, \overline{L_w}, N_T\}$.

Exemple 1.1. — $1/r_i, p_i = 1/L_{max}$ *: Traiter un ensemble des tâches indépendantes et sans préemption ayant des temps de traitement tous identique et égaux à* 1 *et des dates de disponibilité et des dates échues arbitraire sur une seule machine. L'objectif est de minimiser la grande tardiveté.*

— $O3/pmtn, r_i/\overline{F}$ *: Traiter un ensemble des tâches indépendantes sur* 3 *machines spécialisées dans un système open shop. Les opérations sont préemptibles et caractérisées par des temps de traitement et des dates de disponibilité arbitraires. L'objectif est la minimisation du temps de flot moyen.*

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

Diagramme de Gantt est un outil graphique développé par "Henry Gantt" 1917 qui permet de représenter la planification de tâches nécessaires à la réalisation d'ordonnancement. Pour un atelier, celui-ci sera composé de plusieurs lignes horizontales, représentant chacune un équipement ou une machine. L'axe horizontal représente le temps. Les tâches seront représentées par des barres d'une longueur proportionnelle à leurs durées et seront positionnées sur la ligne correspondant à l'équipement sur lequel elles se dérouleront. On observera donc sur le diagramme l'occupation des machines, l'enchainement des tâches sur celles-ci ainsi que les dates de début et de fin de chaque tâche [10].

Exemple 1.2. *: On désire traiter* 4 *tâches sur une machine :*

T_i	T_1	T_2	T_3	T_4
p_i	3	2	1	2

TABLE 1.1 – Les données de l'exemple 1.2

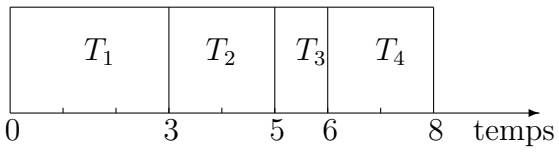


FIGURE 1.5 – Diagramme de Gantt d'un ordonnancement de l'exemple 1.2

1.7 Notions de complexité

1.7.1 Les principes fondamentaux de complexité

Il est divisé plusieurs sections, notamment :

1.7.1.1 Complexité d'un algorithme :

L'objectif de la notion de complexité est d'analyser les coûts de résolutions surtout en termes de temps de calcul.
Elle vise aussi à classer les problèmes d'ordonnancement à des problèmes faciles ou difficiles.[1]
En général, la complexité algorithmique se mesure par rapport à deux paramètres :
- Le temps alloué pour l'exécution de l'algorithme : il est relatif au nombre d'instructions à exécuter ainsi qu'à la taille des données manipulées.
- Espace mémoire requis : associé à la taille d'instance d'un problème donné.

1.7.1.2 Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire consiste à déterminer un meilleur élément parmi un nombre fini d'éléments mais souvent très élevé. Pour ce type de problème, il est nécessaire de mesurer l'efficacité de tout algorithme mis en ouvre. Ainsi, pour comparer la performance de plusieurs algorithmes résolvant un même problème, il existe une mesure basée sur leur temps d'exécution, c'est la complexité en temps.[2]

1.7.1.3 Problème de décision

Un problème de décision est un énoncé auquel la réponse peut être uniquement "oui" ou "non". Chaque problème d'optimisation possède un problème de décision correspondant.[4]
Pour chaque problème d'optimisation, on peut associer un problème de décision défini par la question suivante : "Existe-t-il une solution telle que son évaluation est majorée (resp. minorée) par une constante fixée ?". Ceci est pour un problème d'optimisation, dont la fonction objectif est la minimisation (resp. maximisation).
Alors si la réponse à la question existe et est obtenue grâce à un algorithme polynomial alors il existera un algorithme polynomial résolvant le problème d'optimisation associé. Cette propriété nous permet de classer les problèmes d'optimisation grâce à leurs pro-

blèmes de décision. Ces problèmes peuvent être classés en deux classes principales : la classe P et la classe NP-complet [15].

1.7.2 Les classes de complexité

[6]

Classe NP

La classe NP (Non déterministe Polynomial) est celle des problèmes d'existence dont une proposition de solution est Oui et qui est vérifiable polynomialement. Parmi les problèmes décidables, les plus simples à résoudre sont regroupés dans la classe NP.

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 l'ensemble de tous les problèmes de reconnaissance polynomiaux. Pour le reste de la classe NP, on n'est pas sûr qu'il n'existe pas un algorithme polynomial pour résoudre chacun de ses problèmes. Ainsi, on sait que P est inclue dans NP mais on n'a pas pu prouver que P n'est pas NP.

Classe NP-Complet

La classe NP-Complet regroupe les problèmes les plus difficiles de la classe NP. Elle contient les problèmes de la classe NP tels que n'importe quel problème de la classe NP leur est polynomiale-ment réductible. Entre eux, les problèmes de la classe NP-Complet sont aussi difficiles.

Un problème de décision est dit NP-complet s'il appartient à la classe NP et il résolu, au mieux, en un temps exponentiel.

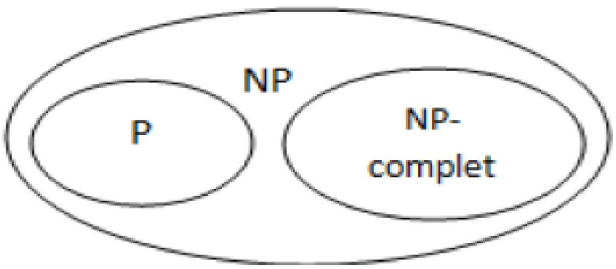


FIGURE 1.6 – Schéma représentatife décision de classes de complexité

Définition : Un problème d'optimisation est dit NP-Difficile, si le problème de décision associé est NP-complet[4].

1.7.3 Réduction entre problèmes d'ordonnancement

Nous donnons ci-dessous quelques réductions (transformations polynomiales) entre problème d'ordonnancement. Une tâche de X à Y signifié que la résolution d'un modèle avec le paramètre Y soit plus difficile que celle du même modèle avec le paramètre X, ces relations nous aide à déterminer la complexité des problèmes [1].

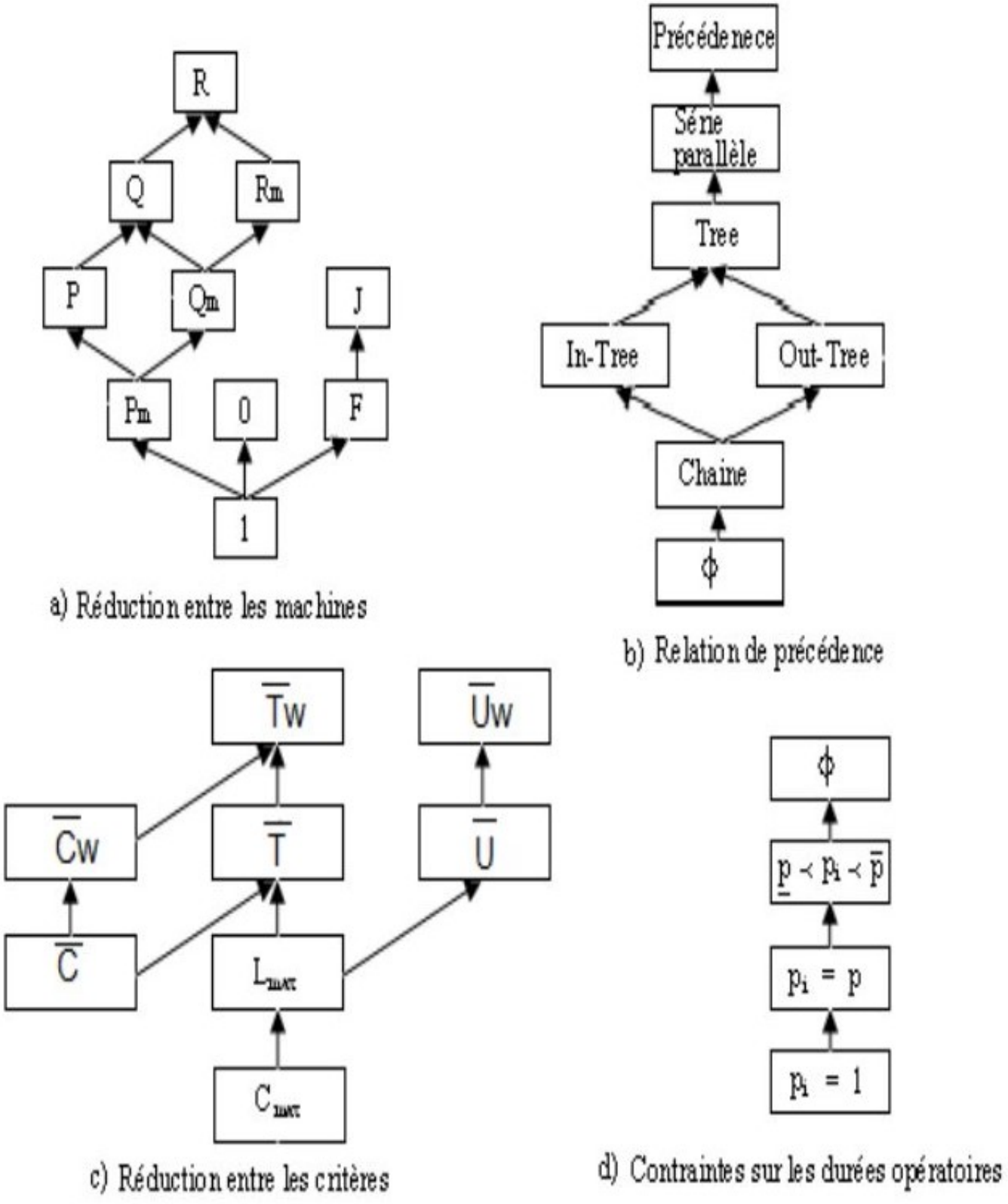


FIGURE 1.7 – Hiérarchie de complexité entre différents problèmes

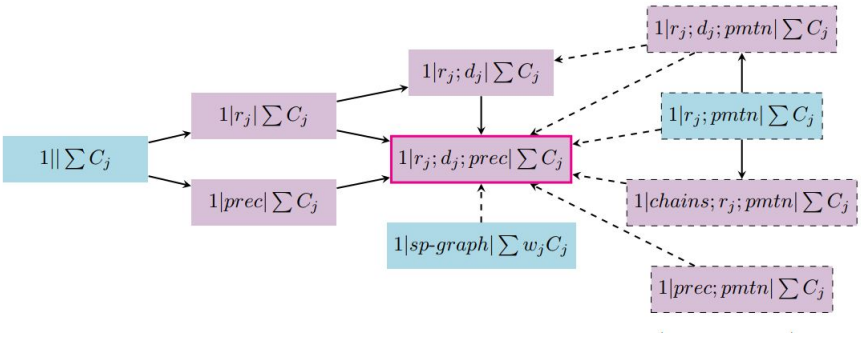


FIGURE 1.8 – Hiérarchie de complexité des relaxations de problèmes

1.8 Conclusion

Dans ce chapitre, nous avons définit un problème d'ordonnancement, nous avons présenté les éléments nécessaires pour formuler un problème d'ordonnancement à savoir les tâches, les ressources, les contraintes et les objectifs, ainsi que la classification $\alpha/\beta/\gamma$, et la notion de complexité des différents problèmes d'ordonnancement.

Chapitre 2

Quelques problèmes avec temps de fin de traitement moyen

2.1 Introduction

Nous concentrons dans ce chapitre sur quelques problèmes d'ordonnancement sur une seule machine pour minimiser le temps de fin de traitement moyen à savoir deux problèmes faciles et un problème difficile.

2.2 Problème $1//\overline{C}$

2.2.1 Description du problème

Traiter un ensemble de tâches indépendantes et sans préemption, ayant des temps de traitement arbitraires sur une seule machine.
L'objectif est la minimisation du temps de fin de traitement moyen.

2.2.2 Complexité du problème

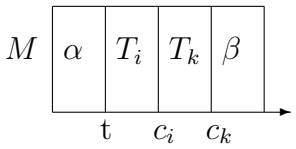
Théorème 2.1. *Le problème* $1//\overline{C}$ *est résolu en O(n log n)[12].*

2.2.3 Résolution de problème

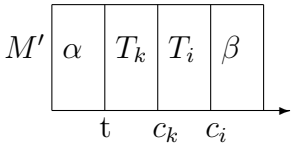
La règle SPT : Ranger les tâches suivant l'ordre croissant de leurs dates de traitements.

Théorème 2.2. *La règle SPT résout le problème* $1//\overline{C}$*.[12]*

Preuve : Supposons qu'il existe un solution $\alpha T_i T_k \beta$ de tâches avec $p_i > p_k$ c'est à dire : l'ensemble des tâches n'est pas ordonné suivant la règle SPT notons y le coût de cet ordonnancement.



Formons, à partir de cette solution une nouvelle solution $\alpha T_k T_i \beta$ et notons y' le coût de cette ordonnancement.



Nous avons :

$$y = \sum_{i=1}^{n} C_i = (t + p_i) + (t + p_i + p_k) + c$$

$$y' = \sum_{i=1}^{n} C_i = (t + p_k) + (t + p_k + p_i) + c$$

Où :

t : est la date de fin de traitement des tâches de la sous séquence α.

c : le coût des tâches des sous séquence α et β.

Alors : $y' - y = p_k - p_i < 0$ donc $y' < y$.

D'où, il existe une solution optimale où l'ensemble des tâches est ordonné suivant la règle SPT.

Exemple 2.1. *Étant donné le tableau suivant :*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7
p_i	1	2	5	7	4	2	1

TABLE 2.1 – Les temps de traitement de l'exemple 2.1

On applique la règle SPT on obtient la séquence optimale suivante : $T_1T_7T_2T_6T_5T_3T_4$ *qui est représentée par la figure 2.1 avec* $\overline{C} = 41.14$.

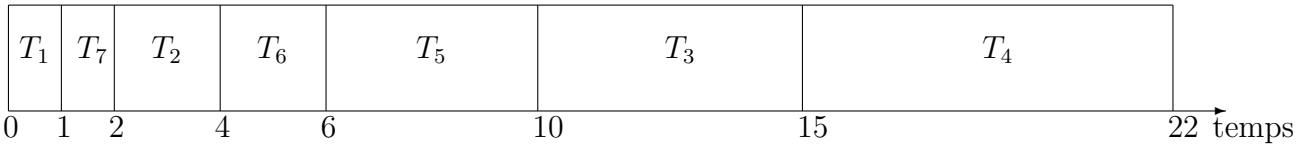


FIGURE 2.1 – Solution optimale pour l'exemple 2.1

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

2.3.1 Description du problème

Traiter un ensemble de tâches indépendantes et avec préemption, ayant des dates de disponibilité et des temps de traitement arbitraires sur une seule machine.
L'objectif est la minimisation du temps de fin de traitement moyen.

2.3.2 Complexité du problème

Théorème 2.3. *Le problème* $1/r_i, pmtn/\overline{C}$ *est résolu en O(n log n)[7].*

2.3.3 Résolution de problème

La règle (SRPT) : À chaque date de disponibilité ou date de fin de traitement, ordonnancer parmi les tâches restantes, la tâche disponible ayant le plus petit temps de traitement restant.

Théorème 2.4. *La règle SRPT résout le problème* $1/r_i, pmtn/\overline{C}$*.[7]*

Preuve :[7] La preuve suit également par un argument d'échange, mais dans ce cas nous ne faisons pas échanger deux emplois, mais éventuellement des parties des deux emplois. Considérons un horaire optimal S dans lequel un travail T_k est en cours de traitement à l'instant t alors qu'il existe un travail inachevé l tel que $r_l \leq t$, c'est-à-dire que le travail T_l est disponible à l'instant t, et $x_k > x_l$, où x_k et x_l sont les montants restants des deux emplois.
Notez que la machine traite les travaux T_k et T_l pour $x_k + x_l$ unités après le temps t. Construire un nouveau horaire S' en traitant uniquement le travail T_l sur les x_l premières

unités de ces temps, et en traitant le travail T_k sur les unités x_k restantes. Puisque $x_l < x_k$, dans le nouveau travail planifié T_l se termine strictement avant l'heure à laquelle le travail T_k ou T_l s'est terminé dans l'ancien programme. C'est $C_l(S') < min\{C_k(S), C_l(S)\}$.
A noter également que $C_k(S) = max\{C_k(S), C_l(S)\}$ et $C_i(S') = C_i(S)$ pour tous les autres $T_i \neq T_k, T_l$. Ainsi, $\overline{C}$ décroît strictement ce qui est une contradiction.

Exemple 2.2. *Étant donné le tableau suivant :*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7
r_i	3	8	0	10	3	4	13
p_i	8	6	3	7	6	1	2

TABLE 2.2 – Les données de l'exemple 2.2

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
$t = 0$	8	6	$\underline{3}$	7	6	1	2
$t = 3$	$\underline{8}$	6	0	7	$\underline{6}$	1	2
$t = 4$	$\underline{8}$	6	0	7	$\underline{5}$	$\underline{1}$	2
$t = 5$	$\underline{8}$	6	0	7	$\underline{5}$	0	2
$t = 8$	$\underline{8}$	$\underline{6}$	0	7	$\underline{2}$	0	2
$t = 10$	$\underline{8}$	$\underline{6}$	0	$\underline{7}$	0	0	2
$t = 13$	$\underline{8}$	$\underline{3}$	0	$\underline{7}$	0	0	$\underline{2}$
$t = 15$	$\underline{8}$	$\underline{3}$	0	$\underline{7}$	0	0	0
$t = 18$	$\underline{8}$	0	0	$\underline{7}$	0	0	0
$t = 25$	$\underline{8}$	0	0	0	0	0	0
$t = 33$	0	0	0	0	0	0	0

TABLE 2.3 – Résolution de l'exemple 2.2

On applique la règle SRPT on obtient la solution optimale suivante : avec $\overline{C} = 15.57$.

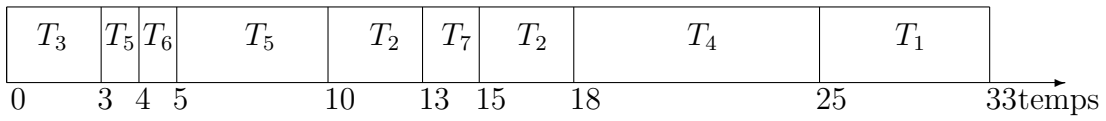


FIGURE 2.2 – Solution optimale de l'exemple 2.2

2.4 Problème $1/r_i/\overline{C}$

2.4.1 Description du problème

Traiter un ensemble de tâches indépendantes et sans préemption, ayant des dates de disponibilité et des temps de traitement arbitraires sur une seule machine.
L'objectif est la minimisation du temps de fin de traitement moyen.

2.4.2 Complexité du problème

Théorème 2.5. *Le problème $1/r_i/\overline{C}$ est NP-difficile [14].*

Définition [14, 17]

Le problème de 3-partition est un problème NP-complet. Le problème est de décider si un multi-ensemble donné d'entiers peut être partitionné en triplets qui ont tous la même somme. Plus précisément :
- L'entrée du problème est un multi-ensemble S de $n = 3m'$ entiers positifs. La somme de tous les entiers est $m'T$.
- Le problème indique s'il existe ou non une partition de S en m' triplets $S_1, S_2, S_{m'}$ de telle sorte que la somme des nombres dans chacun soit égale à T. Le $S_1, S_2, , S_{m'}$ doit former une partition de S dans le sens où ils sont disjoints et qu'ils couvrent S.

Le problème à 3-partition reste NP-complet sous la restriction que chaque entier de S est strictement compris entre $T/4$ et $T/2$.

Preuve :[14] Nous allons montrer que le problème de la 3-partition se réduit au problème de décision de $1/r_i/\overline{C}$. Considérons une instance de 3-partition, composée des entiers positifs $T_1, ..., T_{3m'}, b$, avec $b/4 < T_i < b/2$ pour tout i et $\sum_i T_i = m'b$. Rappel qu'il s'agit d'une instance oui si et seulement si l'ensemble d'index $T = \{1, ..., 3t\}$ peut être partitionné en t sous-ensembles de 3 éléments mutuellement disjoints $S_1, ..., S_t$ avec $\sum_{i \in S_h} T_i = b$ pour $h = 1, ..., t$. Nous allons définir une instance de $1/r_i/\overline{C}$ et un entier Z tel qu'il existe un ordonnancement de valeur $\overline{C} \leq Z$ si et seulement si l'instance de 3-partition est un oui-instance.
L'instance de planification a trois types de tâches. Premièrement, pour chaque $T_i \in T$, il existe un travail T_i avec date de lancement 0 et temps de traitement p_i. Deuxièmement,

pour chaque $T_h \in \{1, ..., t1\}$, il y a v emplois $K_k^{(h)}$ avec la date de sortie hb et le temps de traitement 0 ($k = 1, ..., v$). Enfin, il y a w jobs L_l avec une date de sortie tb et un temps de traitement 1 ($l = 1, ..., w$). Les valeurs de Z, v et w seront définies ultérieurement.

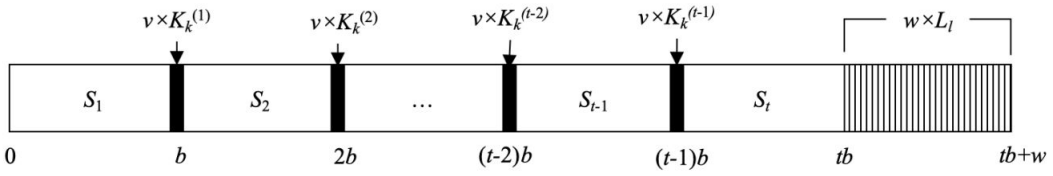


FIGURE 2.3 – Distribution de tâches dans un ordonnancement basé le problème de 3-partition

Supposons que nous ayons une instance oui de 3-partition et considérons ce qui suit calendrier (voir Figure 2.3). Les trois jobs T_i avec $T_i \in S_h$ sont traités dans l'intervalle $[(h1)b, hb]$, pour $i = 1, ..., t$; la somme de leurs temps d'exécution est bornée par :

$$Z_I = \sum_{h=1}^{t} 3hb = \frac{3}{2}t(t+1)b$$

Les emplois $K_k^{(h)}$ devraient commencer à leurs dates de sortis ; leur temps de fin de traitement moyen est égal :

$$Z_K = \sum_{h=1}^{t-1} vhb = \frac{1}{2}vt(t-1)b$$

Les emplois L_l sont traités consécutivement dans l'intervalle $[tb, tb+w]$; leur temps de fin de traitement moyen est donné par :

$$Z_L = \sum_{l=1}^{w}(tb+l) = wtb + \frac{1}{2}w(w+1)$$

Nous définissons maintenant $v = Z_J$,$w = Z_J + Z_K$ et $Z = Z_J + Z_K + Z_L$. Le calendrier correspondant à l'instance oui de 3-partition a une valeur $\overline{C} \leq Z$.
Inversement, considérons un ordonnancement satisfaisant $\overline{C} \leq Z$. Nous affirmons que, dans un tel horaire, tous les travaux T_i sont terminés à l'heure tb. Si ce n'est pas le cas, alors au moins un travail T_i ainsi que tous les travaux T_l se terminent après tb, de sorte

que :

$$\overline{C} > \sum_{l=1}^{w+1}(tb+l) = ZL + tb + w + 1 > Z$$

Nous pouvons supposer que, pour tout h, tous les emplois $K_k^{(h)}(k = 1, ..., v)$ sont traités en conséquence activement, cela découle d'un argument d'échange direct. Nous revendiquons désormais également que, si $\overline{C} \leq Z$, alors tous les travaux $K_k^{(h)}$ commencer à leurs dates de sortie. Sinon, au moins v de ces travaux sont retardés d'une unité de temps, de sorte que :

$$\overline{C} > Z_K + v + Z_L = Z$$

Nous concluons que, dans tout ordonnancement de valeur $\overline{C} \leq Z$, tous les travaux T_i sont traités dans l'intervalle $[0, tb]$, interrompu par des tâches de temps zéro à chaque instant hb $(h = 1, ..., t-1)$.
Ainsi, dans chaque intervalle $[(h-1)b, hb]$, trois jobs T_i sont traités pour une durée totale de b unités de temps. Cela implique que nous avons une instance oui de 3-partition.
Notez que le nombre d'emplois est proportionnel à b, de sorte que l'exactitude de la réduction dépend essentiellement de la forte NP-complétude de 3-partition.
Enfin, mentionnons que l'on peut modifier la réduction de manière à ce que tous les emplois aient une longueur positive.

2.5 Conclusion

Dans ce chapitre nous avons présenté deux problèmes faciles d'ordonnancement avec temps de fin de traitement moyen comme les problèmes faciles, nous avons présenté des algorithmes pour résoudre ces problèmes ainsi qu'un exemple d'application, nous avons présenté aussi un problème difficile et la preuve de sa complexité.

Chapitre 3

Implémentation et heuristiques proposées pour résoudre le problème $1/r_i/\overline{C}$

3.1 Introduction

Dans ce chapitre nous présentons la formulation mathématique du problème $1/r_i/\overline{C}$, nous proposons trois heuristiques pour résoudre ce problème qui est NP-difficile, ainsi que nous implémentons ces heuristiques sur Matlab et nous les comparons utilisant des exemples différents.

3.2 Modèle mathématique du problème $1/r_i/\overline{C}$

Les variables :

$$x_{ik} = \begin{cases} 1 & si\ la\ tâche\ T_i\ est\ en\ position\ k\ sur\ la\ machine \\ 0 & sinon \end{cases}$$

Avec : $i = \overline{1,n}$ et $k = \overline{1,n}$

t_k : La date de début de traitement de la tâche en position k sur la machine ($k = \overline{1,n}$).

Les contraintes :

1) $\sum_{k=1}^{n} x_{ik} = 1 \qquad i = \overline{1,n}$

2) $\sum_{i=1}^{n} x_{ik} = 1 \qquad k = \overline{1,n}$

3) $t_k - \sum_{i=1}^{n} x_{ik} r_i \geqslant 0 \qquad k = \overline{1,n}$

4) $t_k + \sum_{i=1}^{n} p_i x_{ik} \leqslant t_{k+1} \qquad k = \overline{1,n-1}$

5) $\frac{1}{n} \sum_{k=1}^{n} (t_k + \sum_{i=1}^{n} p_i x_{ik}) \leqslant y \qquad y \in R$

La 1ère contrainte impose le traitement de toutes les tâches.

La 2ème contrainte exige le traitement d'une seule tâche par position.

La 3ème contrainte empêche le démarrage d'une tâche avant sa date de disponibilité.

La 4ème contrainte impose â la tâche en position $k+1$ de commencer son exécution après la fin d'exécution de la tâche en position k.

La 5ème contrainte indique que la date de fin de traitement moyen de chaque tâche doit être $\leqslant y$.

La formulation mathématique du problème est :

$$\begin{cases} \min y \\ \sum_{k=1}^{n} x_{ik} = 1, i = \overline{1,n} \\ \sum_{i=1}^{n} x_{ik} = 1, k = \overline{1,n} \\ t_k - \sum_{i=1}^{n} x_{ik} r_i \geqslant 0, k = \overline{1,n} \\ t_k + \sum_{i=1}^{n} p_i x_{ik} \leqslant t_{(k+1)}, k = \overline{1,n-1} \\ \frac{1}{n} \sum_{k=1}^{n} (t_k + \sum_{i=1}^{n} p_i x_{ik}) \leqslant y \\ x_{ik} \in \{0,1\}, i = \overline{1,n}, k = \overline{1,n} \\ t_k \geqslant 0, k = \overline{1,n} \\ y \in R \end{cases}$$

3.3 Heuristiques pour résoudre le problème $1/r_i/\overline{C}$

3.3.1 Définition

Les heuristiques sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables. Elles se basent sur des règles simplifiés 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.[11]

3.3.2 Heuristiques proposés

Dans cette session nous proposons trois heuristiques gui fournissent rapidement une solution réalisable proche de l'optimum pour le problème d'ordonnancement $1/r_i/\overline{C}$ gui est NP-difficile, où notre but est de trouver l'heuristique de bonne qualité.

3.3.2.1 Principe de la $1^{\text{ère}}$ heuristique

L'heuristique est basée sur l'idée de ranger les tâches par ordre croissant des dates de disponibilité.

Algorithm 1 Algorithme de H_1

Début
Ranger les tâches par ordre croissant de leurs dates de disponibilité ;
$t_1 = r_1$;
$C_1 = t_1 + p_1$;
pour $i = 2$ à n
faire
$t_i = max\{C_{i-1}, r_i\}$;
$C_i = t_i + p_i$;
fait
$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$;
Fin

3.3.2.2 Principe de la $2^{\text{ème}}$ heuristique

L'heuristique est basée sur l'idée de ranger les tâches par ordre croissant des temps de traitement.

Algorithm 2 Algorithme de H_2

Début
Ranger les tâches par ordre croissant de leurs temps de traitement ;
$C_0 = 0$;
pour $i = 1$ à n
faire
$t_i = max\{C_{i-1}, r_i\}$;
$C_i = t_i + p_i$;
fait
$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$;
Fin

3.3.2.3 Principe de la 3ème heuristique

Cette heuristique est basée sur l'idée qu'une tâche de plus petit temps de traitement est choisie parmi les tâches de plus petite date de disponibilité.

Algorithm 3 Algorithme de H_3

Début
$t = min\{r_i\}$;
tant que $T \neq \phi$
faire
$T' = \{T_i \in T / r_i \leq t\}$
Choisir $T_i \in T'$ tel que
$p_i = min\{p_b / T_b \in T'\}$;
Ordonnancer T_i à l'instant t ;
$C_i = t + p_i$;
$T = T \setminus \{T_i\}$;
$t = max\{C_i, min\{r_{i+1}\}\}$;
fait
$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$;
Fin

3.3.3 Exemple d'application

Dans cette cession nous tritons traiter 7 tâches indépendantes sur une seule machine, dont les temps de traitement, les dates de disponibilité sont donnés dans le tableau suivant :

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7
r_i	8	3	0	10	3	4	13
p_i	8	6	3	7	6	1	2

TABLE 3.1 – Les données de l'exemple d'application

Nous appliquons cet exemple sur les trois heuristiques proposées précédemment.

3.3.3.1 Application de la 1ère heuristique

Nous appliquons notre heuristique H_1 : Ranger les tâches par ordre croissant de leurs dates de disponibilité. Nous trouvons la liste L suivante :

L=$\{T_3, T_2, T_5, T_6, T_1, T_4, T_7\}$

$t_1 = r_1 = 0$

$C_1 = t_1 + p_1 = 0 + 3 = 3$

Itération 1 : $t_2 = max\{C_1, r_2\} = max\{3, 3\} = 3$

$C_2 = t_2 + p_2 = 3 + 6 = 9$

Itération 2 : $t_3 = max\{C_2, r_3\} = max\{9, 3\} = 9$

$C_3 = t_3 + p_3 = 9 + 6 = 15$

Itération 3 : $t_4 = max\{C_3, r_4\} = max\{15, 4\} = 15$

$C_4 = t_4 + p_4 = 15 + 1 = 16$

Itération 4 : $t_5 = max\{C_4, r_5\} = max\{16, 8\} = 16$

$C_5 = t_5 + p_5 = 16 + 8 = 24$

Itération 5 : $t_6 = max\{C_5, r_6\} = max\{24, 10\} = 24$

$C_6 = t_6 + p_6 = 24 + 7 = 31$

Itération 6 : $t_7 = max\{C_6, r_7\} = max\{31, 13\} = 31$

$C_7 = t_7 + p_7 = 31 + 2 = 33$

Donc : $\overline{C} = 18.71$

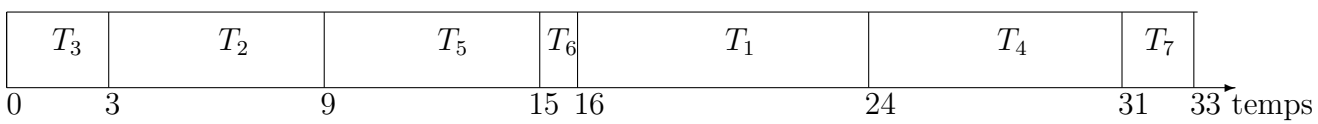


FIGURE 3.1 – Solution de l'exemple par l'heuristique H_1

3.3.3.2 Application de la 2ème heuristique

Nous appliquons notre heuristique H_2 : Ranger les tâches par ordre croissant de leurs temps de traitement. Nous trouvons la liste L suivante :

L=$\{T_6, T_7, T_3, T_2, T_5, T_4, T_1\}$

$C_0 = 0$

Itération 1 : $t_1 = max\{C_0, r_1\} = max\{0, 4\} = 4$

$C_1 = t_1 + p_1 = 4 + 1 = 5$

Itération 2 : $t_2 = max\{C_1, r_2\} = max\{5, 13\} = 13$

$C_2 = t_2 + p_2 = 13 + 2 = 15$

Itération 3 : $t_3 = max\{C_2, r_3\} = max\{15, 0\} = 15$

$C_3 = t_3 + p_3 = 15 + 3 = 18$

Itération 4 : $t_4 = max\{C_3, r_4\} = max\{18, 3\} = 18$

$C_4 = t_4 + p_4 = 18 + 6 = 24$

Itération 5 : $t_5 = max\{C_4, r_5\} = max\{24, 3\} = 24$

$C_5 = t_5 + p_5 = 24 + 6 = 30$

Itération 6 : $t_6 = max\{C_5, r_6\} = max\{30, 10\} = 30$

$C_6 = t_6 + p_6 = 30 + 7 = 37$

Itération 7 : $t_7 = max\{C_6, r_7\} = max\{37, 8\} = 37$

$C_7 = t_7 + p_7 = 37 + 8 = 45$

Donc : $\overline{C} = 24,85$

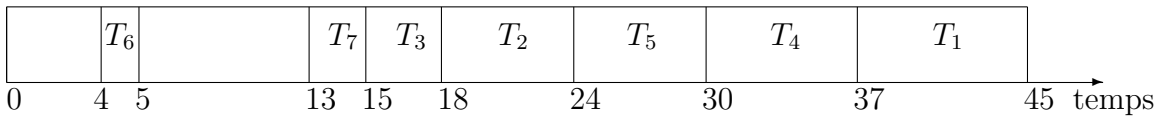


FIGURE 3.2 – Solution de l'exemple par l'heuristique H_2

3.3.3.3 Application de la 3ème heuristique

Nous appliquons notre heuristique H_3 : Nous mettons toutes les tâches dans la liste T suivante :

$T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$

t=min$\{r_i\} = 0$

Itération 1 : $T' = \{T_3\}$

$p_i = min\{p_3\} = p_3 = 3$

$C_3 = t + p_3 = 0 + 3 = 3$

$T = \{T_1, T_2, T_4, T_5, T_6, T_7\}$

$t = max\{C_3, min\{r_i\}\} = max\{3, 3\} = 3$

Itération 2 : $T' = \{T_2, T_5\}$

$p_i = min\{p_2, p_5\} = min\{6, 6\} = 6$

$C_2 = t + p_2 = 6 + 3 = 9$

$T = \{T_1, T_4, T_5, T_6, T_7\}$

$t = max\{C_2, min\{r_i\}\} = max\{9, 4\} = 9$

Itération 3 : $T' = \{T_1, T_5, T_6\}$

$p_i = min\{p_1, p_5, p_6\} = min\{8, 6, 1\} = 1$

$C_6 = t + p_6 = 9 + 1 = 10$

$T = \{T_1, T_4, T_5, T_7\}$

$t = max\{C_6, min\{r_i\}\} = max\{10, 8\} = 10$

Itération 4 : $T' = \{T_1, T_4, T_5\}$

$p_i = min\{p_1, p_4, p_5\} = min\{8, 7, 6\} = 6$

$C_5 = t + p_5 = 10 + 6 = 16$

$T = \{T_1, T_4, T_7\}$

$t = max\{C_5, min\{r_i\}\} = max\{16, 10\} = 16$

Itération 5 : $T' = \{T_1, T_4, T_7\}$

$p_i = min\{p_1, p_4, p_7\} = min\{8, 7, 2\} = 2$

$C_7 = t + p_7 = 16 + 2 = 18$

$T = \{T_1, T_4\}$

$t = max\{C_7, min\{r_i\}\} = max\{18, 13\} = 18$

Itération 6 : $T' = \{T_1, T_4\}$

$p_i = min\{p_1, p_4\} = min\{8, 7\} = 7$

$C_4 = t + p_4 = 18 + 7 = 25$

$T = \{T_1\}$

$t = max\{C_4\} = 25$

Itération 7 : $T' = \{T_1\}$

$p_i = min\{p_1\} = p_1 = 8$

$C_1 = t + p_1 = 25 + 8 = 33$

$T = \phi$

$t = max\{C_1\} = 33$

Donc : $\overline{C} = 16.28$

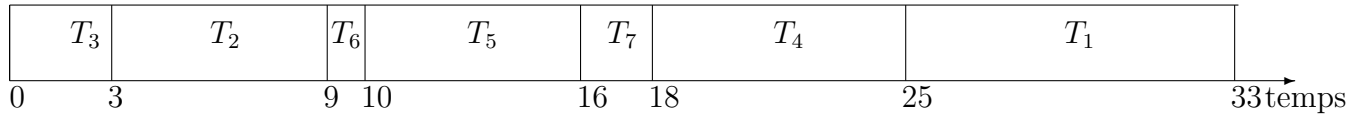


FIGURE 3.3 – Solution de l'exemple par l'heuristique H_3

3.4 Résolution du problème et implémentation

Nous arrivons maintenant à la dernière étape, qui est le développement du langage doté d'une interface claire et accessible, facile à utiliser.

3.4.1 Choix du langage

Le programme Matlab a été choisi car son utilisation répond aux critères suivants :
La maniabilité du langage : constitué d'un ensemble de possibilités faisant en sorte que le programmeur travaille avec aisance, assuré d'une part par la syntaxe du langage et d'autre part par un aspect visuel clair représentatif à la fois du détail et du global.
Le bagage du langage : il contient une interface graphique puissante ainsi qu'une grande variété de méthodes scientifiques implémentées (prédéfinies).

3.4.2 Généralités sur le langage Matlab

Matlab est un logiciel de calcul numérique matriciel à syntaxe simple. C'est devenu un logiciel de programmation largement utilisé en finance et en ingénierie financière. Avec ses nombreuses fonctions spécialisées pré-codées et ses différentes "tool box" disponibles, Matlab permet une prise en main rapide et efficace, moyennant une certaine rigueur.
Matlab est dit "interpréteur" : les instructions sont lues et exécutées ligne par ligne. Il existe deux modes de fonctionnement :
Mode interactif : le logiciel exécute les instructions au fur et à mesure qu'elles sont entrées par l'utilisateur dans la fenêtre de commande.
Mode exécutif : le logiciel exécute un "M-file", qui lui-même peut appeler d'autres M-file, que l'on nomme alors "fonction".

3.4.3 Exemple illustratif

Pour illustrer notre heuristiques nous proposons des exemples avec les données suivants :

- Le nombre de tâches $n = 4, 5, 6, 7, 10, 15$ et 19.

- La date de disponibilité des tâches r_i.

- Le temps de traitement des tâches p_i.

Exemple 1 :

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}	T_{13}	T_{14}	T_{15}
r_i	0	3	2	2	5	7	3	5	4	8	9	11	15	12	4
p_i	4	3	3	4	2	2	2	1	1	1	3	2	5	6	4

TABLE 3.2 – Les données de l'exemple 1

Exemple 2 :

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}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}
r_i	5	3	8	4	7	11	6	1	0	10	7	12	16	0	5	20	8	25	13
p_i	7	8	6	2	9	2	10	1	1	3	9	4	5	10	2	3	6	8	10

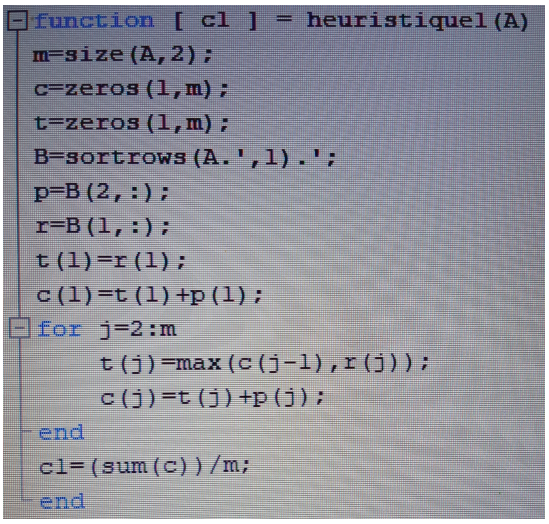
TABLE 3.3 – Les données de l'exemple 2

3.4.4 Résultats numériques

Dans cette partie, nous avons utilisé nos programmes MATLAB sur plusieurs exemples avec nombre de tâches différents.

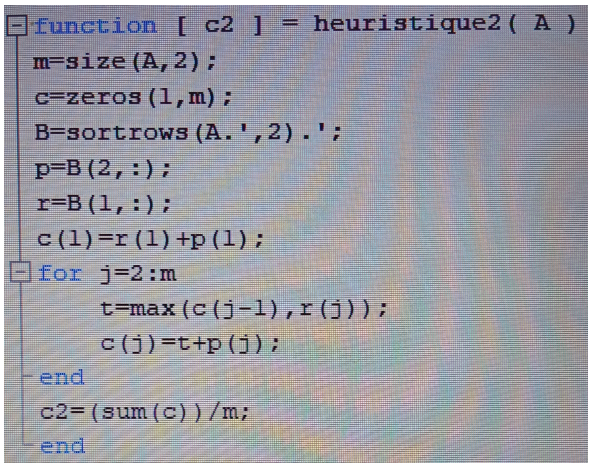
Pour chaque exemple les dates de disponibilité r_i et les temps de traitement p_i sont aléatoires, nous calculons la valeur de la fonction objective de chaque exemple pour les trois heuristiques.

Nous présentons les algorithmes des trois heuristiques que nous avons programmés dans MATLAB :



```
function [ c1 ] = heuristique1(A)
m=size(A,2);
c=zeros(1,m);
t=zeros(1,m);
B=sortrows(A.',1).';
p=B(2,:);
r=B(1,:);
t(1)=r(1);
c(1)=t(1)+p(1);
for j=2:m
    t(j)=max(c(j-1),r(j));
    c(j)=t(j)+p(j);
end
c1=(sum(c))/m;
end
```

FIGURE 3.4 – Programme Matlab de l'algorithme H1



```
function [ c2 ] = heuristique2( A )
m=size(A,2);
c=zeros(1,m);
B=sortrows(A.',2).';
p=B(2,:);
r=B(1,:);
c(1)=r(1)+p(1);
for j=2:m
    t=max(c(j-1),r(j));
    c(j)=t+p(j);
end
c2=(sum(c))/m;
end
```

FIGURE 3.5 – Programme Matlab de l'algorithme H2

```
function [ C3] = heuristique3( A )
m=size(A,2);
c =[ ];
r = A(1,:);p = A(2,:);
t = min(r);
for k=1:m
    [~, n]=size(A);
    r = A(1,:);p=A(2,:);
    min_p=[ ];
    for i=1:n
        if r(i)<=t
            min_p=[min_p p(i)];
        end
    end
    min_p;
    c(k)=min(min_p)+t;
    c=[c];
    for j=1:n
    if p(j)==min(min_p) && r(j)<=t
        A(:,j)=[ ];
        break
    end
    end
    A;
    t=max(c(k),min(r));
end
C3=(sum(c))/m;
end
```

FIGURE 3.6 – Programme Matlab de l'algorithme H3

Pour confirmer les résultats obtenus du programme, nous donnons deux exemples :

1/ Exemple 6, $n = 6$

```
Command Window
>> A=A6,c1=heuristique1(A),c2=heuristique2(A), c3=heuristique3(A)

A =

     2     5     0     3     0     2
     4     3     2     4     5     1


c1 =

   11.1667


c2 =

   10.8333


c3 =

    9.1667

fx >>
```

FIGURE 3.7 – Résultats de l'exemple 6 pour $n = 6$

2/ Exemple 2, $n = 15$

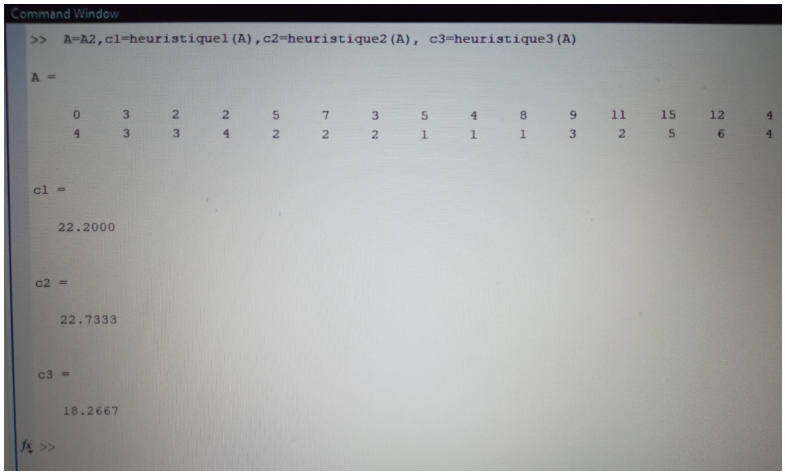


FIGURE 3.8 – Résultats de l'exemple 2 pour $n = 15$

Exemple	1	2	3	4	5	6	7	8	9	10
*Heristique*1	54.2632	22.2	13.6	11.9	16.1429	11.1667	12	10	12.8	4.25
*Heristique*2	46	22.7333	12.1	11.2	18.8571	10.8333	13.8333	8.6	15	6.5
*Heristique*3	42	18.2667	11.1	9.5	13	9.1667	9.667	8.2	12.4	4.25

TABLE 3.4 – Les résultats des trois heuristiques

3.4.5 Comparaison entre les heuristiques proposées

Pour comparer les heuristique proposées, nous avons calculer le ration d'approximation R avec :

$$R = \frac{\overline{C}(H)}{B}$$

_ $\overline{C}(H)$: Temps de fin de traitement moyen de toutes les tâches obtenue par l'heuristique proposée H.

_ B : Est la borne inférieure du problème $1/r_i/\overline{C}$ avec :

$$B = t + \frac{\sum_{i=1}^{n}(n - i + 1)p_i}{n}$$

Où $t = min\{r_i\}$ et n est le nombre des tâches.

Lorsque R est proche de 1 alors l'approximation est de bonne qualité.

Dans ce tableau, nous calculons la borne inférieure et le ration d'approximation pour chaque exemple pour les trois heuristiques, puis les comparons :

Ex	1	2	3	4	5	6	7	8	9	10
B	38.5263	16.8667	11.1	9.4	12	8.8333	8.8333	7.6	11	4
R_1	1.5327	1.3162	1.2252	1.2659	1.3452	1.7514	1.2641	1.3584	1.3157	1.1636
R_2	1.3030	1.3478	1.0900	1.1914	1.7514	1.2264	1.5660	1.1315	1.3636	1.625
R_3	1.1926	1.0830	1	1.0106	1.0833	1.0377	1.0943	1.0789	1.1272	1.0625

TABLE 3.5 – Comparaison entre les trois heuristiques

A partir des résultats obtenues, nous remarquons que l'heuristique $H3$ donne la meilleur valeur de la fonction objectif car la valeur de $\overline{C}$ obtenue est proche à la valeur de la borne B, donc le ratio d'approximation R est proche de 1, par contre les heuristiques $H1$ et $H2$ donnent des valeurs plus grand de $\overline{C}$.
Alors pour ces 10 exemples, l'heuristique $H3$ est plus performant des heuristiques $H1$ et $H2$.

Remarque 3.1. *Nous avons traité les trois heuristiques sur* 10 *exemples, nous avons remarqué que l'heuristique* $H3$ *donne de meilleur solution que les heuristiques* $H1$ *et* $H2$ *pour ces* 10 *exemples.*

3.5 Conclusion

Dans ce chapitre nous avons traité le problème d'ordonnancement $1/r_i/\overline{C}$. *Nous avons commencé par le modèle mathématique, nous avons proposé trois heuristiques et nous avons utilisé le programme Matlab pour implémenter ces heuristiques.*

Conclusion Générale

Les problèmes d'ordonnancement avec temps de fin de traitement moyen sur une seule machine est l'un des problèmes d'ordonnancement intensivement étudiés. Dans ce travail, nous avons traité quelques problèmes, sur une seule machine à savoir la complexité de chaque problème. Ainsi que nous avons proposées trois heuristiques pour le problème $1/r_i/\overline{C}$ *qui est NP-difficile pour parvenir à une solution acceptable.*

La première partie de ce mémoire est une présentation générale des problèmes d'ordonnancement, leurs notions fondamentales et les éléments nécessaires pour caractériser un problème d'ordonnancement sont aussi évoqués : les tâches, les ressources, les contraintes et les objectifs, le deuxième point nous avons abordé la classification, le diagramme de Gantt comme un outil de représentation de ces problèmes.

Nous nous somme intéressé dans la deuxième partie à étudier quelques problèmes d'ordonnancement sur une seule machine qui peuvent être classés comme des problèmes faciles et difficiles, nous avons commencé par deux problèmes faciles, pour chaque problème nous avons donné un méthode pour son résolution, ainsi nous avons également utilisé le problème de 3-partition pour prouver la difficulté du problème $1/r_i/\overline{C}$.

Dans la troisième partie, nous avons proposé trois heuristiques pour le problème $1/r_i/\overline{C}$ *qui est NP-difficile, tout d'abord nous avons représenté ce problème par son modèle mathématique et nous avons donné sa définition. La fin de cette partie est consacré a l'implémentation sur Matlab de ces trois heuristiques qui sont proposés pour trouver une solution satisfaite.*

Enfin, notre perspective est d'implémenter ces trois heuristiques sur d'autres programmes comme ***Delphi****, par exemple, pour afficher clairement les résultats, ou bien de proposer d'autres heuristiques pour résoudre le problème* $1/r_i/\overline{C}$.

Bibliographie

[1] ***H. ADJIRI***, Les problèmes d'ordonnancement d'atelier : M-Machine identique en parallèle, *Mémoire présenté pour l'obtention du diplôme de Master Académique, Université Mohamed Boudiaf - M'sila, 2017/2018.*

[2] ***A. BENKACI***, *Note de cours* Problèmes d'optimisation combinatoire, *Pour M2 Ro Université de Médéa, 2016.*

[3] ***M. BENSADIA & A. HAMANI***, Problème d'ordonnancement avec dates échues sur une seule machine, *Mémoire de Master, Université de Médéa, 2020.*

[4] ***M. BENTTALEB***, Gestion de production sous incertitudes, *Thèse de doctorat de l'UTT, Université de Technologie Troyes, 2018.*

[5] ***M. BERRANDJIA***, Approche multi-agents pour l'ordonnancement par coopération multi-ressources avec délais de production, *Mémoire présenté en vue de l'obtention du diplôme de Magister en Informatique, Ecole doctorale STIC Sciences et Technologies de l'information et de la Communication Option : Systèmes d'information et Connaissances "SIC", 2014/2015.*

[6] ***P. BRUCKER***, Scheduling Algorithms, First Edition,*Book from the internet, Osnabrck ,Octobre 2006.*

[7] ***CO 454***, Scheduling Theory, Lecture 4 : Single Machine Environments (Release Dates), *May 14th, 2009.*

[8] ***N. DERADRA & N. KEDJAR***, Ordonnancement du projet de rénovation de la base de vie de la station de pompage SP3-OB1 (W. MSila), *Mémoire de Master, Université de Béjaa, 2016/2017.*

[9] ***A. FERDI & M. SLIMI***, Optimisation et Planification de l'Ordonnancement des Organes véhicule Cas : SNVI-VIR, *Mémoire de Master, Université Mohamed Bougara de Boumerdès, 2019/2020.*

[10] ***A. HADRI***, L'ordonnancement par insertion en temps reel de la production dans un atelier flexible, *Mémoire Présenté En vue de l'obtention du diplôme de Magister Spécialité Génie Industriel Option Génie Industriel et Productique,2011/2012.*

[11] ***B. Hela***, Sur 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, 2009.*

[12] ***S. KOUIDER AIAD***, Note de cours, Formulation des problèmes d'ordonnancement, *pour Master 2 Recherche opérationnelle, Université de Médéa, 2022.*

[13] ***I. LARIBI***, Résolution de problèmes d'ordonnancement de type Flow-Shop de permutation en présence de contraintes de ressources non-renouvelables, *thèse de Doctorat, Université Tlemcen, 12 décembre 2018.*

[14] ***J. LENSTRA & D. SHMOYS***, Elements of Scheduling, *fragments of a book on machine scheduling, 1977.*

[15] ***M. MEBROUKI & M. GASTI***, Étude de certains problèmes d'ordonnancement avec temps de préparation, *Mémoire de Master, Université de Médéa, 2022.*

[16] ***L. MOUSS***, Ordonnancement des systèmes de production sous contraintes technologiques ou contextuelles : Modélisation, étude de complexité et approches intégrées de résolution, *thèse de Doctorat, Université de Batna 2, 22 Octobre 2017.*

[17] ***R. SRITHARAN & M. ESCHEN & and***, The complexity of the list partition problem for graphs, *West virginia University, 2008.*