République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Faculté des sciences

Département des Mathématiques et Informatique



MEMOIRE

présenté pour l'obtention du

Diplôme de Master en Mathématiques

Spécialité : Recherche Opérationnelle

présenté par :

BENSADIA Meriem & ***HAMANI Amal***

Thème

Problème d'ordonnancement avec dates échues sur une seule machine

Soutenu devant le jury composé de :

M_{me}	K.SERIER ABDALLAH	Maitre assistant	UYF Médéa	Présidente
M_r	A.BENKACI	Maitre assistant	UYF Médéa	Examinateur
M_r	K.TABLENNEHAS	Maitre assistant	UYF Médéa	Examinateur
M_{me}	S.KOUIDER AIAD	Maitre assistant	UYF Médéa	Promotrice

Septembre 2020

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.

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

Nous tenons à remercier **Monsieur A.BENKACI et Monsieur k.TABLENNEHAS** Maitre assistant à l'Université Dr.Yahia Farés de Médéa, pour nous avoir fait l'honneur d'être membres du jury. Ainsi que pour avoir consacré une partie de leur 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

À la plus belle créature qu'Allah a créée sur terre..

Je dédie ce travail :

à mes très chers parents : Mon Père "Mohamed", Ma Mère "Fatiha"qui m'ont guidé durant les moments les plus pénibles de ce long chemin, mes chers parents qui ont sacrifié toute leur vie afin de me voir devenir ce que je suis, Maman j'adore ,Papa j'adore.

À mes sœurs : "Hadjira", "Hassiba", "Farah", "Karima", leur mari "Abdalnour", grande sœur, leur mari "Moussa" et RAHAF

À mon fiancé "Karim" qui a toujours était à mes cotés.

À tous ma famille "HAMANI" et la famille "KACI".

À ma binôme "Meriem", mon intime amie et ma sœur qui n'existait pas, merci pour tous les beaux moments que nous avons vécus, dieu m'a toujours préserver pour moi.

À mes chère amies : "Loubna", "Asma", "Souad", "Salima".

. Ainsi que tout mes collègues de la promo RO

À tous ceux qui, par un mot, m'ont donné la force de continuer. . .

Amal

Dédicaces

À la plus belle créature qu'Allah a créée sur terre

Je dédie ce travail :

À la source de l'amour et à la rivière de tendresse qui coule qui ne se dessèche pas, à celle dont le cœur m'a vu avant que ses yeux ne me voient, et elle m'a bénie avec ses prières , m'a encouragée et est restée les nuits pour moi, comme elle brûlait comme une bougie pour éclairer mon chemin, à ceux qui possèdent le paradis sous ses pieds et la langue ne peut pas la remercier, le secret de mon existence et le baume de mon cœur. "Que Dieu lui fasse miséricorde et j'habiterai dans ses jardins."

mon très chère père "Ahmed" qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence ma la vie. Papa j'adore.

À mon cher mari "Hakim" Pour ses aides précieuses, son encouragement et sa patience, à qui je dois offrir ma réussite, qu'il trouve ici l'expression de mon affection et une récompense de ses sacrifices.

À ma très chère fille "Rahoufa".

À mes chères sœurs et frères "samia", "Naima", "Amina", "Amirouch", "Messaoud", "Samir" ,"hamza","mohemed "et leurs enfants .

À mes deuxièmes parents "fatima zohra" et"mhamad".

À tous ma famille "BENSADIA" et la famille "MADJBER".

À ma binôme "AMAL", mon intime amie et ma sœur qui n'existait pas, merci pour tous les beaux moments que nous avons vécus, dieu m'a toujours préserver pour moi.

À mes chère amies : "Loubna", "Asma","loubna", "Nawel".

tout mes collègues de la promo RO .

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

Meriem

Résumé

La théorie de l'ordonnancement est une branche de la recherche opérationnelle qui 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 composes d'une seule opération qui nécessite la même machine.

Dans ce travail, nous nous intéressons à étudier quelques problèmes d'ordonnancement avec dates échues sur une seule machine.

Mots clés : Ordonnancement, $1/r_i/L_{max}$, une seule machine, heuristique.

Abstract

Scheduling is a solution to the scheduling problem. It is defined by the schedule for performing tasks and allocating resources and aims to meet one or more objectives. For scheduling with a single machine, all the tasks to be performed are done by a single machine. The tasks then are composed of a single operation which requires the same machine.

In this work, we are interested in studying some scheduling problems with expired dates on a single machine.

Keywords : Scheduling, $1/r_i/L_{max}$, single machine, heuristics

ملخص

نظرية الجدولة هي فرع من بحث العمليات ،يحدده الجدول الزمني لتنفيذ المهام وتخصيص الموارد ويهدف لتحقيق هدف واحدا او اكثر .
من اجل الجدولة في الة واحدة مجموعة المهام يتم تنفيذها بواسطة الة واحدة. تتكون المهام من عملية وحيدة التي تتطلب نف المهام.

في هذا العمل نحن مهتمون بدراسة بعض مشاكل الجدولة مع تواريخ انتهاء الصلاحيةعلى جهاز واحد.

الكلمات المفتاحية: الجدولة، $1/r_i/L_{max}$، . الة واحدة، ارشادية

Table des matières

Introduction Générale **1**

1 Concepts de base **3**
1.1 Introduction . . . 3
1.2 Présentation d'un problème d'ordonnancement . . . 3
1.2.1 Définition d'un problème d'ordonnancement . . . 3
1.2.2 Définition de l'ordonnancement . . . 3
1.2.3 Éléments d'un problème d'ordonnancement . . . 4
1.3 Classification d'un problème d'ordonnancement . . . 7
1.4 Représentation des problèmes d'ordonnancement par un Diagramme de Gantt 9
1.5 Analyse de complexité . . . 9
1.5.1 Les principes fondamentaux de complexité . . . 9
1.5.2 Les classes de complexité . . . 10
1.5.3 Réduction entre problème d 'ordonnancement . . . 12
1.6 Conclusion . . . 13

2 Résolution de quelques problèmes faciles **14**
2.1 Introduction . . . 14
2.2 Minimisation de la grande tardivité . . . 14
2.2.1 avec temps de traitement arbitraire (Problème $1//L_{max}$) . . . 14
2.2.2 avec contrainte de pécédence Problème $1/prec/L_{max}$ . . . 15
2.2.3 avec temps de traitement égal 1 et contrainte de précédence (Problème $1/prec, p_i = 1/L_{max}$) . . . 17
2.2.4 avec préemption et date de disponibilité et contrainte de précédence (problème $1/prec, pmtn, r_i/L_{max}$) . . . 18
2.2.5 avec préemption et date de disponibilité (Problème $1/pmtn, r_i/L_{max}$) 20
2.3 Minimisation du nombre de tâche en retard . . . 21
2.3.1 avec temps de traitement arbitraire (Problème $1//\sum_{i=1}^{n} u_i$) . . . 21
2.3.2 avec date de disponibilité (Problème $1/r_i < r_j \longrightarrow d_i < d_j/\sum_{i=1}^{n} u_i$) 24
2.3.3 avec date de disponibilité et temps de traitement égale 1(Le problème $1/r_i, p_i = 1/\sum_{i=1}^{n} w_i u_i$) . . . 26
2.4 Conclusion . . . 27

3 Problèmes difficiles **28**
3.1 Introduction . . . 28
3.2 Minimisation de la grande tardivité $1/r_i/l_{max}$ . . . 28

3.3 Minimisation de retard 29
3.3.1 Problème $1//\sum_{i=1}^{n} T_i$ 29
3.3.2 Problème $1//\sum_{i=1}^{n} w_i T_i$ 31
3.3.3 problème $1//\sum_{i=1}^{n} w_i u_i$ 34
3.4 Conclusion 35

4 Heuristiques proposées **36**
4.1 Introduction 36
4.2 Représentation du problème $1/r_i/L_{max}$ 36
4.2.1 Le modèle mathématique du problème 36
4.2.2 Définition du problème 37
4.3 Les heuristiques pour résoudre le problème $1/r_i/L_{max}$ 37
4.3.1 Définition 37
4.3.2 Les heuristiques proposés 37
4.3.3 Exemple d'application 39
4.4 Comparaison entre les heuristiques proposées 43
4.5 Conclusion 44

Conclusion Générale **45**

BIBLIOGRAPHIE **46**

Table des figures

1.1 Caractérisation d'une tâche T_i 5
1.2 Diagramme de Gantt d'un ordonnancement 9
1.3 Schéma représentative de classe de complexité 11
1.4 Réduction des machines 12
1.5 Relation de précédence 12
1.6 Réduction entre les critères 12
1.7 contrainte sur les temps de traitement 12

2.1 Solution optimale pour l'exemple 2.1 15
2.2 Réseau pour l'exemple 2.2. 16
2.3 Solution optimale pour l'exemple 2.2 16
2.4 Solution optimale pour l'exemple 2.3 18
2.5 Ordonnancement de S et $S*$ pour le problème $1/prec, pmtn, r_i/L_{max}$. . . . 19
2.6 Les contraintes de précédence de l'exemple 2.4. 19
2.7 Solution optimale pour l'exemple 2.4 20
2.8 Solution optimale pour l'exemple 2.5 21
2.9 Solution optimale pour l'exemple 2.6. 24
2.10 Solution optimale pour l'exemple 2.7 25
2.11 Graphe bipartie qui présente l'exemple 2.8. 27
2.12 Solution optimale pour l'exemple 2.8. 27

3.1 Partition réduite au problème $1/r_i/L_{max}$ 29
3.2 Partition réduite au problème $1//\sum_{i=1}^{n} w_i T_i$ 33

4.1 Solution générale de l'exemple par l'heuristique $H1$ 40
4.2 Solution générale de l'exemple par l'heuristique $H2$ 41
4.3 Solution générale de l'exemple par l'heuristique $H3$ 43

Liste des algorithmes

1 Algorithme de $1/prec, p_i = 1/L_{max}$ 17
2 Algorithme de moore 22
3 Algorithme Pour le calcul de E 24
4 Algorithme de Créneaux 26
5 Algorithme de H1 38
6 Algorithme de H2 38
7 Algorithme de H3 39

Liste des abréviations

EDD	Earliet Due Date.
NP	No-Polynomial.
NPC	NP-Complet .
OPT	Optimized Production Technology.
P	Polynomial.
PL	Programmation Linaire
WSPT	Weighted Shortest Processing Time.

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 veut d'aboutir au résultant vise ou meilleur résultant possible, elle fait partie aide à la décision dans la mesure ou elle propose des modèles conceptuels en vue d'analyse et de maitriser des situations complexe pour permettre aux décideurs de comprendre, d'évaluer les enjeux et d'arbitrer ou de faire les choix les plus efficace, ce domaine fait largement appel au raisonnement mathématique (logique, probabilités, analyse des données) et la modélisation des procédures, il est fortement lie à l'ingénierie des systèmes, ainsi qu'au management du système d'information.

Des le $XVII$ ème siècle, des mathématiciens comme Christiaan Huygens ou Blais Pascal tentent de résoudre des problèmes de décision dans l'incertain avec l'espérance mathématique, d'autre au $XVIII$ ème et XIX ème siècle résolvent des problèmes combinatoires. Au début du XX ème siècle, l'étude de la gestion de stock peut être considère comme étant à l'origine de la recherche opérationnelle moderne avec la formule du lot économique (dite formule de Wilson) proposée par Harris en 1913 mais ce n'est qu'avec la seconde guerre mondiale que la pratique va s'organiser pour la première fois en acquérir son non. En 1940 Patrick Blackett est appelé par l'état-major anglais à diriger la première équipe de recherche opérationnelle, pour résoudre certains problèmes tels que l'implantation optimal de radars de surveillance ou la gestion des convois d'approvisionnement le qualification opérationnelle vient du fait que la première application d'un groupe de travail organise dans cette dicipline avant trait aux opérations militaires. La domination est restée par la suit même si le domaine militaire n'est plus le principal champ d'application de cette discipline après la guerre, les techniques se sont considérablement développées grâce notamment à l'explosion des capacités de calcul des ordinateurs, les domaines d'application se sont également multipliés.

La théorie d'ordonnancement est une branche de la recherche opérationnelle qui s'intéresse au calcule 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 considèrè comme un sous-problème de planification dans lequel il s'agit de décider de l'exécution opérationnelle des tâches planifiées.

Les problèmes d'ordonnancement d'ateliers consistent surement pour les entreprises une des difficultés importantes de leurs systèmes de gestion et de pilotage de la production, en effet, c'est ce niveaux 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 ordonnancement concerne un produit ou un lot de produit à fabriquer en respectant les gammes de fabrication.

Dans notre mémoire nous sommes intéressés par l'ordonnancement avec dates échue sur une seul machine, oú nous donnons quelques problèmes qui sont NP-difficiles et d'autre polynomiaux et nous proposons quelques heuristiques pour le résoudre et trouver une solution approchée. Notre étude est constitué en quatre chapitres, il est organisé comme suit :

Dans le premier chapitre, nous introduisons les problèmes d'ordonnancement, les paramètres nécessaires pour caractériser ce dernier ainsi que la représentation de ces problèmes sur une seule machine, et la complexité de la résolution des problèmes ordonnancement, est également aborder.

Le deuxième chapitre est consacré aux problèmes faciles, nous allons présenter des algorithmes pour résoudre ce type des problèmes et nous donnons des exemples illustratifs.

Dans le troisième chapitre nous allons porter les problèmes difficiles, nous avons les prouver par le problème de 3- partition et quelques problèmes qui sont NP-difficiles

Nous avons présentés dans le quatrième chapitre la représentation du problème $1/r_i/L_{max}$ et nous allons proposer trois heuristiques pour avoir la résolution satisfaite, la dernière partie concerne l'application de ces heuristiques sur un exemple donné.

Chapitre 1

Concepts de base

1.1 Introduction

Dans ce chapitre on s'intéresse aux problèmes d'ordonnancement de manière générale, il s'agit de présenter les éléments fondamentaux du problème d'ordonnancement ainsi les concepts de la théorie de complexité de ce type de problème.

1.2 Présentation d'un problème d'ordonnancement

1.2.1 Définition d'un problème d'ordonnancement

Les problèmes d'ordonnancement apparaissent dans tous les domaines : informatique, construction, industrie, administration, etc... Nombreuses sont les définitions proposées au problème d'ordonnancement. Dans ces définitions, on retrouve l'aspect commun de l'affectation de ressources aux tâches, en recherchant une certaine optimisation. On peut évoquer parmi ces définitions ce qui suit : « Etant donné un ensemble de tâches à accomplir, un problème d'ordonnancement consiste à déterminer quelles opérations doivent être exécutées, et à assigner des dates et des ressources à ces opérations de façon à ce que les tâches soient, dans la mesure du possible, accomplies en temps utile, au moindre coût et dans les meilleures conditions ».
« Résoudre un problème d'ordonnancement consiste à ordonnancer i.e. programmer ou planifier dans le temps, l'exécution des tâches en leur attribuant les ressources nécessaires, matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation » [16]
. D'une façon générale, il y a problème d'ordonnancement quand il existe :

- Un ensemble de travaux (tâches ou encore jobs ou lots) à réaliser.
- Un ensemble de ressources à utiliser par ces travaux.
- Un programme (calendrier) à déterminer, pour allouer convenablement les ressources aux tâches[14].

1.2.2 Définition de l'ordonnancement

Un ordonnancement décrit l'exécution des tâches et l'allocation des ressources au cours du temps et vise à satisfaire un ou plusieurs objectifs. De manière plus précise, on

parle d'ordonnancement lorsqu'on réserve le terme séquencèrent au cas où seul l'ordre relatif des tâches est fixé indépendamment des dates d'exécution. En partant de ces trois définitions, les propriétés élémentaires d'un problème d'ordonnancement découlent directement. Un problème d'ordonnancement fait évoquer généralement les points suivants [24] :

- Existence d'un nombre fini de tâches.
- Partage de ressources limitées entre ces tâches.
- Détermination de la date de début d'exécution pour chaque tâche.
- Respect des contraintes imposées sur les tâches et les ressources [22].

1.2.3 Éléments d'un problème d'ordonnancement

Les différentes données d'un problème d'ordonnancement sont les tâches, les contraintes, les ressources et la fonction économique.

Définition 1.1. *Une tâche est une entité élémentaire localisée dans le temps, par une date de début et/ou de fin, et dont la réalisation nécessite une durée préalablement définie. Elle 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[5]. On distingue deux types de tâches [10] :*

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.*

Exemple 1.1. *:*

— *En construction : Pose des dalles, peinture, plomberie.*
— *En informatique : Exécution d'un programme, impression d'un document, etc... .*
— *En transport ferroviaire : Occupation d'une portion de voie par un train.*

L'ensemble des tâches est généralement notée par T, le nombre de tâche notée par n et chaque tâche est noté par Ti. et elle est décrite par l'ensemble des caractéristiques suivantes [02] :

- t'_i : date de début d'exécution de la tâche T_i.
- r_i : date de disponibilité de la tâche T_i (date de début au plus tôt).
- P_i : durée d'exécution de la tâche T_i (temps de traitement de la tâche T_i).
- d_i : date échue
- C_i : date de fin d'exécution de la tâche T_i,

$$c_i = t_i + p_i$$

- W_i : poids associé à la tâche.
- L_i : la tardivité de la tâche T_i,

$$l_i = c_i - d_i$$

- t_i : retard d'exécution de la tâche T_i,

$$\max_{1 \leqslant i \leqslant n} \{c_i - d_i, 0\}$$

- U_i : indicateur de retard de la tâche,

$$u_i = \begin{cases} 1, sic_i \geqslant d_i \\ 0, sinon \end{cases}$$

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

D'une façon plus schématique ou présente la figure suivante :

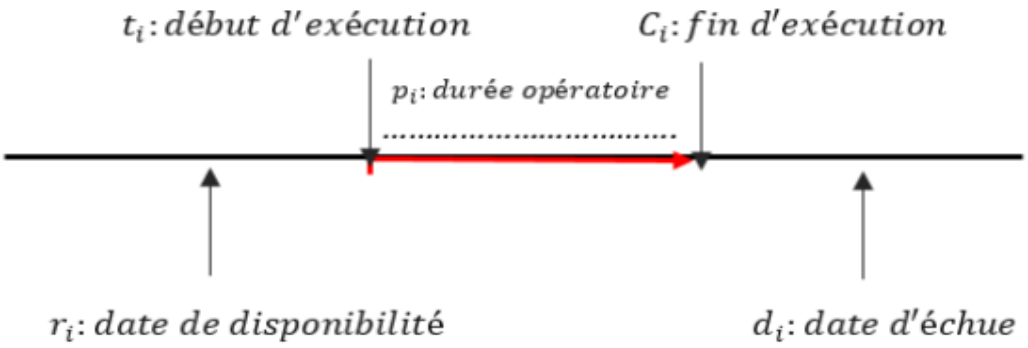


FIGURE 1.1: Caractérisation d'une tâche T_i

Définition 1.2. *Une ressource est un moyen destiné à être utilisé pour la réalisation d'une tâche, et disponible en quantités limitées[18].*
On distingue deux types des ressources [06] :

Ressources renouvelables vs. ressources consommables : *Une ressource est dite renouvelable, si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité comme : les hommes, les machines, l'espace, etc.... Par contre, une ressource est consommable, lorsqu'elle devient non disponible après avoir été utilisée, comme la matière première, l'énergie, etc.... .*

Ressources disjonctives vs. ressources cumulatives : *Une ressource est dite disjonctive (ou non partageable) : si elle ne peut être affecté qu'à une seule tâche à la fois. Ce type concerne surtout les ressources renouvelables. Par contre, une ressource cumulative (partageable) : peut être utilisée par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail, ...)[16].*

Définition 1.3. *Une contrainte exprime des restrictions sur les valeurs que peuvent prendre simultanément les variables représentant les relations reliant les tâches et les ressources. On distingue deux types de contraintes, les contraintes temporelles et les contraintes de ressources[07].*

Les contraintes temporelles : *Les contraintes temporelles concernent les délais de fabrication imposés. Ces contraintes peuvent être :*

Des contraintes de dates butoirs : *certaines tâches doivent être achevées avant une date préalablement fixée.*

Des contraintes de précédence : *une tâche T_i doit précéder la tâche T_i.*

Des contraintes de dates au plus tôt : *: liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches.*

Les contraintes de ressources : *Ces contraintes concernent la limitation de la quantité des ressources de chaque type. Dans ce cadre, deux types de contraintes de ressources sont distingués :*

Les contraintes disjonctives : *induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource.*

Les contraintes cumulatives : *impliquant la limitation du nombre de tâches à réaliser en parallèle[07].*

Définition 1.4. *Généralement l'objectif d'un problème d'ordonnancement Consiste à optimisation un ou plusieurs critère(s) ou à trouver une solution admissible qui respect les contraintes du problème[14].*

Les critères liés aux paramètres temporels caractérisant les tâches. Les plus utilisés sont[18] :

- Longueur d'ordonnancement : $C_{max} = \max_{1 \leqslant i \leqslant n} \{c_i\}$
- La date de fin des tâches moyenne : $\overline{C} = \frac{1}{n} \sum_{i=1}^{n} c_i$
- Le temps de fin de traitement moyen pondéré :$\overline{C_w} = \frac{\sum_{i=1}^{n} w_i c_i}{\sum_{i=1}^{n} w_i}$
- La grande tardivité : $L_{max} = \max_{1 \leqslant i \leqslant n} \{l_i\}$
- La tardivité moyen :$\overline{L} = \frac{1}{n} \sum_{i=1}^{n} l_i$
- La tardivité moyen pondéré : $\overline{L_w} = \frac{\sum_{i=1}^{n} w_i l_i}{\sum_{i=1}^{n} w_i}$
- Le grand temps de flow : $F_{max} = \max_{1 \leqslant i \leqslant n} \{f_i\}$
- Le temps de flow moyen :$\overline{F} = \frac{1}{n} \sum_{i=1}^{n} f_i$

- Le temps de flow moyen pondéré : $\overline{F_w} = \dfrac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$
- 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 tâche en retard :$N_T = \sum_{i=1}^{n} u_i$

1.3 Classification d'un problème d'ordonnancement

Pour des raisons de clarté et de compréhension, il est important d'adopter un descripteur qui soit basé sur l'existant et homogénéisant les notations actuelles. La notion ($\alpha/\beta/\gamma$) introduire par Graham et al. est couramment utilisée pour distinguer les différents problèmes d'ordonnancement entre eux et les classifier[08] :

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 égal à m

Champ β : Contraintes
Permet d'obtenir des information 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 tâches 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.
$\beta_6 = S_i$: S'il existe un temps de préparation.

Champ γ : Critère d'optimisation
Le troisième champ γ décrit le critère d'optimalité. On cherche à minimiser Y avec $Y \in \{C_{max}, \overline{C}, \overline{C_w},$
$T_{max}, L_{max} \ldots\}$.

Exemple 1.2. *:*

- $R2//L_{max}$ *: Traiter un ensemble des tâches indépendantes sans préemption ayant des temps de traitement et des dates échues sont arbitraire et des dates de disponibilité nulles sur des machines parallèles générales l'objectif est de minimiser le grand tardiveté.*
- $1/tree, pmtn, r_i/T_{max}$ *: Traiter un ensemble des tâches dépendantes, où le graphe de précédente est donné sous forme d'un arbre avec préemption ayant des temps de traitement arbitraires et des dates de disponibilité sont arbitraire sur un machine. L'objectif est de minimiser le grand retard.*

1.4 Représentation des problèmes d'ordonnancement par un Diagramme de Gantt

Les ordonnancements sont représentés par un diagramme de Gantt, ce dernier a été développé par H.-L. Gantt en 1910. Le diagramme de Gantt est utilisé aussi dans les entreprises pour la planification des projets. Dans le cas de la gestion de la production, ce diagramme se compose de plusieurs lignes horizontales (ou des étages) qui représentent les machines, les tâches exécutées sur une machine donnée sont représentées par des rectangles, de longueur proportionnelle à leur durée. Par conséquent, le diagramme de Gantt donne, un aperçu des tâches exécutées sur l'ensemble des machines au cours du temps[08].

Exemple 1.3. *: On désire traiter 5 tâches sur une machine*

T_i	T_1	T_2	T_3	T_4	T_5
p_i	*2*	*1*	*4*	*2*	*3*

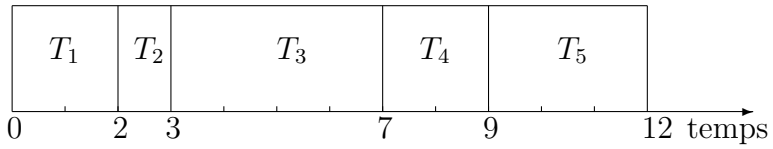


FIGURE 1.2: Diagramme de Gantt d'un ordonnancement

1.5 Analyse de complexité

1.5.1 Les principes fondamentaux de complexité

1.5.1.1 Définition de complexité

La complexité d'un algorithme est le nombre asymptotique élémentaire (+,-,*, /, test) qu'il doit effectuer en fonction de la taille des données La complexité algorithmique s'attache à comparer les algorithmes selon leur temps d'exécution[1].

1.5.1.2 Un algorithme

Est une procédure de résolution composé d'une suite finie d'opération élémentaire qui pour toute instance du problème à traiter, transforme les données initiales (entrées) en résultants (sorties) , le mot algorithme vient du nom du mathématicien ALKAWARIZM (9 ème siècle)qui a fournit la premier algorithme pour résoudre les équations linéaires et quadratiques[4].

1) **Un algorithme est dit efficace :** Ou encore polynomial lorsque le nombre total d'opération élémentaires (opération mathématique, comparaison) effectuées pour aboutir à la solution est borné par un polynôme en la taille du problème en entrée, donc un problème est dit polynomial s'il existe un algorithme polynomial pour le résoudre, il est dit pseudo–polynômial si sa complexité est bornée par un polynôme en fonction de la taille de la plus grand instance du problème[18].
2) **Un algorithme non polynômial :** Est un algorithme dont le nombre d'opérations n'est pas donné par un polynôme en n.[01]

Notation	Désignation	Notation	Désignation
O(1)	Constance	O(n^2)	Quadratique
O(log(n))	Logarithmique	O(n^3)	Cubique
O(n)	Linaire	O(a^n),a>1	Exponentiel
O(nlog(n))	Quasi-linaire	O(n !)	Factorielle

1.5.1.3 Problème d'optimisation combinatoire

Un problème d'optimisation combinatoire consiste à déterminer un meilleur élément parmi un nombre fini d'élément 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[4].

1.5.1.4 Problème de décision

Chaque problème d'optimisation combinatoire peut se réduire à un problème de décision, *i.e.* un problème formulé comme une question dont la réponse est oui ou non, un problème qui n'est pas formulé de cette manière peut être très simplement transformé en un problème de décision équivalent, un problème de décision $D1$ est dit réductible à un autre problème de décision $D2$(noté $D1$ à $D2$)s'il existe une fonction polynômiale f qui transforme chaque énoncé de $D1$ en un autre énoncé de $D2$ de telle manière que la réponse pour $D1$ et oui si seulement si la réponse pour $D2$ est oui. Il est toujours possible de faire correspondant un problème d'optimisation à un problème de décision. Dans le cas, par exemple, de la minimisation d'une fonction objective, le problème peut être posé de la façon suivant : 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 objective[17].

1.5.2 Les classes de complexité

Le but de la théorie de la complexité est la classification des problèmes de décision suivant leur degré de difficulté de résolution. Dans la littérature, il existe plusieurs classes de complexité mais les plus connus sont les suivants [21] :

1) **La classe NP :** La classe des problèmes NP (Non détermination Polynomiale) est la classe des problèmes des décisions qui peuvent être résolus par une machine de Turing(un modèle abstrait du fonctionnement des appareils mécanique de calcul, tel un ordinateur) non déterministe en temps polynomial. Dans la classe des problèmes NP, on distingue deux grandes classes : la classe des problèmes polynomiaux (la classe P) et la classe des problèmes NP Complet (la classe NPC).

2) **La classe P** Un problème est dit appartenir à la classe P s'il existe un algorithme polynomial pour le résoudre, on dit que les problèmes de la classe P sont faciles.
3) **La classe NP- Complet** Un problème de décision est dit NP-Complet si tout problème de la classe NP peut se rendre polynomiale à lui. Tous les problèmes NP-difficiles ne sont pas de difficulté identique, on rencontre par exemple des problèmes que ne peuvent pas être résolus en temps polynomiale avec un codage binaire, mais qui peuvent l'être avec un codage unaire ,ces problèmes sont dits NP-difficiles au sens ordinaire ou simplement NP-difficiles ,les algorithmes pour cette classe de problème sont appelés pseudo-polynomiaux ,pour d'autre problèmes ,on peut ne pas trouver d'algorithme polynomiale quelque soit le codage ,ceux la sont dits NP-difficile au sens fort .

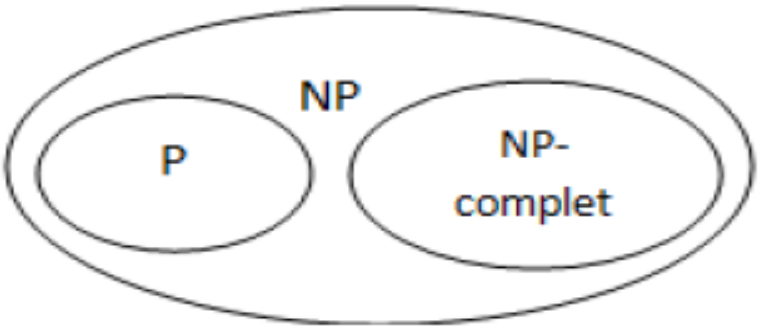


FIGURE 1.3: Schéma représentative de classe de complexité

Un problème d'optimisation combinatoire est dit NP-difficile si le problème de décision qui lui correspondant est NP-complet.

Quelques problèmes de décision NP-Complets : Problème de 3 partitions :

Le problème des 3 partitions est similaire au problème de partition, qui à son tour est lié au problème de somme des sous-ensembles. Dans le problème de partition, le but est de partitionner S en deux sous-ensembles à somme égale. Dans 3 partitions, l'objectif est de partitionner S en m sous-ensembles (ou n / 3 sous-ensembles), pas seulement deux sous-ensembles, avec une somme égale.

Le problème des 3 partitions est un problème NP-complet dans l'optimisation combinatoire. Le problème est de décider si un multiset donné d'entiers peut être partitionné en triplets qui ont tous la même somme. Plus précisément, étant donné un multi-ensemble S de n = 3 m entiers positifs, S peut-il être partitionné en m triplets S1, S2,. . . , Sm de telle sorte que la somme des nombres dans chaque sous-ensemble soit égale ? Les sous-ensembles S1, S2,. . . , Sm doivent former une partition de S dans le sens où ils sont disjoints et ils couvrent S. Soit B la somme (souhaitée) de chaque sous-ensemble Si, ou de manière équivalente, la somme totale des nombres dans S est m B. Le problème des 3 partitions reste fortement NP-complet lorsque chaque entier dans S est strictement compris entre B / 4 et B / 2. Le problème des 3 partitions reste NP-complet même lorsque les entiers de S sont bornés ci-dessus par un polynôme de n. En d'autres termes, le problème reste NP-complet même lors de la représentation des nombres dans l'instance d'entrée en unaire. C'est-à-dire que la partition 3 est NP-complète au sens fort ou fortement NP-complète. Cette propriété, et les 3 partitions en général, sont utiles dans des nombreuses réductions où les nombres sont naturellement représentés en unaire. En revanche, le problème de partition est connu pour être NP-complet uniquement lorsque les nombres sont codés dans

un système non unaire et ont une valeur exponentielle dans n.

1.5.3 Réduction entre problème d 'ordonnancement

Nous donnons ci-dessous quelques réductions (transformations polynomiales) entre problème d'ordonnancement .Une flèche de X à Y signifie 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[17].

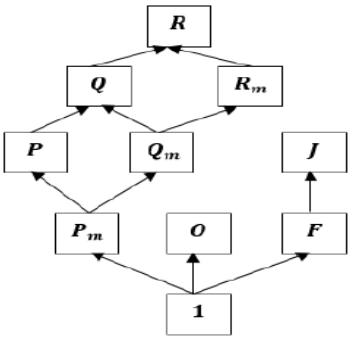


FIGURE 1.4: Réduction des machines

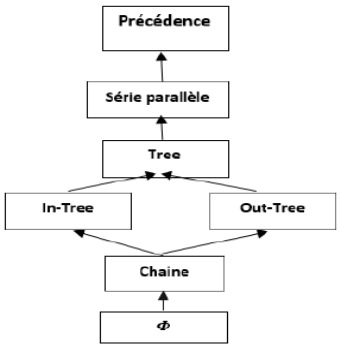


FIGURE 1.5: Relation de précédence

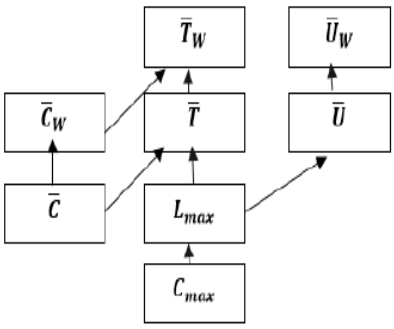


FIGURE 1.6: Réduction entre les critères

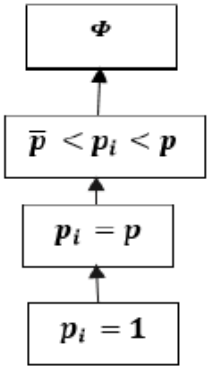


FIGURE 1.7: contrainte sur les temps de traitement

Exemple 1.4. *:*

— *le problème $Q//L_{max}$ est un cas particulier du problème $Q//T_w$ où toutes les machines auraient la même vitesse tout algorithme développé pour $Q//T_w$ peut être appliqué au problème $Q//L_{max}$*

1.6 Conclusion

Nous avons présenté dans ce chapitre le problème d'ordonnancement en précisant sa définition et les différents éléments qui le déterminent : les tâches, les ressources, les contraintes, les objectives ainsi en présentant les critères d'optimisation comme élément de grande importance et la représentation de ces problèmes, puis en traitant la complexité de problème d'ordonnancement.

Chapitre 2

Résolution de quelques problèmes faciles

2.1 Introduction

Résoudre un problème d'ordonnancement c'est choisir pour chaque tâche une date de début de tell sorte que les contraintes du problème soient respectées (la solution est alors dite admissible ou réalisable) et qu'un ou plusieurs critères donnés soient optimisés.

2.2 Minimisation de la grande tardivité

:

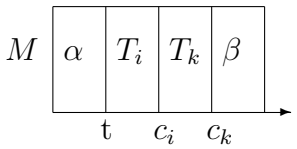
2.2.1 avec temps de traitement arbitraire (Problème $1//L_{max}$)

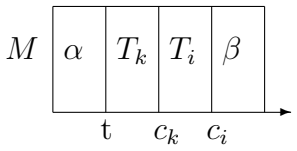
Pour résoudre le problème $1//L_{max}$ on applique **la règle EDD (Earliet Due Date)**.

La règle EDD (Earliet Due Date) : Range les tâches suivant l'ordre croissant de leurs dates échues

Théorème 2.1. *la règle EDD résout le problème* $1//L_{max}$ *[18].*

Preuve :
$d_i > d_k$, la règle EDD n'est pas vérifier





$L_{max} = \max_{\{1\leq i\leq\leq n\}} l_i = \max_{1\leq i\leq n} \{1, L_i, L_k\}$ Avec : l : la tardivité de α etβ
$L_k = t + p_i - d_i$, $L_i = t + p_i + p_k - d_k$
$d_i > d_k$, la règle EDD est vérifiée
$L'_{max} = \max_{\{1\leq i\leq\leq n\}} l'_i = \max_{1\leq i\leq n} \{1, L'_i, L'_k\}$ Avec
l : la tardivité de α etβ $L'_i = t + p_i - d_{(i)}$,$L'_k = t + p_i + p_k - d_{(k)}$ On a $L'_i < L_k$ et $L'_k < L_k$ donc $\max L'_i, L'_k < L_k$ Alors $\max\{L'_i, L'_k\} < \max\{L_i, L_k\} \longrightarrow \max 1, d'_i, L'_k < \max\{1, L_i, L_k\}$ alors $L'_{max} < L_{max}$[18].

Exemple 2.1. *: Etant 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*
d_i	*11*	*9*	*6*	*13*	*7*	*4*	*8*

On applique la règle EDD on obtient la séquence optimal suivant :$T_6T_3T_5T_7T_2T_1T_4$ *qui est représenté par le figure 2.1* $L_{max} = 9$ *talque* $L_{max} = \max_{1\leq i\leq n}\{l_i\}$

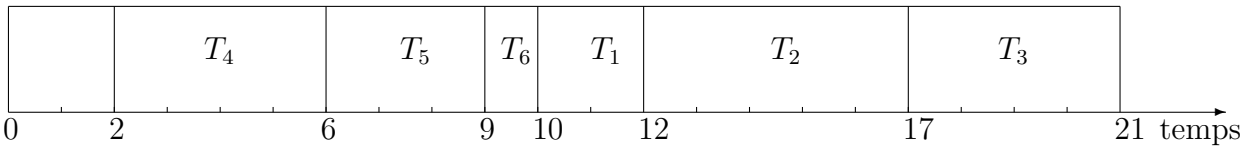


FIGURE 2.1: Solution optimale pour l'exemple 2.1

2.2.2 avec contrainte de pécédence Problème $1/prec/L_{max}$

Pour résoudre le problème $1/prec/L_{max}$ il suffit de faire une modification des dates échues de telle sorte qu'ils soient compatibles avec les contraintes de précédence. On a si $T_i \longrightarrow T_j$ et $d_i < d_j$ alors la règle EDD respecte les contraintes de précédence, tel que : La règle EDD range les tâches suivant l'ordre croissant de leurs dates échues, c'est-à-dire que, dans ce cas, EDD est optimal. Donc pour que la fonction L_{max} n'augmente pas dans aucun ordonnancement réalisable, on a la règle de modification suivante [19] :

$$SiT_i \longrightarrow T_j \quad alors \quad d'_i = \min\{d_i, d_j - p_i\} \quad avecj = nbas1 \tag{2.1}$$

D'où d'algorithme de ce problème consiste deux étapes :

— Construire des dates échues cohérente :

$$d'_i = \min\{d_i, \min_{j/T_(i) \longrightarrow T_j} \{d_j - p_i\} j = nbas \quad 1\}$$

— Appliquer la règle EDD avec des dates échues modifiées.

Exemple 2.2. *: Etant donné le tableau suivant :*

	T_1	T_2	T_3	T_4	T_5
p_i	2	3	1	6	4
d_i	5	8	3	8	7

Les contraintes de précédence sont représentées dans la figure suivante :

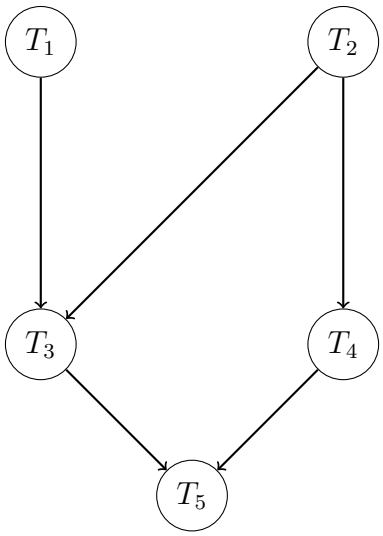


FIGURE 2.2: Réseau pour l'exemple 2.2.

— *On fait la modification des dates échues d_i par la règle de modification 2.1 on obtient :*

	T_1	T_2	T_3	T_4	T_5
d'_i	2	2	3	2	7

— *On applique l'algorithme on obtient la séquence optimal suivant :$T_2T_1T_4T_3T_5$ qui est représenté par le figure 2.3 avec $L_{max} = 1$ talque $L_{max} = \max_{1 \leq i \leq n} \{l_i\}$*

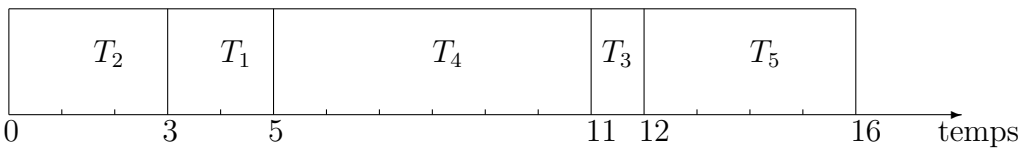


FIGURE 2.3: Solution optimale pour l'exemple 2.2

2.2.3 avec temps de traitement égal 1 et contrainte de précédence (Problème $1/prec, p_i = 1/L_{max}$)

Ce problème est un cas particulier de problème $1/prec, r_i, p_i = 1/f_{max}$ ($f_{max} = \max_{1 \leqslant i \leqslant n} f_j(c_i)$..... f_j monotone pour tout i).
On ajoute la contrainte $p_i = 1$ au problème précédent, et on le résolu plus efficacement, en utilisant une idée de Monma 1982, définit comme suit [9] :

- Commencer par modifier les dates échues. On suppose que toutes les dates échues sont positives ($L_{max} \geq 0$).
- Toutes les tâches sont ordonnancées dans l'intervalle [0 ; n], ce qui implique qu'aucune tâche i avec $d_i \leq n$ n'est en retard
- Pour trier les tâches, nous pouvons utiliser la méthode de Bucket sorting, on construit les ensembles :

$$B_k = \begin{cases} \{i : d_i = k\}, si 0 \leq k \leq\leq n-1 \\ \{i : d_i \geq n\}, sik = n \end{cases}$$

Peut être construit une séquence optimale π par l'algorithme suivant :

Algorithm 1 Algorithme de $1/prec, p_i = 1/L_{max}$

Pour $k = 0$ à n faire
$B_k = \emptyset$
Pour $j = 1$ à n faire
Si $(d_j < n)$ alors $B_{(d_j)} = B_{(d_j)} \cup T_j$
Sinon $B_n = B_n \cup T_j$;
$j = 1$;
Pour $k = 0$ à n faire
Tantque $(\exists T_i \in B_k)$ faire Début
$\pi(j) = i$;
$B_k = B_k / T_i$;
$j = j + 1$;
Fin.

Exemple 2.3. *: On considère le même exemple précédent talque $p_i = 1$*

On applique l'algorithme 2.1 :

$B_k = \emptyset, \forall (j = \overline{1,5})$

- $j = 1 \longrightarrow d_i = 2 < 5 alors B_2 = \{T_1\}$
- $j = 2 \longrightarrow d_i = 2 < 5 alors B_2 = \{T_1, T_2\}$
- $j = 3 \longrightarrow d_i = 3 < 5 alors B_3 = \{T_3\}$
- $j = 4 \longrightarrow d_i = 2 < 5 alors B_2 = \{T_1, T_2, T_4\}$
- $j = 5 \longrightarrow d_i = 7 > 5 alors B_5 = \{T_5\}$
- $j = 1$

— $B_0 = \emptyset$
— $B_1 = \emptyset$
— $B_2 = \{T_1, T_2, T_4\} \longrightarrow \pi(1) = 1 \longrightarrow B_2 = \{T_2, T_4\} \longrightarrow j = 2$
— $B_2 = \{T_2, T_4\} \longrightarrow \pi(2) = 2 \longrightarrow B_2 = \{T_4\} \longrightarrow j = 3$
— $B_2 = \{T_4\} \longrightarrow \pi(3) = 4 \longrightarrow B_2 = \emptyset \longrightarrow j = 4$
— $B_2 = \{T_3\} \longrightarrow \pi(4) = 3 \longrightarrow B_3 = \emptyset \longrightarrow j = 5$
— $B_4 = \emptyset$
— $B_5 = \{T_5\} \longrightarrow \pi(5) = 5 \longrightarrow B_5 = \emptyset$....*stop*

Donc la séquence optimal est :$T_1T_2T_4T_3T_5$ *qui est représenté par le figure 2.4 avec* $L_{max} = 9$ *talque* $L_{max} = \max\limits_{1 \leq i \leq n} \{l_i\}$

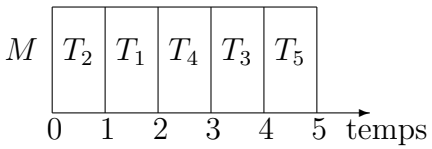


FIGURE 2.4: Solution optimale pour l'exemple 2.3

2.2.4 avec préemption et date de disponibilité et contrainte de précédence (problème $1/prec, pmtn, r_i/L_{max}$)

— Ce problème est un cas particulier de problème $1/prec, pmtn, r_i/F_{max}(f_{max} = \max\limits_{1 \leqslant i \leqslant n} f_j(c_i)$.....$f_j$ monotone pour tout i).
— Réduction directe par **la règle EDD préemptive**.

La règle EDD préemptive :
Ordonnancer les tâches en commençant à la plus petite valeur de r_i. A chaque point de décision t donné par une date de disponibilité ou par la fin d'une tâche, ordonnancer une tâche T_i telle que :

— $r_i \leq t$;
— tous les prédécesseurs de T_i sont ordonnancés ;
— T_i à la plus petite date échue modifiée[18].

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

Preuve Supposons que les deux ordonnancements S etS^* coïncident jusqu'au moment t, Ensuite nous avons une situation comme indiqué dans la figure 2.5, où $s > t$ est le temps minimal auquel la tâche T_i est traité dans S^*. Si nous éliminons la partie de la tâche T_i dans S^* à partir du moment s, déplacer la tâche T_k et les successeurs possibles de T_k ordonnancée entre t et s à droite, et réordonnancée la partie de T_i avec longueur $l = r - s$ dans les intervalles de temps créés vides, puis encore nous avons un ordonnancement optimal. En continuant ce processus nous obtenons après un nombre fini des étapes, un ordonnancement optimal qui coïncide avec S.

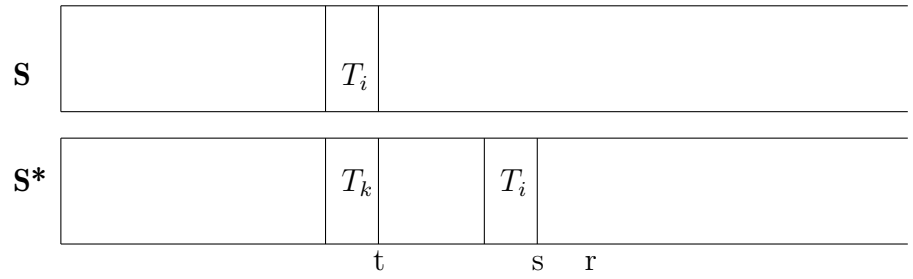


FIGURE 2.5: Ordonnancement de S et $S*$ pour le problème $1/prec, pmtn, r_i/L_{max}$.

Exemple 2.4. *: Etant donné le tableau suivant :*

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

Les contraintes de précédence sont représentées dans la figure suivante :

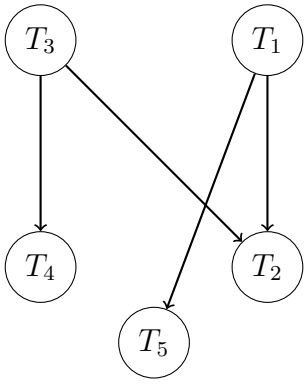


FIGURE 2.6: Les contraintes de précédence de l'exemple 2.4.

- *On fait la modification des dates échues d_i par la règle de modification 2.1 on obtient :*

	T_1	T_2	T_3	T_4	T_5
d'_i	*-4*	*1*	*5*	*4*	*-5*

- *On applique la règle EDD préemptive :*
- $t = 0 \longrightarrow F = \{T_1, T_4\}$*et on a le prédécesseur de* T_4 *n'est pas ordonnancé donc* $s = \{T_1\}$

- $t = 1 \longrightarrow F = \{T_1, T_4, T_5\}$ *et on a les prédécesseurs de* T_5 *et* T_4 *ne sont pas ordonnancés donc* $s = \{T_1\}$
- $t = 2 \longrightarrow F = \{T_2, T_4, T_5\}$ *on a les prédécesseurs de* T_2 *et* T_4 *ne sont pas ordonnancés alors* $s = \{T_1, T_5\}$
- $t = 3 \longrightarrow F = \{T_2, T_4\}$ *on a les prédécesseurs de* T_2 *et* T_4 *ne sont pas ordonnancés alors* $s = \{T_1, T_5\}$
- $t = 4 \longrightarrow F = \{T_2, T_4, T_3\}$ *et on a* $d'_3 < d'_4$ *et* $d'_3 < d'_2$ *donc* $s = \{T_1, T_5, T_3\}$
- $t = 5 \longrightarrow F = \{T_2, T_4\}$ *et on a* $d'_2 < d'_4$ *donc* $s = \{T_1, T_5, T_3, T_2\}$
- $t = 6 \longrightarrow F = \{T_4\}$ *donc* $s = \{T_1, T_5, T_3, T_2, T_4\}$

Donc la séquence optimal est : $T_1T_5T_3T_2T_4$ *qui est représenté par le figure 2.7 avec* $L_{max} = 1$ *talque* $L_{max} = \max\limits_{1 \leq i \leq n} \{l_i\}$

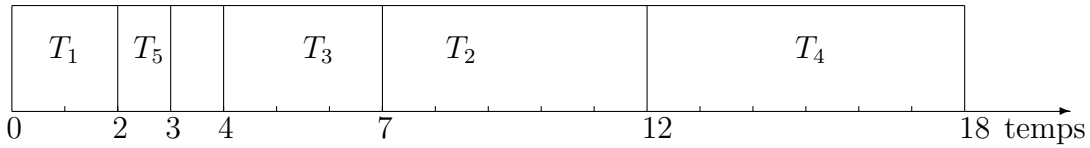


FIGURE 2.7: Solution optimale pour l'exemple 2.4

2.2.5 avec préemption et date de disponibilité (Problème $1/pmtn, r_i/L_{max}$)

Pour résoudre le problème $1/pmtn, r_i/L_{max}$ on utilisant la règle **EDD modifiée (Eartiest Due Date)**.

La règle EDD modifiée (Eartiest Due Date) :

— On pose $t = \min\limits_{1 \leqslant i \leqslant n} \{r_i\}$

— On ordonnancer parmi les tâches disponible à l'instant t les tâches ayant la plus petite date échue.

Théorème 2.3. *La règle EDD modifiée résout le problème* $1/pmtn, r_i/L_{max}$ *[18].*

Exemple 2.5. *: Etant donné le tableau suivant :*

	T_1	T_2	T_3	T_4	T_5	T_6
p_i	*4*	*3*	*2*	*4*	*5*	*1*
r_i	*2*	*5*	*0*	*3*	*0*	*2*
d_i	*10*	*3*	*1*	*1*	*2*	*4*

- *On applique la règle EDD madéfiée on obtient :*

p_i	p_1	p_2	p_3	p_4	p_5	p_6
$t=0$	*4*	*3*	***2***	*4*	5	*1*
$t=2$	4	*3*	*0*	*4*	***5***	1
$t=3$	4	3	*0*	***4***	4	1
$t=5$	4	3	*0*	***2***	4	1
$t=7$	4	3	*0*	*0*	***4***	1
$t=11$	4	***3***	*0*	*0*	*0*	1
$t=14$	4	*0*	*0*	*0*	*0*	***1***
$t=15$	***4***	*0*	*0*	*0*	*0*	*0*
$t=19$	*0*	*0*	*0*	*0*	*0*	*0*

— *Donc la séquence optimal est :* $T_3T_5T_4T_5T_2T_6T_1$ *qui est représenté par le figure 2.8 avec* $L_{max}=11$ *talque* $L_{max}=\max\limits_{1\leq i\leq n}\{l_i\}$

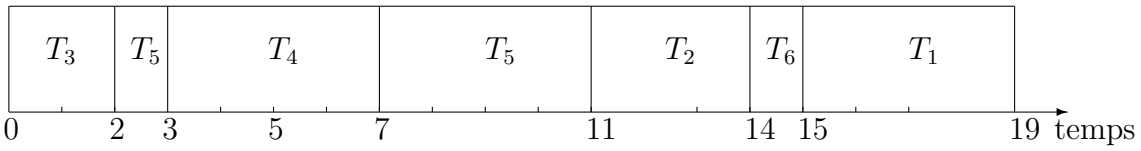


FIGURE 2.8: Solution optimale pour l'exemple 2.5

2.3 Minimisation du nombre de tâche en retard

2.3.1 avec temps de traitement arbitraire (Problème $1//\sum_{i=1}^{n} u_i$)

L'algorithme de Moore résout le problème $1//\sum_{i=1}^{n} u_i$

Idée de l'algorithme de Moore

— Construire un ensemble S maximal des tâches à temps.

— Tâches de S ordonnancées en ordre croissant de dates échues.

— Les tâches n'appartenant pas à S sont en retard et peuvent être exécutées dans un ordre arbitraire après les tâches de S.

— Essayer d'ordonnancer les tâches dans l'ordre des dates échues.

— Si la tâche suivante T_i à ordonnancer est en retard, on retire une tâche T_j avec le plus grand temps d'exécution de S[12].

Algorithm 2 Algorithme de moore

Début
$t = 0$
$S = \emptyset$
Pour $i = 1$ à n faire
$S = S \bigcup T_i$
$t = t + p_i$
Si $(t > d_i)$ alors
Trouver une tâche T_j de S ayant le plus long temps de traitement
$S = S/T_j$
$t = t - p_j$
Fin
Fin
Fin

Théorème 2.4. *l'algorithme* $1//\sum_{i=1}^{n} u_i$ *construit l'ensemble S des premières tâches dans un ordonnancement optimal.[8]*

Preuve : Nous désignons un programme par $P = (S, F)$, où S est l'ensemble des premier tâches et F est l'ensemble des tâches en retard. Soit $T_j$la première tâche qui à l'étape 8 $(S := S/\{T_j\}$) de l'algorithme est supprimée de l'ensemble S. Nous prouvons d'abord qu'il existe un ordonnancement optimal $P = (S, F)$ avec $T_j \in F$. Notons T_k la tâche qui a été ajoutée à S dans la précédente étape $4(S = S \cup \{T_i\})$. Ensuite $p_j = \max_{i=1} pi$. De plus, lors de la planification de la séquence partielle $T_1, ..., T_{j-1}, T_{j+1}, ..., T_k$ aucune tâche n'est en retard. Cela découle du fait que dans $T_1, T_2, ..., , T_{k-1}$ aucun tâche n'est en retard et $p_k \leqslant p_j$. Nous remplaçons T_j par T_k et réorganisons les tâches en fonction des dates échues non décroissantes. Considérons maintenant un ordonnancement optimal $P' = (S', F')$ avec $T_j \in S'$. Il existe une séquence. $T_\pi : T_{\pi_1}, ..., T_{\pi_m}, ..., T_{\pi_r}, T_{\pi_{r+1}}, ..., T_{\pi_n}$ Avec

$$F' = T_{\pi_{r+1}}, ..., T_{\pi_n} \tag{2.2}$$

$$d_(\pi_1) \leqslant ... \leqslant d_{\pi_n} \tag{2.3}$$

$$\{T_{\pi_1}, ..., T_{\pi_m}\} \subseteq \{T_1, ..., T_k\} \tag{2.4}$$

$$\{T_{\pi_{r+1}}, ..., T_{\pi_n}\} \subseteq \{T_(k+1), ..., T_n\} \tag{2.5}$$

$$\{T_j\} \in \{T_{\pi_1}, ..., T_{\pi_m}\} \tag{2.6}$$

Puisque $d_1 \leqslant d_2 \leqslant ... \leqslant d_n$ il existe toujours m tel que (2.4) et (2.5) sont satisfaits. De plus, (2.6) est vrai car $T_j \in S' \bigcap \{T_1, ..., T_k\}$. Depuis $\{T_{\pi_1}...T_{\pi_m} \subseteq S'\}$, pas de tâche dans $\{T_{\pi_1}...T_{\pi_m}\}$ est en retard. Sur d'autre part, il y a une tâche tardive en $\{T_1....T_k\}$. dans n'importe quel programme. Ainsi nous avoir $\{T_{\pi_1}...T_{\pi_m} \subsetneq T_1....T_k\}$.Ce qui implique que nous avons une tâche $T_h(T_1 \leqslant T_h \leqslant T_k)$ avec $T_h \in \{T_{\pi_1}...T_{\pi_m}\}$. Nous supprimons la tâche $T_j$de $\{T_{\pi_1}...T_{\pi_m}\}$ et remplacez-le par T_h. Si nous ordonner tous les tâches dans $\{T_{\pi_1}...T_{\pi_m}\} \bigcup \{T_h\}$ $\{T_j\} \subseteq \{T_1....T_k\}$ $\{T_j\}$ selon aux dates échues non décroissantes, tous les tâches sont en temps car $\{T_1....T_k\}$ $\{T_j\}$ a cette propriété. Si l'on additionne les tâches $T_{\pi_{m+1}}, ..., T_{\pi_r\}}$à l'ensemble $\{T_{\pi_1}...T_{\pi_m}\} \bigcup \{T_h\}$ $\{T_j\}$ et triez tous les tâches en fonction de la non décroissance dates échues, tous les tâches sont en temps car $p_h \leqslant p_j$ implique que

$\sum_{i=1}^{m}\{p_{\pi_i}\} - p_j + p_h \leqslant \sum_{i=1}^{m}\{p_{\pi_i}\}$ .On obtient ainsi un ordonnancement optimal $P = (S, F)$ avec $T_j \in F$. Maintenant, il n'est pas difficile de prouver le théorème par induction sur le nombre n des tâches. Clairement, l'algorithme est correct si $n = 1$. Supposons que l'algorithme est correct pour tous les problèmes avec $n - 1$ tâches. Soit $P = (S, F)$ l'ordonnancement construit par l'algorithme et laissez $P' = (S', F')$, un ordonnancement optimal avec $T_j \in F'$. Ensuite, par optimalité, nous avons $|S| \leqslant |S'|$. Si nous appliquons l'algorithme à l'ensemble des tâches $\{T_1, ..., T_(j-1), T_(j+1), ..., T_n\}$ nous obtenir un ordonnancement optimal du formulaire $(S', F'\ T_j)$. Parce que $(S, F\ T_j)$ est faisable pour le problème réduit, nous avons $|S'| \leqslant |S|$. Ainsi $|S| = |S'|$ et P est optimal[08].

Exemple 2.6. *: Etant donné le tableau suivant :*

	T_1	T_2	T_3	T_4	T_5
P_i	*1*	*4*	*3*	*2*	*8*
d_i	*3*	*9*	*7*	*5*	*11*

Ranger les tâches suivantes par la règle EDD : $T_1T_4T_3T_2T_5$

- ***Itération 0*** *:*
  $S = \emptyset, t = 0.$
- ***Itération 01*** *:*
  $S = S\bigcup\{T_1\} = \{T_1\} \longrightarrow t = 1$
  Test : $t < d_1 = 3$
- ***Itération 02*** *:*
  $S = S\bigcup\{T_4\} = \{T_1, T_4\} \longrightarrow t = 3$
  Test : $t < d_4 = 5$
- ***Itération 03*** *:*
  $S = S\bigcup\{T_3\} = \{T_1, T_4, T_3\} \longrightarrow t = 6$
  Test : $t < d_3 = 7$
- ***Itération 04*** *:*
  $S = S\bigcup\{T_2\} = \{T_1, T_4, T_3, T_2\} \longrightarrow t = 10$
  Test : $t > d_2 = 9$
  Alors : Trouver une tâche T_j *de* S *ayant le plus long temps de traitement.*
  Tell que $S = S/\{T_j\}$
  D'où : $S = S/\{T_2\} = \{T_1, T_4, T_3\}, t = 6$
- ***Itération 05*** *:* $S = S\bigcup\{T_5\} = \{T_1, T_4, T_3, T_5\} \longrightarrow t = 14$ *Test :* $t > d_5 = 11$ *Alors : Trouver une tâche* T_j *de* S *ayant le plus long temps de traitement. Tell que* $S = S/\{T_j\}$ *D'où :* $S = S/\{T_5\} = \{T_1, T_4, T_3\}, t = 6$

D'où :
L'ensemble des tâches en temps est $S = \{T_1, T_4, T_3\}$ *et l'ensemble des tâches en retard est* $T/S = \{T_2, T_5\}$ *La séquence optimale est* $T_1T_4T_3T_2T_5$ *qui est présentée dans la figure 2.9.*
Avec
$\sum_{i=1}^{n} u_i = |T/S| = 2$

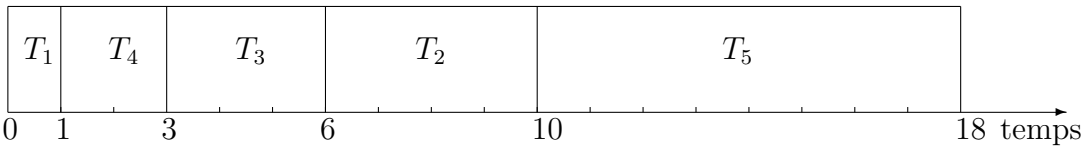


FIGURE 2.9: Solution optimale pour l'exemple 2.6.

2.3.2 avec date de disponibilité (Problème $1/r_i < r_j \longrightarrow d_i < d_j / \sum_{i=1}^{n} u_i$)

Pour un ensemble de tâches donné, les dates de disponibilités et les dates échues sont appelées cohérentes, si $r_i < r_j \longrightarrow d_i < d_j$ pour toutes les tâches T_i, T_j Nous commençons par ordonner des tâches selon les deux, des temps de disponibilité et des dates échues non décroissants, sans perte de généralité, nous pouvons supposer que les tâches sont déjà correctement indexées, c'est-à-dire $r_1 \leqslant r_2 \leqslant ... \leqslant r_n$ et $d_1 \leqslant d_2 \leqslant ... \leqslant d_n$. Tout ordonnancement S peut encore être décrit par une partition du groupe de tâches τ dans le temps fixé $\tau^{\leqslant}$ et ensemble tardif $\tau^{>}$. Les tâches de $\tau^{\leqslant}$ sont traitées dans l'ordre EDD, elles sont donc classées en fonction de leurs indices. Soit $\tau^{\leqslant} = \{T_{k_1}, ..., T_{k_m}\} k_1 < ... < k_2$. La date de fin de traitement C_{k_i} de la tâche T_{k_i} dans cet ordonnancement est donné par :

$$C_{k_1} = r_{k_1} + p_{k_1}$$

$$C_{k_i} = \max\{C_{k_{i-1}}, r_{k_i}\} + p_{k_i}.$$

Ensuite, la dernière tâche de $\tau^{\leqslant}$ est terminée au moment $C(\tau^{\leqslant}) = C_{k_m}$. L'algorithme suivant génère des ordonnancements optimales pour les sous-ensembles $\{T_1, \ldots, T_i\}$dans l'ordre de $i = 1, \ldots, n$ Soit un ordonnancement optimal pour $\{T_1, \ldots, T_i\}$être spécifié par les sous-ensembles $E_i \subseteq \{T_1, \ldots, T_i\}$des tâches à temps. Ensuite, ensemble En donnera un ordonnancement optimal pour τ[9.18].

Algorithm 3 Algorithme Pour le calcul de E

Début
Ordonner les tâches en fonction des dates de disponibilités et des dates échues non décroissantes
$E_0 = \emptyset$
Pour $j = 1$ à n faire
$E_j = E_{j-1} \bigcup T_j$ Si $C(E_j) > d_j$ Alors
Choisir $T_K \in E_j$ tel que $S(E_j - T_K)$ dans l'ordre EDD soit de longueur minimale ;
$E_j = E_j - T_K$ Fin :

Exemple 2.7. *: Etant donné le tableau suivant :*

	T_1	T_2	T_3	T_4	T_5	T_6
d_i	*2*	*4*	*5*	*8*	*19*	*22*
r_i	*0*	*0*	*1*	*2*	*4*	*5*
p_i	*2*	*5*	*4*	*4*	*3*	*1*

Les tâches sont déjà ordonné en fonction des d_i et r_i croissante :

- ***Itération 0*** *:* $E_0 = \emptyset, E_1 = \{T_1\}$
  Test : $c(E_1) = 2 \leqslant d_1 = 2$
- ***Itération 01*** *:* $E_0 = \emptyset, E_2 = E_1 \bigcup \{T_2\} = \{T_1, T_2\}$
  Test : $c(E_2) = 7 > d_2 = 4$
  Alors : on choisit $T_k \in E_j$ *de telle sorte que la sous-ordonnancement obtenue pour* $E_j/\{T_j\}$*. Dans l'ordre EDD soit de longueur minimale.*
  $E_2 = E_2/\{T_1\} = \{T_2\}$
- ***Itération 02*** *:* $E_3 = E_2 \bigcup \{T_3\} = \{T_3, T_2\}$
  Test : $c(E_3) = 11 > d_3 = 5$
  Alors : on choisit $T_k \in E_j$*de telle sorte que la sous-ordonnancement obtenue pour* $E_j/\{T_j\}$*. Dans l'ordre EDD soit de longueur minimale.*
  $E_3 = E_3/\{T_2\} = \{T_3\}$
- ***Itération 03*** *:* $E_4 = E_3 \bigcup \{T_4\} = \{T_3, T_4\}$
  Test : $c(E_4) = 15 > d_4 = 8$
  Alors : on choisit $T_k \in E_j$ *de telle sorte que la sous-ordonnancement obtenue pour* $E_j/\{T_j\}$*. Dans l'ordre EDD soit de longueur minimale.*
  $E_4 = E_4/T_3 = \{T_4\}$
- ***Itération 04*** *:* $E_5 = E_4 \bigcup \{T_5\} = T_4, T_5$
  Test : $c(E_5) = 18 < d_5 = 19$
- ***Itération 05*** *:* $E_6 = E_5 \bigcup \{T_6\} = \{T_4, T_5, T_6\}$ *;*
  Test : $c(E_6) = 19 < d_6 = 22$

D'où :

— *L'ensemble des tâches en temps est* $\tau^{\leqslant}\{T_4, T_5, T_6\}$ *et l'ensemble des tâches en retard est* $\tau^{>}\{T_1, T_2, T_3\}$

— *La séquence optimale est* $T_4T_5T_6T_1T_2T_3$ *qui est présentée dans la figure 2.10 Avec* $\sum_{i=1}^{n} u_i = 3$

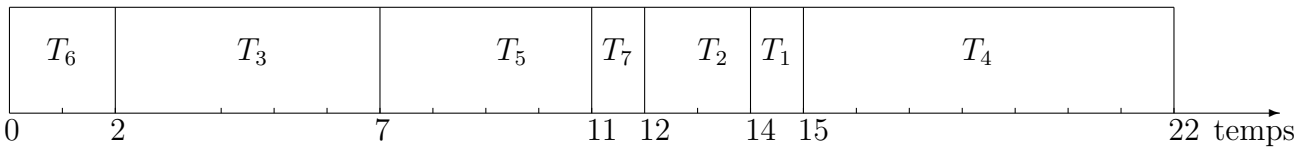


FIGURE 2.10: Solution optimale pour l'exemple 2.7

2.3.3 avec date de disponibilité et temps de traitement égale 1(Le problème $1/r_i, p_i = 1/\sum_{i=1}^{n} w_i u_i$)

: Nous nous intéressons tout d'abord au problème plus général $1/r_i, p_i = 1/\sum_{i=1}^{n} f_i$ avec f_i fonction monotone de c_i pour tout T_i.

- Si le créneau de temps t est affecté à la tâche T_i , alors le coût correspondant est $f_i(t+1)$.
- Puisque les f_i sont monotones, les tâches devraient être ordonnancées le plus tôt possible.
- Supposons que les tâches sont ordonnées de sorte que $r_1 \leqslant r_n \leqslant, ... \leqslant r_n$. Les n créneaux aux quelles les tâches peuvent être ordonnancées au plus tôt peuvent être calculées par :

Algorithm 4 Algorithme de Créneaux

$t_1 = r_1$
Pour $i = 2$ à $i \leqslant n$ faire
$t_i = \max\{r_i, t_{i-1} + 1\}$

- Il existe un ordonnancement qui utilise tous les créneaux de temps $t_i, i = 1, \ldots, n$
- Définissons un graphe biparti complet avec

$$v_1 = \{t_1, \ldots, t_n\}$$

$$v_2 = \{T_1, \ldots, T_n\}$$

Ainsi que des coûts

$$c_{ij} = \begin{cases} f_i(t_j + 1), sir_i < t_j \\ \infty, sinon \end{cases}$$

$1/r_i, p_i = 1/\sum_{i=1}^{n} f_i$ se réduit à un problème d'affectation dans ce graphe biparti[12].

Exemple 2.8. *: Etant donné le tableau suivant :*

	T_1	T_2	T_3	T_4
r_i	*5*	*1*	*2*	*4*
d_i	*3*	*6*	*4*	*1*
w_i	*5*	*8*	*11*	*6*

On applique l'algorithme 2.4 pour définir les Créneaux :

- ***Initialisation*** *:* $t_1 = r_1 = 1$
- ***Itération 01*** *:* $t_2 = \max\{r_2, t_1 + 1\} = \max\{2, 2\} = 2$
- ***Itération 02*** *:*$t_3 = \max\{r_2, t_1 + 1\} = \max\{4, 3\} = 4$

. ***Itération 03*** :$t_4 = \max\{r_4, t_3 + 1\} = \max\{5, 5\} = 5$

Définissons un graphe biparti complet avec

$$v_1 = \{t_1, \ldots, t_4\}$$

$$v_2 = \{T_1, \ldots, T_4\}$$

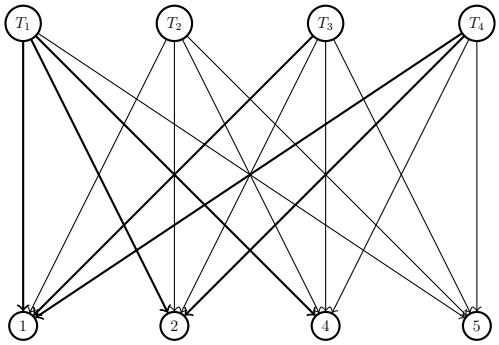


FIGURE 2.11: Graphe bipartie qui présente l'exemple 2.8.

Dans la figure 2.11 l'arête tracée en ligne mince exprime que la tâche T_i disponible en le temps t_j, sinon les tâches n'est pas disponible.
Pour ordonnance ces tâche en réduit ce problème à un problème d'affectation dans ce graphe biparti c'est-à-dire on choisir une seul tâche qui adjacent on le temps t_j en suite on va traitée cette tâche dans ce temps.
La séquence optimale est $T_2T_3T_4T_1$ qui est présentée dans la figure 2.12 Avec $\sum_{i=1}^{n} w_i u_i = 11$

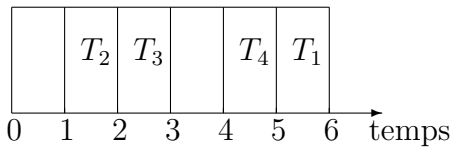


FIGURE 2.12: Solution optimale pour l'exemple 2.8.

2.4 Conclusion

Dans ce chapitre nous avons présenté quelques problèmes d'ordonnancement sur une seule machine avec dates échues comme les problèmes faciles, nous avons précisé des algorithmes pour résoudre ce type des problèmes ainsi que quelques exemples illustratifs.

Chapitre 3

Problèmes difficiles

3.1 Introduction

Dans ce chapitre, nous abordons les problèmes d'ordonnancement difficiles sur une seule machine avec date échue et nous montrons la complexité de ces problèmes dans deux sections.
Nous traitons dans la première section un problème de minimisation de la grande tardivité et dans la deuxième section, les problèmes de minimisation du nombre de tâches en retard.

3.2 Minimisation de la grande tardivité $1/r_i/l_{max}$

Ce problème avec des tâches ayant des dates de disponibilité arbitraire et qui ne permet pas la préemption, est notamment plus difficile. L'ordonnancement optimal n'est pas nécessairement un ordonnancement sans retard.

Théorème 3.1. *Le problème $1/r_i/L_{max}$ est de complexité NP- difficile au sens fort [21].*

Preuve La preuve est basé sur le fait que 3-Partition est réduit à $1/r_i/L_{max}$.
Soient $a_1, \ldots, a_3t, b$, tel que $b/4 < a_i < b/2$ et $\sum_{i=1}^{3t} a_i = tb$ l'instance suivante de $1/r_i/L_{max}$ peut être construit. Le nombre des tâches n est égale à $4t-1$ Et

$r_i = ib + (i-1).$	$p_i = 1.$	$d_i = ib + i.$	$i = 1, \ldots, t-1$
$r_i = 0.$	$p_i = a_i - t + 1.$	$d_i = tb + (t-1)$	$i = t, \ldots, 4t-1.$

Un ordonnancement avec $L_{max} \leq 0$ existe si et seulement si chaque tâche T_i avec $i = 1, \ldots, t-1$, peut être traité entre r_i et $d_i = r_i + p_i$.Cela peut être fait si et seulement si les tâches restantes peuvent être partitionnées sur les t intervalles de longueur b, ce qui peut être fait si et seulement si 3-Partition admet une solution[21].

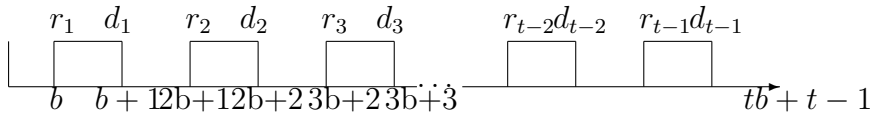


FIGURE 3.1: Partition réduite au problème $1/r_i/L_{max}$

Plusieurs algorithmes exacts ont été proposés pour ce problème, mais ils sont tous basés sur des méthodes énumératives et leur temps de calcul augmente de manière exponentielle avec la taille du problème. Une procédure de séparation et évaluation est proposé pour résoudre ce problème.

3.3 Minimisation de retard

3.3.1 Problème $1//\sum_{i=1}^{n} T_i$

l'objectif de $1//\sum_{i=1}^{n} T_i$ est également important dans la pratique .Minimiser le nombre des tâches tardif $1//\sum_{i=1}^{n} u_i$, en pratique ne peut pas être le seul objectif à mesurer comment les dates échues sont respectées. Certaines tâches devront peut-être attendre longtemps si le nombre de tâches en retard est minimisé. Si, au contraire, la somme des retards sont minimisés, il est moins probable que l'attente d'une tâche donnée soit trop longue.
Le modèle $1//\sum_{i=1}^{n} T_i$ a reçu une énorme attention dans la littérature. Pendant de nombreuses années, sa complexité de calcul est restée ouverte, jusqu'à sa dureté NP a été établie en 1990.
Comme $1//\sum_{i=1}^{n} T_i$ est $NP - difficile$ dans l'ordinaire sens, il permet un algorithme de temps pseudo-polynomial basé sur la programmation dynamique. L'algorithme est basé sur deux préliminaires résultats[21].

Lemme 3.1. *Si $p_i \leq p_k$ et $d_i \leq d_k$, alors il existe une séquence optimale dans laquelle la tâche T_i est ordonnancée avant la tâche T_k [21].*

Ce type de résultat est utile lorsqu'un algorithme à développer pour un chapeau problématique est $NP - dur$. Un tel résultat, souvent appelé critère d'élimination peut être considéré comme un ensemble des contraintes de priorité sur les tâches, permet souvent de ne pas tenir compte d'un assez grand nombre des séquences. Plus de contraintes de priorité crées grâce à ces résultats, plus le problème devient facile. Dans le lemme suivant, la sensibilité d'une séquence optimale au les dates sont prises en compte. Deux problèmes sont envisagés, les deux avoir n tâches avec des temps de traitement $p_1, ..., p_n$. La première instance a des dates échues $d_1, ..., d_n$. Soit C'_k être le dernier temps d'achèvement possible de tâche T_k dans une séquence, disons S", pour cet exemple. La deuxième instance a des dates échues $d_1, ..., d_{k-1}, max(d_k, C'_k), d_{k+1}, ..., d_n$. Soit S" la séquence optimale en ce

qui concerne ce deuxième ensemble de dates échues et C_i'' la fin du la tâcheT_i sous cette deuxième séquence[21].

Lemme 3.2. *Toute séquence optimale pour la deuxième instance est également optimal pour la première instance[21].*

preuve Soit V'(S) le retard total sous une séquence arbitraire S par rapport au premier ensemble de dates échues et que V"(S) désigne le retard total sous la séquence S par rapport au deuxième ensemble de dates échues.
Maintenant

$$V'(S') = V''(S') + A_k$$

Et

$$V'(S'') = V''(S'') + B_k$$

Où, si $C_k' \leq d_k$ les deux ensembles de dates échues sont les mêmes et la séquence S" est optimal pour le deuxième ensemble est donc également optimal pour le premier ensemble. Si $d_k \leq C_k'$, puis

$$A_k = C_k' - d_k$$

$$Et$$

$$B_k = max(0, min(C_k'', C_k') - d_k)$$

Il est clair que $A_k \geq B_k$. Comme S'' est optimal pour la deuxième instance $V''(S') \geq V''(S'')$. Donc $V'(S') \geq V'(S'')$ qui complète la preuve. Dans le reste de cette section, il est supposé que (sans perte de généralité) tous les temps de traitement sont différents, si nécessaire après une perturbation infinitésimale. Supposons que $d_1 \leq, ... \leq d_n$ et $p_k = max(p_1, ..., p_n)$. Autrement dit, la tâche dont la date échue est la plus petite a le temps de traitement le plus long. Il ressort du lemme 3.1 qu'il existe une séquence optimale dans laquelle les tâches $\{T_1, ..., T_(k-1)\}$ apparaissent dans un certain ordre, avant la tâche T_k. Parmi les$n-k$ tâches restantes, c'est-à-dire les tâches $\{T_(k+1), ...T_n$, certains peuvent apparaître avant la tâche $T_(k+1)$ et certains peuvent apparaître après la tâche T_k. Le lemme suivant se concentre sur ces $n-k$ tâches[21].

Lemme 3.3. *Il existe un entier σ, $0 \leq \sigma \leq n-k$, tel qu'il existe une séquence optimale S dans laquelle la tâche T_k est précédé de tous les tâches T_i avec $i \leq k+\sigma$ et suivi de tous les tâches T_i avec $k+\sigma < i$[21].*

preuve Soit C'_k désigne le dernier temps d'exécution possible du $T_k$dans n'importe quelle séquence ce qui est optimal par rapport aux dates échues données $d_1, ..., d_n$. Soit S" une séquence optimale par rapport dates échues $d_1, ..., d_(k-1), max(C'_k, d_k), d_{k+1}, ..., d_n$et qui remplit la condition énoncée dans le lemme 2.1. Soit C'_k indiquent le temps d'exécution de la tâche T_k dans cette séquence. Par la séquence de Lemme 3.2 S" est également optimal par rapport aux dates échues d'origine. Cela implique que $C''_k \leqslant max(C'_k, d_k)$. On peut supposer que la tâche T_k n'est pas précédée dans S par une tâche dont la date échue est postérieure à max (C'_k, d_k) (si tel avait été le cas cette tâche serait en temps et repositionner cette tâche en l'insérant immédiatement après la tâche T_k n'augmenterait pas la fonction objectif). De plus, la tâche T_k doit être précédé de tous les tâches dont la date échue est antérieure à $max(C'_k, d_k)$ (sinon lemme 3.1 serait violé). Donc δ peut être choisi pour être le plus grand entier tel que $d_k + \delta \leqslant max(C'_k, d_k)$. Ceci complète la preuve. .
Dans l'algorithme de programmation dynamique, un sous-programme est requis pour générer un ordonnancement optimal pour l'ensemble des tâches $T_1, ..., T_l$ commence par le traitement de cet ensemble au temps t.
Soit T_k une tâche avec le plus long temps de traitement parmi ces l tâches. Il ressort du lemme 3 que pour certains σ $(0 \leq \sigma \leq l-k)$,il existe une séquence optimale commençant à t qui peut être considérée comme une concaténation de trois sous-ensembles des tâches, à savoir

— les tâches $T_1, T_2, ..., T_{k-1}, T_{k+1}, ..., T_{k+\sigma}$ dans un certain ordre, suivi de la tâche T_k, suivi les tâches $T_{k+\sigma+1}, T_{k+\sigma+2}, ..., T_l$ dans un certain ordre.

Le temps de réalisation du la tâche T_k, $C_k(\sigma)$, est donné par $C_k(\sigma) = \sum_{i \leq k+\sigma} p_i$.
Il est clair que pour la séquence entière soit optimale, le premier et le troisième sous-ensemble doivent être séquencés de manière optimale en eux-mêmes. Cela suggère un procédure de programmation dynamique qui détermine une séquence optimale pour un plus grand ensemble des tâches après avoir déterminé des séquences optimales pour les sous-ensembles appropriés du plus grand ensemble. Les sous-ensembles J utilisés dans cette procédure récursive sont d'un type très spécial. Un sous-ensemble comprend tous les tâches d'un ensemble $\{i, i+1, ..., l-1, l\}$ avec les temps de traitement plus petit que le temps de traitement $p_k$de la tâche T_k. Un tel sous-ensemble est noté par $J(i, l, k)$. Soit $V(J(i, l, k), t)$ le retard total de ce sous-ensemble sous une séquence optimale, en supposant que ce sous-ensemble commence au temps t. La procédure de programmation dynamique peut être$T_i, T_{i+1}, \ldots, T_{l-1}, T_l, T_k$[21].

3.3.2 Problème $1//\sum_{i=1}^{n} w_i T_i$

— $1/r_i, p_i = 1/\sum_{i=1}^{n} w_i T_i$ peut être résolu en O(n^3) (cas particulier de $1/r_i, p_i = 1/\sum_{i=1}^{n} f_i$,

— mais $1/prec, p_i = 1/\sum_{i=1}^{n} T_i$,est NP-difficile.

— Algorithme pseudo-polynomial de Lawler pour $1//\sum_{i=1}^{n} w_i T_i$ si les poids w_j sont **acceptables** pour les temps d'exécution p_j , c'est-à-dire si $p_i < p_j$ implique $w_i \geq w_j$ pour tout $i; j = 1, \ldots, n$[08].

Lemme 3.4. *Supposons que les poids w_j sont acceptables pour les temps d'exécution p_j. Alors il existe une séquence optimale π dans laquelle T_i précède T_j si $d_i \leq d_j$ et$p_i < p_j$, et dans laquelle toutes les tâches à temps sont en ordre croissant des dates échues[8].*

preuve soit π une séquence optimale. Supposons que T_i suive T_j dans π, où $d_i \leq d_j$ et $p_i < p_j$. Parce que les tâches sont agréablement pondérés,$p_i < p_j$ implique $w_j \leq w_i$. Par conséquent, $w_j max\{0, T - d_j\} \leq w_j max\{0, T - d_i\}$ vrai pour tous les tâches T_i. . Ainsi, si nous déplaçons la tâche T_j à la position immédiatement après la tâche T_i , alors ce nouveau ordonnancement doit être également optimal. De plus, si la tâche T_i suit la tâche T_j où $d_i \leq d_j$ et T_i et T_j sont en temps, puis en déplaçant T_j à la position immédiatement après T_i garde $T_j$en temps. Ainsi, nous obtenons une séquence pour laquelle le retard pondéré total n'augmente pas. L'application répétée de ces deux règles donne une séquence optimale satisfaisante les conditions du lemme. Afin de simplifier quelque peu l'exposition, supposons que tous les traitements des temps sont distincts. Si ce n'est pas le cas, ils peuvent être perturbés infiniment sans violer l'hypothèse d'une pondération agréable et sans changer le problème de manière significative.

Théorème 3.2. *Le problème $1//\sum_{i=1}^{n} w_i T_i$ est de complexité NP- difficile au sens fort[23].*

preuve La preuve est faite en réduisant 3-Partition à $1//\sum_{i=1}^{n} w_i T_i$ La réduction est basée sur la transformation suivante. Encore une fois, le nombre des tâches n est choisi égal à $4t - 1$
Et

$$\begin{array}{llll} d_i = 0 & p_i = a_i & w_i = a_i & i = 1, \ldots, 3t \\ d_i = (i - 3t)(b + t) & p_i = 1 & w_i = 2 & i = 3t + 1, \ldots, 4t - 1. \end{array}$$

Soit

$$z = \sum_{i,k=1}^{3t} a_i a_k + \frac{1}{2}(t - 1)tb.$$

.

On peut montrer qu'il existe un ordonnancement avec une valeur objective z si et seulement s'il existe une solution au problème de 3-Partition. Les 3 premières tâches ayant un rapport $\frac{w_i}{p_i}$ égal à 1 et sont dus au moment 0. Il y a$t-1$ tâches avec $\frac{w_i}{p_i}$ rapport égal à 2 et leurs dates d'échues sont à$b+1, 2b+2$, et ainsi de suite. Une solution avec la valeur z peut être obtenue si ces $t - 1$ tâches peuvent être traités exactement pendant les intervalles $[b, b+1], [2b+1, 2b+2], \ldots, [(t-1)b+t-2, (t-1)b+t-1]$ Pour adapter ces $t-1$ tâches à ces $t-1$ intervalles, le premier $3t$ tâches doit être partitionnées en $t$sous-ensembles de trois tâches chacune avec la somme des trois temps de traitement dans chaque sous-ensemble étant égal à b. Il peut être vérifié que dans ce cas la somme des retards pondérés est égale à z. Si une telle partition n'est pas possible, il existe au moins un sous-ensemble dont la somme des trois temps de traitement est supérieure à b et un autre sous-ensemble de dont la somme des trois temps de traitement est inférieure à b . On peut vérifier que dans ce cas la somme des retards pondérés est supérieure à z.
Généralement une méthode approchée est utilisée pour le problème à $1//\sum_{i=1}^{n} w_i T_i$. Le plus souvent, les ordonnancements sont construits à partir de la fin. Au i^{im}niveau de l'arbre

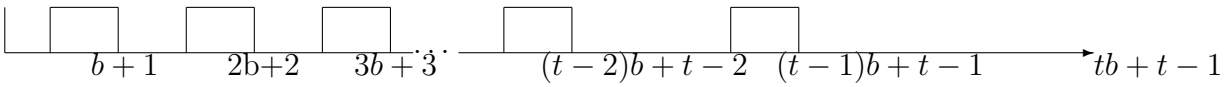


FIGURE 3.2: Partition réduite au problème $1//\sum_{i=1}^{n} w_i T_i$

de recherche, les tâches sont placées dans la $(n-i+1)^{me}$ position. Donc de chaque nœud au niveau $i-1$ il y a $n-i+1$ branches allant au niveau i. Il n'est pas nécessaire d'évaluer tous les nœuds possibles. Les résultats de dominance tels que celui décrit dans le lemme 3.3.1 peut éliminer un certain nombre des nœuds. La borne supérieure sur le nombre des nœuds au niveau j est $\frac{n!}{(n-1)!}$.. L'argument pour construire la séquence en arrière est que les termes plus grands dans la fonction objective sont susceptibles de correspondre à des tâches qui se positionnent plus vers la fin de l'ordonnancement. Il semble avantageux de traiter ces derniers en premier.
Il existe de nombreuses techniques de délimitation différentes. L'un des plus élémentaires techniques de délimitation est basé sur une relaxation du problème à un problème de transport. Dans cette procédure, chaque tâche $T_(i)$avec un temps de traitement (entier) $p_(i)$est divisée en $p_(i)$ tâches, chacun avec un temps de traitement unitaire. Les variables de décision x_{ik}sont 1 si une unité du tâche $T_(i)$ est traitée pendant l'intervalle de temps $[k-1,k]$ et0 sinon.
Ces variables de décision x_{ik} doivent satisfaire deux ensembles de contraintes :
$x_{ik} = \sum_{k=1}^{c_{max}} p_i \ , \qquad \overline{1,n}$
$x_{ik} = \sum_{j=1}^{n} p_i \ , \qquad k = \overline{1,c_{max}}$
Clairement, une solution satisfait ces contraintes ne garantit pas un réalisable ordonnancement sans préemptions. Définir les coefficients de coût c_{ik} qui satisfont $\sum_{k=l-p_i+1}^{l} c_{ik} \leq w_i max(c_i - d_i, 0)$, pour $i = 1, ..., n$,$l = 1, ..., c_{max}$.
Ensuite, la solution de coût minimum fournit une borne inférieure, puisque pour toute solution du problème de transport avec $x_{ik} = 1$ Pour $k = c_i - p_i + 1, \ldots, c_i$ la relation suivante :

$$\sum_{k=1}^{l} c_{ik} x_{ik} = \sum_{k=c_j-p_j+1}^{c_j} c_{ik} \leq w_i max(c_i - d_i, 0)$$

.
Il est assez facile de trouver des fonctions de coût qui répondent à cette relation. Par exemple, l'ensemble
$c_{ik} = \begin{cases} 0, pour k \leq d_i \\ w_i, pour k > d_i \end{cases}$
La solution du problème des transports prévoit une borne inférieure pour $1//\sum_{i=1}^{n} w_i T_i$.
Cette technique englobant est appliquée à l'ensemble des tâches non planifiés à chaque

nœud de l'arbre. Si la borne inférieure est plus grande que la solution de l'ordonnancement connu, alors le nœud peut être éliminé[23,21].

3.3.3 problème $1//\sum_{i=1}^{n} w_i u_i$

Étant donné n tâches T_i avec $i = 1, ..., n$ avec le temps de traitement p_i et les dates échues d_i, nous devons ordonnancer ces tâches de telle sorte que $\sum_{i=1}^{n} w_i u_i$ est minimisé où $w_i \geq 0$ pour $i = 1, ..., n$. Supposons que les tâches sont énumérés selon aux date échues non décroissantes :

$$d_1 \leq d_2 \leq ... \leq d_n \tag{3.1}$$

Il existe alors un ordonnancement optimal donné par une séquence du formulaire $T_1, T_2, ..., T_s, T_{s+1}, ...T_n$, où $T_1 < T_2 < ... < T_s$ sont en temps et les tâches $T_{s+1}, ...T_n$, sont en retard. Cela peut être montré facilement en appliquant la transformation suivant. Si une tâche T_i est en retard, nous pouvons mettre T_i à la fin du l'ordonnancement sans augmenter la fonction objectif. Si T_i et T_j sont des précoces (en temps) avec $d_i \leq d_j$ tel que T_i ne soit pas programmé avant T_j, alors on peut déplacer le bloc de toutes les tâches programmées entre T_j et T_i conjointement avec T_i à gauche par le temps p_j unités et programmez T_j immédiatement après ce bloc. Parce que T_i n'étais pas en retard et $d_i \leq d_j$, cela ne crée aucun tâche tardif.

Pour résoudre le problème, nous calculons récursivement pour $t = 0, 1, ..., T := \sum_{i=1}^{n} p_i$. et $j = 1, ..., n$,la valeur minimale du critère $F_j(t)$ pour les j premiers tâches, sous la contrainte que le temps de traitement total de la ponctualité tâches est tout au plus t. Si $0 \leq t \leq d_j$ et la tâche T_j est à temps dans un ordonnancement ce qui correspond à $F_j(t)$, puis $F_j(t) = F_{j-1}(t - p_j)$. Autrement $F_j(t) = F_{j-1}(t) + w_j$. Si $t > d_j$, alors $F_j(t) = F_j(d_j)$ car toutes les tâches $T_1, T_2, ..., T_n$ sont en retard et $d_n \geq ... \geq d_1$. Ainsi, pour $j = 1, ..., n$ nous avons la récursivité

$$F_j(t) = \begin{cases} min F_{j-1}(t - p_j), F_{j-1}(t) + w_j, pour 0 < t < d_j \\ F_j(d_j), pour d_j < t < T \end{cases}$$

avec $F_j(t) = \infty$ pour $t < 0$, $j = 0, ..., n$ et $F_0(t) = 0$ pour $t \geq 0$. Notez que $F_n(d_n)$ est la solution optimale au problème.

L'algorithme suivant calcule toutes les valeurs $F_j(t)$ pour $j = 1, ..., n$ et $t = 0, ..., d_j$. Nous supposons que les tâches sont énumérés de telle sorte que (1.1) est vrai. p_{max} indique le temps de traitement le plus long. Le temps de calcul de cet algorithme est limité par $O(n \sum_{i=1}^{n} p_i)$[8].

La généralisation de ce problème avec des poids, c'est-à-dire $1//\sum_{i=1}^{n} w_i u_i$ est connu pour être NP-dur. Le cas spécial où toutes les dates échues sont égaux est équivalent au soi-disant problème de sac à dos. La date échue équivaut à la taille du sac à dos, les temps de traitement des tâches sont équivalents à les tailles des articles et les poids sont équivalents aux avantages obtenus permettre les articles dans le sac à dos. Une heuristique populaire pour ce problème est la Règle WSPT qui séquence des tâches dans l'ordre décroissant de $\frac{w_i}{p_i}$. Un pire de cas l'analyse montre que cette heuristique peut mal fonctionner arbitrairement, c'est-à-dire que le rapport $\frac{w_i u_i(WSPT)}{w_i u_i(OPT)}$ peut être arbitrairement grand[21].

3.4 Conclusion

Nous avons exposé dans ce chapitre les problèmes d'ordonnancement difficiles sur une seule machine, nous avons utilisé le problème de 3-partition et problème de programmation dynamique pour les preuves.

Chapitre 4

Heuristiques proposées

4.1 Introduction

Dans ce chapitre nous pressentons le problème $1/r_i/L_{max}$ ainsi que nous proposons quelques heuristiques pour résoudre ce problème qui est NP-Difficile

4.2 Représentation du problème $1/r_i/L_{max}$

4.2.1 Le modèle mathématique du problème

Les variables :

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

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

5) $t_k + \sum_{i=1}^{n} (p_{(i)} - d_i) x_{ik} \leqslant y \qquad y \in R$

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 empâche le démarrage d'une tâche avant sa date de disponibilité.
La 4ème contrainte impose à la tâche en position $k+1$ de commencer son exécution après la fin exécution de la tâche en position k.

La $5^{\text{ème}}$ contrainte indique que la tardiveté de chaque tâche doit être $\leqslant y$

La formulation mathématique du problème est :

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

Ce problème est une problème linéaire en nombres entiers.

4.2.2 Définition du problème

Traiter un ensemble de tâches indépendantes sans préemption ayant des dates Échues, des temps de traitement et es dates de disponibilité sont arbitraires sur une seule machine. L'objectif est de minimiser la grande tardivité.

4.3 Les heuristiques pour résoudre le problème $1/r_i/L_{max}$

4.3.1 Définition

Les heuristiques sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables. Elles se basent sur des règles simplifies 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[5].

4.3.2 Les heuristiques proposés

Dans cette session nous proposons des heuristiques lesquelles fournissent rapidement une solution réalisable proche de l'optimum pour le problème d'ordonnancement $1/r_i/L_{max}$, ou notre but est de trouver l'heuristique de bonne qualité.

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

La première phase de cette heuristique consiste à ranger les tâches selon la règle EDD en suite ordonnancer la tâche T_i ayant la plus petite date de disponibilité.

Algorithm 5 Algorithme de H1

Début
Ranger les tâches selon la règle EDD
$t_1 = r_1$
$c_1 = t_1 + p_1$
tant que $T \neq \emptyset$
faire
choisir T_i tel que
$r_i = min\{r_i/p_i = max\{p_i\}\}$
ordonnancer T_i à l'instant t
$T = T/\{T_i\}$
$c_i = t + p_i$
$l_i = c_i - d_i$
$t = c_i$
fait
$L_{max} = max\{l_i\}$
Fin

4.3.2.2 Principe de la 2ème heuristique

La première phase de cette heuristique consiste à trouver les tâches disponible à l'instant $\max\{c_{i-1}, \min\{r_i\}\}$, en suite ordonnancer la tâches T_i ayant la plus petite date échue.

Algorithm 6 Algorithme de H2

Début
$t_i = \min\{r_i\}$
Tant que$T \neq \emptyset$
faire
$T' = \{T_i \in T/r_i \leqslant t\}$
choisir $T_i \in T'$ tel que
$d_i = \min\}\{d_r\}/p_i = \max\{p_i\}$
ordonnancer T_i à l'instant t
$T = T\ \{T_i\}$
$c_i = t + p_i$
$l_i = c_i - d_i$
$t = \max\{c_i, \min\{r_i\}\}$
fait
$L_{max} = max\{l_i\}$
Fin

4.3.2.3 Principe de la 3ème heuristique

La première phase de cette heuristique consiste à ranger les dates de disponibilité et les dates échues par ordre croissant dans une liste S, en suite ordonnancer les tâches selon la liste S tel que T_i n'est pas ordonnancer.

Algorithm 7 Algorithme de H3

Début
Ranger les dates de disponibilité et les dates échues par ordre croissante dans une liste S
$t_1 = 0$
$c_1 = r_1 + p_1$
Tant que $T \neq \emptyset$
ordonnancer la tâche T_i à l'instante t
$T = T/\{T_i\}$
$c_i = t + p_i$
$l_i = c_i - d_i$
$t = c_i$
fait
$L_{max} = max\{l_i\}$
fin

4.3.3 Exemple d'application

Dans cette session nous désirons traiter 12 tâches indépendantes sur une machine dont les temps de traitement, les dates échues et les dates de disponibilité sont donnes dans le tableau suivant :

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	4	3	3	4	2	2	2	1	1	1	3	2
d_i	7	4	5	2	6	8	3	10	12	7	5	4
r_i	0	3	2	2	5	7	3	5	4	8	9	11

Nous appliquons cet exemple sur les trois heuristique proposées précédemment.

4.3.3.1 Application de la 1ère heuristique

Nous appliquons notre heuristique $H1$: Ranger les tâches suivantes par la règle EDD : $T_4T_7T_2T_{12}T_3T_{11}T_5T_1T_{10}T_6T_8T_9$

- **Itération 0** :
  $t_1 = r_4 = 2 \rightarrow \quad F = \{T_4\} \rightarrow \quad c_1 = t_1 + p_4 = 6 \rightarrow \quad T = T/T_4.$
- **Itération 01** :
  $t_2 = c_1 = 6 \rightarrow \quad F = \{T_4, T_7\} \rightarrow \quad c_2 = t_2 + p_7 = 8 \rightarrow \quad T = T/T_7.$
- **Itération 02** :
  $t_3 = c_2 = 8 \rightarrow$ on a $d_1 = d_{12}$ alors $min\{r_2, r_{12}\} = min\{3, 11\} = 3$ donc $F = \{T_4, T_7, T_2\} \rightarrow \quad c_3 = t_3 + p_2 = 11 \rightarrow \quad T = T/T_2.$
- **Itération 03** :
  $t_4 = c_3 = 11 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}\} \rightarrow \quad c_4 = t_4 + p_{12} = 13 \rightarrow \quad T = T/T_{12}.$
- **Itération 04** :
  $t_5 = c_4 = 13 \rightarrow$ on a $d_3 = d_{11}$ alors $min\{r_3, r_{11}\} = min\{2, 9\} = 2$ donc $F = \{T_4, T_7, T_2, T_{12}, T_3\} \rightarrow \quad c_5 = t_5 + p_3 = 16 \rightarrow \quad T = T/T_3.$
- **Itération 05** :
  $t_6 = c_5 = 16 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}\} \rightarrow \quad c_6 = t_6 + p_{11} = 19 \rightarrow \quad T = T/T_{11}.$

- **Itération 06** :
  $t_7 = c_6 = 19 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5\} \rightarrow \quad c_7 = t_7 + p_5 = 21 \rightarrow \quad T = T/T_5$.
- **Itération 07** :
  $t_8 = c_7 = 21 \rightarrow$ on a $d_1 = d_{10}$ alors $min\{r_1, r_{10}\} = min\{0, 8\} = 0$ donc $F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_1\} \rightarrow \quad c_8 = t_8 + p_1 = 25 \rightarrow \quad T = T/T_1$.
- **Itération 08** :
  $t_9 = c_8 = 25 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_1, T_{10}\} \rightarrow \quad c_9 = t_9 + p_{10} = 26 \rightarrow T = T/T_{10}$.
- **Itération 09** :
  $t_{10} = c_9 = 26 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_1, T_{10}, T_6\} \rightarrow \quad c_{10} = t_{10} + p_6 = 28 \rightarrow \quad T = T/T_6$.
- **Itération 10** :
  $t_{11} = c_{10} = 28 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_1, T_{10}, T_6, T_8\} \rightarrow \quad c_{11} = t_{11} + p_8 = 29 \rightarrow \quad T = T/T_8$.
- **Itération 11** :
  $t_{12} = c_{11} = 29 \rightarrow \quad F = \{T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_1, T_{10}, T_6, T_8, T_9\} \rightarrow \quad c_{18} = t_{12} + p_9 = 30 \rightarrow \quad T = T/T_9 = \emptyset$ stop.

Nous avons trouvé la solution présenté par la figure 4.1 avec $L_{max} = 20$

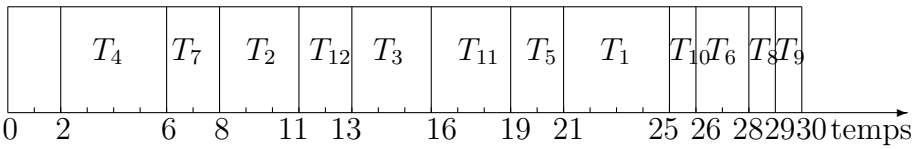


FIGURE 4.1: Solution générale de l'exemple par l'heuristique $H1$

4.3.3.2 Application de la $2^{\text{ème}}$ heuristique

Nous appliquons notre heuristique $H2$:

- **Itération 01** :
  $t_1 = 0 \rightarrow T = \{T_1\} \rightarrow F = \{T_1\}$
- **Itération 02** :
  $t_2 = max\{2, 4\} = 4 \rightarrow T = \{T_2, T_3, T_4, T_7, T_9\}$ on a $min\{d_i\} = d_4$ donc $F = \{T_1, T_4\}$
- **Itération 03** :
  $t_3 = max\{2, 8\} = 8 \rightarrow T = \{T_2, T_3, T_5, T_6, T_7, T_8, T_9T_{10}\}$ on a $min\{d_i\} = d_7$ donc $F = \{T_1, T_4, T_7\}$
- **Itération 04** :
  $t_4 = max\{2, 10\} = 10 \rightarrow T = \{T_2, T_3, T_5, T_6, T_8, T_9T_{10}, T_{11}\}$ on a $min\{d_i\} = d_2$ donc $F = \{T_1, T_4, T_7, T_2\}$

- **Itération 05** :
  $t_5 = max\{2, 13\} = 13 \rightarrow T = \{T_3, T_5, T_6, T_8, T_9T_{10}, T_{11}, T_{12}\}$ on a $min\{d_i\} = d_{12}$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}\}$
- **Itération 06** :
  $t_6 = max\{2, 15\} = 15 \rightarrow T = \{T_3, T_5, T_6, T_8, T_9T_{10}, T_{11}\}$ on a $min\{d_i\} = d_3$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3\}$
- **Itération 07** :
  $t_7 = max\{3, 18\} = 18 \rightarrow T = \{T_5, T_6, T_8, T_9, T_{10}, T_{11}\}$ on a $min\{d_i\} = d_{11}$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}\}$
- **Itération 08** :
  $t_8 = max\{4, 21\} = 21 \rightarrow T = \{T_5, T_6, T_8, T_9, T_{10}\}$ on a $min\{d_i\} = d_5$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5\}$
- **Itération 09** :
  $t_9 = max\{4, 23\} = 23 \rightarrow T = \{T_6, T_8, T_9, T_{10}\}$ on a $min\{d_i\} = d_{10}$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_{10}\}$
- **Itération 10** :
  $t_{10} = max\{4, 24\} = 24 \rightarrow T = \{T_6, T_8, T_9\}$ on a $min\{d_i\} = d_6$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_{10}, T_6\}$
- **Itération 11** :
  $t_{11} = max\{4, 26\} = 26 \rightarrow T = \{T_8, T_9\}$ on a $min\{d_i\} = d_8$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_{10}, T_6, T_8\}$
- **Itération 12** :
  $t_{12} = max\{4, 27\} = 27 \rightarrow T = \{T_9\}$ on a $min\{d_i\} = d_9$ donc $F = \{T_1, T_4, T_7, T_2, T_{12}, T_3, T_{11}, T_5, T_{10}, T_6, T_8, T_9\}$
- **Itération 13** :
  $t_13 = 29 \rightarrow T = \{\emptyset\}$ .....stop

Nous avons trouvé la solution présenté par la figure 4.2 avec $L_{max} = 18$

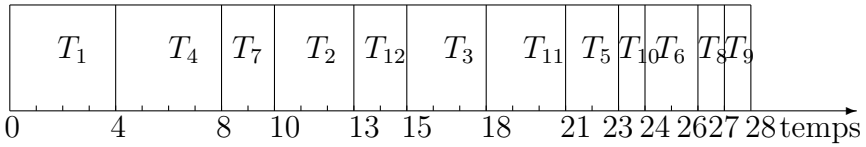


FIGURE 4.2: Solution générale de l'exemple par l'heuristique $H2$

4.3.3.3 Application de la $3^{\text{ème}}$ heuristique

Sur le même exemple, nous appliquons notre heuristique $H3$:
Ranger les tâches suivants selon r_i et d_i : $S = \{r_1, r_3, r_4, d_4, r_2, r_7, d_7, r_9, d_2, d_{12}, d_3, r_5, r_8, d_{11}, d_5, d_1, r_6, d_{10}d_6, d_{10}, r_{11}, r_8, r_{12}, d_9\}$

- **Itération 00** :
  $t_0 = 0 \rightarrow c_1 = 0 + 4 = 4 \rightarrow F = \{T_1\}$ donc $T = T/T_1 = \{T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$
- **Itération 01** :
  $t_1 = 4 \rightarrow c_3 = 4 + 3 = 7 \rightarrow F = \{T_1, T_3\}$ donc $T = T/T_1 = \{T_2, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$.
- **Itération 02** :
  $t_2 = 7 \rightarrow c_4 = 7 + 4 = 11 \rightarrow F = \{T_1, T_3, T_4\}$ donc$T = T/T_4 = \{T_2, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$.
- **Itération 03** :
  $t_3 = 11 \rightarrow c_2 = 11 + 3 = 14 \rightarrow F = \{T_1, T_3, T_4, T_2\}$ donc$T = T/T_2 = \{T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$.
- **Itération 04** :
  $t_4 = 14 \rightarrow c_7 = 14 + 2 = 16 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7\}$ donc$T = T/T_7 = \{T_5, T_6, T_8, T_9, T_{10}, T_{11}, T_{12}\}$.
- **Itération 05** :
  $t_5 = 16 \rightarrow c_9 = 16 + 1 = 17 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9\}$ donc$T = T/T_9 = \{T_5, T_6, T_8, T_{10}, T_{11}, T_{12}\}$.
- **Itération 06** :
  $t_6 = 17 \rightarrow c_{12} = 17 + 2 = 19 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}\}$ donc$T = T/T_{12} = \{T_5, T_6, T_8, T_{10}, T_{11}\}$.
- **Itération 07** :
  $t_7 = 19 \rightarrow c_5 = 19 + 2 = 21 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}, T_5\}$ donc$T = T/T_5 = \{T_6, T_8, T_{10}, T_{11}\}$.
- **Itération 08** :
  $t_8 = 19 \rightarrow c_8 = 19 + 2 = 21 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}, T_5, T_8\}$ donc$T = T/T_8 = \{T_6, T_{10}, T_{11}\}$.
- **Itération 09** :
  $t_9 = 21 \rightarrow c_{11} = 21 + 1 = 22 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}, T_5, T_8, T_{11}\}$ donc$T = T/T_{11} = \{T_6, T_{10}\}$.
- **Itération 10** :
  $t_{10} = 21 \rightarrow c_6 = 22 + 3 = 25 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}, T_5, T_8, T_{11}, T_6\}$ donc$T = T/T_6 = \{T_{10}\}$.

- **Itération 11** :
  $t_{11} = 25 \rightarrow c_{10} = 25 + 2 = 27 \rightarrow F = \{T_1, T_3, T_4, T_2, T_7, T_9, T_{12}, T_5, T_8, T_{11}, T_6, T_{10}\}$
  donc$T = T/T_{10} = \{\}$.

Nous avons trouvé la solution présenté par la figure 4.3 avec $L_{max} = 21$

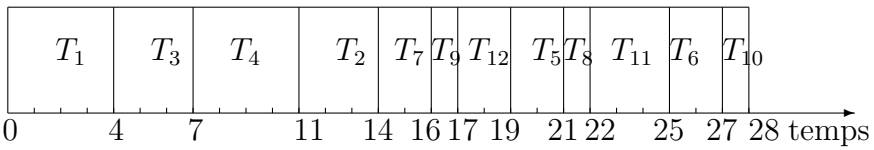


FIGURE 4.3: Solution générale de l'exemple par l'heuristique $H3$

4.4 Comparaison entre les heuristiques proposées

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

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

- $L_{max}(H)$: est la grande tardivité obtenue par l'heuristique proposée H.
- Q : est la borne inférieur du problème $1/r_i/L_{max}$ avec :

$$Q = \sum_{i=1}^{n} p_i - d_{max}$$

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

Calculons alors la borne inférieur de notre exemple :

$$Q = \sum_{i=1}^{n} p_i - d_{max} = 28 - 12 = 16$$

Pour l'heuristique $H1$

On a $L_{max} = 20$ et donc $R = \frac{L_{max}(H_1)}{Q} = \frac{20}{16} = 1.25$
Alors cette heuristique est de bonne qualité car $R = 1.25$ est proche de 1.

Pour l'heuristique $H2$
On a $L_{max} = 18$ et donc $R = \frac{L_{max}(H_2)}{Q} = \frac{18}{16} = 1.125$
Alors cette heuristique est de bonne qualité car $R = 1.125$ est proche de 1.
Pour l'heuristique $H3$
On a $L_{max} = 21$ et donc $R = \frac{L_{max}(H_3)}{Q} = \frac{21}{16} = 1.3125$
Alors cette heuristique est de bonne qualité car $R = 1.3125$ est proche de 1.

A partir des résultats obtenues, nous remarquons que l'heuristique $H2$ donne la meilleur valeur de la fonction objectif car la valeur de L_{max} obtenue est broche à la valeur de la borne Q, donc le ratio d'approximation R est proche de 1, par contre les heuristiques $H1$ et $H3$ donnent des valeurs plus grand de L_{max}.
Alors pour cet exemple, l'heuristique $H2$ est plus performant des heuristiques $H1$ et $H3$.

Remarque 4.1. *Nous avons traité les trois heuristiques sur un seul exemple, nous avons remarqué que l'heuristique $H2$ donne de meilleur solution que les heuristiques $H1$ et $H3$ pour cet exemple, mais pas forcement pour d'autres exemples.*

4.5 Conclusion

Nous traitons dans ce chapitre le problème d'ordonnancement $1/r_i/L_{max}$,nous avons commence par le modèle mathématique et la définition de ce problème ainsi nous proposons trois heuristiques pour trouver une solution approchée de ce dernier

Conclusion Générale

Les problèmes d'ordonnancement sont présents dans tous les secteurs de l'économie et constituent une fonction important en gestion de production. Un problème d'ordonnancement consiste à allouer dans le temps des tâches à des ressources qui existent en quantité limité, tout en satisfaisant un ensemble de contrainte.

Les problèmes d'ordonnancement avec date échue sur une seule machine est l'un des problèmes d'ordonnancement intensivement étudiés. Dans ce travail, nous avons traité ces problèmes, les types de ces derniers selon leurs complexités et d'application des heuristiques proposent 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, ainsi que nous discutons sur quelques concepts de la théorie de complexité.

Nous nous somme intéressé dans la deuxième et la troisième partie à étudier quelques problèmes d'ordonnancement sur une seule machine qui peuvent être classés comme des problèmes faciles et difficiles, nous avons commencer par les problèmes faciles, nous avons exposé quelques problèmes et nous avons donné les algorithmes pour leurs résolution, ainsi que nous avons utilisé le problème de 3-partition et les problèmes de programmation dynamique pour prouver la difficulté de certains problèmes.

Dans la quatrième partie, nous avons proposé trois heuristiques pour résoudre le problème $1/r_i/L_{max}$ qui est NP-difficile , tout d'abord nous avons représenté ce problème par leur modèle mathématique et nous avons donné sa définition, la fin de cette partie est consacré a appliquer ces trois heuristiques sur un exemple donné pour trouver une solution satisfaite.

Notre perspective est d'implémenter les trois heuristiques proposées sur un logiciel comme par exemple Matlab pour bien visionner les résultats.

Bibliographie

[1] **A.BENKACI**,Note de cours «*Problèmes d'optimisation combinatoire* » Pour M2 Ro université de médéa ,2016 .

[2] **A.BEY MASTAPHA et K.BOUKEDAKDJI** «*Problème de minimisation de la longueur d'ordonnancement sur machines parallèles identique*» ,Mémoire de fin de master , université de médéa ,2018-2019.

[3] **A.BITAR** , «*Ordonnancement sur machines parallèles appliqué a la fabrication de semi-conducteurs :ateliers de photolithographie* » ,École national supérieure des mines de saint, 2015.

[4] **A.BOUCHOU**,Note de cours «*Modélisation*» , Pour M2 Ro université de médéa,2020.mémoire de docteur

[5] **H.BOUKEF**, «*Sur l'ordonnancent d'ateliers job-shop flexibles et flow-shop en industries phormaceutiques : optimisation par algorithme génétique et essaims particulaires* » mémoire Pour obtenir le grade de docteur, université de TINIS EL MANAR , 2011.

[6] **M.BOUMEDIENE**, «*Les problèmes d'ordonnancement sur une seule machine parallèles des tâches dépendent*» ,Mémoire de magister,université de SAAD DAHLAB de blida, 2016 .

[7] **M.BOURZIK**, «*Problèmes d'ordonnancement d'atelier*» , Mémoire de licence,université SIDI BEN ABDELLAH ,juin 2016.

[8] **P.BRUCKER**, «*Scheduling algorithms*» , Fifth edition spring, 2006.

[9] **P.BRUCKER**, «*Complex scheduling*» ,édition Spring, 2006.

[10] **A,CHERGUI ,A.DAHMANI**, «*Ordonnancement d'un flow-shop par metaheuristique hybride* » , Mémoire de fin de master,université ABOU BEKR BELKAID de tilamcan, 2017.

[11] **R.CHRÉTIENNE**, «*Ordonnancement sur une machine*» ,cour MOP spécialité IAD, 2015.

[12] **B.FORTZ**, «*Modèles mathématiques et algorithmes pour l'ordonnancement* » , INFO-F-425,2008-2009.

[13] **A.GARA-ALI**, «*Ordonnancement de tâche et périodes d'indisponibilité du durée variable* » mémoire Pour obtenir le grade de docteur ,université GRENOBLE ALPE.17 aout 2016.

[14] **GOTHA** «*Les problèmes d'ordonnancement*»,RAIRO recherche opérationnelle tome 27,n01,1993.

[15] **JEFFREY W. HERRMAN et al**, «*Handbook of Production Scheduling* »,édition Springer,2006.

[16] **C.JOSSIN**, «*Planification et ordonnancement*,J-TRAVAIL Planification-et-ordonnancement.doc,2005.

[17] **M.KEBABLA**, «*Utilisation des stratégies métaheuristique pour l'ordonnancement des ateliers de type job-shop*»,mémoires de magister,université EL-HADJ LAKHDHAR de batna,02 juillet 2008.

[18] **S.KOUIDER AIAD**, Note de cours «*Formulation des problèmes d'ordonnancement*»,pour M2 Ro université de médéa,2020.

[19] **Muminu O. Adamu**, Note de cours «*A survey of single machine scheduling to minimize weighted number of tardy jobs*»,University of Kwazulu-Natal, 2014.

[20] **NIEBERG Tim**, Note de cours «*Scheduling Single Machine Scheduling, Research Institute for Discrete Mathematics*», 2010.

[21] **M-L.PINEDO**, «*Schuding theory algorithms and systems* »,third edition spring,1994.

[22] **O.SELT**, «*Utilisation des stratégies métaheuristique pour l'ordonnancement des ateliers de type job-shop*»,mémoires de magister ,université EL-HADJ LAKHDHR de batna,2008.

[23] **H.SEMRANI et A.ANNANE**, «*Problème d'ordonnancement sur une seule machine leurs complexités et les différentes méthodes de résolution*»,mémoires de fin mastre, université de médéa ,2018-2019.

[24] **C.SOLON**, «*Réduction de problème combinatoire et optimisation par colonie de fournis*»,2006.