الجمهـــوريـــة الجزائـــريـــة الديمقـــراطيـــة الشـعبيـــة
République Algérienne Démocratique et Populaire
وزارة التـــعلـــيـــم العـــالـــي والبـــحـــث العـــلمـــي
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

جامعة وهران 2 محمد بن أ حمد
Université d'Oran 2 Mohamed Ben Ahmed

معهد الصيانة و الأمن الصناعي
Institut de Maintenance et de Sécurité Industrielle

Département de Maintenance en Instrumentation

MÉMOIRE

Pour l'obtention du diplôme de Master

Filière : Génie Industriel
Spécialité : GI

Thème

Conception d'un simulateur d'aide à l'ordonnancement des systèmes automatisés de production

Présenté et soutenu publiquement par :

ZITOUNI Laid & KELLEL Hamza

Devant le jury composé de :

Nom et Prénom	Grade	Etablissement	Qualité
Mr. Djilali NEKROUF	MCB	Univ. Oran-2, IMSI	**Président**
Mr. Abderrahim BENFEKIR	MCB	Univ. Oran-2, IMSI	**Encadreur**
Mr. Taha BENARBIA	MCB	Univ. Oran-2, IMSI	**Examinateur**

Juin 2023

Remerciements

***A**u terme de ce travail, nous devons remercier tout d'abord dieu qui nous a donné la force et le courage de suivre nos études et d'arriverà ce stade et à nos parents qui nous ont beaucoup soutenus pendent tousle long de notre parcourt.*

***U**n grand merci à mon encadreur **Mr. Abderrahim BENFEKIR** qui nous a beaucoup aidé, soutenu et nous a permis d'arriver à ce niveau-là et pour ses excellents conseils et surtout pour son temps passé avec nous et sa patience, sans lui on n'aurait pas pu réaliser ce modeste travail et pour sa confiance en nous.*

***N**ous tenons aussi à remercier les membres de jury qui nous ont fait honneur d'examiner ce travail.*

***E**nfin, nous renouvelons nos remerciements à ceux qui nous ont aidés de près ou de loin pour réaliser ce travail sans oublier les enseignants qui ont contribués à notre formation.*

Dédicace

Merci mon dieu de m'avoir donné la capacité d'écrire et de réfléchir, la patience d'aller jusqu'au bout du rêve J'ai l''honneur de dédier ce modeste travail :

A ma mère celle qui m'a donné la vie, le symbole de tendresse qui s'estsacrifiée pour mon bonheur et ma réussite.

A mon père, école de mon enfonce, qui a été mon ombre durant toutes les années des études et qui a veillé tout au long de ma vie à m'encourager à me donner l'aide et à me protéger.

A mes frères et à mes sœurs, je vous souhaite un avenir plein de joie de bonheur de réussite et de sérénités, je vous exprime à travers ce travail mes sentiments de fraternité et d'amour.

*A Ma petite princesse **MARAM***

A tous les membres de ma famille, petite et grande.

*A mon encadreur **Mr. Abderrahim BENFEKIR**.*

A mes chers ami(e)s : Radouane, Sofiane, Ben_dhiba, SI Ali…..

A tous ceux qui ont sacrifié leurs temps pour la science et à tous ceux qui utilisent la science pour le bien et la prospérité de l'humanité.

Hamza kellel

Dédicace

A la mémoire de ma chère grand-mère mima « Jamila » qui nous a quittétrès tôt je dédie ce travail a son angélique âme et à mon grand-père paternel que dieu les accueille dans son propre paradis.

A mon petit frère Djamel.

A toute ma chère nombreuse famille ZITOUNI.

A mon chère ami 'Sidiadda Bessafi' pour sa sincère amitié,

« je te souhaite un avenir éclaircissant plein de joie santé et réussite ».

*A mes deux petite sœur '**Rayen**' et '**kawter**'.*

A ma source de bonheur « mes chers parents LAKHDER et AICHA » en particulier, Leurs prières et leurs conseils m'ont toujours accompagné. Je lesremercié pour tous ce que m'ont donné (Confiance, moyens et amour) en dirigeant et en mettant tout ce qui est nécessaire pour fournir la bonne atmosphère pour compléter mon travail, et leur souhaite une longue et heureuse vie pleine de joie, santé et honneur...

LAID ZITOUNI

Résumé

Le présent projet de fin d'études (PFE) se concentre sur résolution des problèmes d'ordonnancement d'atelier, une tâche cruciale dans la gestion de la production industrielle. L'ordonnancement d'atelier vise à déterminer la séquence d'exécution des différentes tâches de production dans un environnement manufacturier afin de maximiser l'efficacité, de réduire les coûts et de respecter les contraintes temporelles, afin de résoudre ces problèmes nous pouvons les résoudre de différentes méthodes où nous pouvons les regroupe de résolutions de problèmes d'ordonnancement d'atelier NP-difficiles en deux classes principales : la classe de méthodes exactes et la classe des méthodes approchées distinguent deux types de méthodes : les heuristiques et Métaheuristiques. La résolution de tels problèmes nécessite des méthodes dédiées, tandis que les méthodes exactes ne peuvent pas résoudre ces types de problèmes vu le temps de calcul énorme, les heuristiques offrent la possibilité de trouver une solution réalisable en un temps raisonnable. Dans ce travail nous avons simulé les méthodes de résolution des problèmes d'ordonnancement d'atelier par **MATLAB** pour afficher le temps optimal et leurs graphes avec concevoir un simulateur développé de cette méthodes permettre de résoudre efficacement les problèmes d'ordonnancement d'atelier complexes dans un court laps de temps.

Mots clés : Ordonnancement, Job Shop, flow shop, heuristique, Métaheuristiques, tache.

Abstract

This graduation project (PFE) focuses on solving shop scheduling problems, a crucial task in industrial production management. Shop scheduling aims to determine the sequence of execution of different production tasks in a manufacturing environment in order to maximize efficiency, reduce costs and meet time constraints, in order to solve these problems, we can solve them of different methods where we can group them for solving NP-hard workshop scheduling problems into two main classes: the class of exact methods and the class of approximate methods distinguish two types of methods: heuristics and metaheuristics. The resolution of such problems requires dedicated methods, while exact methods cannot solve these types of problems due to the enormous computation time, heuristics offer the possibility of finding a feasible solution in a reasonable time. In this work we simulated the methods of solving shop scheduling problems by MATLAB to display the optimal time and their graphs with designing a simulator developed from this method to effectively solve complex shop scheduling problems in a short time.

Keywords: Scheduling, Job Shop, flow shop, heuristic, metaheuristic, task.

Sommaire

Chapitre 3 : *Conceptions de l'outil de simulation*

Liste des figures

Liste des tableaux

Introduction générale

Introduction

Les systèmes automatisés de production occupent une place de plus en plus prépondérante dans l'industrie, l'efficacité de leur ordonnancement revêt une importance capitale. En effet, un ordonnancement optimal permet d'optimiser l'utilisation des ressources, de minimise le temps et les coûts de production et d'améliorer la satisfaction des clients. Dans cette optique, la conception d'un simulateur d'aide à l'ordonnancement se présente comme une solution prometteuse pour atteindre ces objectifs.

L'ordonnancement des systèmes automatisés de production consiste à déterminer la séquence d'exécution des tâches à effectuer, ainsi que l'affectation des ressources nécessaires à leur réalisation. Cette tâche complexe est influencée par de nombreux paramètres tels que les contraintes de temps, les capacités des machines, les contraintes de stockage, les priorités des tâches, etc. Les méthodes traditionnelles d'ordonnancement reposent souvent sur des heuristiques ou des règles prédéfinies, qui ne tiennent pas compte de manière exhaustive de toutes les variables et interactions du système.

La conception d'un simulateur d'aide à l'ordonnancement prend tout son sens. Ce simulateur (**MATLAB**) vise à modéliser le système automatisé de production dans son ensemble, en prenant en compte tous les éléments pertinents, tels que les machines, les ressources, les flux de production, les contraintes de temps, etc. Il permet ainsi de simuler différentes configurations d'ordonnancement, d'évaluer leur performance et d'identifier les solutions les plus efficientes.

Dans ce travail nous sommes intéressés à la conception d'un simulateur d'aide à l'ordonnancement des systèmes automatisés de production, qui conduit à l'étude de planification des systèmes de production adaptés aux différentes classifications d'enchaînements des lignes de production industrielles [14]), ceci dans l'esprit de l'automatisation flexibles. Les travaux que nous avons traités dans ce thème ont principalement concerné l'exposition des déférentes méthodes d'ordonnancements d'ateliers la programmation, la simulation et l'élaboration d'un outil de simulation pour

des systèmes de production. L'accent sera mis notamment sur la possibilité de proposé un simulateur d'aide pour l'ordonnancement des chaines de production à la phase de conception d'une unité industrielle.

Alors, après une introduction générale notre mémoire est devisé en trois chapitres. Les deux premiers chapitres sont conçus essentiellement pour déterminer le cadre générale du travail et pour définir et positionner le problème d'ordonnancement à étudier. Par contre le troisième chapitre a pour objectif de décrire l'approche proposée.

Dans **Le premier chapitre** on a donné une généralité sur les problèmes d'ordonnancement (ses notions de base, leurs différentes classes et les méthodes utilisées pour les résoudre). Nous abordons ensuite le problème d'ordonnancement en temps réel et les deux types d'approche trouvés dans la littérature pour résoudre ce type de problème.

Le deuxième chapitre on a définies les types de problèmes d'ordonnancement d'atelier pour chaque types nous avons donnez deux méthodes avec des exemples par chaque types pour les résoudre. L'objectif du **troisième chapitre** est de simuler des méthodes de problèmes d'ordonnancement d'atelier par un logiciel de simulation **MATLAB** avec affichage du résultat final et leurs plots et Nous avons développé une application sur MATLAB permettant de résoudre le problème d'ordonnancement d'atelier avec une interface .

Enfin, ce travail sera complété par une conclusion générale à travers laquelle on exposera les principaux points retirés et on donnera les perspectives à envisager comme suite à ce travail.

Chapitre1 : Généralités sur l'ordonnancement

I.1 Introduction :

L'ordonnancement est un domaine essentiel de la gestion des opérations et de la planification. Il concerne la manière dont les tâches ou les ressources sont planifiées, organisées et exécutées dans le but d'optimiser l'efficacité, la productivité et les performances globales d'un système.

L'objectif principal de l'ordonnancement est d'affecter les tâches aux ressources disponibles de manière à minimiser les temps d'attente, les coûts ou d'autres critères de performance spécifiques. Il concurrents.

Ce chapitre sur les généralités de l'ordonnancement et introduire quelques notion sur les problèmes d'ordonnancement dans les systèmes de production, Nous rappelons d'abord les différents paramètres et classifications dans tel problème, puis nous expliquons sur la théorie de la complexité, ensuiteles principes généraux des méthodes de résolution approchée et exacte

I.2 Définition de l'ordonnancement :

L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de la production qui vise à améliorer l'efficacité d'une entreprise en termes de coûts de production et de délais de livraison. [1]

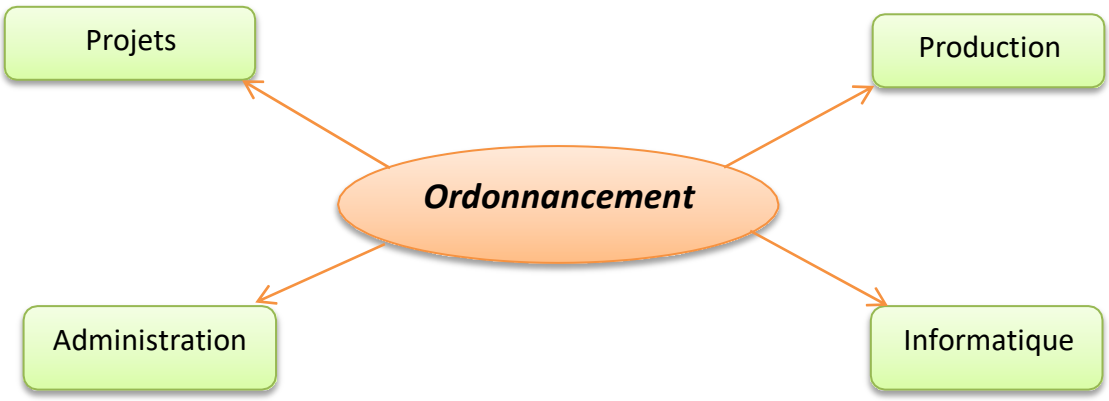
Selon [2], « Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. ». Présentons une autre définition qui est plus explicite : « Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps, et vise à satisfaire un ou plusieurs objectifs. Plus précisément, on parle de problème d'ordonnancement lorsqu'on doit déterminer les dates de début et les dates de fin des tâches, alors qu'on réserve le terme de problème de séquencement au cas où l'on cherche seulement à fixer un ordre relatif entre les tâches qui peuvent être en conflit pour l'utilisation des ressources. Un ordonnancement induit nécessairement un ensemble unique de relations de séquencement. »

L'ordonnancement se déroule en trois étapes qui sont:

- la planification qui vise à déterminer les différentes opérations à réaliser, les dates correspondantes et les moyens matériels et humains à y affecter.
- l'exécution qui consiste à mettre en œuvre les différentes opérations définies dans la phase de planification.

La Figure (**1.1**) suivante présenter les Domaine concerné par l'ordonnancement :

Gestion de projets… Ateliers de production...



Gestion des ressources humaine
emplois du temps

Exécution des programmes
optimisation de code…

Figure 1.1 Domaine concerné par l'ordonnancement

I.3. Formulation d'un problème d'ordonnancement :

Il est possible de rencontrer des problèmes d'ordonnancement dans plusieurs domaines tel quel'informatique avec la gestion des tâches dans les processeurs ; dans l'industrie avec la gestion de la production et les chaines de distribution, mais aussi dans d'autres domaines tel que la construction ou l'administration.

Les différentes données d'un problème d'ordonnancement peuvent varier d'un domaine à un autre. Dans les systèmes de production elles sont groupées en tâches, ressources, contraintes, etobjectifs ou critères d'optimisation.

I.3.1. Les tâches :

C'est l'ensemble des opérations requises pour la fabrication d'un produit. Une tâche, « *Job* » en anglais, J_i est une unité fondamentale présentée dans le temps par une date de début et une date de fin ainsi qu'un temps d'exécution sur une ou plusieurs machines [3].

Chaque tâche i où $i = \{0, 1, 2, \dots, N-1\}$, où N est le nombre de tâches, est composée d'une ou plusieurs d'opérations notées $\{O_{i,0}, O_{i,1}, \dots, O_{i,j}\}$, $j = \{0, 1, \dots, N_i - 1\}$. Où $O_{i,}$ est la $j^{ème}$ opération du job i et N_i le nombre total d'opérations requises pour achever le job i (peut être aussi dénotée m_{ij}). Chaque tâche $O_{i,}$ doit être exécutée sur une machine où $m_{i,}$ représente son numéro d'identification pendant un certain temps p_{ij}.

Une tâche, comme présenté dans la Figure 1.2, possède une date d'arrivée r_i (aucune

opérationde la tâche i ne peut être exécutée avant la date r_i), un temps d'exécution p_i (Équation 1.1), et une date d'échéance d_i. La tâche doit se terminer avant cette date sinon une pénalité peut être appliquée. Une tâche possède aussi une date de fin d'exécution C_i, qui correspond au moment de fin de toutes les opérations de cette tâche. Il existe aussi un autre attribut pouvant être associéà une

tâche ou à une opération appelé « *setup-time* » ou temps de préparation noté s_{ijk}, i pour la tâche, j pour l'opération si le setup-time concerne celle-ci et k pour la machine. Il représentele temps requis par une machine pour procéder à l'exécution d'un certain type de tâches/opérations.

La Figure (**1.2**) suivante les taches :

$$= \sum_{l=0}^{Ni-1} p_{ij} \qquad (1.1)$$

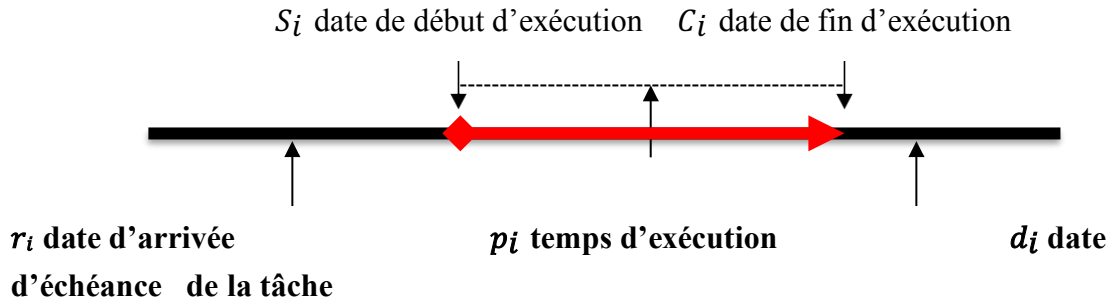


Figure 1.2 Exemple d'une tâche

Il est aussi possible d'associer une priorité d'exécution à chaque tâche afin de répondre aux besoins des clients ou à cause de la nature du produit, si le produit ou la matière première utiliséesont périssables par exemple. A ces tâches on affecte des poids w_i relatifs à leurs degrés de priorité.

On distingue deux types de tâches :

- **Les tâches morcelables :** ou on parle de préemption, qui peuvent être exécutées en plusieurs fois, les opérations pouvant ainsi être stoppées et reprisent plus tard, ce qui facilite la résolution de certains problèmes.
- **Les tâches non morcelables** : ou indivisibles qui doivent être exécutées en une seule foiset ne peuvent être interrompues qu'une fois une des opérations soient terminées.

Pour certains types de tâches on peut parler de « *gamme de fabrication* ». C'est-à-dire, afin ded'achever un produit (tâche/job) celui-ci doit passer par un certain nombre de machines définisau préalable. Il existe trois types de gammes :

1-Gamme linéaire : l'ordre des opérations est déterminé et doit être respecté.

2- Gamme libre : l'ordre est totalement libre mais toutes les opérations doivent êtreachevées.

3- Gamme mixte : est un mélange des gammes linéaire et libre. C'est-à-dire que certaines opérations doivent respecter un ordre de passage bien spécifique et d'autres non.

I.3.2. Les ressources :

L'exécution des tâches ou opérations nécessite la mise en place et l'affectation d'un ensemble de moyens techniques et/ou de ressources humaines indispensables à la réalisation de ces tâches/opérations durant les intervalles de disponibilité.

Une ressource (machine) M_k où $k = \{0, 1, 2, \dots, M-1\}$, avec M désignant le nombre de machines, peut-être physique ou virtuelle, et d'une capacité limitée ou virtuellement illimitée, permettant l'exécution d'une opération (ou une tâche). On distingue plusieurs types de ressources [1] :

- **Ressources renouvelables**, sont disponible et peuvent être réutilisées après être allouéesà une tâche comme les machines ou le personnel,
- **Ressources consommables**, peuvent être épuisées après l'exécution d'une tâche oud'une opération, par exemple la matière première ou les ressources financières,
- **Ressources partageables,** sont des ressources pouvant être partagées entre plusieurstâches/opérations.

Une autre classification des ressources existe aussi dans la littérature [5] :

- **Ressources disjonctives**, ne peuvent exécuter qu'une seule tâche à la fois. Les autres tâches ayant besoin de cette ressource doivent attendre dans une file d'attente la fin de la tâche en cours,
- **Ressources cumulatives**, contrairement aux disjonctives, peuvent exécuter plusieurs tâches en parallèle (deux ou plus).

Une décomposition de type ressource est illustré par la figure (1.3).

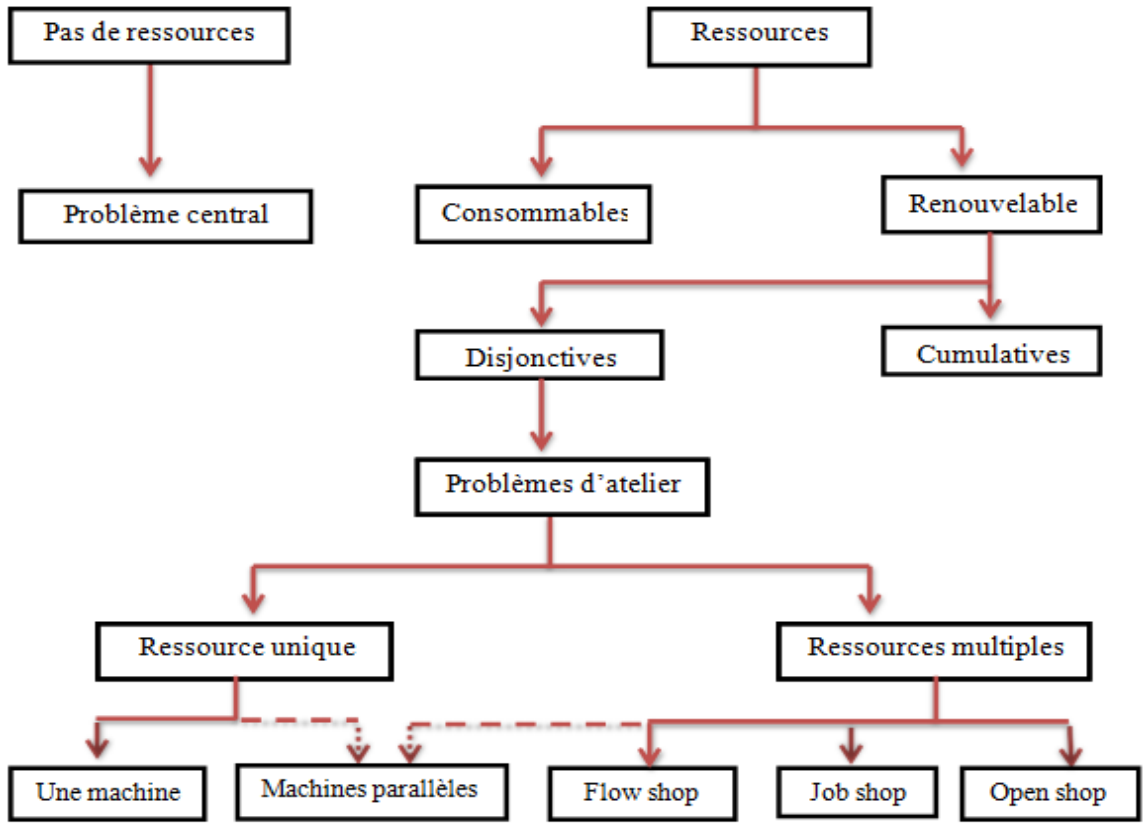


Figure1.3 Typologie des problèmes d'ordonnancement par les ressources [2].

Dans certains problèmes d'ordonnancement, on trouve la notion « *d'état de la ressource* » quidoit être prise en considération. Un changement d'état de la ressource, de fonctionnel à défaillant ou en maintenance, peut survenir et devrait être pris en considération lors de la mise en place de d'un ordonnancement. On parle d'ordonnancement dynamique lorsque ces types d'évènements sont pris en considération.

I.3.3. Les contraintes :

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision. On distingue deux types de contraintes, les contraintes liées au tempset celles liées aux ressources [5]–[6].

- **Contraintes temporelles :**

Les contraintes de temps sont généralement liées à l'intervalle de temps pour l'exécution d'unetâche ou d'une opération, c'est-à-dire que l'exécution d'une tâche ne doit pas commencer avantune date r_i et se terminer avant une date d_i

- **Contraintes technologiques :**

Le deuxième type correspond aux contraintes technologiques,en général décrites dans les gammes de fabrication des produits.

- **Contraintes d'enchaînement :**

Nous qualifions de contrainte d'enchaînement ou desuccession, une contrainte qui lie le début ou la fin de deux activités par une relation linéaire. Ce sont des contraintes imposées généralement par la cohérence technologique (les gammesopératoires dans le cas d'ateliers) qui décrivent des positionnements relatifs devant êtrerespectés entre les tâches. [7]

Une contrainte temporelle peut être spécifiée ainsi (Équation 1.2) :

$$t_i \leq d_i \text{ et } r_i \leq C_i \leq d_i \qquad (1.2)$$

Où t_i (*Start time*) est l'instant où la tâche i commence son exécution effective. Dans les figures ci-dessous, on illustre un exemple d'exécution d'une tâche i avec un temps de disponibilité $r_i = 0$ et une date limite $d_i = 5$. Dans la Figure 1.4, l'ordonnancement respecte les contraintes, par contre dans la Figure (1.5), l'ordonnancement ne respecte pas les contraintes temporelles.

La Figure suivante présenter les taches :

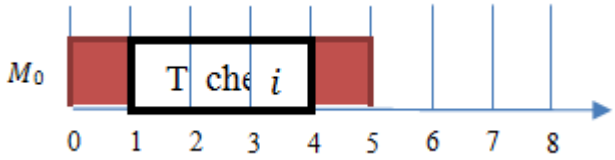


Figure 1.4 Ordonnancement réalisable.

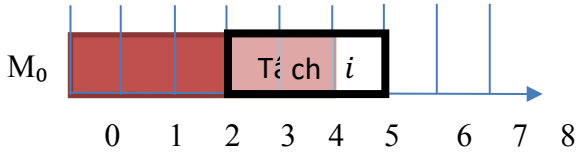


Figure 1.5 Ordonnancement non réalisable.

Parmi contraintes temporelles on peut aussi définir les relations précédence, c'est-à-dire qu'uneopération O_i doit s'exécuter avant l'opération O_j (gamme) la condition à respecter est donc datede fin d'exécution de l'opération i doit être avant la date de début de l'exécution de la tâche j (Équation 1.3).

$$C_i \leq t_j \qquad (1.3)$$

- **Contraintes des ressources :**

Ces contraintes spécifient la capacité ainsi que la disponibilité des ressources. Une ressource est renouvelable si elle est toujours disponible avec la même capacité initiale après l'exécution d'une opération. Elle est considérée comme consommable si après une exécution d'une tâche sa capacité est réduite et pouvant être par conséquent indisponible après un certain nombre d'opérations.

Une seconde contrainte consiste dans le nombre de tâches qu'une ressource peut opérer en même temps. La ressource est cumulative si elle peut faire passer plusieurs tâches à la fois et disjonctif si elle ne permet l'exécution que d'une seule et unique opération à la fois.

I.3.4. Les objectifs ou les critères d'évaluation :

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettant d'évaluer la qualité de l'ordonnancement établi [8]. Les critères que doit satisfaire un ordonnancement sont variés. D'une manière générale, on distingue plusieurs classes d'objectifs concernant un ordonnancement.

-Les objectifs liés au temps : On trouve par exemple la minimisation du temps total d'exécution, du temps moyen d'achèvement, des durées totales de réglage ou des retards par rapport aux dates de livraison.

-Les objectifs liés aux ressources : maximiser la charge d'une ressource ou minimiser le nombre de ressources nécessaires pour réaliser un ensemble de tâches sont des objectifs de ce type.

-Les objectifs liés au coût : ces objectifs sont généralement de minimiser les coûts de lancement, de production, de stockage, de transport, etc. [7]

I.4. Notation des problèmes d'ordonnancement :

Une méthode de notation qui est toujours d'actualité a été proposée par Graham et al. [9] qui permet de représenter un problème d'ordonnancement d'une manière simple et intuitive. Cette notation est composée de trois champs α, β, et γ, et est présentée sous la forme $(\alpha \mid \beta \mid \gamma)$.

I.4.1. Le champ α :

Le champ α spécifie le type de l'atelier ou le type de machines ainsi que leur nombre. Il est à son tour composé de trois éléments : α_1 , α_2 , et α_3 . α_1 peut prendre l'une des valeurs suivantes $\{o, P, Q, R, O, F, J\}$.

- $\alpha_1 = o$: l'atelier ne possède qu'une seule machine et par conséquent, les tâches (jobs) ne sont composées que d'une seule opération avec un temps d'exécution noté p_i,
- $\alpha_1 = P$: l'atelier est composé de plusieurs machines parallèles et identiques. Le temps d'exécution d'une tâche est similaire sur toutes les machines,
- $\alpha_1 = Q$: l'atelier est composé de machines parallèles uniformes. Chaque machine possède sa propre vitesse b_i (peut aussi être notée s_i). Le temps d'exécution des tâches est par conséquent redéfini. Le temps d'exécution d'une tâche j sur une machine M_i estdéfinit alors par : $P_{ij} = p_j/b_i$,
- $\alpha_1 = R$: l'atelier est composé de machines parallèles et le temps d'exécution nécessairepour achever une tâche varie d'une machine à une autre,
  - $\alpha_1 = O$: atelier de type open shop,
  - $\alpha_1 = F$: atelier de type Flow Shop,
  - $\alpha_1 = J$: atelier de type Job Shop.
  - $\alpha_2 \in \mathbb{Z} +$: définit le nombre de machines.
- $\alpha_3 = o$: le nombre de machines m est variable.

I.4.2. Le champ β :

Dans ce champ est défini les contraintes liées aux tâches et aux machines comme par exemplela contrainte de précédence entre tâches. Le champ est composé de 5 sous-champs, à savoir :

$\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$.

$\beta_1 = pmtn$: la préemption entre tâches est tolérée, c'est-à-dire que l'exécution d'une tâche peut être interrompue et reprise plus tard,

- $\beta_1 = o$: pas de préemption,
- $\beta_1 = split$: les tâches peuvent être décomposées en sous-tâches et donc être exécutéesen parallèle.
- $\beta_2 = Prec$: les opérations des tâches doivent respecter la relation de précédence,
- $\beta_2 = Tree$: la relation de précédence entre opérations est sous forme d'arbre,
- $\beta_2 = o$: la précédence n'est pas prise en considération.
- $\beta_3 = r_i$: chaque tâche i possède une date d'arrivée,
- $\beta_3 = o$: toutes les tâches arrivent à l'instant $t = 0$.
- $\beta_4 = p_i = 1$: si $\alpha_1\{o, P, Q\}$, toutes les tâches possèdent le même temps d'exécutionégal à 1,

I.4.3. Le champ γ :

Ce dernier champ indique la fonction objectif utilisée. Parmi les critères C_{max}, L_{max}, T_{max}, E_{max}, …. (cf. Section 1.2.4 sur les objectifs).

Une représentation légèrement différente de celle proposée par Graham et al. [9] existe aussidans la littérature et est suggérée par Garey & Johnson [10]. Elle est aussi composée de trois champs $(a \mid b \mid c)$:

- Dans le champ a on décrit le type d'atelier, le nombre de machines, ainsi que le nombrede tâches (jobs) à réaliser,
- Le champ b et c spécifie respectivement les contraintes et la fonction objectif commeles champs β et γ introduits par Graham et al. [9],

Exemple : soit un problème d'ordonnancement Job Shop composé de 40 tâches et 4 machines avec prise en considération des contraintes de précédence, de contraintes sur les dates d'arrivées,et les dates d'échéance. La fonction objectif est de minimiser la valeur du makespan.

Les champs a, b et c sont alors définis ainsi :

- Le champ $a = J, 40, 4$. J pour le type du problème Job Shop, 40 pour le nombre detâches et 4 le nombre de machines,
- Le champ $b = Prec, r, .$ $Prec$ est la contrainte de précédence, r_i pour les datesd'arrivée des jobs (release date) et d_i pour les dates d'échéance (due date),
- Enfin, le champ c contient la ou les critères d'optimisation. Dans cet exemple il estdéfini par C_{max} (makespan).

La représentation de ce problème est ainsi donnée par l'Équation 1.4 :

$$J, 40,4 \mid Prec, r_i, d_i \mid C_{max} \qquad (1.4)$$

I.5. Les classes d'ordonnancement :

Dans [5] et [11], les auteurs précisent que lorsqu'on tente de trouver un ordonnancement, on découvre plus d'une solution répondant aux critères sélectionnés. Par exemple, les dates de début de certaines opérations peuvent être retardées sans qu'il n'y ait d'effet sur la valeur du critère. Le but des classes d'ordonnancement (voir Figure 1.6) est de limiter la recherche à un sous-ensemble d'ordonnancements constitué de meilleures solutions, et ainsi éviter d'explorer tout l'espace de solutions et accélérer la recherche.

La Figure(**1.6**) suivante présenter les classes d'ordonnancement :

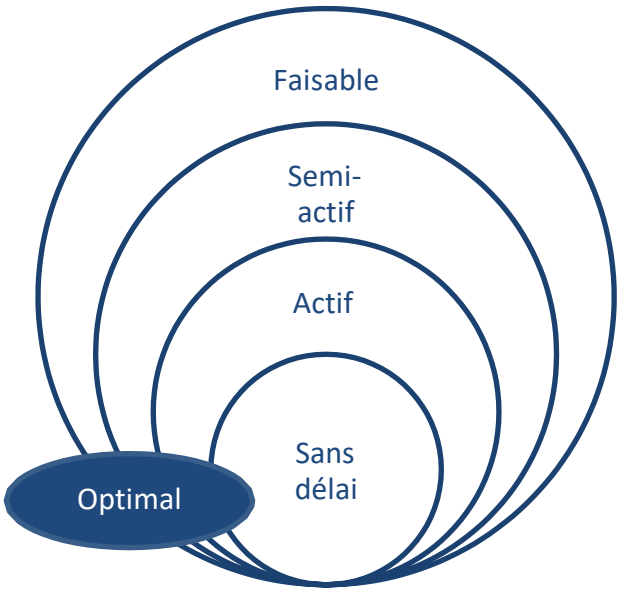


Figure 1.6 Les classes d'ordonnancemen

I.5.1. Ordonnancement faisable :

Un ordonnancement est dit faisable si et seulement si toutes les tâches respectent l'ensemble des contraintes du problème d'ordonnancement.

I.5.2. Ordonnancement semi-actif :

Un ordonnancement est semi-actif s'il n'est pas possible de commencer une opération plus tôtsans pour autant violer une contrainte ou sans changer l'ordre des opérations sur les ressources (machines). Dans ce type d'ordonnancement la seule solution pour optimiser la fonction « Objectif » est de réordonnancer toutes les tâches/opérations sur les ressources. Si un ordonnancement semi-actif (l'ordre de passage des tâches) est représenté avec un diagramme de Gantt, il est impossible de décaler une tâche à gauche.

I.5.3. Ordonnancement actif :

Un ordonnancement est actif si tout décalage oblige à retarder la date de début d'exécution d'une autre opération. Il est aussi impossible d'insérer une nouvelle opération entre de deux tâches sans remettre en cause l'ordre d'exécution ou violer certaines contraintes. Un ordonnancement actif est aussi en même temps semi-actif. Sur un diagramme de Gantt, un ordonnancement est considéré comme actif s'il est impossible de décaler une tâche à gauche.

I.5.4. Ordonnancement sans délai :

Un ordonnancement est « *sans délai* » si à chaque instant toute opération qui dispose des ressources nécessaires est commencée. C'est-à-dire qu'une machine ne doit pas rester inactive alors qu'une tâche/opération est présente dans la file d'attente de la machine. On ne doit pas retarder l'exécution d'une tâche si celle-ci est en attente et si la ressource est disponible.

I.5.5. Ordonnancement optimal :

Un ordonnancement est optimal s'il donne les meilleures valeurs possibles étant donné un ensemble d'objectifs à optimiser. Dans un problème d'ordonnancement il est possible de trouver plus d'une solution qui donnent le même résultat (même valeur de la fonction objectif)

Afin de concevoir un algorithme d'optimisation, il faut impérativement choisir l'une des classes. Parmi ces classes, la tendance se porte sur celle contenant le plus faible nombre d'ordonnancements. Pourtant, si l'on veut que l'algorithme puisse trouver la solution optimale, il faudrait que cette classe la contienne.

I.6. Représentation des problèmes d'ordonnancement :

Il existe trois sortes de représentations possibles d'un problème d'ordonnancement: le diagramme de Gantt, le graphe Potentiel-Tâches et la méthode PERT.

I.6.1. Le diagramme de Gantt :

Le diagramme de Gantt est un outil permettant de représenter les séquences de traitement des jobs sur chaque machine requise, en précisant la date de début et de fin de chacune d'elles. Il s'agit d'un outil élabore en 1917 par Henry L. Gantt. En effet, le diagramme de Gantt représente en ordonnée la liste des taches, notées O_i à exécuter par les machines notées M_i et en abscisse l'échelle du temps par lequel on précise la date de début, la durée de la tâche et la date de fin de chaque tache.

La Figure (I.7) représente un exemple de diagramme de Gantt d'un ordonnancement de quatre jobs sur trois machines :

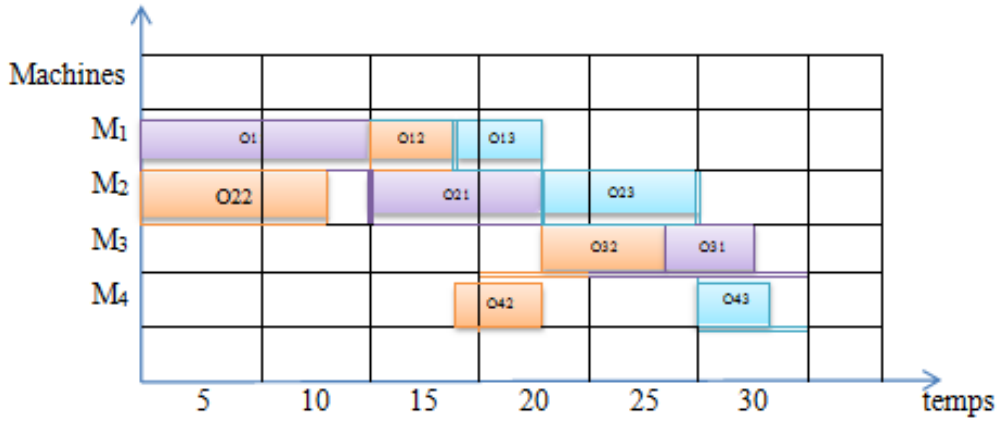


Figure 1.7 Diagramme de Gantt

I.6.2. Graphe Potentiel-Tâches :

Cette outil graphique a été développé grâce à la théorie des réseaux de Pétri qui ont surtout servi à modéliser les systèmes dynamiques à évènements discrets [1].

Dans ce genre de modélisation, les tâches sont représentées par des nœuds et les contraintes par des arcs, comme le montre la Figure (I.8). Ainsi, les arcs peuvent être de deux types :

- les arcs conjonctifs illustrant les contraintes de précédence et indiquant les durées des tâches,
- les arcs disjonctifs indiquant les contraintes de ressources.

Tâches	Durées	Antériorités
A	0	-
A	6 mois	A
B	3 mois	A
C	6mois	A
D	2 mois	B
E	4 mois	B
F	3 mois	d – a
Ω	0	c-e-f

Tableau 1.1 La table initiale d'ordonnancement T pour la réalisation d'un graphe potentiel-tâches [1]

Pour qu'une tâche puisse commencer, il est nécessaire que toutes les tâches qui la relient à la tâche du début **S** du projet, soient réalisées. On définit donc :

- **la date au plus tôt de la tâche**, qui correspond à la date de début, au plus tôt, de l'exécution de la tâche.

Exemple : la tâche f ne peut s'exécuter que si a et d ont été réalisées. Donc, pour exécuter a il faut 6 mois et pour exécuter d il faut 2+3 mois. La tâche f ne pourra commencer au plus tôt que 6 mois après le début du projet: c'est donc le plus long chemin entre a et f.

- **la durée du projet**, qui correspond au plus long chemin entre α (tâche de début du projet) et ω (tâche de fin du projet).

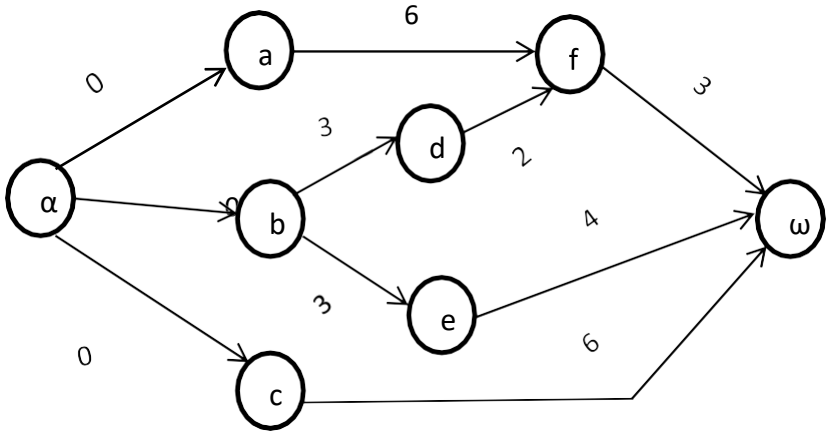


Figure 1.8 Graphe Potentiel-Tâches d'un ordonnancement [1]

I.6.3. Méthode PERT (Program Evaluation and Research Task):

Cette représentation, semblable à la précédente, permet de représenter une tâche par un arc, auquel est associé un chiffre qui représente la durée de la tâche. Entre les arcs, figurent des cercles, appelés sommets ou événements, qui marquent l'aboutissement d'une ou de plusieurs tâches. Ces cercles sont numérotés afin de suivre l'ordre de succession des divers évènements.Les méthodes graphiques ont connu une très importante évolution surtout avec l'apparition des Réseaux de Pétri (RdP), qui permettent de traduire plusieurs notions fondamentales ayant un lien avec les problèmes d'ordonnancement, telles que : les conflits sur les ressources, les durées opératoires certaines, les durées opératoires aléatoires, les gammes, les disponibilités, les multiplicités et les capacités de ressources la répétitivité, etc …

I.7. Complexité des problèmes d'ordonnancement :

D'une manière générale, les problèmes d'ordonnancement d'ateliers étant des problèmes d'optimisation combinatoires difficiles, il n'existe pas de méthodes universelles permettant de résoudre tous les cas. Plusieurs algorithmes peuvent être utilisés pour résoudre un problème d'ordonnancement mais tous ne sont pas équivalents. [7]

On distingue deux types de complexité: algorithmique et problématique [4]

I.7.1. Complexité algorithmique :

Le temps et l'espace mémoire sont les paramètres déterminant la performance des algorithmes pouvant résoudre un problème donné. La performance d'un algorithme se mesurepar rapport au temps de calcul nécessaire.

I.7.2. La complexité problématique :

La complexité problématique dépend de la difficulté du problème à résoudre et du nombre des opérations élémentaires qu'un algorithme peut effectuer pour trouver l'optimum en fonction de la taille du problème.

Selon son degré de complexité, un problème peut appartenir à l'une des quatre classes suivantes [8] :

- **Les problèmes les plus difficiles :** ce sont les problèmes indécidables pour lesquels il n'existepas d'algorithme pour leur résolution.
- **Les problèmes de classe P :** un problème de décision est dit facile où appartenir à la classe P s'il existe au moins un algorithme polynomial pour le résoudre.
- **Les problèmes de la classe NP :** Ce sont des problèmes NP-difficiles, qui ne peuvent à priori être résolus en un temps polynomial que par des méthodes approchées (heuristiques),

- **Les problèmes NP-Complets** : un problème de décision A est dit NP-Complet s'il appartient à laclasse NP et si pour tout A' de NP :

  - il existe une application polynomiale qui transforme toute instance I' de A' en une instanceI de A.
  - A' admet une réponse "oui" pour l'instance I', si et seulement si A admet une réponse "oui" pour l'instance I.

Autrement dit, s'il existe un algorithme polynomial pour résoudre A, alors, pour tout le reste des problèmes de la classe NP-Complets, il existe des algorithmes polynomiaux pour les résoudre. [8]

I.8. Comment résoudre les problèmes d'ordonnancement?

Au fil des années, de nombreuses méthodes de résolution de problèmes ont été proposées. Ainsi, une grande variété de concept et principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodesde résolution de problèmes NP-difficiles en deux classes principales : la classe de méthodes exactes et la classe des méthodes approchées.

I.8.1. Méthodes exactes :

Ces méthodes sont généralement utilisées pour résoudre des problèmes de petite taille. Dans ce cas, le nombre de combinaisons possibles est suffisamment faible pour pouvoir explorer l'espace de solutions en un temps raisonnable. On distingue trois sous-classes de méthodes exactes : la procédure de séparation et d'évaluation Branch and Bound, la programmation dynamique et la programmation linéaire.

I.8.2. Méthodes approchées :

La résolution d'un problème d'optimisation combinatoire, de taille comparable à ceux rencontrés dans la pratique, se heurte à des tailles mémoire et des temps de calcul trop importants. L'objectif n'est plus alors d'obtenir systématiquement l'optimum mais plutôt d'obtenir une solution proche de l'optimum ou de « bonne qualité » en un temps minimal. Ainsi, au lieu d'effectuer une recherche exhaustive, les méthodes approchées échantillonnent l'espace de recherche, n'en considèrent qu'une partie, et fournissent ainsi, en un temps raisonnable, la meilleure configuration rencontrée.

On distingue deux types de méthodes : les heuristiques et métaheuristiques. [8][14]

A. Les heuristiques :

Les heuristiques sont des méthodes empiriques basées sur des règles simplifiées pour

optimiser un ou plusieurs critères. Le principe général de ces méthodes est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum, tout en essayant de l'obtenir en un temps de calcul raisonnable. [8]

B. Les Métaheuristiques :

Une métaheuristiques est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace derecherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales

L'ensemble des métaheuristiques proposées dans la littérature sont partagées en deux catégories : des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions.

○ *Les Métaheuristiques à base de solution unique :*

Les métaheuristiques à base de solution unique débutent la recherche avec une seule solution initiale. Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations.

De nombreuses méthodes à base de solution unique ont été proposées dans la littérature, le recuit simulé, la recherche tabou. [14]

○ *Les métaheuristiques à base de population de solutions :*

Les métaheuristiques à base de population de solutions débutent la recherche avec une panoplie de solutions. Elles s'appliquent sur un ensemble de solutions afin d'en extraire la meilleure (l'optimum global) qui représentera la solution du problème traité. L'idée d'utiliser un ensemble de solutions au lieu d'une se

Une solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité.

Une grande variété de métaheuristiques basées sur une population de solutions a été proposée dans la littérature, algorithmes génétiques, et les algorithmes à base d'intelligence par essaims : l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies defourmis. [14].

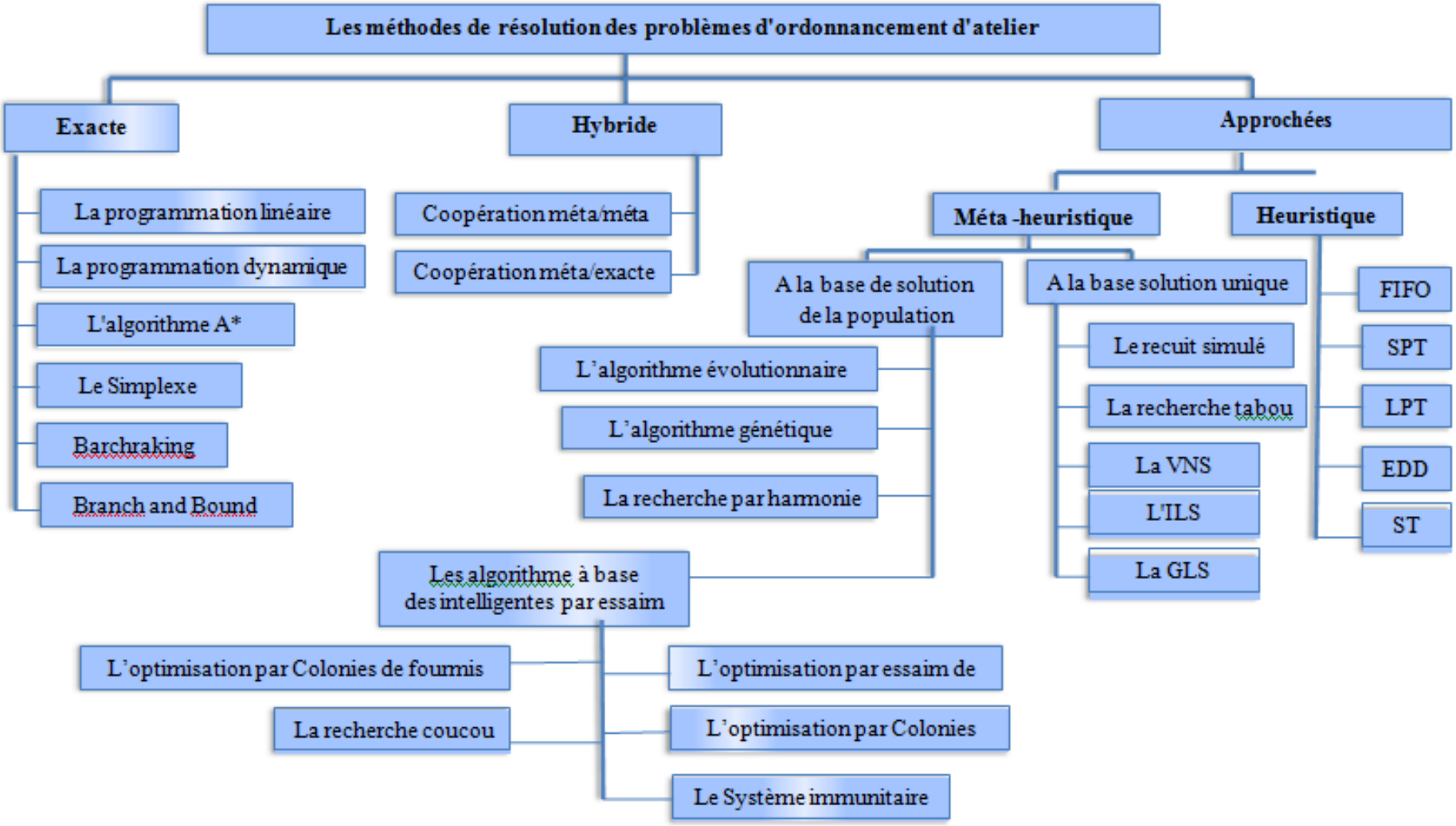


Figure 1.9 Classification des méthodes de résolution des problèmes d'ordonnancement d'atelier [17]

I.9. Conclusion :

Nous avons présenté, dans ce chapitre, les différentes notions et les concepts liées aux problèmes d'ordonnancements ainsi que les méthodes de résolution utilisée dans la littérature. Ces méthodes sont regroupées en deux classes : les méthodes exactes et les méthodes approchées et nous avons résumé quelques des informations importantes liées à notre méthode de résolution qui est les problèmes d'ordonnancements avec la représentation diagramme de Gantt.

En conclusion, l'ordonnancement est un domaine dynamique et passionnant, offrant de nombreuses opportunités d'amélioration des performances opérationnelles. En explorant les défis et les opportunités présentés par l'ordonnancement, nous serons en mesure de proposer des solutions innovantes et efficaces dans chapitre suivant pour répondre aux besoins de planification et d'organisation des systèmes complexes dans divers secteurs d'activité.

Chapitre 2 : Problèmes d'ordonnancement d'ateliers

II.1 Introduction :

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation des tâches ou les opérations, compte tenu d'un certain nombre de contraintes afin d'atteindre un certain objectif appelé fonction économique.

L'ordonnancement joue un rôle essentiel dans de nombreux secteurs d'activités : la conception (de bâtiments, de produits, de systèmes, . . .), l'administration (gestion d'emplois du temps, gestion du personnel), l'industrie (gestion de la production), l'informatique (ordonnancement de processus, ordonnancement de réseaux). Les méthodes d'ordonnancement foisonnent dans la littérature. Elles se différencient par la nature du problème considéré (nombre de ressources, structure particulière du problème, . . .), la nature des contraintes prises en compte, les objectifs à satisfaire (minimisation des coûts, de la durée totale de mise en œuvre, . . .) et la nature de l'approche de résolution adoptée (heuristiques, méthodes exactes, méta heuristiques, approches par contraintes, . . .).

Dans ce chapitre, nous introduisons quelques notions sur les problèmes d'ordonnancement dans les systèmes de production, Nous rappelons d'abord les différents paramètres et classifications dans tel problème, puis nous expliquons Les méthodes de résolution des problèmes d'ordonnancement d'atelier avec des exemples.

II.2. Les problèmes d'ordonnancement d'atelier :

Dans un problème d'atelier, une pièce doit être usinée ou assemblée sur différentes machines. Chaque machine est une ressource disjonctive, c'est-à-dire qu'elle ne peut exécuter qu'une tâche à la fois, et les tâches sont liées exclusivement par des contraintes d'enchaînement. Plus précisément, les tâches sont regroupées en n entités appelées travaux ou lots. Chaque lot est constitué de m tâches à exécuter sur m machines distinctes, et dans le cas des problèmes d'atelier, une tâche est une opération, une ressource est une machine et chaque opération nécessite pour sa réalisation une machine. Dans le modèle de base de l'ordonnancement d'atelier, l'atelier est constitué de m machines, n travaux (jobs), disponibles à la date 0, doivent être réalisés, un travail i est constitué de n_i opérations, l'opération j du travail i est notée (i,j) avec (i, 1) < (i, 2) < ... < (i, n_i) si le travail i possède une gamme (A < Bsignifie A précède B). Une opération (i,j) utilise la machine [16].

$m_{i,j}$ pendant toute sa durée $p_{i,j}$ et ne peut être interrompue.

Un atelier se définit par le nombre de machines qu'il contient et par son type. Une classification peut exister selon le nombre des machines et l'ordre d'utilisation des machines, pour réaliser un travail (par exemple fabrication d'un produit qui dépend de la

nature de l'atelier) [16].

Solon les caractéristiques des machines on peut distinguer plusieurs types des problèmes (II.1) :

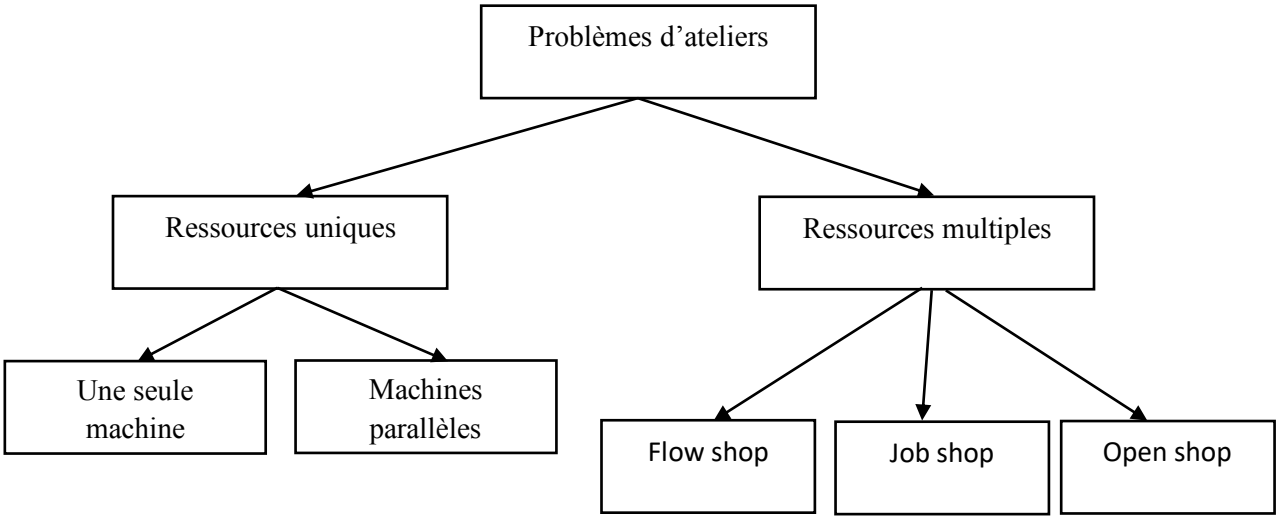


Figure II.1: Les problèmes d'ordonnancement d'ateliers

II.3 Ressources uniques :

II.3.1 Le problème à une machine :

Un problème est dit problème à une machine (ou ressource unique) si on ne dispose qu'une seule ressource pour réaliser un ensemble de travaux. Ces derniers sont constitués d'une seule tâche pour chacun d'eux. Dans ce type de problème la résolution consiste à trouver l'ordre optimal d'exécution de ces tâches vis-à-vis d'un critère donné [15].

Figure (II.2) présenter l'Ordonnancement à une machine :

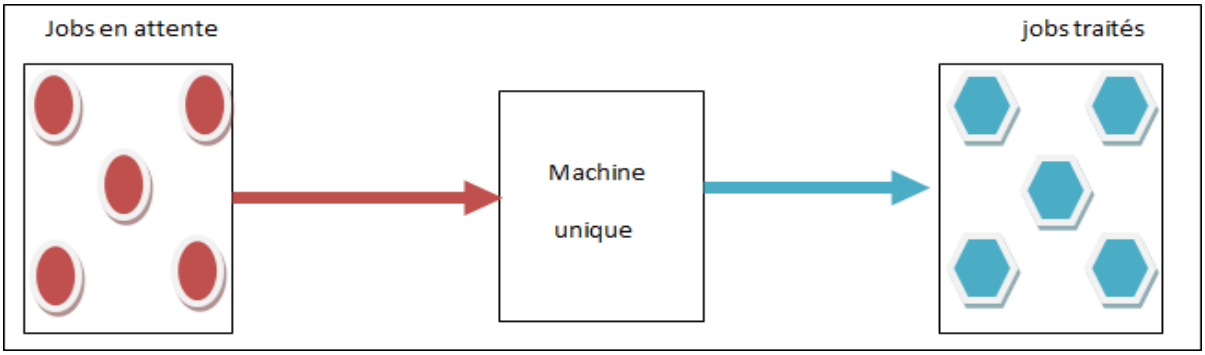


Figure II.2: Ordonnancement à une machine.

A. Algorithme Tri à bulles :

Le tri à bulles est un algorithme de tri classique. Son principe est simple, et il est très facile à implémenter.

On considère un tableau de nombres T, de taille N. L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en fin de tableau. On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite. Au i-ème passage on fait remonter le i-ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles [19].

On donne ci-dessous l'algorithme de tri à bulles en *pseudo-code* :

```
1: procedure TRI_BULLES(T)
2:     N ← taille de T
3:     pour i de N − 1 à 1 faire
4:         pour j de 0 à i − 1 faire
5:             si T[j] > T[j + 1] alors
6:                 échanger T[j] et T[j + 1]
7:             fin si
8:         fin pour
9:     fin pour
10: fin procedure
```

Exemple :

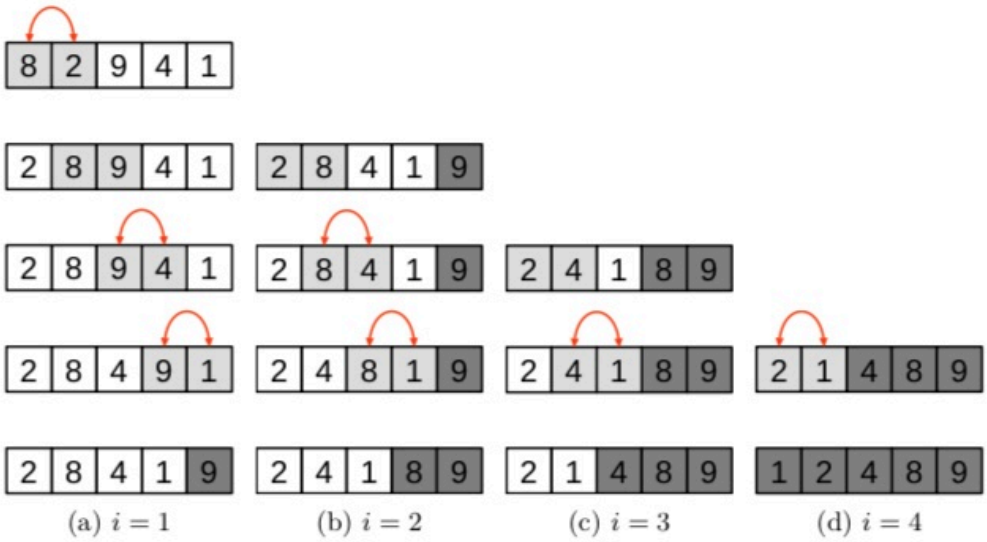


Figure II.3 – Exemple d'exécution de l'algorithme de tri à bulles.

B. Algorithm de SJF (Shortest Job First):

Le SJF (Shortest Job First) utilise une file ordonnée selon le temps d'exécution croissant (le plus court en tête de liste et le plus lent en queue de liste). Si deux processus possèdent le même temps estimé, ils sont traités en FCFS. L'estimation est faite à priori [20].

On donne ci-dessous l'algorithme de SJF :

Algorithm EQ-SJF

```
1.  Check for new event;

2.  if (event == "Arrival"){
3.        if ( (Queue, C) == ( Empty, Null) )
4.         {set C to the new arrival; set DNP = 0;}
5.   else
6.         add new arrival to Queue;
7.  }
8.  else if (event == "Completion")
9.         {set C to Null; set DNP = 0;}

10. if ((Queue, C) == ( Non-Empty, Null) ) {
11.       find process k with the min value
                    r_k =min_j{RT(j)};
12. set C to k; set DNP = 0;

13. }else if ((Queue, C) == ( Non-Empty, Non-Null)
    and ( DNP == 0)) {

14.       if( RT(C) ≤ e * BT(C) ) set    DNP = 1;
15. else{
16.       find process k with max value
                w_k = max_j{ WT(j)- q * BT(j)}
17.       if (w_k > 0){
18.  add C to Queue;
19.          set C = k; set DNP = 1;
20.        } else{
21.   find process k with the min value
          r_k =min_j{RT(j)};
22.         if (r_k < RT(C)){
23.           add C to Queue;
24.         set C = k; set DNP = 0;
25.       }
26.     }
27.   }
28. }
```

Exemple :

Taches	Temps d'exécution
A	21
B	3
C	6
D	2

Tableau **II**.1 : Présentation d'un exemple l'algorithme de Shortest Job First

Dans (shortest job first), le processus le plus court est exécuté en premier :

D→ B→ C→ A

II.3.2 Les problèmes à machines parallèles :

Les problèmes d'ordonnancement à machines parallèles se caractérisent par l'existence de plus d'une ressource. Elles représentent un cas particulier de problèmes multi-machines où les machines sont disposées en parallèle [15]. On peut distinguer trois types des problèmes à machinesparallèles :

- Les problèmes à machines identiques : les durées opératoires sont égales et ne dépendent donc pas des machines,
- Les problèmes à machines uniformes : la durée d'une opération varie uniformément enfonction de la performance de la machine choisie,
- Les problèmes à machines indépendantes (non liées) : les durées opératoires dépendentcomplètement des machines utilisées.

Figure (II.4) présenter les machine parallèles :

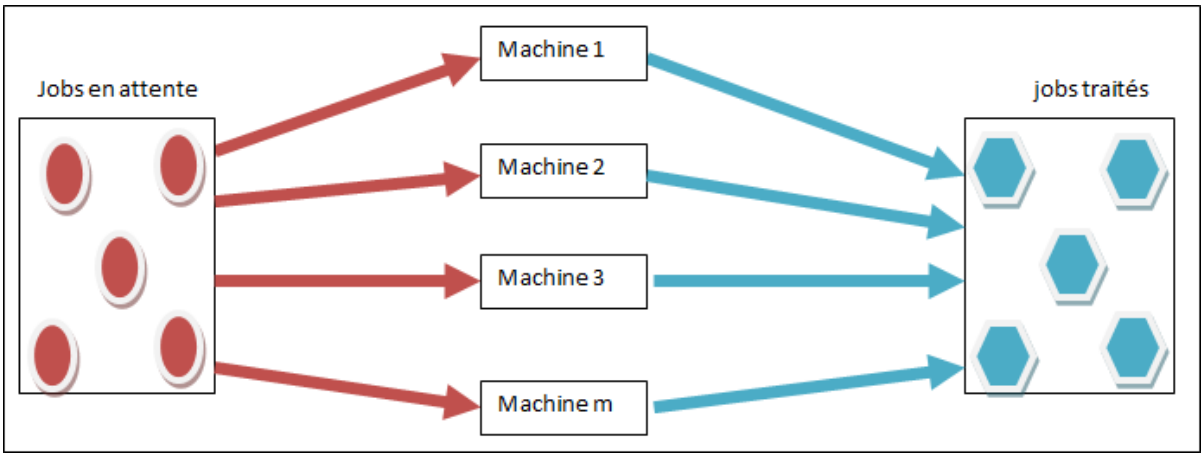


Figure II.4: Ordonnancement à machines parallèles.

A. Premier Arrivé Premier Servi (FCFS: First come First Served FIFO) :

L'algorithme de scheduling du processeur le plus simple est l'algorithme du Premier Arrivé Premier Servi(First Come First Served : FCFS). Avec cet algorithme, on alloue le processeur au premier processus qui le demande. L'implémentation de la politique FCFS est facilement gérée avec une file d'attente FIFO. Quand un processus entre dans la file d'attente des processus prêts, son PCB est enchaînée à la queue de la filed'attente. Quand le processeur devient libre, il est alloué au processeur en tête de la file

d'attente [21].

On donne ci-dessous l'algorithme de FCFS :

```
Entier Defiler() {
    Si EstVide() Alors
        Erreur ( "La file est vide !" )
    Sinon
        nouveau entier temp
        temp <- Tableau[debut]
        debut <- debut + 1
        Si ( debut = taille ) Alors
            debut <- 0
        FinSi
        Si ( fin = debut ) Alors
            vide <- vrai
        FinSi
        Renvoyer Temp

    FinSi
}
```

Exemple :

On considère l'ensemble des processus suivants :

Taches	A	B	C	D	E
Date arrivé	5	11	0	9	3
Date execu	1	6	5	4	4

Tableau II.2 Présentation d'un exemple l'algorithme de FCFS

Les taches sont triées dans l'ordre croissant en fonction de leur Date arrivé :

L'ordonnancement finale **:**

$$\mathbf{C \rightarrow E \rightarrow A \rightarrow D \rightarrow B}$$

B. Earliest Due Date (EDD):

C'est un algorithme simple, il est très utile dans certain cas d'ordonnancement, cette règle est efficace elle donne des résultats approchés pour le problème SMTWT, le principe de fonctionnement de la règle EDD est de planifier la séquence des taches par rapport à leurs date d'échues, la séquence est ordonnancée suivant un ordre croissant, de la tâche qui a la plus petite date d'échus vers la plus grande [22].

On donne ci-dessous l'algorithme de EDD :

Algorithm 1 The Earliest Due Date algorithm.

Data: *operationList* a list containing all the operations for the jobs, *machines* a list with the machines

Result: The completed jobs.

```
1  sort(operationList)
2  maxIndex = operationList.size()
3  time = 1
4  operationIndex = 0
5  while operationIndex < maxIndex do
6      for i = 0; i < machines.size(); i++ do
7          tempMachine = machines.get(i)
8          for j = 0; j < tempMachine.getProcessingRate(); j++ do
9              tempMachine.work(operationList(operationIndex), time)
10             operationIndex += 1
11             if operationIndex >= maxIndex then
12                 break
13             end
14         end
15         if operationIndex >= maxIndex then
16             break
17         end
18     end
19     time += 1
20 end
```

Exemple :

On considère l'ensemble des processus suivants :

Taches	A	B	C	D
Durées	6	4	5	3
Echances	15	12	14	10

Tableau II.3 : Présentation d'un exemple l'algorithme de EDD

Dans (Earliest Due Date) :

EDD = Echances – Durées

Taches	A	B	C	D
Durées	6	4	5	3
Echances	15	12	14	10
EDD	9	8	9	7

Tableau II.4 : Présentation résultats d'exemple l'algorithme de EDD

Les taches sont triées dans l'ordre croissant en fonction de leur critère EDD :

L'ordonnancement finale **:**

$$\mathbf{D \rightarrow B \rightarrow A \rightarrow C}$$

II.4 Ressource multiples :

Les produits sont procédés dans une direction unique selon la même gamme de fabrication. Les machines consacrées à la réalisation de ces produits sont inévitablement disposées en série (Figure II.5). Ce type de problème est souvent rencontré dans les systèmes produisant les matières liquides ou gazeuses [15].

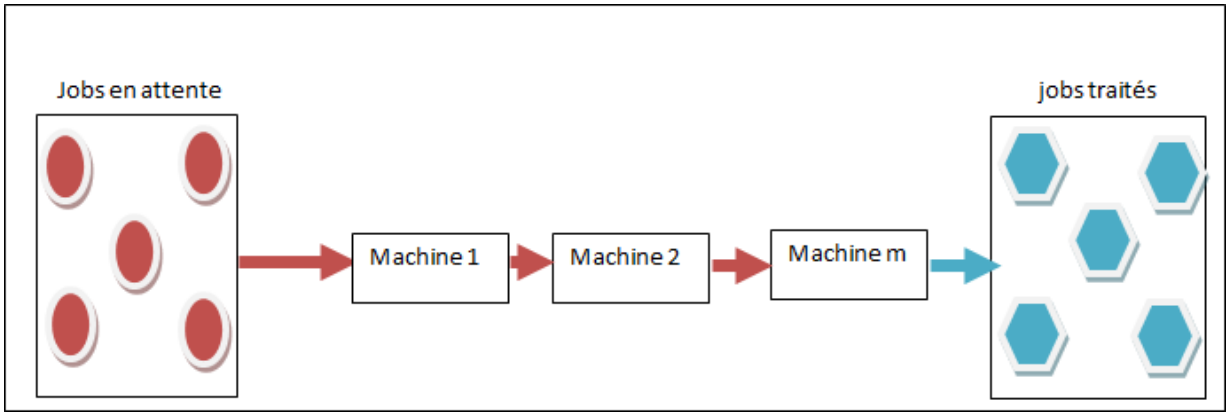


Figure II.5: Ordonnancement d'atelier flow shop.

II.4.1 Flow Shop:

A. Algorithme de Johnson :

L'algorithme de Johnson proposé par S.M Johnson qui calcule l'ordonnancement minimisant le temps total d'exécution des taches. Cet algorithme, et ses variantes, est un moyen rapide d'optimiser l'ordonnancement de processus simples. Son objectif est de minimiser la durée de réalisation d'une file d'attente de n pièces devant toutes passer selon le même ordre sur deux machines. La complexité de l'algorithme de Johnson est de l'ordre :$O\ (n\ log(n))$ [23].

Algorithme à deux machines construit à partir de la règle de Johnson :

Présentation d'algorithme de deux machines (m=2) : la règle de Johnson
❖ A_i : Durée du travail i sur la première machine. ❖ B_i : Durée du travail i sur la seconde machine. • Classer les travaux en deux groupes. • Dans le premier groupe G1, mettre tous les travaux tels que $A_i<B_i$ • Dans le second groupe G2, mettre tous les travaux tels que $B_i \leq A_i$ • Classer G1 par A_i croissants • Classer *G2* par B_i décroissants • La solution optimale est constituée de la séquence *G1* suivi de *G2*. ➢ Cette séquence est optimale.

Exemple :

Soit un problème de job-shop consistant à usiner sur 5 taches soient à exécuter sur 2machines.

Le tableau (3.3) suivant représenté les durées des jobs :

Tâche (i)	A	B	C	D	E
Tâche t_{iA}	50	150	80	200	30
Tâche t_{iB}	60	50	150	70	200

Tableau **II**.5: Présentation d'un exemple l'algorithme de Johnson

Dans le premier groupe *G1*, $A_i < B_i$

- la tache 5 (30<200) donc mise en première position.

1	**2**	**3**	**4**	**5**
E				

- la tache 1 (50 <60) donc est mise en deuxième position.

1	**2**	**3**	**4**	**5**
E	A			

- La tache 3 (80 < 150) donc mise en troisième position.

1	2	**3**	**4**	**5**
E	A	C		

- Dans le deuxième groupe *G2,* $B_i \leq A_i$:
- la tache 2 (50< 150) donc mise en première position.

1	**2**	**3**	**4**	**5**
B				

- la tache 4 (70 < 150) donc mise en deuxième position.

1	**2**	**3**	**4**	**5**
B	D			

- Classer G1 par A_i croissants :

1	2	3	4	5
E	C	A		

- Classer G2 par B_i décroissants :

1	**2**	**3**	**4**	**5**
D	B			

- La solution optimale est constituée de la séquence *G1* suivi de *G2*.

1	2	3	4	5
E	C	A	D	B

L'ordonnancement obtenu est optimal

E → C→ A → D→ B

B. Algorithme de Johnson généralisé :

L'algorithme de Johnson généralisé est une extension de l'algorithme de Johnson, qui est utilisé pour ordonner les tâches sur des machines dans le cadre de problèmes d'ordonnancement, Il peut s'appliquer sur toute fabrication dont le processus de fabrication est séquentiel avec plus de deux postes de fabrication même si tous les postes ne sont pas utilisés.

Algorithme à m-machines construit à partir de la règle de Johnson généralisé :

Présentation d'algorithme des machines (m) : la règle de Johnson généralisé
Pour chaque pièce : ▪ Réaliser la somme des temps de toutes les phases (N) ▪ Réaliser la somme x des temps des n-1 premières phases ▪ Réaliser la somme y des temps des n-1 dernières phases ▪ Calculer le rapport k=x/y On obtient l'ordre des fabrications grâce à l'ordre croissant de k.

Exemple :

Soit une file d'attente composée de six pièces et devant être fabriquées séquentiellement sur 4 machines, les temps opératoires sont exprimés en centièmes d'heures.

Pieces	M1	M2	M3	M4	Total	x	y	k	Ordre
P1	2000	1720	600	160	4480	4320	2480	1,74	6
P2	3560	3960	3800	3080	14400	11320	10840	1,04	4
P3	280	1880	800	3920	6880	2960	6600	0,45	1
P4	320	2560	480	3760	7120	3360	6800	0,49	2
P5	2440	760	2600	560	6360	5800	3920	1,48	5
P6	40	3200	2640	3120	9000	5880	8960	0,66	3

Tableau **II**.6: Présentation d'un exemple l'algorithme de Johnson généralisé

L'ordre de passage est donc : P3 P4 P 6 P2 P5 P1

II.4.2 job shop :

Le problème de Job Shop consiste à réaliser un ensemble de *n* Job (travail) sur un ensemble de *m* machines en cherchant d'atteindre certain objectifs. Chaque Job *j* est composé de n_j tâches devant être exécutées sur les différentes machines selon un ordre préalablement défini. Les travaux ne s'exécutent pas sur toutes les machines et de plus n'ont pas le même ordre d'exécution. En effet chaque travail emprunte le chemin qui lui est propre (Figure II.6). [15].

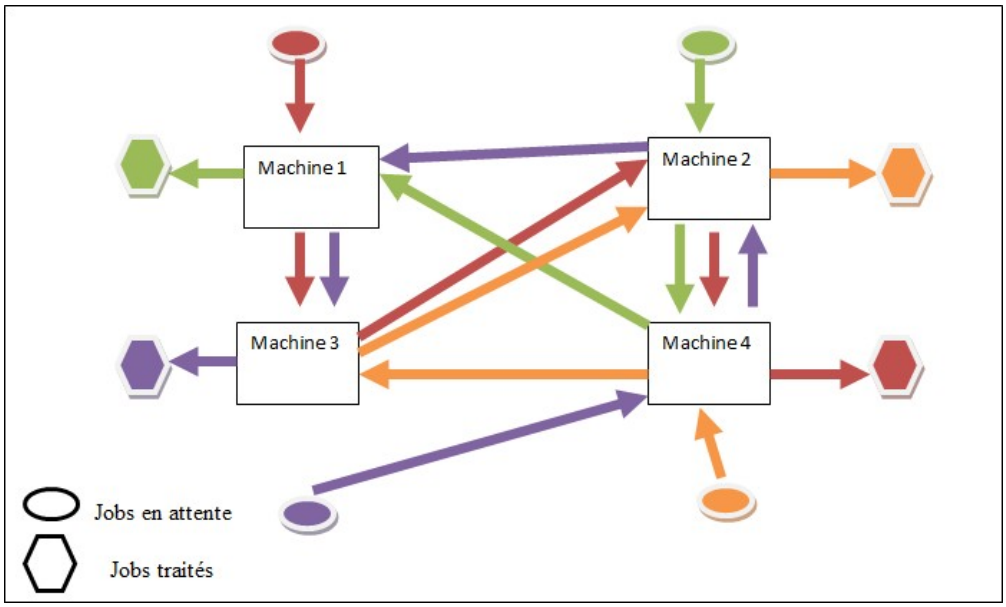


Figure II.6: Ordonnancement d'atelier job shop.

A. L 'heuristique SPT (shortest Processing Time) :

SPT (Shortest Processing Time) en français, DFC (Délai de Fabrication le plus Court) : la priorité est donnée aux opérations/commandes qui ont un délai de fabrication le plus court. On parle aussi de (*Traitement par ordre croissant du temps opératoire*) [24].

Algorithme de SPT :

Présentation d'algorithme de SPT :
Pour chaque JOB : Réaliser la somme des temps de toutes les phases (S) On obtient l'ordre des fabrications grâce à l'ordre croissant de S.

Exemple :

Les taches et leurs temps opératoires pour SPT :

	M1	M2	M3	S
J1	15	4	7	26
J2	10	11	6	27
J3	12	8	2	22

Tableau II.7: Présentation d'un exemple l'algorithme de SPT

D'après l'heuristique SPT la séquence :

J3 → J1→ J2

A. L 'heuristique LPT (Longest Processing Time) :

LPT (Longest Processing Time) en français, TFL (Temps de Fabrication le plus long) : la priorité est donnée aux opérations/commandes qui ont un temps de fabrication le plus long. On parle aussi de (*Traitement par ordre décroissant du temps opératoire*) [24].

Algorithme de LPT :

Présentation d'algorithme de LPT :
Pour chaque JOB : Réaliser la somme des temps de toutes les phases (S) On obtient l'ordre des fabrications grâce à l'ordre *décroissant* de S.

:

Exemple :

Les taches et leurs temps opératoires pour LPT :

	M1	M2	M3	S
J1	15	4	7	26
J2	10	11	6	27
J3	12	8	2	22

Tableau **II**.8: Présentation d'un exemple l'algorithme de LPT

D'après l'heuristique LPT la séquence :

J2 → J1→ J3

II.4.3 Atelier open-shop:

Dans ce type d'ateliers, l'acheminement d'opérations est multiple et libre, autrement dit, il n'existe aucun ordre d'exécution des opérations (les gammes sont libres). On trouve ce type d'atelier dans le cas où la fabrication de chaque produit se traduit par le traitement de plusieurs opérations, dont l'ordre est totalement libre [15].

II.5 Conclusion :

Dans ce chapitre Nous avons abordé, les différents types des problèmes d'ordonnancements ainsi que les méthodes de résolution utilisée dans la littérature. Ces méthodes sont regroupées en deux classes : les méthodes exactes et les méthodes approchées, la plupart des méthodes est approchées, et nous avons résumé quelques informations importantes de chaque méthodes aves des exemples simples, le prochain chapitre va détailler les méthodes avec des algorithmes applique sur "Matlab" et affichages des résultats de chaque méthode avec des "plots".

Chapitre 3 : Conceptions de l'outil de simulation

III.1 Introduction :

Dans ce chapitre, nous avons faire une simulation des méthodes d'ordonnancement d'atelier sur MATAB avec des exemples et l'affichage de résultat final de l'ordonnancement et leurs plots.

Nous avons développé une application sur MATLAB permettant de résoudre des problème d'ordonnancement d'atelier avec une interface utilisateur conviviale qui est affiche le résultat final.

III.2 Le langage de programmation Matlab :

MATLAB (« *matrix laboratory* ») est un langage de script émulé par un environnement de développement du même nom ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran. Les utilisateurs de MATLAB (environ 4 millions en 2019) sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche. [25]

III.3 Simulation des méthodes proposées :

III.3.1 Le problème à une machine :

A. Algorithme Tri à bulles :

Programme :

```
    temps_execution = [5, 2, 8, 1, 6, 7, 11, 12, 3, 14];
 n = length(temps_execution);
for i = 1:n-1
for j = 1:n-i
if temps_execution(j) > temps_execution(j+1)
  [temps_execution(j), temps_execution(j+1)] = deal(temps_execution(j+1),
                                    temps_execution(j));
   end
     end
        end
   temps_ecoule = 0;
   temps_attente_total = 0;
   gantt_labels = cell(1, n);
   gantt_durations = zeros(1, n);
   gantt_start_times = zeros(1, n);
   for i = 1:n
temps_processus = temps_execution(i);
temps_attente = temps_ecoule;
temps_ecoule = temps_ecoule + temps_processus;
```

```
      temps_attente_total = temps_attente_total + temps_attente;
gantt_labels{i} = sprintf('Processus %d', i);
   gantt_durations(i) = temps_processus;
   gantt_start_times(i) = temps_attente;
   fprintf('Processus %d - Temps d"exécution : %d - Temps d"attente : %d\n', i,
temps_processus, temps_attente);
         end
   temps_attente_moyen = temps_attente_total / n;
   fprintf('Temps d"attente moyen : %.2f\n', temps_attente_moyen);
       figure;
       xlabel('Temps');
       ylabel('Processus');
        title('Diagramme de Gantt - Tri à bulles');
           hold on;
             for i = 1:n
      plot([gantt_start_times(i), gantt_start_times(i) + gantt_durations(i)], [i, i], 'r',
'LineWidth', 10);
            end
              hold off;
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[4 2 9 1 5 6 3 7 8 10]*

Et on a présenté dans la figure III.1 :

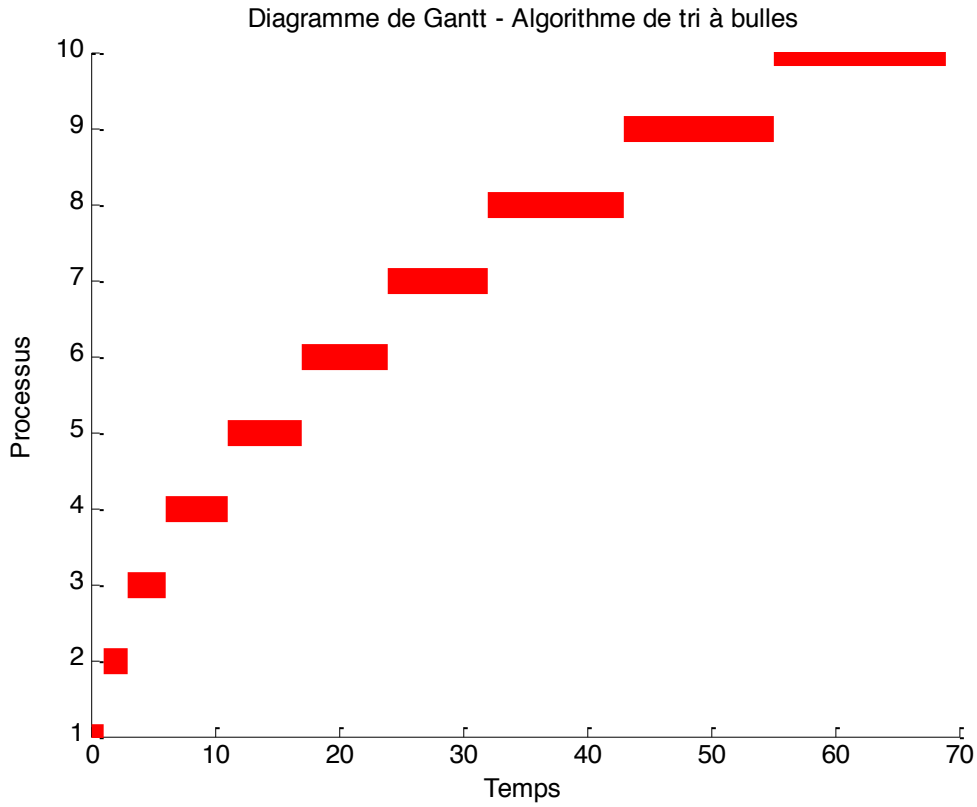


Figure III.1 diagramme de gantt algorithme de Tri à bulles

Commentaire :

Le diagramme de Gantt représente visuellement l'algorithme de tri à bulles appliqué à une liste de temps d'exécution. Les barres rouges horizontales représentent les processus, et leur longueur correspond à la durée de chaque processus. Les processus sont exécutés de gauche à droite.

B. Algorithme SJF :

Programme :

```
t = [5, 2, 8, 1, 6, 7, 11, 12, 3, 14];
  [sorted_t, order] = sort(t);
   et = 0;
   tat = 0;
   labels = cell(1, length(order));
   durations = zeros(1, length(order));
     starts = zeros(1, length(order));
      for i = 1:length(order)
       tt = sorted_t(i);
         ta = et;
           et = et + tt;
            tat = tat + ta;
             labels{i} = sprintf('P%d', order(i));
              durations(i) = tt;
                starts(i) = ta;
               fprintf('P%d - TE : %d - TA : %d\n', order(i), tt, ta);
     end
     tata = tat / length(order);
     fprintf('TATA : %.2f\n', tata);
     figure;
     xlabel('Temps');
     ylabel('Processus');
     title('Diagramme de Gantt - SJF');
     hold on;
     for i = 1:length(order)
      plot([starts(i), starts(i) + durations(i)], [i, i], 'r', 'LineWidth', 10);
       end
         hold off;
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[4 2 9 1 5 6 3 7 8 10]*

Et on a présenté dans la figure III.2 :

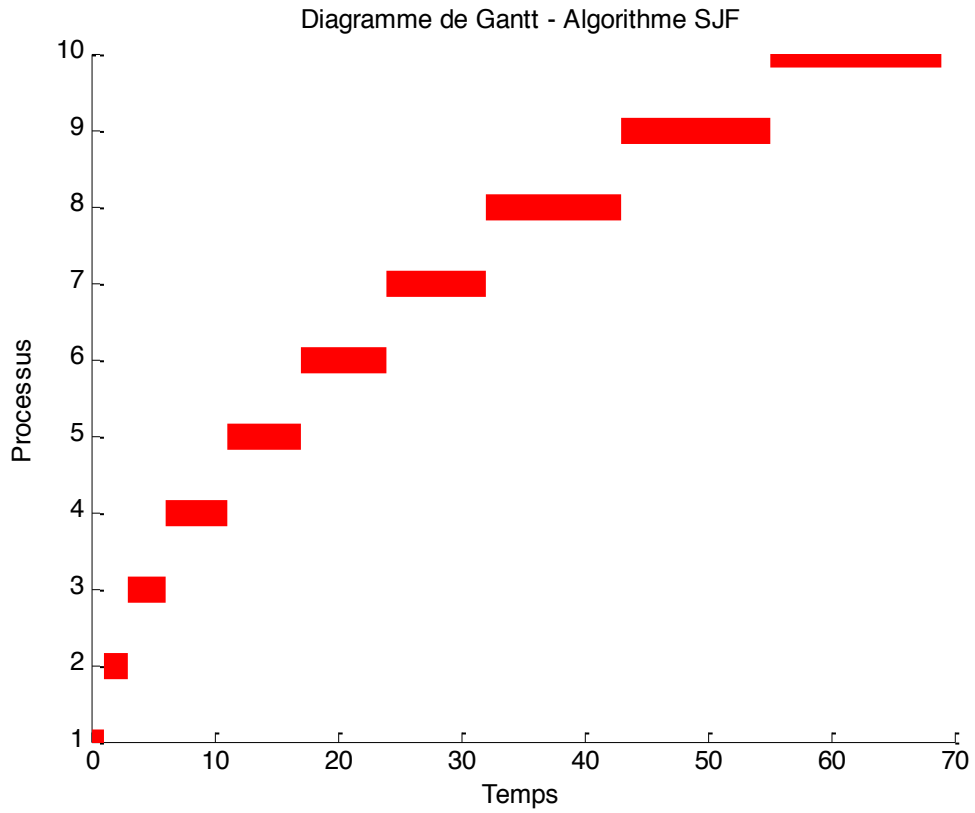


Figure III.2 diagramme de gantt SJF

Commentaire :

Le diagramme de Gantt généré par ce programme représente l'ordonnancement des processus en utilisant l'algorithme SJF (Shortest Job First). Cet algorithme consiste à exécuter en priorité les processus ayant les temps d'exécution les plus courts.

Le diagramme de Gantt montre les temps de début des processus en ajoutant des lignes verticales pour chaque processus. La ligne rouge représentant chaque processus est dessinée de manière à partir du temps d'attente et se prolonge jusqu'à la fin de l'exécution du processus.

III.3.2 Les problèmes à machines parallèles :

A. Premier Arrivé Premier Servi (FCFS) :

Programme :

```
taches = [
    struct('date_arrivee', 6, 'duree_execution', 5)
    struct('date_arrivee', 8, 'duree_execution', 2)
    struct('date_arrivee', 0, 'duree_execution', 3)
    struct('date_arrivee', 2, 'duree_execution', 6)
    struct('date_arrivee', 4, 'duree_execution', 4)
];
    taches = sortTasksByArrival(taches);
    temps_actuel = 0;
    temps_attente_total = 0;
    n = length(taches);
   gantt_labels = cell(1, n);
gantt_durations = zeros(1, n);
gantt_start_times = zeros(1, n);
for i = 1:n
      temps_attente = max(0, taches(i).date_arrivee - temps_actuel);
      temps_actuel = taches(i).date_arrivee + taches(i).duree_execution;
      temps_attente_total = temps_attente_total + temps_attente;
      gantt_labels{i} = sprintf('Tâche %d', i);
      gantt_durations(i) = taches(i).duree_execution;
      gantt_start_times(i) = taches(i).date_arrivee;
    fprintf('Tâche %d : Date d''arrivée = %d, Durée d''exécution = %d, Temps d''attente =
%d\n', ...
        i, taches(i).date_arrivee, taches(i).duree_execution, temps_attente);
      end
    fprintf('Temps d''attente total = %d\n', temps_attente_total);
    figure;
    xlabel('Temps');
    ylabel('Tâches');
    title('Diagramme de Gantt - Algorithme FIFO');
          hold on;
        for i = 1:n
        plot([gantt_start_times(i), gantt_start_times(i) + gantt_durations(i)], [i, i], 'r',
'LineWidth', 10);
          end
       hold off;
       set(gca, 'yticklabel', gantt_labels);
      function sortedTasks = sortTasksByArrival(tasks)
    [~, order] = sort([tasks.date_arrivee]);
    sortedTasks = tasks(order);
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[3 4 5 1 2]*

Et on a présenté dans la figure III.3 :

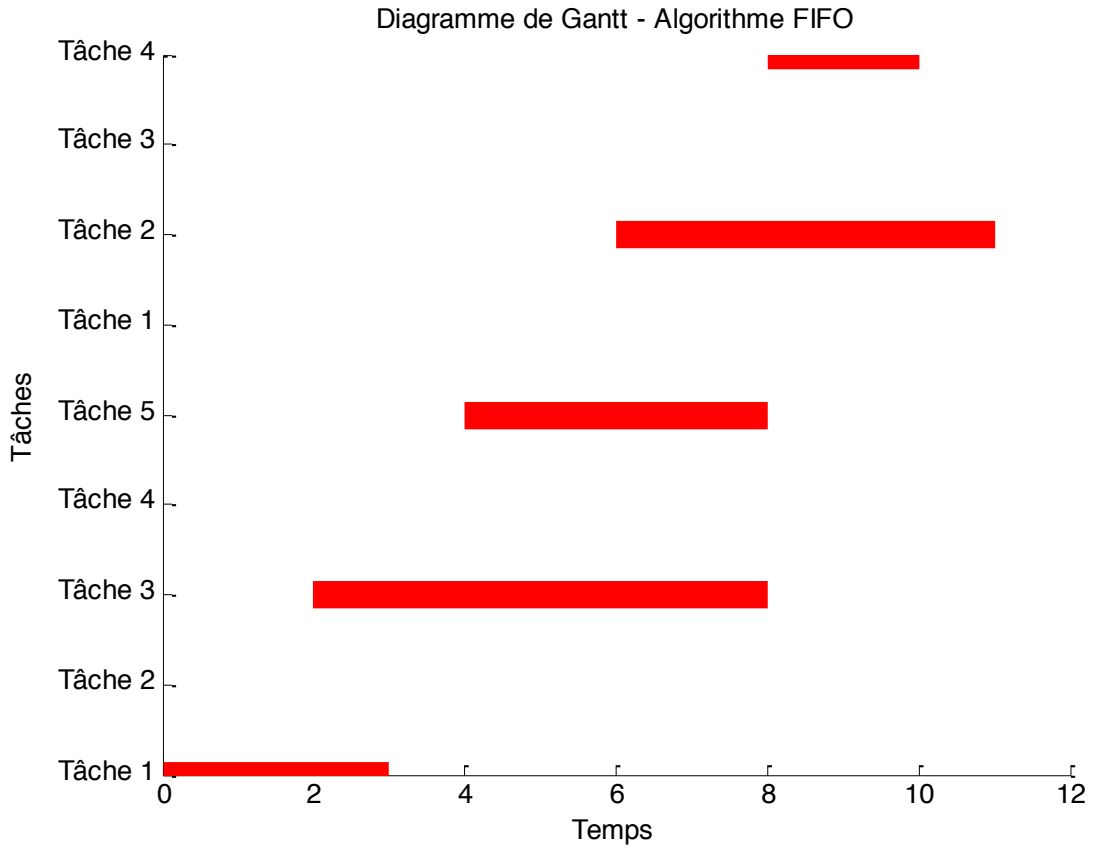


Figure III.3 diagramme de Gantt 'FIFO'

Commentaire :

Le diagramme de Gantt généré par ce programme représente l'exécution des tâches selon l'algorithme FIFO (First-In, First-Out). Chaque tâche est affichée sous forme d'une barre horizontale, dont la longueur correspond à la durée d'exécution de la tâche. Les barres sont placées sur une ligne temporelle, indiquant les instants de début et de fin de chaque tâche.

A. Earliest Due Date (EDD):

Programme :

```
function EDD_Gantt()
taches = [
    struct('date_arrivee', 6, 'duree_execution', 5, 'date_echeance', 8),
    struct('date_arrivee', 8, 'duree_execution', 2, 'date_echeance', 9),
    struct('date_arrivee', 0, 'duree_execution', 3, 'date_echeance', 8),
    struct('date_arrivee', 2, 'duree_execution', 6, 'date_echeance', 9),
    struct('date_arrivee', 4, 'duree_execution', 4, 'date_echeance', 7)
];
    EDD = [taches.date_echeance] - [taches.duree_execution];
    [~, ordre] = sort(EDD);
    temps_execution = cumsum([taches(ordre).duree_execution]);
    fprintf('Ordre d"exÈcution des t,ches : ');
    for i = 1:length(taches)
        fprintf('%d ', ordre(i));
    end
    fprintf('\n');
    fprintf('Temps d"exÈcution final de chaque t,che : \n');
    for i = 1:length(taches)
        fprintf('T,che %d: %d jours\n', ordre(i), temps_execution(i));
    end
    temps_attente_total = sum(temps_execution - [taches(ordre).date_arrivee]);
    fprintf('Temps d"attente total = %d\n', temps_attente_total);
    n = length(taches);
    gantt_start_times = zeros(1, n);
    gantt_durations = [taches(ordre).duree_execution];
    gantt_labels = cell(1, n);
    for i = 1:n
        gantt_start_times(i) = max(taches(ordre(i)).date_arrivee,
sum(gantt_durations(1:i-1)));
        gantt_labels{i} = sprintf('T,che %d', ordre(i));
    end
    figure;
    xlabel('Temps');
    ylabel('T,ches');
    title('Diagramme de Gantt - Algorithme FIFO');
    hold on;
    for i = 1:n
        plot([gantt_start_times(i), gantt_start_times(i) + gantt_durations(i)], [i, i], 'r',
'LineWidth', 10);
    end
    hold off;
    set(gca, 'yticklabel', gantt_labels);
end
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[1 4 5 3 2]*

Et on a présenté dans la figure III.4 :

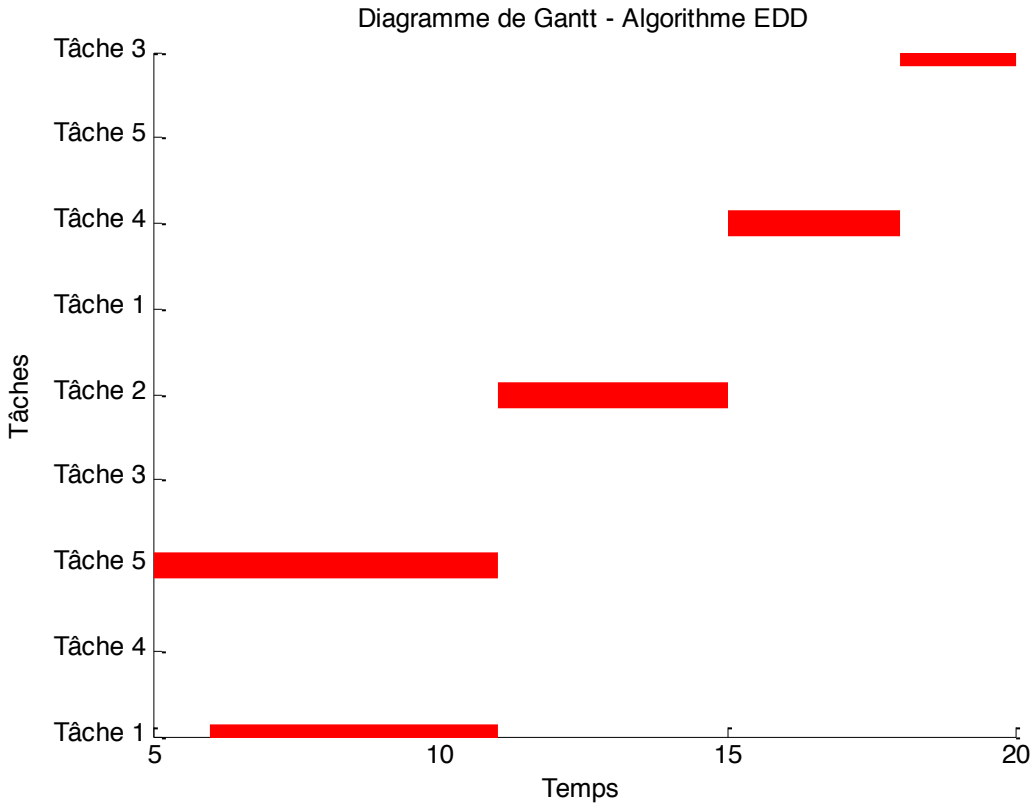


Figure III.4 diagramme de gantt ' EDD'

Commentaire :

Le diagramme de Gantt généré par ce programme utilise l'algorithme EDD pour déterminer l'ordre d'exécution des tâches en fonction de leur date d'échéance. Les tâches sont triées dans l'ordre croissant des dates d'échéance, et le temps d'exécution cumulatif de chaque tâche est calculé. Le temps d'attente total est également calculé en fonction des temps d'exécution et des dates d'arrivée des tâches, Chaque tâche est représentée par une barre horizontale rouge sur le graphique, dont la longueur correspond à la durée d'exécution de la tâche. Les étiquettes des tâches sont affichées sur l'axe vertical.

III.3.3 Flow shop :

A. Algorithme de Johnson :

Programme :

```
temps_exec = [2 1; 3 4; 5 3; 1 3; 4 2; 2 5];
temps_traitement_modif = max(temps_exec(:, 1), temps_exec(:, 2)) +
sum(temps_exec(:, 2));
[~, index_trie] = sort(temps_traitement_modif);
ordonnancement = zeros(size(temps_exec));
machine1_temps = 0;
machine2_temps = sum(temps_exec(:, 2));
                        for i = 1:size(temps_exec, 1)
tache = index_trie(i);
temps_tache = min(temps_exec(tache, :));
machine1_temps = machine1_temps + temps_tache;
machine2_temps = machine2_temps - temps_exec(tache, 2);
ordonnancement(i, :) = [tache machine1_temps + machine2_temps];
end
disp(ordonnancement(:, 1)');
figure;
hold on;
machine1EndTime = 0;
machine2EndTime = 0;
for i = 1:size(ordonnancement, 1)
jobIndex = ordonnancement(i, 1);
machine1StartTime = machine1EndTime;
machine1EndTime = machine1StartTime + temps_exec(jobIndex, 1);
machine2StartTime = max(machine1EndTime, machine2EndTime);
machine2EndTime = machine2StartTime + temps_exec(jobIndex, 2);
rectangle('Position', [machine1StartTime, jobIndex - 0.4,
temps_exec(jobIndex, 1), 0.8], 'FaceColor', 'b');
rectangle('Position', [machine2StartTime, jobIndex - 0.4,
temps_exec(jobIndex, 2), 0.8], 'FaceColor', 'r');
text(machine1StartTime + temps_exec(jobIndex, 1)/2, jobIndex,
sprintf('%d', jobIndex), 'HorizontalAlignment', 'center',
'VerticalAlignment', 'middle', 'Color', 'w');
end
xlabel('Temps');
ylabel('Machine');
ylim([0, size(temps_exec, 1) + 1]);
xlim([0, machine2EndTime]);
 yticks(1:size(temps_exec, 1));
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[4 6 2 3 5 1]*

Et on a présenté dans la figure III.5 :

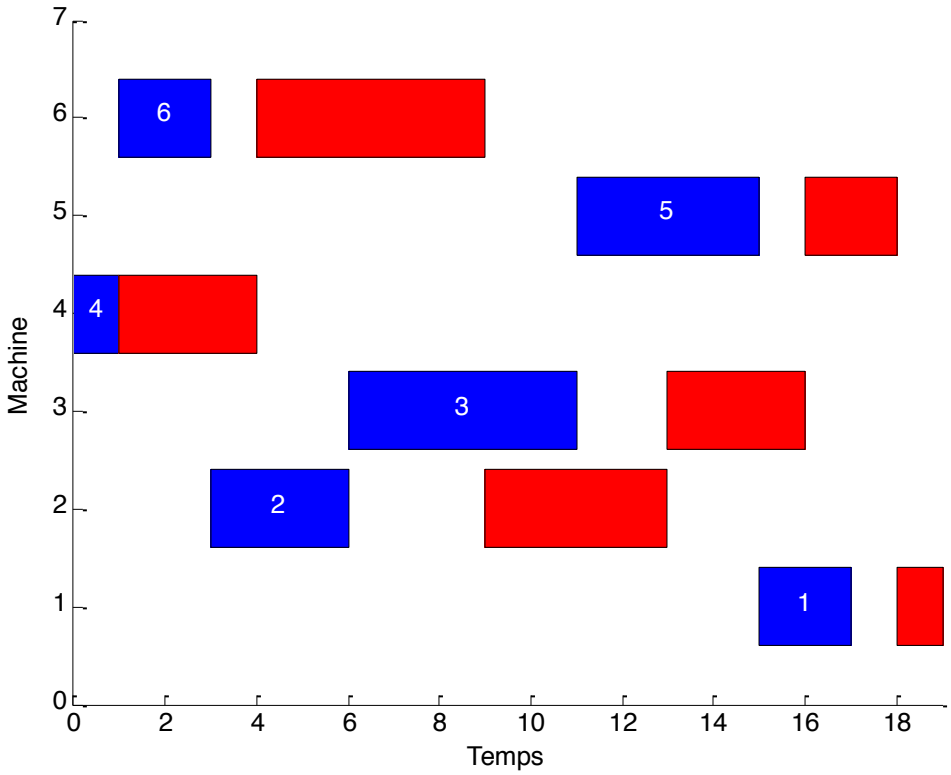


Figure III.5 diagramme de gantt Johnson

Commentaire :

le diagramme de Gantt est tracé pour visualiser l'ordonnancement des tâches sur les deux machines. Chaque tâche est représentée par un rectangle bleu pour la première machine et un rectangle rouge pour la deuxième machine. Le numéro de la tâche est affiché au centre de chaque rectangle.

B. Algorithme de Johnson généralisé :

Programme :

```
P=[2000 1720 600 160;3560 3960 3800 3080;280 1880 800 3920;320 2560
480 3760;2440 760 2600 560;40 3200 2640 3120]
% Calcul des temps de traitement les plus courts pour chaque t,che
[n m]=size(P);
Y = P(:, 2);
X = P(:, 1);
K=zeros(size(P),1);
for i = 1:n
    X(i) = sum(P(i, 1:m-1));
    Y(i) = sum(P(i, 2:m));
    K(i)=X(i)/Y(i)
end
% Tri des t,ches en fonction du temps de traitement sur la premiËre machine
[~, order] = sort(K)
bar(order,'stacked');
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[6 4 1 2 5 3]*

Et on a présenté dans la figure III.6 :

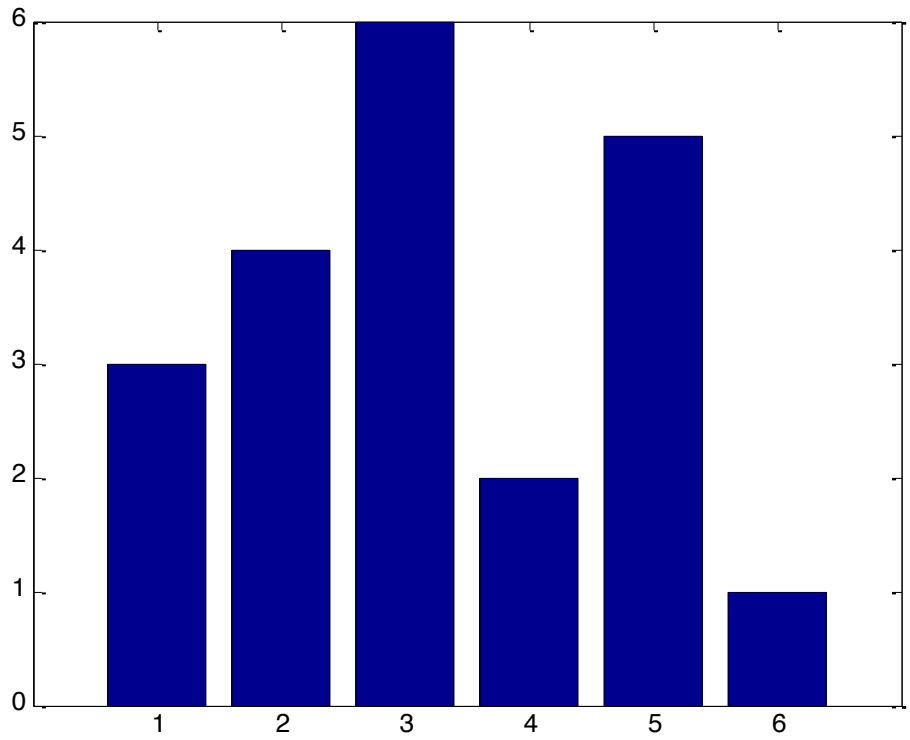


Figure III.6 plots Johnson généralisé

Commentaire :

Le programme fournit un graphique à barres bleu empilées qui représente l'ordre de traitement des tâches en fonction de K. Chaque barre représente une tâche, et sa hauteur est déterminée l'ordre des taches des machines globale.

III.3.4 job shop :

A. L 'heuristique SPT (shortest Processing Time) :

Programme :

```
tempsTraitement = [15 4 7; 10 11 6; 12 8 2];
[~, ordre] = sort(sum(tempsTraitement, 1));
tempsDebut = zeros(size(tempsTraitement));
tempsFin = zeros(size(tempsTraitement));
tempsFin(ordre(1), 1) = tempsTraitement(ordre(1), 1);
for j = 2:size(tempsTraitement, 2)
    tempsDebut(ordre(1), j) = tempsFin(ordre(1), j-1);
    tempsFin(ordre(1), j) = tempsDebut(ordre(1), j) + tempsTraitement(ordre(1),
j);
    for i = 2:size(tempsTraitement, 1)
        tempsDebut(ordre(i), j) = max(tempsFin(ordre(i-1), j), tempsFin(ordre(i), j-
1));
        tempsFin(ordre(i), j) = tempsDebut(ordre(i), j) + tempsTraitement(ordre(i),
j);
    end
end
tempsTotalExecution = max(tempsFin(:));
figure;
hold on;
couleurs = ['r', 'g', 'b', 'c', 'm', 'y'];
for j = 1:size(tempsTraitement, 2)
    for i = 1:size(tempsTraitement, 1)
        x = [tempsDebut(i, j) tempsFin(i, j) tempsFin(i, j) tempsDebut(i, j)];
        y = [j-0.4 j-0.4 j+0.4 j+0.4];
        couleur = couleurs(mod(i, numel(couleurs)) + 1);
        fill(x, y, couleur);
    end
end
title('Diagramme de Gantt - SPT-JS');
xlabel('Temps');
ylabel('Machine');
ylim([0.5 size(tempsTraitement, 2)+0.5]);
yticks(1:size(tempsTraitement, 2));
yticklabels(1:size(tempsTraitement, 2));
xlim([0 tempsTotalExecution]);
disp(tempsTotalExecution);
disp(tempsDebut);
disp(tempsFin);
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[2 3 1]*

Et on a présenté dans la figure III.7 :

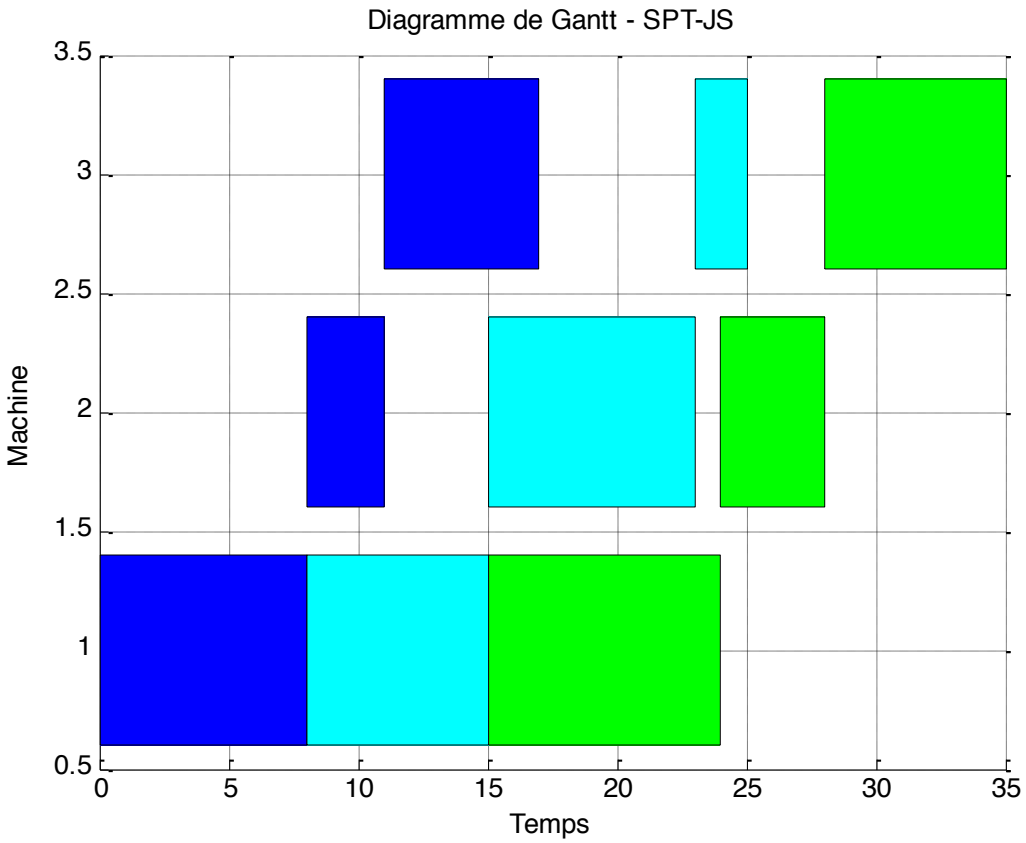


Figure III.7 diagramme de gantt SPT

Commentaire :

le diagramme de Gantt représente les tâches sur différentes machines. Chaque tâche est représentée par un rectangle coloré, et la position horizontale du rectangle indique le temps de début et de fin de chaque tâche sur la machine correspondante. Les couleurs différentes sont utilisées pour distinguer les tâches (le bleu présenté la tache2)(le bleu clair présenté la tache3)(le vert présenté la tache1).

A. L 'heuristique LPT (Longest Processing Time) :

Programme :

```
tempsTraitement = [15 4 7; 10 11 6; 12 8 2];
tempsTotalTraitement = sum(tempsTraitement, 1);
[~, ordre] = sort(tempsTotalTraitement, 'descend');
tempsDebut = zeros(size(tempsTraitement));
tempsFin = zeros(size(tempsTraitement));
tempsDebut(ordre(1), 1) = 0;
tempsFin(ordre(1), 1) = tempsTraitement(ordre(1), 1);
for i = 2:numel(ordre)
tempsDebut(ordre(i), 1) = tempsFin(ordre(i-1), 1);
tempsFin(ordre(i), 1) = tempsDebut(ordre(i), 1) + tempsTraitement(ordre(i), 1);
end
for j = 2:size(tempsTraitement, 2)
tempsDebut(ordre(1), j) = tempsFin(ordre(1), j-1);
tempsFin(ordre(1), j) = tempsDebut(ordre(1), j) + tempsTraitement(ordre(1), j);
for i = 2:numel(ordre)
tempsDebut(ordre(i), j) = max(tempsFin(ordre(i-1), j), tempsFin(ordre(i), j-1));
tempsFin(ordre(i), j) = tempsDebut(ordre(i), j) + tempsTraitement(ordre(i), j);
end
end
tempsTotalExecution = max(tempsFin(:));
couleurs = ['r', 'g', 'b', 'c', 'm', 'y'];
figure;
grid on;
hold on;
for j = 1:size(tempsTraitement, 2)
for i = 1:size(tempsTraitement, 1)
x = [tempsDebut(i, j) tempsFin(i, j) tempsFin(i, j) tempsDebut(i, j)];
y = [j-0.4 j-0.4 j+0.4 j+0.4];
couleur = couleurs(mod(i, numel(couleurs)) + 1);
fill(x, y, couleur);
end
end
title('Diagramme de Gantt - LPT-JS');
xlabel('Temps');
ylabel('Machine');
ylim([0.5 size(tempsTraitement, 2)+0.5]);
yticks(1:size(tempsTraitement, 2));
yticklabels(1:size(tempsTraitement, 2));
xlim([0 tempsTotalExecution]);
disp(tempsTotalExecution);
disp(tempsDebut);
disp(tempsFin);
```

Nous avons exposé les résultats sous forme de vecteurs V :

V : *[1 2 3]*

Et on a présenté dans la figure III.8 :

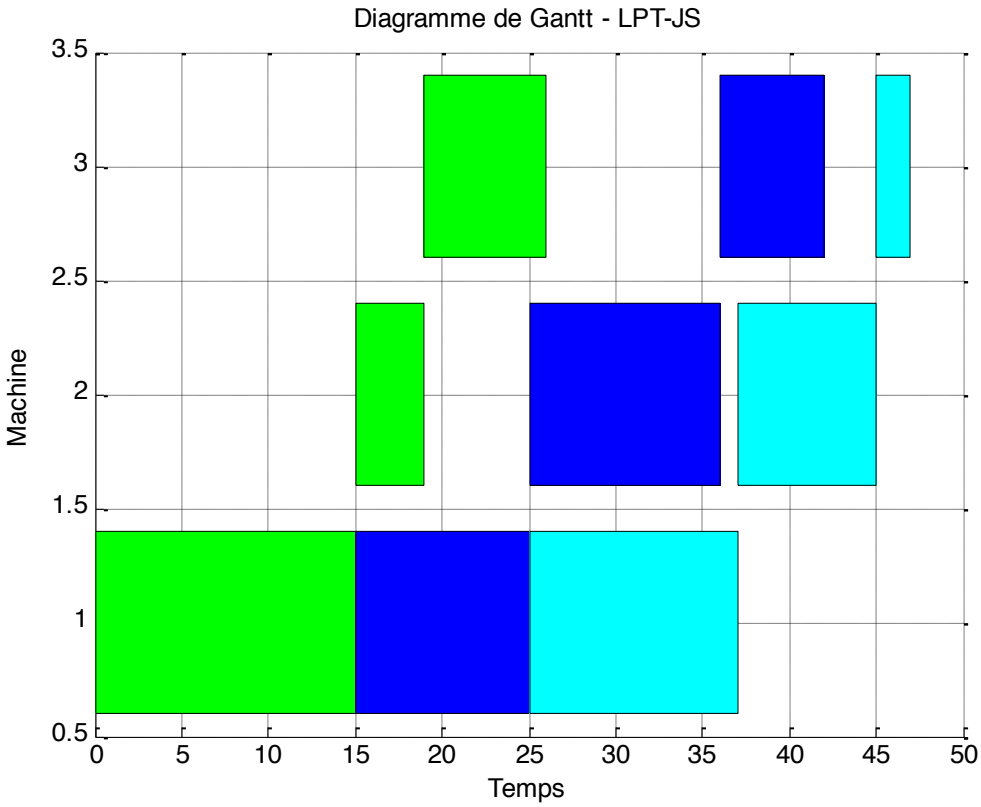


Figure III.8 diagramme de gantt LPT

Commentaire :

le diagramme de Gantt représente les tâches sur différentes machines. Chaque tâche est représentée par un rectangle coloré, et la position horizontale du rectangle indique le temps de début et de fin de chaque tâche sur la machine correspondante. Les couleurs différentes sont utilisées pour distinguer les tâches (le vert présenté la tache1) (le bleu présenté la tache2) (le bleu clair présenté la tache3).

III.4 Application pour résoudre les problèmes d'ordonnancement d'atelier :

III.4.1 Présentation d'application :

Nous avons développé une application sur MATLAB permettant de résoudre le problème d'ordonnancement d'atelier avec une interface utilisateur conviviale ou les utilisateurs peuvent saisir les paramètres suivants : le nombre de taches date d'arrivée, dure d'exécution, date d'échéance et le nombre de machine en fonction du problème traité. Nous avons programmé des méthodes exactes comme : algorithme Johnson et algorithme SJF et des méthodes approchées comme : les algorithmes EDD, FIFO, TRI, SPT, LPT qui

est présenté à l'utilisateur le résultat final dans l'interface pour permettre aux utilisateurs de visualiser et d'analyser les solutions obtenues.

III.4.2 Interface graphique de l'application :

A. Modèle statique :

L'interface de l'application conçue contient une seule fenêtre. Afin d'appliquer les méthodes de résolution décrites dans le chapitre 3, l'utilisateur doit saisir le nombre des taches, le nombre des machines et la durée des taches de chaque machine pour créer les listes. Le menu principal de l'application est donné par la figure suivante :

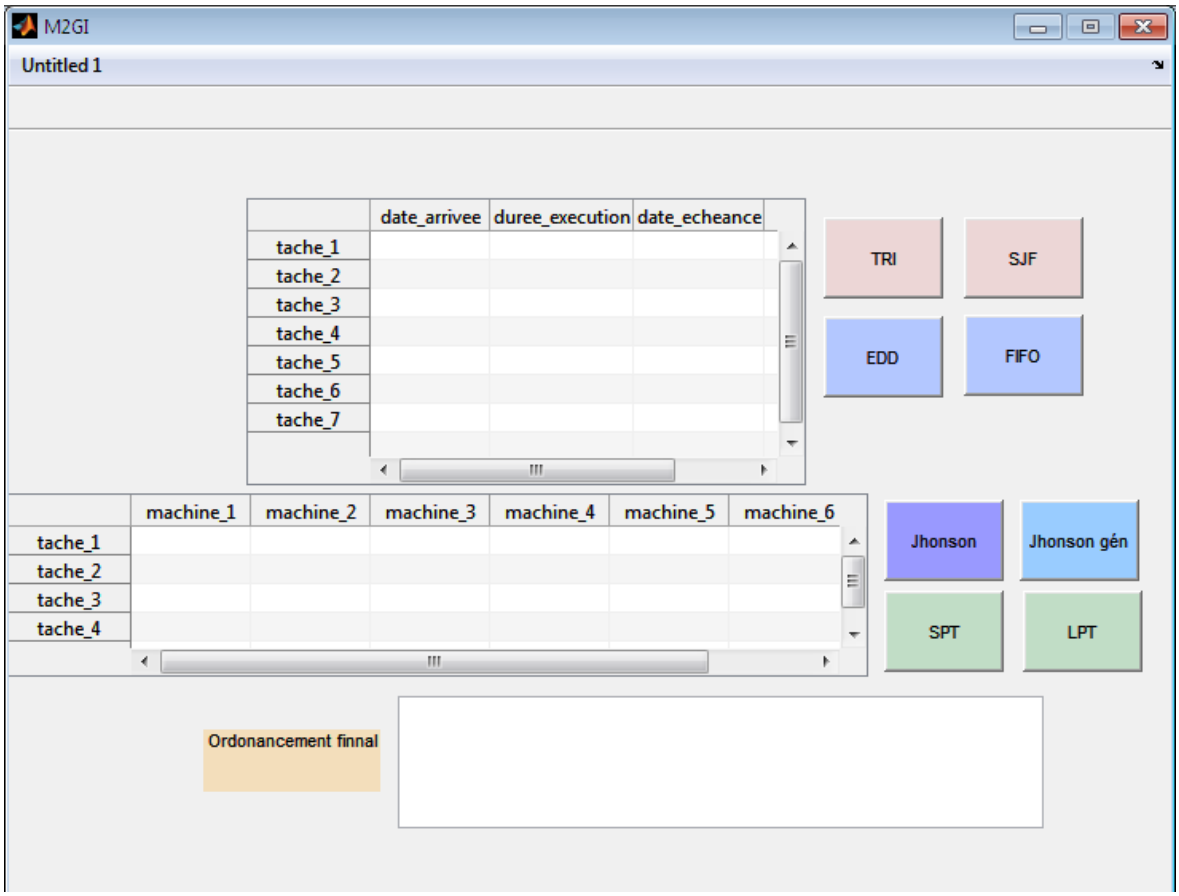


Figure III.9 Page de l'interface.

L'application présenté par l'organigramme suivant :

Figure III.10 Organigramme de fonctionnement.

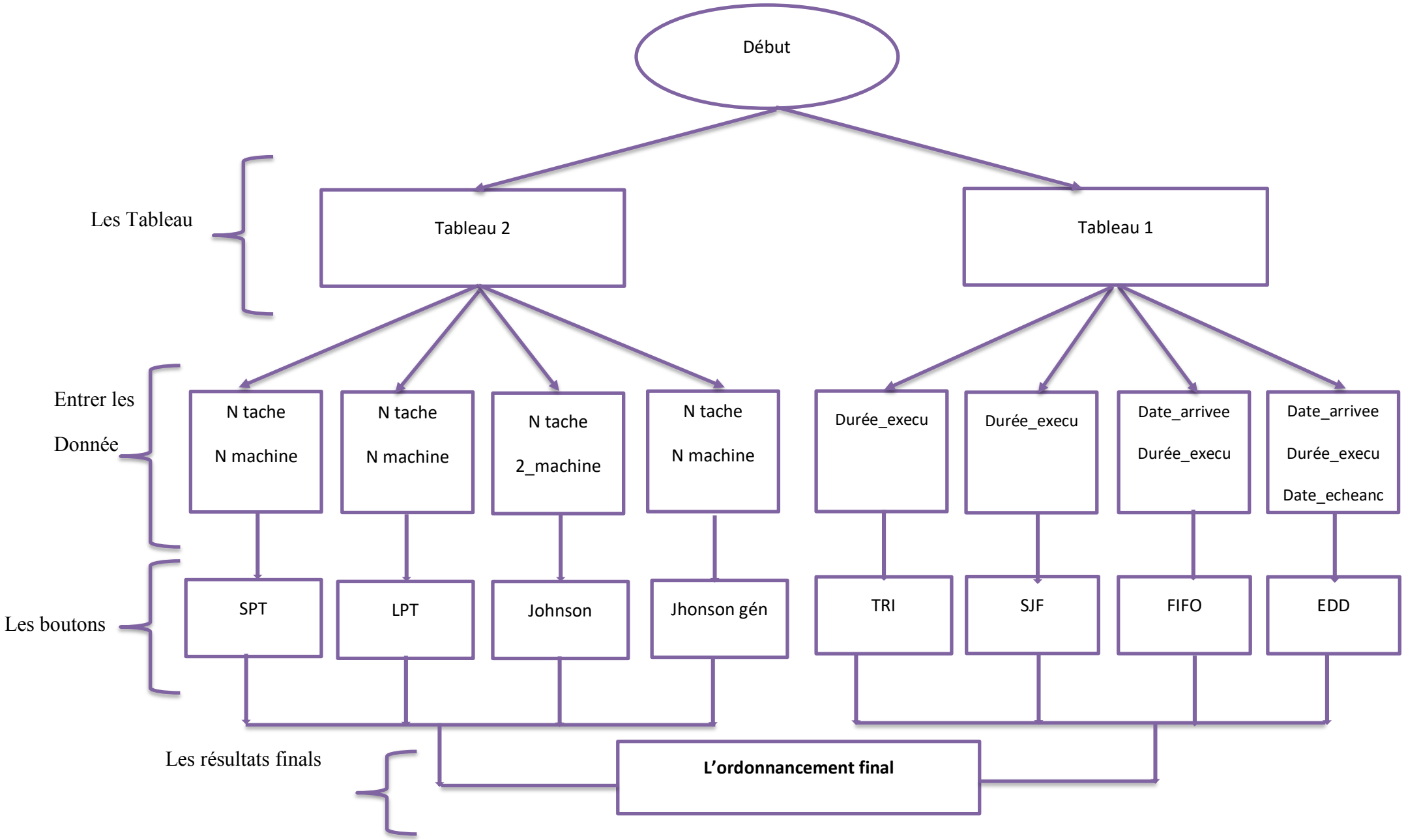


Figure III.10 Organigramme de fonctionnement.

B. L'utilisation de l'application :

Application dans MATLAB qui résout les problèmes d'ordonnancement. Voici comment un utilisateur peut l'utiliser :

1. Lancez l'application : ouvrez MATLAB et exécutez le fichier principal de l'application.
2. L'interface utilisateur : une fenêtre de l'interface utilisateur s'affiche avec différents éléments interactifs, tel que des cases vide pour entrer les paramètres précédents et des boutons qui sont représenté par les algorithmes précédents.
3. Saisie des paramètres : utilisez les éléments de l'interface pour entrer les paramètres du problème d'ordonnancement d'atelier par exemple, vous pouvez saisir nombres des taches et nombres des machines, et cliquez sur le bouton ''Johnson généralisé ''dans l'interface pour lancer le processus de résolution du problème.
4. Attendez les résultats : l'application utilise méthodes exactes comme : algorithme Johnson et algorithme SJF et des méthodes approchées comme : les algorithmes EDD, FIFO, TRI, SPT, LPT en fonction des paramètres que vous avez spécifiés. Attendez que l'application effectue les calculs.
5. Affichage des résultats : une fois que les calculs sont terminés, les résultats sont affichés dans l'interface dans la fenêtre ''ordonnancement final''.
6. Analyses des résultats : l'utilisateur peut maintenant analyser les résultats affichés dans l'interface .il peut les interpréter, les comparer ou les utiliser pour prendre des décisions en matière d'ordonnancement d'atelier.
7. Répétez si nécessaire : si vous souhaitez résoudre le problème avec nouveaux paramètres ou exécuter une autre instance du problème d'ordonnancement d'atelier, vous pouvez modifier les paramètres dans l'interface et cliquer à nouveau sur le bouton des algorithmes précédents.

   Cet exemple montre comment un utilisateur peut utiliser une application MATLAB avec interface pour des problèmes d'ordonnancement d'atelier.

C. Exemple :

On a choisi la méthode de (EDD) pour faire un exemple de simulation sur notre application.

Nous avons pris le même exemple dans la simulation De méthode (EDD) dans ce chapitre :

	Date_arrivee	Duree_execution	Date_echeance
Tache_1	6	5	8
Tache_2	8	2	9
Tache_3	0	3	8
Tache_4	2	6	9
Tache_5	4	4	7

Tableau **III**.1 : Présentation d'un exemple l'algorithme de EDD.

Taper les valeur sur le premier tableau (voir figure 11) :

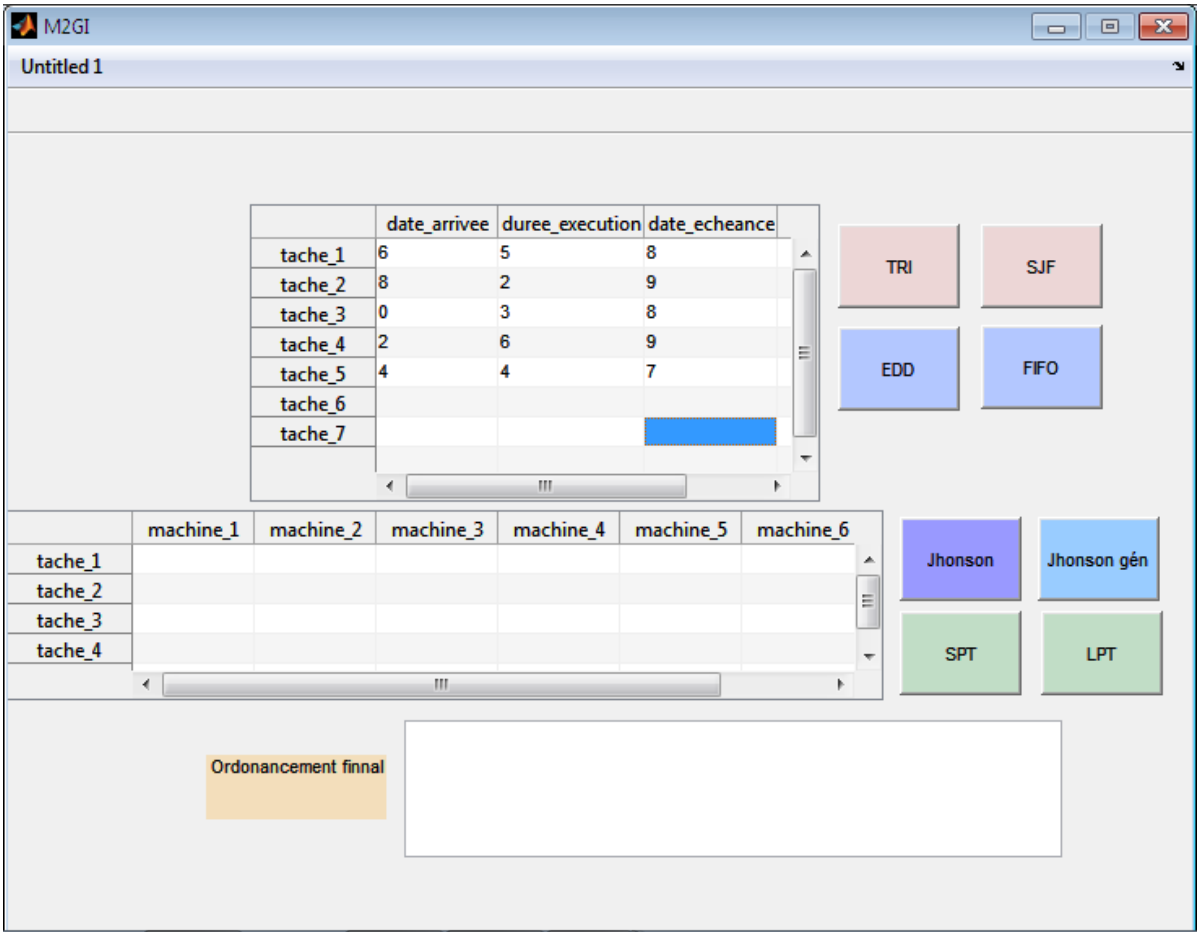


Figure III.11 Page de l'interface (exemple).

Les résultats de simulation (voire la figure 12) :

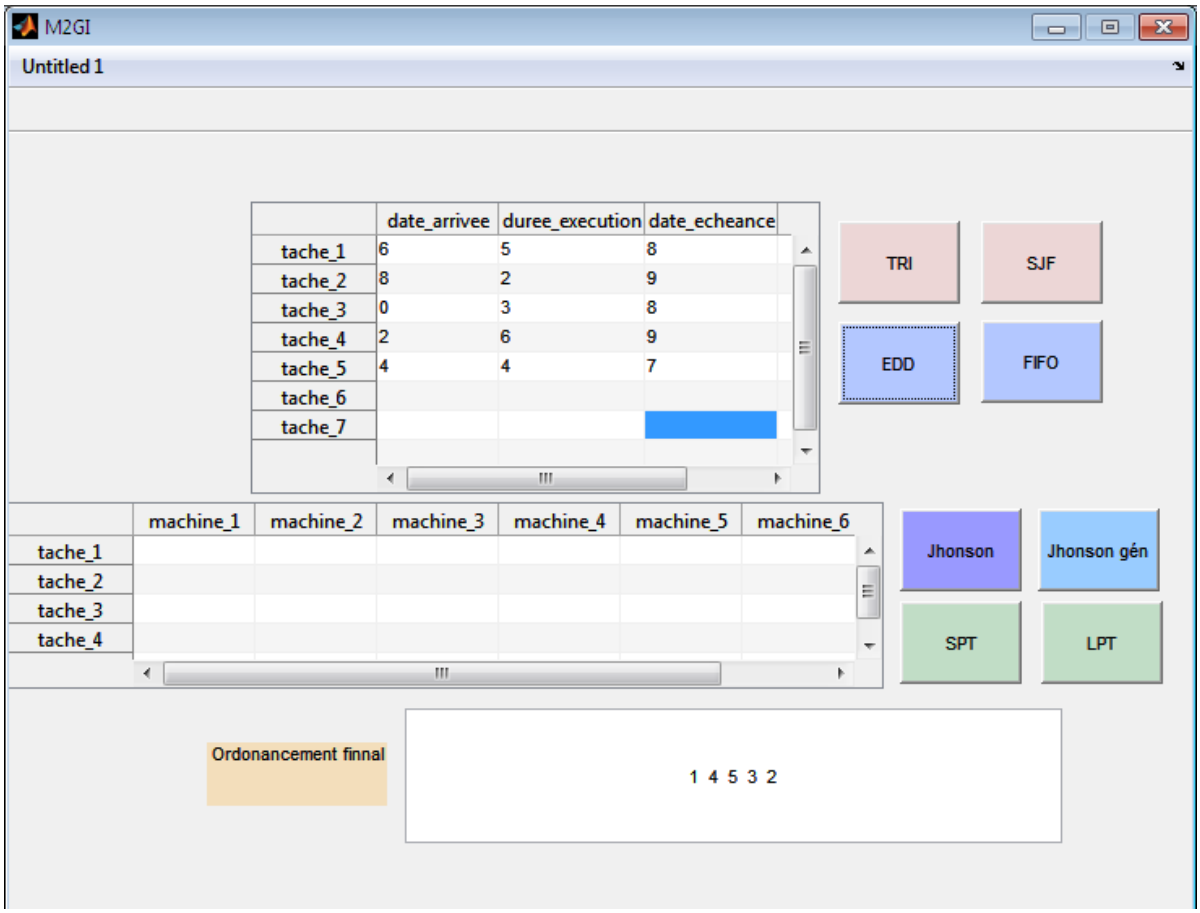


Figure III.12 Page de l'interface(résultats).

d'après ce exemple nous avons remarqué que le programme nous permet de trouver la meilleure solution avec un pourcentage presque de 100%. Les résultats obtenus sont très motivants.

III.5 Conclusion :

Nous avons présenté dans ce chapitre les méthodes de résolution les problèmes d'ordonnancement d'atelier simulé sur MATLAB pour améliore l'efficacité de la planification des opérations industrielles avec des exemples et leurs résultats.

Nous avons développé une application sur MATLAB permettant de résoudre les problèmes ordonnancements d'atelier avec une interface utilisateur conviviale.

En somme, ce chapitre nous a permis de mieux comprendre les méthodes de résolution des problèmes d'ordonnancement d'atelier.

Conclusion générale

Conclusion générale

Dans ce projet, nous avons abordé, les principales caractéristiques des problèmes d'ordonnancement de la production en générale et les différents types de problèmes d'ordonnancement d'atelier. Ces problèmes sont présentés par l'occurrence des nouvelles commandes qui doivent être insérées dans le programme de production courant de ce système.

Nous avons traité en particulier la simulation des méthodes de problèmes d'ordonnancement d'atelier par **MATLAB** avec des exemples et leurs résultats finals sous formes des vecteurs et leurs plots.

L'objectif principal de ce projet de fin d'étude est de concevoir un tel simulateur d'aide à l'ordonnancement des systèmes automatisés de production. Nous avons développé un modèle de simulation réaliste, capable de reproduire fidèlement le fonctionnement du système étudié. Ce simulateur sera équipée d'un moteur de calcul performant, permettant de résoudre efficacement les problèmes d'ordonnancement d'atelier complexes dans un court laps de temps.

Références bibliographiques

[1] H. Boukef ben othman, Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques optimisation par algorithmes génétiques et essaims particulaires, Tunis, Doctorat, 2009.

[2] N. Mouhoub, Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet, Setif, Doctorat, 2011.

[3] Harrath, "Contribution à l'ordonnancement conjoint de la production et de la maintenance : Application au cas d'un job shop," Université de Franche-Comté, 2003.

[4] K. Mesghouni, Application des algorithmes évolutionnistes dans les problèmes d'optimisation en ordonnancement de la production, Lille, Doctorat, 1999.

[5] M. A. Shahzad, "Une Approche Hybride de Simulation-Optimisation Basée sur la Fouille de Données pour les Problèmes d'ordonnancement," École polytechnique de l'Université de Nantes, 2011.

[6] R. Djeridi, "Contribution à la maîtrise de la disponibilité de systèmes complexes : Proposition d'une méthode de l'ordonnancement proactif de la maintenance," École Nationale Supérieure d'Arts et Métiers ParisTech, 2010.

[7] A. Hassam ahmed, Développement et analyse de méthodes d'ordonnancement temps réel pour les systèmes flexibles de production, Tlemcen, Doctorat, 2012.

[8] A. Karray, Contribution à l'ordonnancement d'ateliers agroalimentaires utilisant des methodes d'optimisation hybrides, Tunis, Doctorat, 2011.

[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey," *Ann Discret.Math.*, vol. 5, pp. 287–326, 1979.

[10] M. R. Garey and D. S. Johnson, "Computer and intractability," *A Guid. to Theory NP- Completeness*, 1979.

[11] A. Caumond, "Le problème de jobshop avec contraintes : modélisation et optimisation,"Université Blaise Pascal Clermont Ferrand II, 2006.

[12] Y. Bahmani, Optimisation multicritère de l'ordonnancement des activités de la production et de la maintenance intégrées dans un atelier Job Shop, Batna, Doctorat, 2017.

[13] G. Javel, Organisation et gestion de la production, 4ème édition, Dunod, 2010.

[14] Driss, Analyse d'un système job shop aspect ordonnancement, Batna, Doctorat, 2016.

[15] HADRI Abdelkader, « L'ORDONNANCEMENT PAR INSERTION EN TEMPS REEL DE LA PRODUCTION DANS UNATELIER FLEXIBLE », Mémoire de Magister, Université de *Batna*, (2012).

[16] BOURZIK MOHAMMED, « Problèmes d'Ordonnancement d'Atelier », Mémoire de Licence, UNIVERSITE SIDI MOHAMED BEN ABDELLAH, FES, (2016).

[17] Z. FATHI, « Etude d'un problème d'ordonnancement Job-Shop avec des contraintes de transport », Mémoire Master informatique, départ. D'informatique, Univ. M'SILA, Sept.2021, Algérie.

[18] M. SOUIER, « Algorithmes, Modèles et Principes de base d'ordonnancement de la production », Support de Cours et exercices, École supérieure en sciences appliquées, univ. Tlemcen, juin.2016, Algérie.

[19] M.Pegon «TP 7 Spéciale BCPST », Support de TP, Université de Lille, , 2016.

[20] M.Mokaddem «Systèmes d'exploitation », Support de Cours, Université de Oran 1 Ahmed Ben Bella , consulté le 29/05/2023.

[21] SILBERSCHATZ, A. et P.B. GALVIN, Operating System Concepts. 8th Edition, Addison Wesley.2012.

[22] ELLACHE Nour El Islam « Développement et implémentation d'un solveur bio inspiré pour la résolution d'un problème d'ordonnancement d'atelier » Mémoire de Master