REPUBLIQUE ALGERIENNE DEMOCRATIQU ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DE MATHEMATIQUES ET D'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'études

Présenté pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Informatique Décisionnel et Optimisation

Par : DRIF Fadila et KESSARI Nadia

SUJET

LE Flow Shop Hybride à deux étages

Soutenu publiquement le : / /2022devant le jury composé de :

Dr.	**UMB M'sila**	**Président**
Dr. MOUHOUB Nassereddine	**UMB M'sila**	**Rapporteur**
Dr.	**UMB M'sila**	**Examinateur**

Promotion : 2021 /2022

Dédicaces

A nos très chers parents

Remerciements

On remercie Dieu le tout puissant, qui nous a donné la force et la patience pour l'accomplissement de ce travail.

Mes remerciements s'adressent tout naturellement d'abord à notre directeur de thèse, Monsieur Mohoub Nassereddine, Professeur à l'université de M'sila d'avoir accepté de diriger notre thèse. Nous souhaitons lui exprimer nos profonde gratitude pour son accompagnement constant, ses conseils, son soutien, sa disponibilité, ses encouragements.

nous tiens aussi à adresser mes remerciements les plus sincères et mon profond respect à Monsieur Ben Azzi Makhlouf, Professeur à l'université de M'sila .

À Monsieur Pr. Gasmi Abdelkader et Monsieur Mr. Khettaf Abdelouahab , Maîtres enseignants à l'université de M'sila, qui nous ont fait l'honneur d'accepter d'être membres de ce jury de thèse et d'avoir consacré une partie de leur temps pour examiner ce travail .

Nous tenons également à remercier les personnes avec qui nous avons directement travaillé ou qui nous ont soutenu. Nous ne citons pas les noms pour ne pas oublier personne, mais nous voudrons ici les remercier tous très chaleureusement, leu s souhaitant plein de bonheur et de réussite .

Un grand Merci à nos parents qui nous ont donné la chance de poursuivre notre études et qui nous ont appris à surpasser les moments difficiles .

TABLE DES MATIERES :

Liste des Figures

Liste des Tables

Liste des Algorithmes

Introduction générale

L'industrie actuelle se caractérise par une forte demande de produits personnalisés de bonne qualité et à bas prix, dans des délais de plus en plus raccourcis. En effet, l'ouverture des marchés internationaux, ainsi que l'évolution et la mondialisation ont poussé les industriels à se diriger vers des systèmes de fabrication de plus en plus flexibles, ce qui a imposé la mise en cause de plusieurs habitudes de production et en particulier la gestion des ateliers de production qui joue un rôle crucial dans la productivité et le raccourcissement des délais de production. Un système de production est dit flexible s'il peut assurer la production simultanée de plusieurs types de pièces avec des quantités variables et s'il est capable de s'adapter à la production de nouveaux produits pour lesquels le système n'a pas été étudié. [12], [13].

La productivité peut être affectée directement par la qualité de l'ordonnancement des opérations sur les machines, car un atelier de production peut réaliser une grande variété de produits avec des coûts réduits, grâce à une meilleure utilisation des ressources. Le domaine d'application de l'ordonnancement est vaste : par exemple, la gestion de la charge des processus en informatique, la gestion de la production dans l'industrie, la gestion de projets, etc.

Le problème d'ordonnancement est classé parmi les problèmes fortement combinatoires, et il est toujours renouvelable, car jusqu'à maintenant, il n'existe aucune méthode de résolution générale et de faible complexité algorithmique. [14].

On trouve une grande variété de problèmes d'ordonnancement qui sont liés à plusieurs paramètres : les tâches ou opérations (préemptives et non préemptives, indépendantes ou non), les paramètres relatifs aux ressources (renouvelables, consommables), types de contraintes sur les tâches (précédence, disjonctions), critères d'optimalité (minimisation de la durée totale d'achèvement de toutes les tâches « makespan », minimisation du retard total des tâches, minimisation de la charge des machines, etc.).

La résolution du problème d'ordonnancement consiste à attribuer une ressource à chaque opération et à organiser l'ordre d'exécution des tâches en attribuant une date de début à chaque tâche, de telle sorte à respecter toutes les contraintes considérées dans le but d'optimiser un certain nombre d'objectifs.

Intuitivement, la résolution d'un problème d'ordonnancement passe par deux étapes fondamentales : la première étape qui consiste à identifier et à modéliser le problème en décrivant les contraintes qui doivent être respectées et en cernant les critères à optimiser. La deuxième étape se concrétise par la recherche de la méthode la plus adéquate pour la résolution du problème considéré.

Dans cette optique, ce travail propose un modèle de résolution du problème d'ordonnancement d'atelier de type Flow Shop Hybride (FSH). On trouve que ce type d'atelier correspond à une réalité industrielle où des applications de différents domaines industriels seront présentées.

Soit un processus de production en série. Les produits fabriqués passent dans un premier temps dans un premier étage, puis dans une seconde, etc. Supposons qu'il y a « k » étages en série. Dans chacun d'eux, on considère que les machines sont regroupées en parallèles. Un groupe contient un ensemble de machines susceptibles d'exécuter une même opération. Elles sont alors équivalentes dans leur fonctionnement mais pas forcément dans leurs performances. La durée d'une opération peut dépendre non seulement des ressources choisies dans le groupemais aussi du nombre de ressources affectées à l'opération (si la ressource est une machine, on peut parler alors de machines parallèles identiques, uniforme ou différentes). On supposera qu'il n'existe qu'un groupe de ressources par étage (où chaque produit subit un nombre d'opérations au plus égal à k, k étant le nombre d'étages).

Le problème d'ordonnancement des ateliers Flow Shop Hybride est difficile, car de point de vue complexité, il fait parti de la classe NP-Difficile.

Ce mémoire est organisé de la manière suivante :

Dans le chapitre 1, nous introduisons la généralité sur l'ordonnancement. Ensuite, nous enchainons avec les différents formulations et représentations d'un problème d'ordonnancement, ainsi que sa complexité. puis on cernant les classifications nécessaires des ateliers et des problème d'ordonnancement.

Dans le chapitre 2, nous présentons les formulations des problèmes d'optimisation et ses méthodes de résolution à savoir les approches exactes et les approches approximatives.

Le chapitre 3 explicite algorithme Guirchoun et Martineau adaptés aux problèmes d'ordonnancement d'atelier de type Flow Shop Hybride . Au début, nous présentons les différentes notations et formulations mathématique. Ensuite, nous passerons aux cas particuliers : problème avec $p_{i,2} \leq 1$, $p_{i,2} > 1$ . .Après, nous présenterons l'algorithme Guirchoun et Martineau .

Le chapitre 4 portera sur l'implémentation de notre démarche d'ordonnancement, nous proposons des exemples pour illustrer et nous ferons une expérimentation .

Nous finirons notre travail par une conclusion générale qui présente un résumé de ce qu'a été étudié dans ce mémoire, les résultats obtenus et le travail qui reste à faire pour l'accomplissement de cette étude.

CHAPITRE 01

« L'ORDONNANCEMENT »

1.Introduction

La réalisation d'un projet nécessite souvent une succession de tâches auxquelles s'attachent certaines contraintes :

- de temps, relatives aux délais à respecter pour l'exécution des tâches.
- d'antériorité**,** où certaines tâches doivent s'exécuter avant d'autres.
- de production, concernant le temps d'occupation du matériel ou des hommes qui l'utilisent,...

Les techniques d'ordonnancement dans le cadre de la gestion d'un projet ont pour objectif de répondre au mieux aux besoins exprimés par un client, au meilleur coût et dans les meilleurs délais, en tenant compte des différentes contraintes.

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 laphase de planification.
- le contrôle, qui consiste à effectuer une comparaison entre planification et exécution,soit au niveau des coûts, soit au niveau des dates de réalisation.

Ainsi, le résultat d'un ordonnancement est un calendrier précis de tâches à réaliser qui se décompose en trois importantes caractéristiques :

- l'affectation, qui attribue les ressources nécessaires aux tâches.
- le séquencement, qui indique l'ordre de passage des tâches sur les ressources.
- le datage, qui indique les temps de début et de fin d'exécution des tâches sur les ressources.

Dans ce chapitre, quelques généralités sur les problèmes d'ordonnancement dans les ateliers de production. dont les spécificités : les types d'ateliers, les critères d'optimisation et les approches de résolution . Dans un deuxième temps, une description des principales méthodes d'optimisation utilisées.

2. GÈNÈRALITÈS SUR 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. Les problèmes d'ordonnancement sont présents dans tous les secteurs d'activités de l'économie, depuis l'industrie manufacturière jusqu'à l'informatique.

Ordonnancer le fonctionnement d'un système industriel de production consiste à gérer l'allocation des ressources au cours du temps, tout en optimisant au mieux un ensemble de critères. C'est aussi programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.

Ordonnancer peut également consister à programmer l'exécution des opérations en leur allouant les ressources requises et en fixant leurs dates de début de fabrication.

D'une manière plus simple, un problème d'ordonnancement consiste à affecter des tâches à des ressources à des instants donnés pour répondre au mieux aux besoins exprimés par un client, au meilleur coût et dans les meilleurs délais, tout en tenant compte des contraintes.

Les problèmes d'allocation des ressources, d'organisation des tâches, de respect des délais et de prise de décision en temps requis constituent autant de difficultés qu'il est nécessaire de surmonter dans la gestion des systèmes de production en milieu industriel.

Au niveau de l'entreprise, l'ordonnancement concerne plusieurs postes : les ventes, la production, la maintenance, etc. Son rôle est de plus en plus important, car il permet une gestion de ces différents postes qui peut être optimale.

Pour la bonne gestion de ces postes ainsi que des contraintes pouvant y être reliées, il est nécessaire :

- de déterminer les différentes opérations à réaliser, les dates correspondantes, les moyens matériels et humains à y affecter,
- d'exécuter ces opérations et de contrôler les coûts qui en découlent.

C'est ainsi que 'ordonnancement intervient pour permettre la meilleure gestion possible du système de production.

3. FORMULATION D'UN PROBLÈME D'ORDONNANCEMENT

Les problèmes d'ordonnancement apparaissent dans tous les domaines : informatique, industrie, construction, administration, etc .

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

3.1 LES TACHES

Une tâche i est une entité élémentaire localisée dans le temps, par une date de début et/ou defin, et dont la réalisation nécessite une durée préalablement définie.

Elle est constituée d'un ensemble d'opérations qui requiert, pour son exécution, certainesressources et qu'il est nécessaire de programmer de façon à optimiser un certain objectif.

On distingue deux types de tâches :

- *les tâches morcelables (préemptibles)* qui peuvent être exécutées en plusieurs fois,facilitant ainsi la résolution de certains problèmes,
- *les tâches non morcelables (indivisibles)* qui doivent être exécutées en une seule foiset ne sont interrompues qu'une fois terminées.

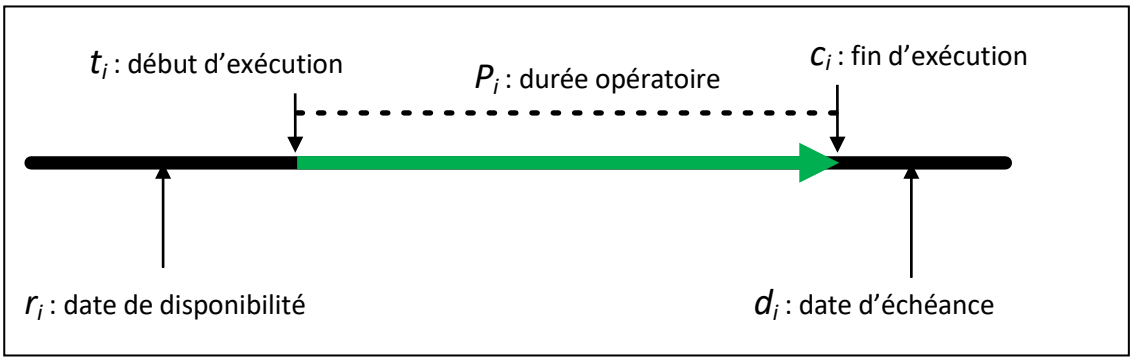


***Figure 1.1** : Caractéristiques d'une tâche i*

3.2 LES RESSOURCES

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. On trouve plusieurs types de ressources :

- *les ressources renouvelables,* qui, après avoir été allouées à une tâche, redeviennent disponibles (machines, personnel, etc),
- *les ressources consommables,* qui, après avoir été allouées à une tâche, ne sont plus disponibles (argent, matières premières, etc).

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. Par ailleurs, dans le cas des ressources renouvelables, on distingue principalement, les ressources disjonctives qui ne peuvent exécuter qu'une tâche à la fois et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité .

3.3 LES CONTRAINTES

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

- *les contraintes de ressources* : plusieurs types de contraintes peuvent être induites par la nature des ressources. A titre d'exemple, la capacité limitée d'une ressource implique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cette ressource.

Les contraintes relatives aux ressources peuvent être disjonctives, induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource, ou cumulatives impliquant la limitation du nombre de tâches àréaliser en parallèle.

- *les contraintes temporelles* : elles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement. 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 *i* doit précéder la tâche *j*,
  - des contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches.

3.4 LES CRITÈRES

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettant d'évaluer la qualité de l'ordonnancement établi.

Les critères dépendant d'une application donnée sont très nombreux; plusieurs critères peuvent être retenus pour une même application. Le choix de la solution la plus satisfaisante dépend du ou des critères préalablement définis, pouvant être classés suivant deux types, réguliers et irréguliers.

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

- *Les critères réguliers* constituent des fonctions décroissantes des dates d'achèvement des opérations. Quelques exemples sont cités ci-dessous:
  - la minimisation des dates d'achèvement des actions,
  - la minimisation du maximum des dates d'achèvement des actions,
  - la minimisation de la moyenne des dates d'achèvement des actions,
  - la minimisation des retards sur les dates d'achèvement des actions,
  - la minimisation du maximum des retards sur les dates d'achèvement des actions.
- *Les critères irréguliers* sont des critères non réguliers, c'est-à-dire qui ne sont pas des fonctions monotones des dates de fin d'exécution des opérations, tels que:
  - la minimisation des encours,
  - la minimisation du coût de stockage des matières premières,
  - l'équilibrage des charges des machines,
  - l'optimisation des changements d'outils.

La satisfaction de tous les critères à la fois est souvent délicate, car elle conduit souvent à des situations contradictoires et à la recherche de solutions à des problèmes complexes d'optimisation.

4 CLASSIFICATION DES ATELIERS

Une classification des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines et leur ordre d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré. Un atelier est caractérisé par le nombre de machines qu'ilcontient et par son type.

Comme le montre la figure1.2, On distingue les trois types d'ateliers suivants : flow-shop, job-shop et open-shop.

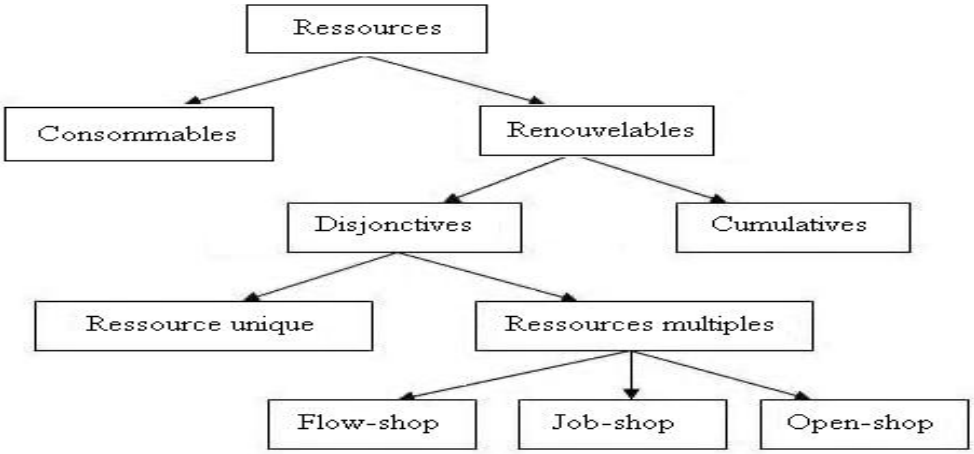


Figure 1. 2 - *Classification des types d'ateliers.*

4.3 LES ATELIERS DE TYPE FLOW-SHOP

Appelés également ateliers à cheminement unique, ce sont des ateliers où une ligne de fabrication est constituée de plusieurs machines en série; toutes les opérations de toutes les tâches passent par les machines dans le même ordre. Dans les ateliers de type *flow-shop hybride*, une machine peut exister en plusieurs exemplaires identiques fonctionnant en parallèle.[A]

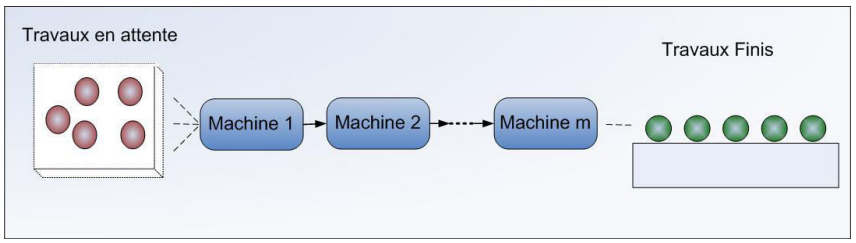


Figure 1. 3 – *La représentation de modèle Flow Shop.*

4.4 LES ATELIERS DE TYPE JOB-SHOP

Appelés également ateliers à cheminement multiple, ce sont des ateliers où les opérations sont

réalisées selon un ordre bien déterminé, variant selon la tâche à exécuter; le *job-shop flexible* est une extension du modèle job-shop classique; sa particularité réside dans le fait queplusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations.

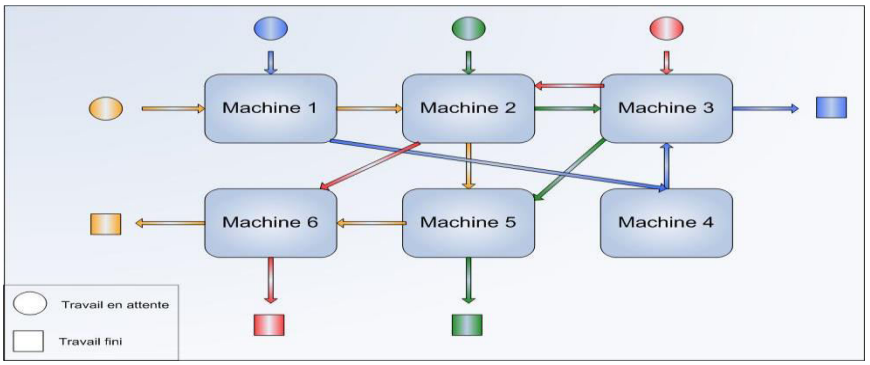


Figure 1. 4 – *La représentation de modèle Job Shop.*

4.5 LES ATELIERS DE TYPE OPEN-SHOP

Ce type d'atelier est moins contraint que celui de type flow-shop ou de type job-shop. Ainsi, l'ordre des opérations n'est pas fixé a priori; le problème d'ordonnancement consiste, d'une part, à déterminer le cheminement de chaque produit et, d'autre part, à ordonnancer les produits en tenant compte des gammes trouvées, ces deux problèmes pouvant être résolus simultanément. Comparé aux autres modèles d'ateliers, l'open-shop n'est pas couramment utilisé dans les entreprises[1].

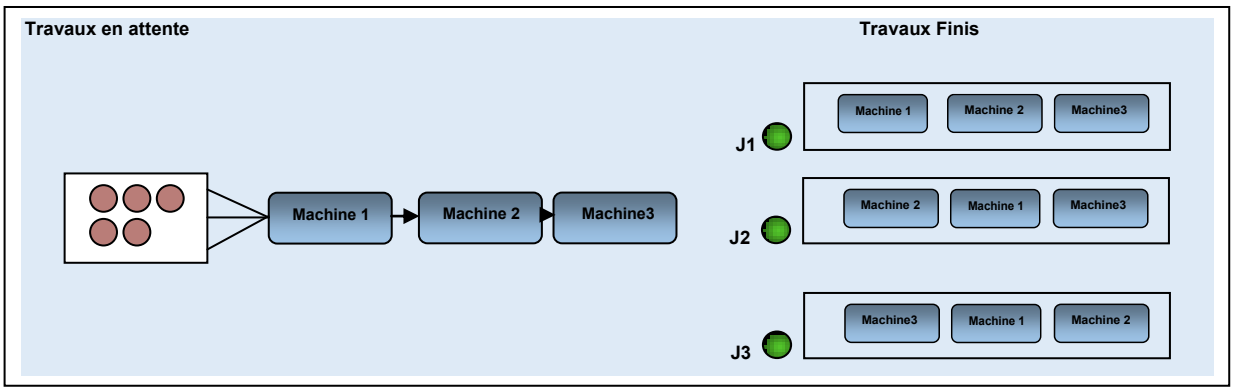


Figure 1. 5 – *La représentation de modèle Open Shop.*

5 REPRÈSENTATION DES PROBLEMES D'ORDONNANCEMENT

5.1 ORDONNANCEMENT ET CRITÈRES D'OPTIMALITÈ

Dans la théorie de l'ordonnancement, sont utilisés souvent les termes suivant : ‹‹ séquence››,‹‹ ordonnancement›› et ‹‹calendrier›› pour présenter ou décrire un problème d'ordonnancement. Ces termes sont souvent interchangeables. Pour mieux distinguer les nuances entre ces mots, il faut comprendre le sens de chaque terme. Ainsi, l'exécution d'une séquence de jobs est simplement l'ordre de passage des jobs sur les machines**.**

Dans une séquence, on n'a pas à préciser le temps de début ou de fin de chaque opération effectuée. Pour exécuter une séquence de jobs, on a besoin d'un calendrier contenant des informations concernant l'ordonnancement des jobs . Ainsi, un diagramme de Gantt permet de visualiser un ordonnancement donné. Chaque bloc de ce diagramme donne le temps de début et de fin d'exécution de chaque job.

Ces termes sont souvent utilisés lors de la résolution des problèmes d'ordonnancement. Dans ces derniers, on trouve aussi les critères de performance ou encore d'évaluation de la qualité d'un ordonnancement. Ces critères sont nombreux . En a distinguée 27. Par ailleurs, on différencie deux classes de critères de performance : les critères de performance réguliers et non réguliers.

Soit C_j et C'_j les dates de fin d'execution d'un job *j* dans deux ordonnancement différents de même tailles de séquence.

Définition 1

un critère de performance est dit régulière, s'il est une fonction *L* qui vérifie cette condition :

$$C_1 \leq C'_1, C^2 \leq C'^2, \dots\dots, C_n \leq C'_n \Rightarrow L(C_1, C^2, \dots, C_n) \leq L(C'_1, C'^2, \dots, C'_n)$$

À partir de cette définition et pour un critère de performance régulier, on peut dire qu'un ordonnancement est meilleur qu'un autre . Or ce n'est pas le cas pour un critère de performance non régulier, il ne vérifie pas cette condition .

Comme illustré à la figure 1.6, on peut voir trois sortes d'ordonnancement :

Ordonnancement actifs, semi-actifs et sans retard.

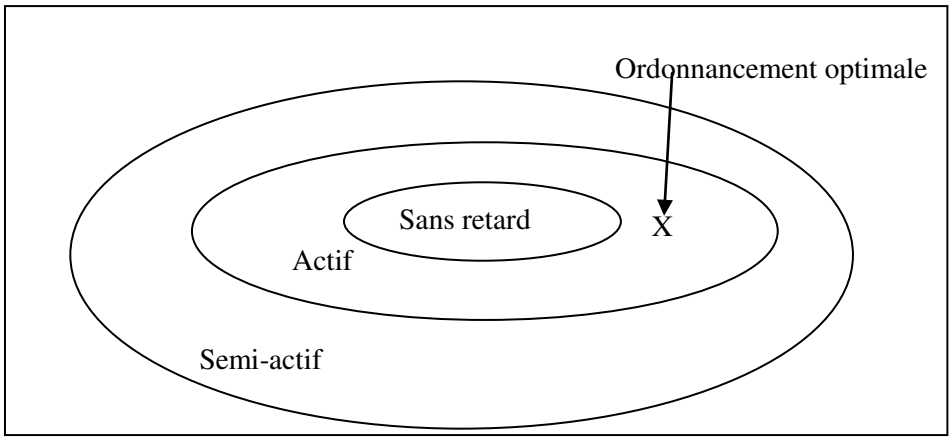


***Figure 1.6**- le diagramme de venn représentant les classes d'ordonnancement.*

Définition 2

Un ordonnancement est dit actif s'il est impossible d'avancer le début d'exécution d'une opération sans devoir retarder une autre tache ou violer une contrainte (de précédence, date de début au plus tôt,…)

Exemple 1-1

Soit , un problème de job-shop avec m = 3 et n = 2 , le tableau suivant présente les temps d'exécution de chaque job.

	J1	J2
M1	1	0
M2	3	3
M3	0	2

***Tableau 1.1** - Temps d'exécution du problème m = 3 et n = 2 de l'exemple 1-1.*

On suppose qu'on a un ordonnancement qui permet d'exécuter sur la machine M2 le job J2 puis le job J1(voir la figure 1-7). Il est alors clair que cet ordonnancement est actif, car si on place le J1 avant le job J2 sur M2 la troisième condition de la Définition 3(ci-dessous) ne sera pas vérifiée

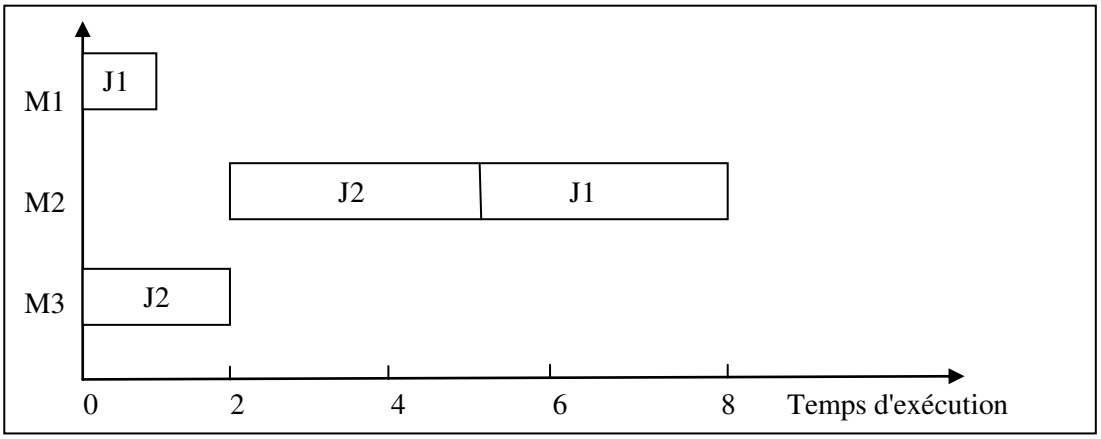


***Figure 1.7** - Un ordonnancement actif.*

Définition 3

Un ordonnancement est dit semi-actif si aucun tache ne peut être exécutée plus tôt sans changer l'ordre d'exécution sur les ressources ou violer une contrainte (de précédence, date de début au plus tôt,…)

Exemple 1-2

Soit , un problème de job-shop avec m = 3 et n = 2 , le tableau suivant présente les temps d'exécution de chaque job.

	J1	J2
M1	1	0
M2	1	2
M3	0	2

Tableau 1.2 *- Temps d'exécution du problème m = 3 et n = 2 de l'exemple 1-2.*

J1 sera exécuté sur M1puis sur M2 alors que J2 sera exécuté sur M2 puis sur M3 .

On suppose qu'on va ordonnancer sur la machine M2 le job J2 avant le job J1 (Figure 1.8). Il est alors clair que le job J2 commence son exécution sur M2 à un temps t = 2, alors que le job J1 à t = 4 . Cet ordonnancement est semi-actif.

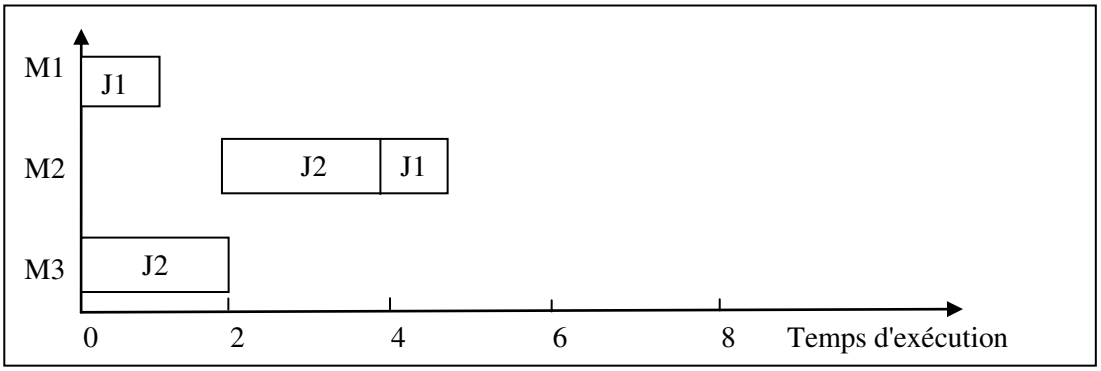


Figure 1.8 *- Un ordonnancement semi-actif.*

Théorème 1 [French, 1982]

Pour minimiser un critère régulier, il est suffisant de considérer un ordonnancement semi-actif.

Dans ce mémoire , notre but est de minimiser la durée totale d'accomplissement des jobs, appelé également le makespan. Il faut alors minimiser la fonction suivante :

$$c_{max} = max_{1<i<n}\{c_i\}$$

Le makespan représente le temps de fin d'exécution de dernier job dans une séquence . Il est l'un des critères les plus utilisés pour évaluer le coût d'un ordonnancement. En minimisant ce critère, on peut améliorer le rendement et réduire le temps moyen d'inactivité des machines . La minimisation du

makespan s'accompagne généralement de contraintes qui peuvent être temporelles ou liées aux ressources . Les contraintes temporelles se divisent en deux catégories : des contraintes de temps alloué (impératif de gestion : délai de livraison , disponibilité, achèvement) et des contraintes d'antériorité(cohérence technologique : gammes de fabrication , inégalité de potentiels : précédence).

Les contraintes liées aux ressources peuvent être des contraintes disjonctives (une tâche i doit s'exécuter avant ou après une tâche j) ou des contraintes cumulatives (respect des capacités des ressources).

On peut aussi considérer d'autres critères de performance , tels que le temps moyen d'achèvement des jobs, le temps totale de traitement, le temps de retard total, le temps d'attente des jobs, le taux d'occupation de machines , le nombre de job en retard , le temps de séjour d'un job dans le système avant sa réalisation , etc.

Dans ce mémoire , nous étudions le problème de flow-shop à deux étages et nous avons pour objectif de minimiser le makespan.

5.2 LE DIAGRAMME DE GANTT

Tout ordonnancement peut être représenté par l'intermédiaire d'un diagramme qu'on appelle diagramme de Gantt. Ce dernier, représentant un tableau mural, est un outil permettant de visualiser dans le temps les diverses tâches composant un projet. Dans un diagramme, on a deux axes perpendiculaires. L'axe horizontal représente les unités de temps , tandis que l'axe vertical représente les machines.

Exemple 1-3

Soit , m = 2 et n = 3 , le tableau suivant présente les temps d'exécution de chaque job.

	J1	J2	J3
M1	4	2	3
M2	2	5	2

Tableau 1.3 *- Temps d'exécution du problème m = 2 et n = 3 de l'exemple 1-3.*

La figure 1.9 représente le diagramme de Gantt associé à un ordonnancement. À partir de ce diagramme on peut déterminer la valeur de makespan ou d'une autre critère tel que : le temps de retard, etc. On peut alors voir si la solution en question est optimale.
Le diagramme de Gantt nous permet de représenter les différents ordonnancements de jobs sur les machines[2] .

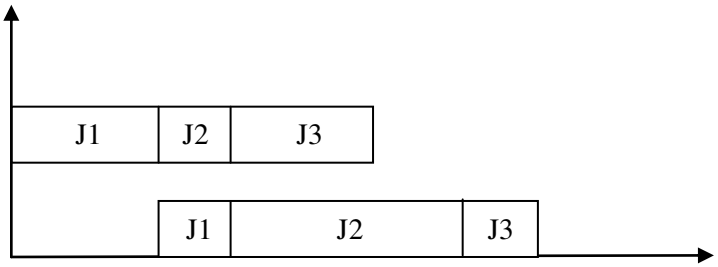


Figure 1.9 *- Diagramme de Gantt.*

5.3 NOTION DE COMPLEXITÈ DES PROBLEMES

La théorie de la complexité a pour but d'apporter des informations sur la difficulté théorique d'un problème à résoudre. Elle permet de classer " du point de vue mathématique "les problèmes selon leur difficulté.

La complexité analyse le temps nécessaire pour obtenir une solution (on peut également s'intéresser à la mémoire nécessaire, mais nous ne nous intéresserons pas ici à cet aspect de la complexité). On peut considérer la durée moyenne ou la durée dans le pire des cas. Nous traitons essentiellement ce dernier cas.

A première vue, comment définir un algorithme efficace ? Pour un problème donné, chercher un algorithme efficace, veut dire trouver un algorithme où le temps nécessaire à son exécution ne soit pas trop important. Un problème est dit facile si on peut le résoudre facilement, c'est-à-dire s'il ne fait pas trop de temps pour arriver à la solution. Donc, s'il existe un algorithme efficace pour un problème donné, alors ce dernier est dit facile. Un problème pour lequel on ne connaît pas d'algorithme efficace, alors ce dernier est dit difficile.

Pour résoudre un problème d'ordonnancement, il ne suffit pas de prouver l'existence d'une solution, il faut également la construire, il est clair que construire la solution est plus difficile que de prouver son existence ce qui nous conduit donc à classer les problèmes comme étant difficiles ou faciles.

Les problèmes indécidables sont ceux pour lesquels aucun algorithme, quel qu'il soit, n'a été trouvé pour les résoudre. A l'opposé, les problèmes décidables sont ceux pour lesquels il existe au moins un algorithme pour les résoudre.

5.3.1 *La classe NP*

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

5.3.2 *La classe P*

Un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quelle que soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissance polynomiaux . Pourle reste de la classe NP, on n'est pas sûr qu'il n'existe pas un algorithme polynomial pour résoudre chacun de ses problèmes. Ainsi, on sait que P est inclue dans NP mais on n'a pas pu prouver que P n'est pas NP.

5.3.3 *La classe NP-Complet*

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

5.3.4 *La classe NP-Difficile*

La classe NP-Difficile regroupe les problèmes (pas forcément dans la classe NP) tels que n'importe quel problème de la classe NP leur est polynomialement réductible.

Pour les problèmes d'ordonnancement à une machine, certains peuvent être résolus par un algorithme polynomial, certains autres sont démontrés NP-difficiles. Certains ne sont ni démontrésNP-difficiles, ni polynomialement résolvables. Ils restent donc ouverts. Et pour les problèmes d'ateliers, la plupart de ces problèmes ont été démontrés NP-difficiles [3].

6 .CLASSIFICATION DES PROBLEMES D'ORDONNANCEMENT

À cause de l'existence d'une très grande variété de problèmes d'ordonnancement , la notion à trois champs α β γ proposée initialement par Graham et ses collègues et reprise par la suite par plusieurs auteurs s'est rapidement émargée en tant que cadre de référence pour donner une classification claire tenant compte des caractéristiques de la plateforme et de l'environnement du problème d'ordonnancement étudié comme indique sur le Tableau 1.4.

Champs	Sous champs	Valeurs
Environnement (α)	Type de machine α( ) Nombre de machine (α_2)	ϕ,Q,P ,O ,F ,J ,FH , R ϕ, k
Contrainte (β)	Préemption (β_1) Ressources additionnelles (β_2) Précédence (β_3) Date de disponibilité (β_4) Durées opératoires (β_5) Dates de fin impérative (β_6) Nombre maximum d'opérations par tache (β_7) Propriété d'attente (β_8)	ϕ, *pmtn* ϕ, *res* ϕ, *prec ,uan ,tree ,chains* ϕ, r_i ϕ, $p_i = p$,$p^- \leq p_i \leq p^+$ ϕ, d_i^{++} ϕ, $n_i \leq n_{max}$ ϕ, *no - wait*
Objectif (γ)	/	Un des critères d'optimalité du Tableau 1.2

Tableau 1.4 *- Signification et valeurs des champs dans la notion de Graham.*

Les valeurs prises par le sous champ α_1 sont décrites dans le Tableau 1.5. Pour le champ β, uniquement les valeurs de la propriété de précédence (sous champ β_3) sont interprétées dans le Tableau 1.6 comme les valeurs affectées aux autres sous champs signifient la présence ou l'absence de la propriété considérée.

Bien que les valeurs affectées aux champ permettent de modéliser un très grand nombre de problèmes d'ordonnancement , multitudes extensions ont été proposées en particulier au niveau des champs β et γ dans le but de supporter d'autres catégories de problèmes particulières telles que la classe des problèmes stochastiques ou multicritères.

Ce processus d'extension est un axe évolutif ou il est très probable dans le future que d'autres contraintes et extensions soient encore créées [4] .

Valeur	Description
ϕ ou 1	Environnement à une seule machine
P	Environnement à machines identique parallèles
Q	Environnement à machines parallèles uniformes
R	Environnement à machines parallèles quelconques
O	Système Open shop
F	Système Flow shop
J	Système Job shop
FH	Système Flow shop Hybride

Tableau 1.5 *- Interprétation des valeurs du champ* α_1

Valeur	Description
Φ	Tâches indépendantes
prec	Tâches avec contraintes de précédence générales
uan	Tâches formant un réseau d'activités uniconnexe
tree	Tâches formant un arbre
chains	Tâches formant union de chaines

Tableau 1.6 *- Interprétation des valeurs du champ β_3.*

7. Conclusion

Dans ce chapitre, nous avons tout d'abord présenté les notions, les classification des ateliers et les différentes méthodes pouvant être utiliser pour la présentation et classifications des problèmes d'ordonnancement.

Le problème d'ordonnancement est un problème combinatoire qui est classé parmi les problèmes d'optimisation les plus difficiles. En effet, la complexité de l'ordonnancement dépend du domaine et de la taille du problème traité. On trouve plusieurs méthodes de résolution du problème d'ordonnancement qui sont classées en deux catégories : méthodes exactes et méthodes approchées. Le chapitre suivant permettra d'expliciter ces méthodes.

CHAPITRE 02

« MÈTHODES D'OPTIMISATION »

1. INTRODUCTION

Nous avons vu précédemment qu'il existe des problèmes d'ordonnancement de complexités différentes. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes de résolution sont largement utilisées pour appréhender les problèmes NP-difficiles.

Dans ce chapitre, nous exposons en générale les méthodes de résolution des problèmes d'optimisation qui sont classées en deux grandes catégories : les méthodes exactes et les méthodes approchées. On explique brièvement les méthodes les plus connues dans chaque catégorie.

2. LES MÈTHODES D'OPTIMISATION

Comme on l'a rappelé dans le paragraphe sur la complexité, la plupart des problèmes d'ordonnancement sont NP-difficiles. On ne peut donc pas espérer trouver un ordonnancement optimal en un temps raisonnable pour des problèmes de taille industrielle. Les méthodes utilisées sont donc des méthodes approchées.

Pour des problèmes de petite taille, nous pouvons obtenir une solution exacte. Une méthode exacte peut servir pour résoudre des sous-problèmes d'un problème de grande taille de manière optimale.

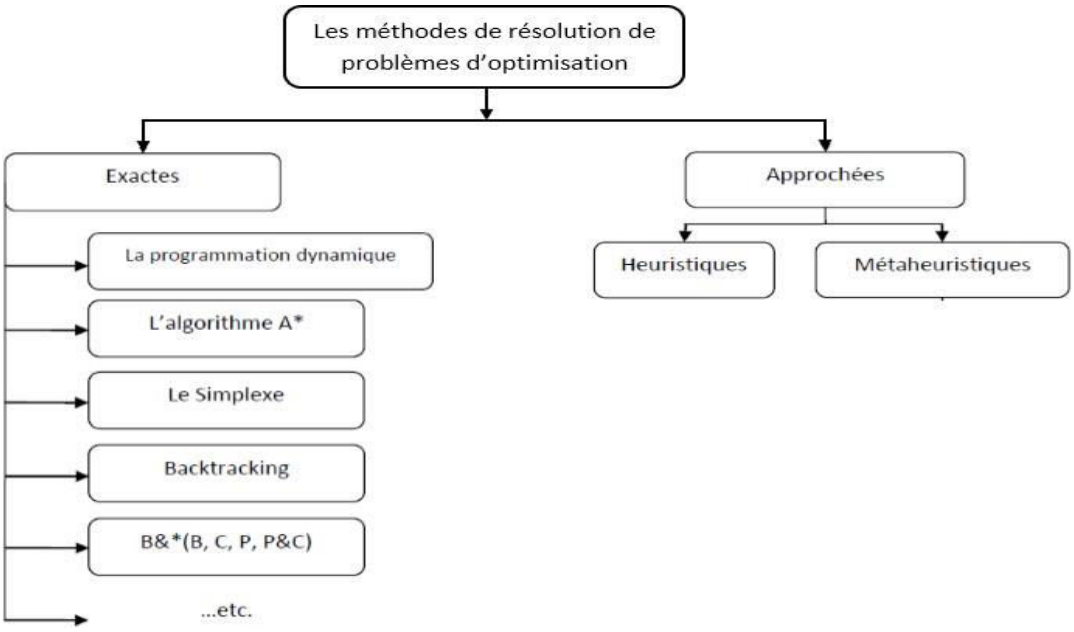


Figure 2.1 *- Classification de méthodes de résolution de problème d'optimisation.*

2.1 LES METHODES EXACTES

Les méthodes exactes utilisent surtout deux approches de résolution très connues: la programmation dynamique et les procédures par séparation et évaluation. Ces méthodes sont souvent utilisées pour résoudre les problèmes combinatoires de manière exacte, en ordonnancement tout particulièrement. Ce sont des méthodes d'énumération implicite:

L'énumération explicite construit toutes les solutions réalisables et retient une parmi les meilleures. L'énumération implicite consiste à explorer l'ensemble de toutes les solutions réalisables en éliminant des sous-ensembles de solutions moins intéressants sans avoir à les construire. Nous pouvons citer trois approches particulièrement célèbres : La programmation dynamique introduite par Bellman dans les années 50 , la méthode par séparation et évaluation (Branch and Bound en anglais : notée B&B) et la programmation linéaire[3].

2.1.1 LA PROGRAMMATION DYNAMIQUE

C'est une méthode d'optimisation introduite par Bellman en, 1957. Cette méthode est basée essentiellement sur la décomposition du problème en une série de sous-problèmes reliés entre eux par une relation de récurrence permettant de décrire la valeur optimale du critère à une étape donnée en fonction de sa valeur à l'étape précédente. La solution optimale du problème est obtenue en calculant les solutions des sous-problèmes les plus petits, pour ensuite en déduire petit à petit les solutions du problème complet. Elle est par exemple utilisée pour développer des algorithmes polynomiaux et pseudo polynomiaux pour résoudre les problèmes à une machine et à machines parallèles).

2.1.2 LA MÈTHODE BRANCH AND BOUND

L'algorithme Branch and Bound consiste à placer progressivement les tâches sur les ressources en explorant un arbre de recherche décrivant toutes les combinaisons possibles. Il s'agit de trouver la meilleure configuration donnée de manière à élaguer les branches de l'arbre qui conduisent à de mauvaises solutions. L'algorithme branch and bound effectue une recherche complète de l'espace des solutions d'un problème donné, pour trouver la meilleure solution. La démarche de l'algorithme Branch and Bound consiste à :

- diviser l'espace de recherche en sous espaces,
- chercher une borne minimale en terme de fonction objectif associée à chaque sous espace de recherche,
- éliminer les mauvais sous-espaces, reproduire les étapes précédentes jusqu'à l'obtention de l'optimum global.

2.1.3 LA PROGRAMMATION LINÈAIRE

C'est un outil d'optimisation très puissant qui permet de résoudre un grand nombre de modèles linéaires. Son utilisation demande que le problème posé puisse se ramener à l'optimisation d'une fonction de forme linéaire, en respectant un ensemble de contraintes elles aussi linéaires, fonctions des mêmes variables positives ou nulles. Des contraintes très variées peuvent être ajoutées de la même façon. L'étape de modélisation du problème peut s'avérer difficile, et la résolution d'un programme linéaire complexe peut demander un temps relativement long. Parmi les logiciels de résolution des problèmes de programmation linéaire, on peut citer LP-Solve, Cplex, Mosel-Xpress (qui est aussi un langage de modélisation et de résolution des problèmes d'optimisation).[5]

2.2 LES METHODES APPROCHÈES

Une méthode approchée est une méthode d'optimisation qui a pour but de trouver une solution réalisable de la fonction objectif en un temps raisonnable, mais sans garantie d'optimalité. L'avantage principal de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, D'un autre côté les algorithmes d'optimisation tels que les algorithmes de recuit simulé, les algorithmes tabous et les algorithmes génétiques ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires.

Les méthodes approchées englobent deux classes : les heuristiques et les métaheuristiques. La particularité qui différencie les méthodes métaheuristiques des méthodes heuristiques c'est que les métaheuristiques sont applicables sur de nombreux problèmes. Tandis que, les heuristiques sont spécifiques à un problème donné.

2.2.1 LES HEURISTIQUES

Les méthodes heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriquesqui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures. Plusieurs définitions des heuristiques ont été proposées par plusieurs chercheurs dans la littérature,parmi lesquelles:

Définition 2.2.1 « Une heuristique (règle heuristique, méthode heuristique) est une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre type de dispositif qui limite considérablement la recherche de solutions dans des espaces problématiques importants. Les heuristiques ne garantissent pas des solutions optimales. En fait, elles ne garantissent pas une solution du tout. Tout ce qui peut être dit d'une heuristique utile, c'est qu'elle propose des solutionsqui sont assez bonnes la plupart du temps ».

Définition 2.2.2 « Une méthode heuristique (ou simplement une heuristique) est une méthodequi aide à découvrir la solution d'un problème en faisant des conjectures plausibles mais faillible de ce qui est la meilleure chose à faire. ».

Définition 2.2.3 « Les heuristiques sont des ensembles de règles empiriques ou des stratégies qui fonctionnent, en effet, comme des règles d'estimation. ».

Parmi Les heuristiques, on peut citer :

- *L'algorithme de Johnson* : développée dans le cadre de la recherche d'une séquence dedurée minimale dans un atelier de type Flow-Shop de permutation à deux machines.
- Les algorithmes de liste dont le principe consiste à trier la liste des tâches selon une stratégie de décision appelée « règles de priorité » telle que :
  - *FIFO (First In First Out*) : la première tâche qui vient est la première tâcheordonnancée,
  - *LIFO (Last In Last Out)* : la dernière tâche qui vient est la première tâche ordonnancée,
  - *SPT (Shortest Processing Time)* **:** la tâche ayant le temps opératoire le plus court esttraité en premier lieu,
  - *LPT (Longest Processing Time)* : la tâche ayant le temps opératoire le plus importantest ordonnancé en premier lieu,
  - *EDD (Earliest Due Date)* : la tâche ayant la date due la plus petite est la plus prioritaire,
  - *SRPT (Shortest Remaining Processing Time)* : cette règle, servant à lancer la tâche ayant la plus courte durée de travail restant à exécuter, est très utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs,
  - *ST (Slack Time)* : à chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.
  - *FAM (First Available Machine)* : affecter les tâches à la première machine libre, etc.

2.2.2 LES MÈTAHEURISTIQUES

Les métaheuristiques d'optimisation sont des algorithmes généraux d'optimisation applicables à une grande variété de problèmes. Elles sont apparues à partir des années 80, dans le but de résoudre au mieux des problèmes d'optimisation. Les métaheuristiques s'efforcent de résoudre tout type de problème d'optimisation. Elles sont caractérisées par leur caractère stochastique, ainsi que par leur origine discrète.

Elles sont inspirées par des analogies avec la physique (recuit simulé, recuit micro- canonique), avec la biologie (algorithmes évolutionnaires) ou encore l'éthologie (colonies de fourmis, essaims particulaires). Cependant, elles ont l'inconvénient d'avoir plusieurs paramètres à régler. Il est à souligner que les métaheuristiques se prêtent à toutes sortes d'extensions, notamment en optimisation mono-objectif et multi-objectif.

Les métaheuristiques utilisent des recherches stratégiques afin d'explorer plus efficacementl'espace de recherche, et souvent se focalisent sur les régions prometteuses. Ces méthodes commencent par un ensemble de solutions initiales ou une population initiale, et après, elles examinent étape par étape une séquence de solutions pour atteindre, ou de s'approcher de la solution optimale du problème. Les métaheuristiques ont plusieurs avantages par rapport aux algorithmes traditionnels. Les deux avantages les plus importants sont la simplicité et la flexibilité. Les métaheuristiques sont souvent simples à implémenter, pourtant, elles sont capables de résoudre des problèmes complexes avec la capacité de s'adapter à plusieurs problèmes d'optimisation du monde réel, à partir du domaine de la recherche opérationnelle, d'ingénierie vers l'intelligence artificielle.

De nos jours les gestionnaires et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs très divers. Le problème à résoudre peut souvent s'exprimer sous la forme générale d'un problème d'optimisation, dans lequel on définit une ou plusieurs fonctions objectif que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés.

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

- heuristique qui vient du verbe heuriskein et qui signifie ‘trouver’.
- méta qui est un suffixe signifiant ‘au -delà’, ‘dans un niveau supérieur’.

2.2.2.1 LES PROPRITES FONDAMENTALES DES MÈTAHEURISTIQUES

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d’une solution optimale.

- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afinde déterminer des solutions (presque) optimales.

- Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure de recherche locale à des processus d'apprentissage complexes.

- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité

- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche.

- Les concepts de base des métaheuristiques peuvent être décrit de manière abstraite, sans faire appel à un problème spécifique.

- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problème traité, mais ces heuristiques sont contrôlées par une stratégie de niveau supérieur.

- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.

2.2.2.2 CLASSIFICATION DE MÈTHAHEURISTIQUE

On peut regrouper les métaheuristiques en deux grandes classes : les métaheuristiques à solution unique (c.à.d. évoluant avec une seule solution) et celles à solutions multiples ou population de solutions. Les méthodes d'optimisation à population de solutions améliorent, au fûr et à mesure des itérations, une population de solutions. L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité.

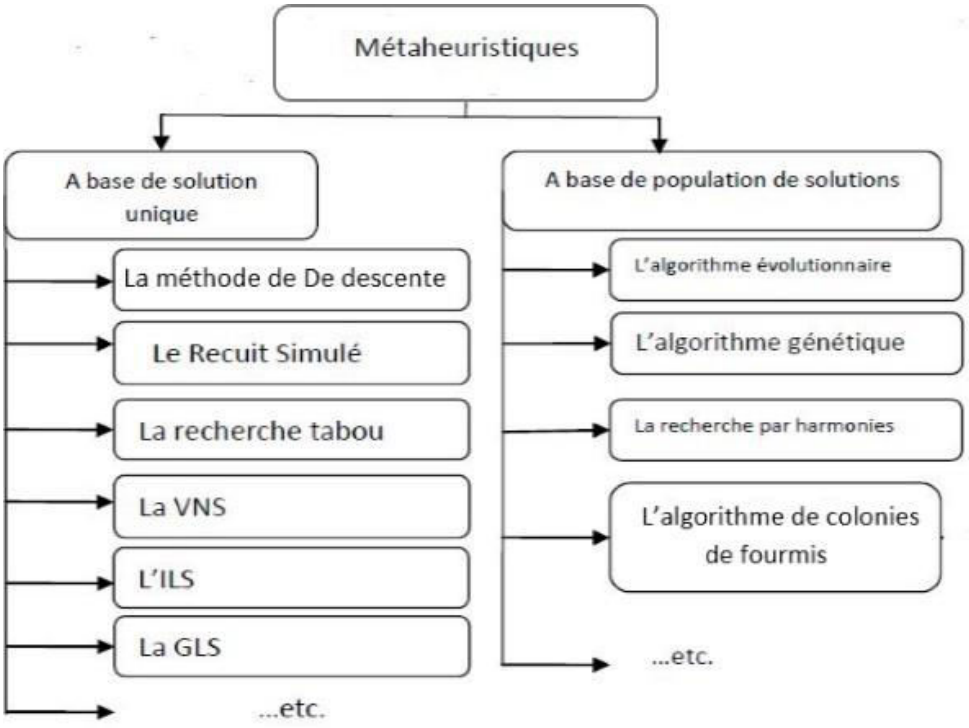


Figure 2.2 *- Classification des mètaheuristiques.*

2.2.2.3 NOTION DE PAYSAGE

La notion de paysage permet de décrire visuellement l'ensemble des valeurs que peut prendre une fonction objectif pour une paramétrisation donnée. Cette description permet notamment de voir s'il existe potentiellement de nombreux optima locaux. La description d'un paysage se fait en analogie avec la notion géographique du paysage. Ainsi un paysage peut être décrit comme une vallée, une plaine, un bassin, Cela correspond à la forme général. Ensuite, il est possible d'apporter une , information sur le nombre d'optima locaux existants. Ainsi, un paysage rugueux ou chaotique correspondra plutôt à un paysage possédant un nombre important d'optima locaux. Au contraire, un paysage tout à fait lisse amènera à priori vers un nombre restreint d'optima locaux. La figure 2.3 propose quelques exemples de forme de paysage.

L'étude du paysage d'une fonction objectif permet de guider le choix de la méthode de résolution à appliquer.

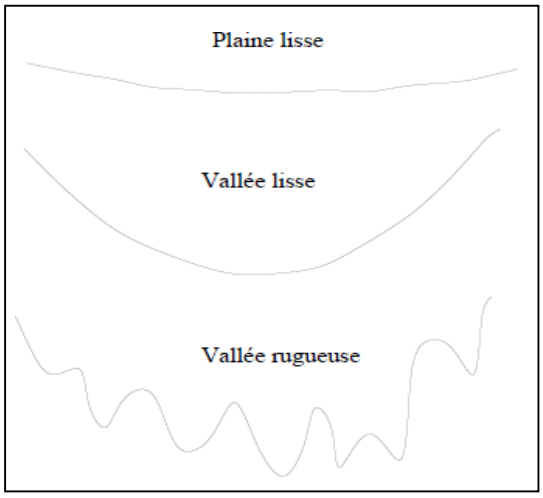


Figure 2.3 *- Exemples de forme de paysage.*

Le principe de diversification d'une méthode d'optimisation donnée correspond à sa capacité de parcourir aisément l'espace de recherche pour obtenir des solutions très différentes les unes des autres. Dans un paysage donné, cela se traduit par la possibilité de faire des déplacements plutôt horizontaux commence la est montré dans la figure 2.4. Cependant, comme cela est visible sur cette figure, la diversification seule n'amène pas forcément à des optima locaux mais à des solutions se trouvant potentiellement dans la zone d'un optimum local non encore trouvé jusqu'à maintenant.

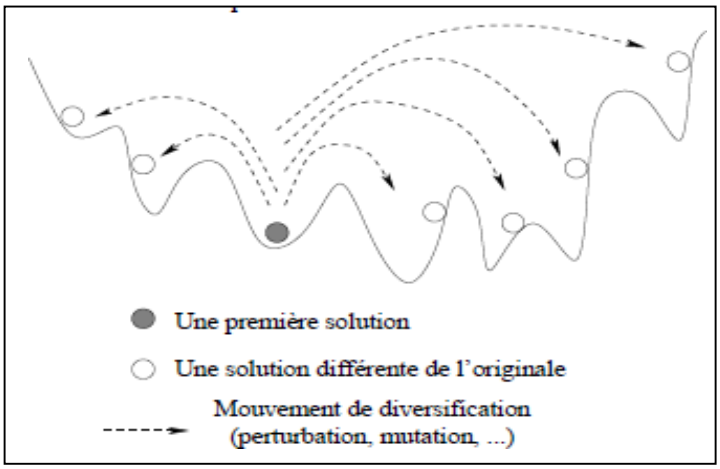


Figure 2.4 *- Processus de diversification d'une solution dans un paysage donné.*

Autant le principe de diversification essaye de déplacer les solutions dans d'autres zones de l'espace de recherche, autant le processus d'intensification vise à forcer une solution donnée à tendre vers l'optimum local de la zone à laquelle elle est attachée.

La figure 2.5 donne un exemple de processus d'intensification pour des solutions de départ données. Comme on peut le voir, une phase d'intensification correspond plutôt à un mouvement vertical dans le paysage pour accéder à un optimum local (peut-être l'optimum global). La force mais aussi la faiblesse de l'intensification est de rester cantonnée à la zone dans laquelle la solution se trouve sans possibilité de pouvoir sortir de celle-ci. Donc si les solutions que l'on veut intensifierne sont pas dans la zone de l'optimum global, il n'y a aucune chance que l'intensification seule permette d'y accéder.

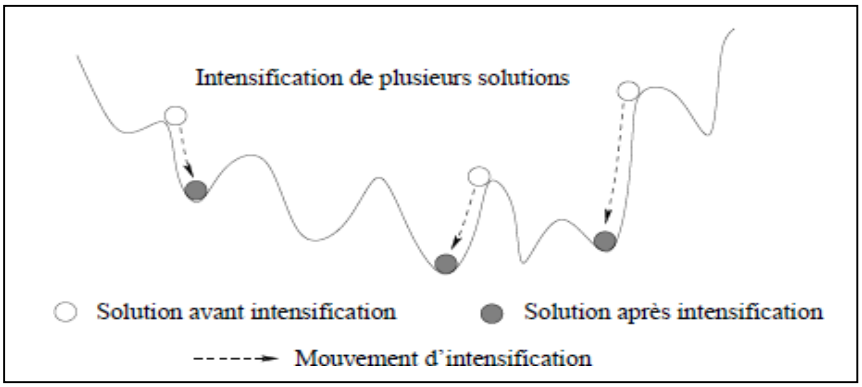


Figure 2.5 *- Processus d'intensification de solutions dans un paysage donné.*

3 LES MÈTAHEURISTIQUES À SOLUTION UNIQUE

Les méthodes à solution unique, plus connues sous le nom de *méthodes de recherche locale* ou encore *méthodes de trajectoire*, sont basées sur l'évolution d'une seule solution dans l'espace de recherche. Typiquement, les méthodes de recherche locale démarrent d'une solution unique, puis à chaque itération, la solution courante est déplacée dans un voisinage local en espérant améliorer la fonction objectif. Les méthodes à solution unique englobent principalement la méthode de descente, la méthode durecuit simulé et la recherche tabou **.**

3.1 LA MÈTHODE DE DESCENTE

La méthode de descente représente la technique la plus intuitive et la plus simple dans le domaine de l'optimisation. Appelée aussi *hill climbing* (par référence à un cas de maximisation de la fonction objectif), cette méthode fonctionne selon le principe suivant : en démarrant d'unesolution aléatoire, à chaque itération la solution courante se déplace vers une meilleure solution présente dans son voisinage jusqu'à ce qu'aucune amélioration ne soit plus possible. Le choix des solutions candidates dans le voisinage peut se faire de deux manières, soit évaluer la fitness de la fonction objectif pour toutes les solutions présentes dans le voisinage puis se déplacer versla meilleure, soit choisir une solution aléatoire dans le voisinage, puis se déplacer vers cette solution si elle améliore le critère, ou sinon choisir une nouvelle solution. La procédure générale de cette méthode est présentée dans l'Algorithme 2.1.

L'inconvénient majeur de la méthode de descente est qu'elle se trouve piégée dans le premier optimum local rencontré. L'amélioration possible de cette méthode consiste à lancer la méthode plusieurs fois en démarrant de différentes solutions initiales, ou dans un schéma plus complexe de réitération, comme la recherche itérée . L'inconvénient majeur de cette méthode peut, dans certains cas, s'avérer un atout considérable (ex. l'espace de recherche est convexe). De plus, la méthode de descente peut être utilisée comme première approche dans un problème d'optimisation pour avoir une idée sur les optima locaux du problème.

Algorithme 2.1 Algorithme de descente

Nécessite : La fonction objectif f

1: **Générer** une solution aléatoire S

2: **Calculer** la fitness f(S) associe a la solution initial S

3: **Initialiser** la solution optimale : S_{opt} Ω S

4: **Tant que** la condition d'arrêt n'est pas vérifiée **Faire**

5: **Générer** la liste des solutions dans le voisinage de la solution courante

6: **Trouver** la meilleure solution S' parmi les solutions voisines

7: **Si** $f(S') < f(S)$ **Alors**

8: $S \Omega S'$

9: $S_{opt} \Omega S$

10: **Fin Si**

11: **Fin Tant que**

12:**Retourner :** La solution optimal S_{opt}

3.2 LA MÈTHODE DE RECUIT SIMULÈ

La méthode du recuit simulé (en anglais "*Simulated Annealing*" (SA)) est une métaheuristique d'optimisation inspirée de la technique de simulation de Metropolis en mécanique statistique basée sur la distribution de Boltzmann. Le processus de recuit en métallurgie consiste à appliquer des cycles de chauffage et de refroidissement contrôlés à un matériau, afin de réorganiser sa structure cristallographique. Le but de cette opération est d'obtenir un matériau homogène de bonne qualité. À l'inverse de la technique de la trempe qui donne naissance à un état métastable, ce procédé permet d'éviter cet état caractéristique des minima locaux de l'énergie.

La transposition de la technique du recuit physique à la métaheuristique d'optimisation du recuit simulé est basée sur les analogies suivantes : la fonction objectif à optimiser est assimilée à l'énergie du matériau et la température est représentée par un paramètre de contrôle définissant le schéma de refroidissement.

L'algorithme d'optimisation du recuit simulé a été proposé pour la première fois par Kirkpa-trick *et al*. Le principe de cet algorithme est le suivant : une configuration (solution) initiale est générée aléatoirement (ou par une heuristique), puis à chaque itération la solution courante est déplacée de façon aléatoire dans son voisinage local, générant ainsi une nouvelle solution. Si cette nouvelle solution est meilleure que la précédente, elle est retenue. Sinon elle est acceptée avec une certaine probabilité. Cette étape est réitérée tant que l'équilibre thermodynamique n'est pas atteint. Une fois l'équilibre atteint, la température est diminuée jusqu'à un nouvel équilibre. Et ainside suite, jusqu'à une condition d'arrêt (nombre d'itérations maximal atteint ou "système figé").

La méthode du recuit simulé est présentée dans l'Algorithme 2.2. Au début de la recherche, la température est élevée, autorisant ainsi des dégradations importantes de la fonction objectif. Au fil des itérations, la température diminue, ce qui réduit la probabilité d'acceptation des dégradations de la fitness. Quand la température tend vers 0, seules les améliorations sont acceptées. Ce schéma se traduit

par une exploration (diversification) de l'espace de recherche, en début de recherche, et une exploitation (intensification) des régions prometteuses, en fin de recherche.

Algorithme 2.2 Algorithme du recuit simulé

Nécessite : La fonction objectif f, la température maximale Γ_{max} la température minimale Γ_{min} et la fonction de diminution de la température *abaisser*(Γ)

1: **Initialiser** la température : $\Gamma \Omega \Gamma_{max}$

2: **Générer** une solution aléatoire S

3: **Calculer** la fitness f(S) associe a la solution initial S

4: **Initialiser** la solution optimale : $S_{opt} \Omega$ S

5: **Tant que** $\Gamma > \Gamma_{min}$ **Faire**

6: **Tant que** l'équilibre thermodynamique n'est pas atteint **Faire**

7: **Tirer** une nouvelle solution S' dans le voisinage de S

8: **Calculer** la variation d'énergie: $A f \Omega f(S') \neq f(S)$

9: **Si** $A f$ E **Alors**

10: **Accepter** la nouvelle solution : S Ω S'

11: **Sinon**

12: **Si** $exp - \frac{\Delta f}{\Gamma} > x$ aléatoire œ[0, 1] **Alors**

13: **Accepter** la nouvelle solution : S Ω S'

14: **Fin Si**

15 **Fin Si**

16: **Si** $f(S') < f(S)$ **Alors**

17: **Mettre à jour** la solution optimale : $S_{opt} \Omega S'$

18: **Fin Si**

19: **Abaisser** la température Γ : $\Gamma \Omega$ *abaisser* (Γ)

20: **Fin Tant que**

21: **Fin Tant que**

22: **Retourner :** la solution optimale S_{opt}

3.3 LA MÈTHODE DE RECHERCHE TABOU

La méthode de recherche tabou (en anglais "*Tabu Search*" (TS)) a été proposée pour la première fois par Glover en 1986. L'objectif de cette méthode est d'introduire un peu d'intelligence dans le processus de recherche, afin de ne passe vouer totalement au hasard, au risque de faire des cycles menant toujours vers le même optimum local. En effet, la méthode de recherche tabou possède une mémoire où sont stockées les leçons tirées du passé. Par ailleurs, cette méthode, dans sa version originale, est dépourvue du caractère stochastique.

Plus concrètement, la méthode TS utilise une mémoire où sont enregistrés les derniers déplacements effectués (ou leurs attributs). L'ensemble de ces déplacements, qu'il est interdit d'effectuer en sens inverse, est regroupé dans la "liste taboue". Cela permet d'éviter le phénomène de cyclage et dirige la recherche vers de nouvelles régions non visitées encore. De même que toutes les méthodes de recherche locale, la recherche tabou nécessite un déplacement de la solution courante vers une nouvelle solution présente dans l'ensemble de ses voisines. À l'inversede la méthode SA qui choisit aléatoirement une seule solution dans son voisinage, la méthode TS explore généralement l'ensemble de son voisinage avant de choisir une direction. Ainsi, aprèsévaluation de la fonction objectif pour toutes les solutions voisines, la meilleure solution non taboue remplace la solution actuelle. Si un optimum local est atteint, la meilleure solution présente dans le voisinage est choisie, même si elle n'améliore pas la solution courante ou si elle estprésente dans la liste taboue. Ce mécanisme appelé *mécanisme d'aspiration* permet d'améliorer la solution actuelle et/ou de visiter de nouvelles régions qui ne peuvent être atteintes qu'en passant par des chemins déjà visités.

La taille de la mémoire réservée à la liste des déplacements tabous agit comme un paramètre de compromis entre diversification et intensification. Une mémoire de taille relativement faible favorise l'intensification, car pour un nombre restreint de mouvements bannis, la recherche aura vocation à visiter souvent les mêmes solutions. En revanche, si la taille de la mémoire augmente, la diversification est favorisée, dans la mesure où de nouvelles régions ont une plus forte probabilité d'être visitées. Un algorithme générique de la méthode de recherche tabou est donné dans l'Algorithme 2.3.

Algorithme 2.3 Algorithme de la recherche tabou

Nécessite : La fonction objectif f et la taille de la liste taboue

1: **Générer** une solution aléatoire S

2: **Calculer** la fitness f(S) associe a la solution initial S

3: **Initialiser** la solution optimale : S_{opt} Ω S

4: **Tant que** la condition d'arrêt n'est pas vérifiée **Faire**

5: **Générer** la liste des candidats non tabous par opération de voisinage

6: **Trouver** la meilleure solution S' parmi les candidats

7: **Si** $f(S') < f(S)$ **Alors**

8: $S \Omega S'$

9: $S_{opt} \Omega S$

10: **Fin Si**

11: **Mettre à jour** la liste taboue

12: **Fin Tant que**

13: **Retourner :** la solution optimale S_{opt}

4 LES MÈTAHEURISTIQUES À POPULATION DE SOLUTION

Contrairement aux algorithmes partant d'une solution singulière, les métaheuristiques à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. Ondistingue dans cette catégorie, les algorithmes évolutionnaires, qui sont une famille d'algorithmesissus de la théorie de l'évolution par la sélection naturelle, énoncée par Charles Darwin et les algorithmes d'intelligence en essaim qui, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels.

4.1 LES ALGOTITHMES GÈNÈTIQUES

Proposé dans les années 1975 par Holland, les algorithmes génétiques doivent leur popularité à Goldberg. Avant la parution de son livre qui est une des références les plus citées dans le domaine de l'informatique, on a pu voir un certain nombre d'autres présentations, citons Goldberg , Holland, Schwefel. Le sujet connaît une très grande popularité. Il existe aujourd'hui plusieurs milliers de références sur le sujet et le nombre de conférences dédiées au domaine (que ce soit sur les techniques elles-mêmes ou sur les applications)ne fait qu'augmenter.

De manière générale, les algorithmes génétiques utilisent un même principe. Une population d'individus (correspondants à des solutions) évoluent en même temps comme dans l'évolution naturelle en biologie. Pour chacun des individus, on mesure sa faculté d'adaptation au milieu extérieur par le fitness.

Les algorithmes génétiques s'appuient alors sur trois fonctionnalités :

-*La Sélection* : Pour déterminer quels individus sont plus enclins à se reproduire, une sélection est opérée. Il existe plusieurs techniques de sélection, les principales utilisées sont la sélection par tirage à la roulette (roulette-wheel sélection), la sélection par tournoi (tournament selection), la sélection par rang (ranking selection), etc.

- *Le Croisement* : L'opérateur de croisement combine les caractéristiques d'un ensemble d'individus parents (généralement deux) préalablement sélectionnés, et génère de nouveaux individus enfants. Là encore, il existe de nombreux opérateurs de croisement, par exemple le croisement en un point, le croisement en n-points (n ≥ 2) et le croisement uniforme 5 (voir figur2.6).

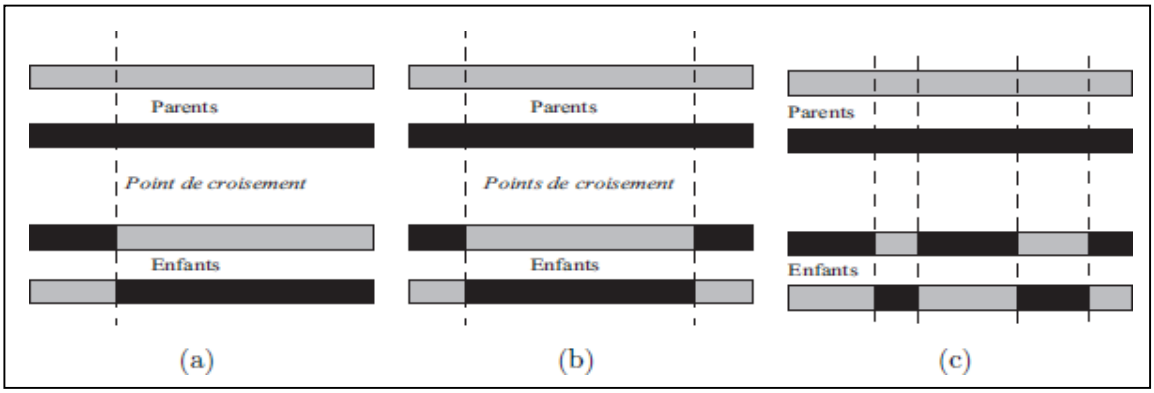


Figure 2.6 *- (a) croisement simple en un point, (b) croisement en deux points, (c) croisement uniforme.*

-*La Mutation et le remplacement*: Les descendants sont mutés, c'est-à-dire que l'on modifie aléatoirement une partie de leur génotype, selon l'opérateur de mutation. Le remplacement (ou sélection des survivants), comme son nom l'indique, remplace certains des parents par certains des descendants. Le plus simple est de prendre les meilleurs individus de la population, en fonction deleurs performances respectives, afin de former une nouvelle population (typiquement de la même taille qu'au début de l'itération).

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluationd'une solution. Le codage phénotypique ou codage direct correspond en général à une représentation de la solution très proche de la réalité. L'évaluation d'une solution représentée ainsiest en général immédiate.

Algorithme 2.4 Algorithme génétique

1: **Initialiser** les paramètres de l'algorithmes génétique
2: **Générer** la population initiale
3: **Tant que** critère d'arrêt non satisfait **Faire**

4: **Evaluer** les individus
5: **Choisir** les parents en se basant sur une strategie de selection
6: **Appliquer** l'operateur de croisement avec un taux de croisement associe
7: **Appliquer** l'operateur de mutation avec un taux de mutation associe
8: **Remplacement** de la population
9: **Fin Tant que**
10: **Retourner** le meilleur individu

4.2 ALGORITHME DE COLONIES DE FOURMIS

Les algorithmes de colonies de fourmis ont été proposés par Colorni,Dorigo et Maniezzo en 1992 et appliquées la première fois au problème du voyageur de commerce. Ce sont des algorithmes itératifs à population où tous les individus partagent un savoir commun qui leur permet d'orienter leurs futurs choix et d'indiquer aux autres individus des choix à suivre ou à éviter. Le principe de cette métaheuristique repose sur le comportement particulier des fourmis, elles utilisent pour communiquer une substance chimique volatile particulière appelée phéromone grâce à une glande située dans leur abdomen. En quittant leur nid pour explorer leur environnementà la recherche de la nourriture (Food), les fourmis arrivent à élaborer des chemins qui s'avèrent fréquemment être les plus courts pour aller du nid vers une source de nourriture. Chaque fourmi dépose alors une quantité de phéromones sur ces pistes qui deviendront un moyen de communication avec leurs congénères, les fourmis choisissent ainsi avec une probabilité élevée les chemins contenant les plus fortes concentrations de phéromones à l'aide des récepteurs situés dans leurs antennes.

La figure 2.7 illustre et confirme ce constat, une expérience a été faite par Gauss et Deneubourg en 1989 appelée expérience du pont à double branche, les fourmis qui retournent au nid rapidement, après avoir visité la source de nourriture, sont celles qui ont choisi la branche courte et les fourmis empruntant cette branche faisant plus d'aller retour, et par conséquent la quantité de phéromones déposée sur la plus courte branche est relativement supérieure que celle présente sur la plus longue branche. Puisque les fourmis sont attirées plus vers les pistes de plus grande concentration en phéromones, alors la branche courte sera la plus s'empruntée par la majorité des fourmis.

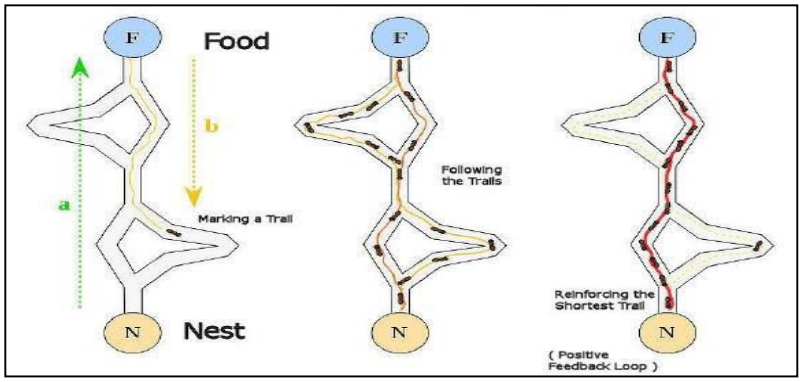


***Figure 2.7**- Optimisation du chemin par les fourmis au cours des itérations.*

Algorithme 2.5 Le schéma générale de l'algorithme de colonies de fourmis

1: **Initialiser** une population de m fourmis
2: **Évaluer** les m fourmis
3: **Tant que** critère d'arrêt n'est pas atteint **Faire**
4: **Pour** i = 1 à m **Faire**
5: **Construire** le trajet de la fourmi i
6: **Déposer** des phéromones sur le trajet de la fourmis i
7: **Fin Pour**
8: **Évaluer** les m fourmis
9: **Évaporer** les pistes de phéromones
10: **Fin Tant que**
11: **Retourner** la ou les meilleures solutions

5 . ALGORITHME DE COLONIES D'ABEILLES

Au cours de la dernière décennie, les algorithmes d'abeilles inspirés de la nature, sont devenus un outil prometteur et puissant. Malheureusement, on ne parvient pas à connaître la date exacte de la première apparition des algorithmes d'abeilles. Ce qui est sûr pour nous c'est qu'ils ont été développés en quelques années de façon indépendante par différent groupes de chercheurs.

D'après la bibliographie, il semble que l'algorithme HONEY-BEE a été réalisé pour la première fois vers 2004 par Craig A.tovey à GEORGIA TECH en collaboration avec SUNIL NAKRANI. A la fin de 2004 et au début de 2005, Xin-She Yang à l'Université de CAMBRIDGE a développé le VIRTUAL BEE ALGORITHM (VBA) pour résoudre des problèmes d'optimisation numérique, cet algorithme permet d'optimiser à la fois les fonctions et les problèmes discrets, cependant ils n'ont

donné comme exemples que les fonctions à deux paramètres. Un peu plus tard en 2005, Haddad, Afshar et leurs collègues ont présenté un algorithme dit Honey-Bee Mating Optimization (HBMO) qui a ensuite été appliqué à la modélisation de réservoirs et de clustering.

En 2006, B.Basturk et D.jarabogo en Turquie, ont développé un algorithme appelé Artificial Bee Colony (ABC) pour l'optimisation de fonction numérique.

Nous remarquons ici que la méthode des abeilles est plus ou moins récente, et qu'avec le temps de nouvelles versions apparaissent, ce qui rend cette méthode de plus en plus populaire et maîtrisable par les chercheurs.

5.1 COMPORTEMENT DES ABEILLES

Comme les fourmis, les abeilles sont des insectes sociaux. Elles sont obligées de vivre en colonie très organisée, formée d'ouvrières, de faux-bourdon et d'une seule reine, et où chacune a un travail bien précis à faire. Les abeilles se nourrissent essentiellement de pollen et de miel. Elles vont butiner les fleurs pour prendre le nectar.

Au cours de sa courte vie (environ 45 jours), l'ouvrière fait plusieurs métiers : elle nettoie les cellules, nourrit les larves, elle range le pollen et le nectar dans les alvéoles, elle ventile la ruche en agitant rapidement ses ailes, elle construit les rayons avec la cire qu'elle produit, elle garde le trou de vol pour chasser les intrus, elle devient butineuse, porteuse d'eau et récolte du pollen et du nectar jusqu'à la fin de sa vie.

L'abeille est capable, par la danse ou par la production de substances chimiques appelées

« phéromone », de communiquer aux autre abeilles l'endroit où elle a découvert de la nourriture. Elle danse en rond (Figure 2.8) quand elle a trouvé du pollen à faible distance (moins de 25 mètres). Elle utilise une danse très compliquée dite la danse frétillante (Figure 2.9), ou danse en huit, si la nourriture se trouve à moins de 10 kilomètres. La direction de la nourriture est exprimée par rapport à la position du soleil. La distance est exprimée par le nombre et la vitesse des tours effectués par l'abeille sur elle-même. Afin de survivre à l'hiver, les abeilles doivent recueillir et stocker environ 15 à 50 Kg de nectar.

Les faux bourdons ne servent que pour la reproduction. Ils sont incapables de se nourrir eux-mêmes (les ouvrières les nourrissent) et ils n'ont pas de dard pour protéger la ruche. Il n'y a qu'une seule reine dans la colonie. Quelques jours après sa naissance, elle s'envole pour la seule fois de son existence pour être fécondée par quelques faux-bourdons. Elle occupera le reste de ses jours (4 à 5 ans) à pondre jusqu'à 2000 œufs par jour.

Les abeilles adultes (âgées de 20 à 40 jours) deviennent habituellement des butineuses. Lesabeilles butineuses jouent en général l'un des trois rôles suivants : butineuses actives, butineuses éclaireuses et butineuses inactives.

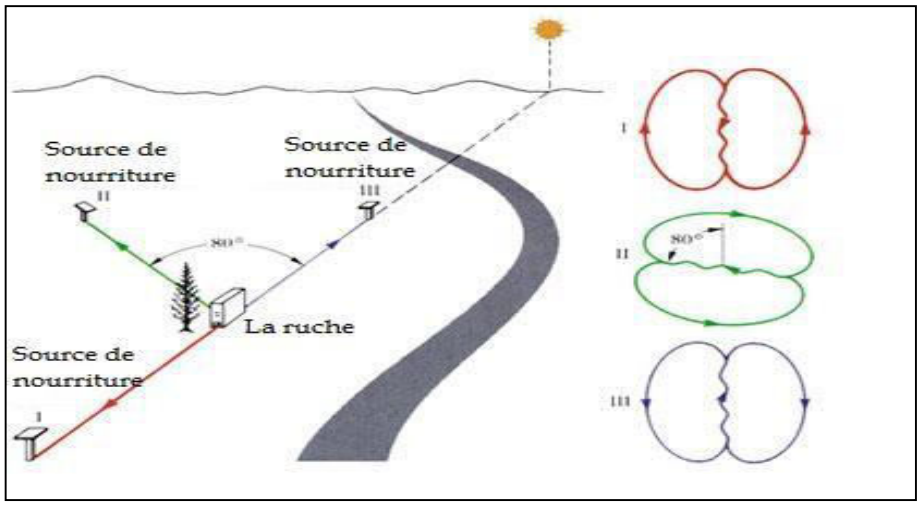


Figure 2.8 *- La danse en rond qu'effectue l'abeille en fonction de la direction de la source de nourriture.*

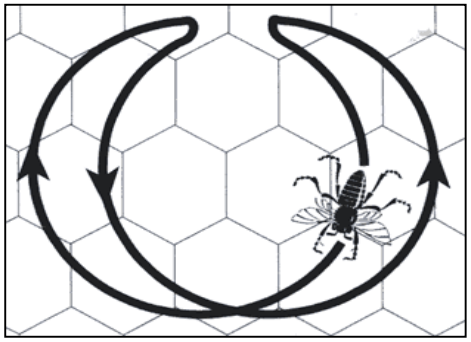


Figure 2.9 - *La danse frétillante, appelée aussi en huit*

5.2 ALGORITHME D'OPTIMISATION PAR COLONIE D'ABEILLES

Dans cet algorithme, l'emplacement de la source de nourriture représente la solution possible au problème, et la quantité du nectar de cette source correspond à une valeur objectifdite fitness.

Les butineuses sont attribués aux différentes sources de nourriture de façon à maximiser l'apport total de nectar. La colonie doit optimiser l'efficacité globale de la collecte. La répartition des abeilles est donc en fonction de nombreux facteurs tels que la quantité du nectar et la distance entre la source de nourriture et la ruche. Ce problème et similaire à la répartition des serveurs d'hébergement web, qui était en fait un des premiers problèmes résolus en utilisant les algorithmes d'abeilles par Nakrani et Tovey en 2004.

Le nombre des butineuses actives ou inactives représente le nombre de solution dans cette populations.

Dans la première étape, l'algorithme génère une population initiale de SN solutions distribuées de façon aléatoire. Chaque solution $x_i(i = 1, 2, \dots, SN)$ qui est initialisée par les éclaireuses, et représente un vecteur de solution au problème d'optimisation. Les variables que contient chaque vecteur doivent être optimisées.

Après l'initialisation, la population des solutions est soumise à des cycles répété $C = 1,2, \dots, C_{\max}$ ces cycles représentent des processus de recherches faits par les butineuses actives, inactives et les éclaireuses.

Les butineuses actives recherchent dans le voisinage de la source précédente x_i de nouvelles sources v_i ayant plus de nectar, Elles calculent ensuite leur fitness. Afin de produire une nouvelle source de nourriture à partir de l'ancienne, on utilise l'expression ci contre :

$$\upsilon_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

Où $k \in \{1, 2, \dots, BN\}$(BN est le nombres des butineuses actives)et $j \in \{1, 2, \dots, SN\}$ sont des indices choisis au hasard. Bien que k est déterminé aléatoirement, il doit être différent de i. ϕ_{ij} est un nombre aléatoire appartenant à l'intervalle [-1, 1], il contrôle la production d'une source de nourriture dans le voisinage de x_{ij}.

Après la découverte de chaque nouvelle source de nourriture v_{ij}, un mécanisme de sélection gourmande est adopté, c'est-à-dire que cette source est évaluée par les abeilles artificielles, sa performance est comparée à celle de x_{ij}. si le nectar de cette source est égale ou meilleur que celui de la source précédente, celle-ci est remplacée par la nouvelle. Dans le cas contraire l'ancienne est conservée.

Pour un problème de minimisation, La fitness est calculée suivant cette formule :

$$fit_i(\vec{x}_i) = \begin{cases} \frac{1}{1 + fit_i(\vec{x}_i)} \; si \; f_i(\vec{x}_i) \geq 0 \\ 1 + abs\left(f_i(\vec{x}_i)\right) si \; f_i(\vec{x}_i) > 0 \end{cases}$$

Telle que $f_i(\vec{x}_i)$ est la valeur de la fonction objectif de la solution $\vec{x}_i$

A ce stade, les butineuses inactives et les éclaireuses qui sont entrain d'attendre au sein dela ruche. A la fin du processus de recherche, les butineuses actives partagent les informations sur le nectar des sources de nourriture ainsi que leurs localisations avec les autre abeilles via la danse frétillante. Ces dernières évaluent ces informations tirées de toutes les butineuses actives, et choisissent les sources de nourriture en fonction de la valeur de probabilité P_i associée a cette source, et calculée par la formule suivante :

$$\mathrm{p_i} = \frac{fit_{\mathrm{i}}}{\sum_{n=1}^{SN} fit_{\mathrm{n}}}$$

Où fit_i est la fitness de la solution i, qui est proportionnelle à la quantité du nectar de la source de nourriture de la position i.

La source de nourriture dont le nectar est abandonné par les abeilles, les éclaireuses la remplacent par une nouvelle source. Si durant un nombre de cycle prédéterminé appelé « limite » une position ne peut être améliorée, alors cette source de nourriture est supposée être abandonnée.[3]

Toutes ces étapes sont résumées dans l'algorithme suivant :

Algorithme 2.6 Algorithme de colonies d'abeilles

1: **Initialiser** la population avec S+W solution aléatoire
2: **Évaluer** la fitness de la population
3: **Tant que** le critère d'arrêt n'est pas satisfait **Faire**
4: **Recruter** des abeilles : Rechercher de nouvelle solution
5: **Évaluer** la fitness de la population
6: **Si** un membre de la population n'est pas amélioré **Faire**
7: **Enregistrer** la solution et remplacer la par une solution aléatoire
8: **Trouver** S solutions aléatoires et remplacer les S membres de la population qui ont la mauvaise fitness
9:**Fin Tant que**
10: **Retourner** la meilleure solution

6. MÈTHODES HYBRIDES

Les différentes méthodes vues précédemment possèdent bien entendu leurs propres avantages et inconvénients, en termes de qualité de solutions fournies, de la complexité temporelle et la simplicité d'implémentation. Une tendance actuelle en optimisation combinatoire consiste à coopérer/hybrider

plusieurs méthodes parmi celles que nous venons de citer. La motivation derrière une telle hybridation est d'avoir des méthodes plus sophistiquées et plus efficaces.

L'hybridation est une technique qui essaie de tirer profit des points forts de chaque méthode de résolution utilisée pour améliorer le comportement global de la méthode hybride. Donc, une méthode hybride est une méthode de recherche constituée d'au moins deux procédures de recherche distinctes, y compris les méthodes exactes et/ou approchées (sus-décrites). Elle consiste à exploiter les avantages respectifs de ces méthodes tout en combinant leurs algorithmes suivant une approche synergétique. Nous citons à titre d'exemple le cas des algorithmes mimétiques ou encore connus sous le nom d'algorithmes génétiques hybrides qui combinent une recherche locale qui assure l'intensification de la recherche et un algorithme génétique qui renforce la diversification.

À ce sujet, Duvivier a proposé une classification des techniques d'hybridation en s'appuyant sur les travaux du groupe PERFORM .Les principales formes hybridation proposées sont classifiées en trois catégories, selon leurs architectures :

- Hybridation séquentielle : consiste à exécuté séquentiellement différentes méthodes de recherche de telle manière que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante comme le montre la figure 2.10. Cette technique d'hybridation est la plus simple et la plus populaire.

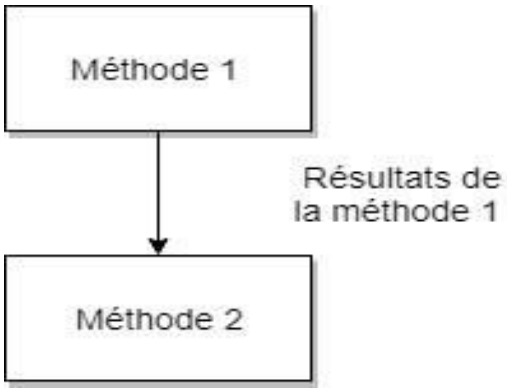


Figure 2.10 *- Hybridation séquentielle.*

-Hybridation parallèle synchrone : consiste à incorporer une méthode de recherche particulière dans un opérateur d'une autre. C'est en quelque sort une hybridation plus fine que la précédente étant donné que la méthode est englobé dans une autre comme le montre la figure 2.11 . Cette technique est plus complexe à mettre en œuvre que la précédente vu qu'il faut tenir compte des fortes interactions entre les méthodes incorporées pour ne pas aboutir à une méthode hybride moins efficace . Un exemple de ce type d'hybridation est de remplacer l'opérateur de mutation d'un algorithme génétique par une recherche taboue.

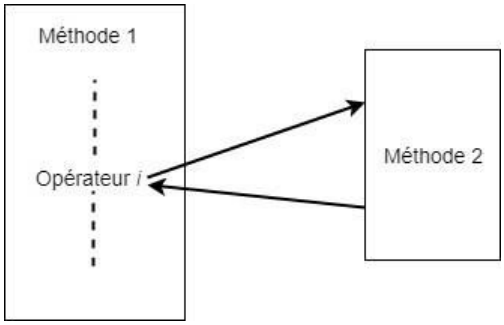


Figure 2.11 *- Hybridation parallèle synchrone.*

-Hybridation parallèle asynchrone : consiste à faire évoluer en parallèle différentes méthodes de recherche. Cette coévolution permet une bonne coopération des méthodes de recherche au travers d'un coordinateur qui est chargé d'assurer les échanges d'informations entre les méthodes de recherche constituant l'hybride (Figure 2.12).[6]

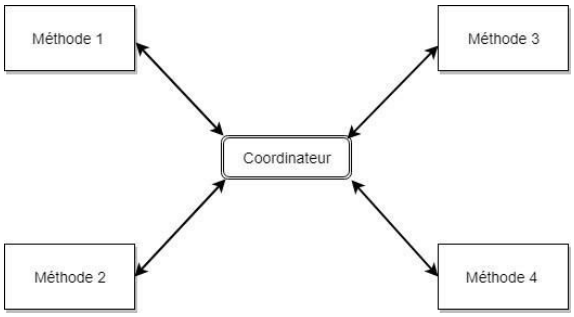


Figure 2.12- *Hybridation parallèle asynchrone* **.**

7. CONCLUSION

Dans ce chapitre nous avons essayé de présenter les principes et les algorithmes de différentes méthodes de résolution de problèmes d'optimisation commençant par les méthodes exactes aux méthodes approchées. Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace

mémoire requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution. Nous avons pu constater que les méthodes approchées peuvent être partagées en deux classes: des méthodes heuristiques et des méthodes métaheuristiques. Une méthode heuristique est applicable sur un problème donné. Tandis qu'une

méthode métaheuristique est plus générique et elle peut être appliquée sur plusieurs problèmes d'optimisation. En outre, nous avons constaté que les méthodes métaheuristiques peuvent être partagées en deux sous classes: des méthodes à base d'une solution unique et des méthodes à base de population de solutions. Les méthodes de la première sous classe (les méthodes à base d'une solution unique) se basent sur la recherche locale pour trouver la solution du problème à traiter. Elles sont souvent piégées par l'optimum local d'un voisinage donné. Par contre, les méthodes de la deuxième classe (les méthodes à base de population de solutions) se basent sur une recherche globale ce qui leur permet d'échapper au problème de la convergence vers l'optimum global et augmente leur possibilité de fournir des solutions de bonnes qualités. Dans la dernière partie du chapitre nous avons présenté l'algorithme de colonies d'abeilles .

CHAPITRE 03

« ALGORITHME GUIRCHOU ET MARTINEAU POUR RÈSOLUTION DU PROBLÈME FLOW SHOP HYBRIDE À DEUX ÈTAGES »

1. INTRODUCTION

Dans un système informatique, le serveur qui envoie des données aux machines est appelé un serveur de réseau. Pendant l'activité de chargement, il n'est pas nécessaire que la machine réalisatrice soit disponible : elle peut traiter un autre travail. En effet, les machines disposent d'un coprocesseur de communication qui leur permet de recevoir informations sur le serveur à tout moment. Ainsi, dans un système informatique, l'activité de chargement peut être considérée comme un travail qui ne nécessite que le serveur pour être exécuté.

Le problème abordé dans ce chapitre concerne un système informatique que nous considérons peut être vu comme un atelier de flow shop hybride à deux étages, avec une seule machine au premier étage, qui est le serveur et le m machines parallèles au deuxième étage, avec une contrainte de no-wait entre les deux étages, et les temps de traitement sur le serveur sont censés être uniques et le but est Pour réduire le temps total de réalisation.

On considère un environnement d'ordonnancement déterministe avec m machines parallèles identiques M_1, M_2,…, Mm et n Tâches à planifier. Chaque tâche doit être traitée sans préemption sur une machine déterminer. Avant son traitement, un travail doit être chargé sur une machine. Cette activité de chargement ou l'activité de configuration, est effectuée par une autre machine spéciale, appelée serveur. Après une configuration, le serveur est à nouveau disponible pour effectuer une autre activité de chargement, c'est-à-dire qu'un seul serveur ne peut gérer qu'un seul tâche à la fois et peut être considérée comme une seule machine. Le chargement d'un travail doit être immédiatement suivie de son traitement. Dans cet environnement, on considère que les temps de réglage nécessitent une unité de temps pour n'importe quel travail. De plus, nous supposons que les temps de transfert entre le serveur et les machines sont non significatif . Le but de cet chapitre est de trouver un ordonnancement réalisable dans un environnement de m = 2 machines, et avec un temps de réalisation total minimum, ce qui correspond à la minimisation du travail en cours dans le réseau.

2. LE FLOW SHOP HYBRIDE

Ce problème d'atelier correspond à une réalité industrielle où des applications de différents domaines industriels seront présentées.

Soit un processus de production en série. Les produits fabriqués passent dans un premier temps dans un premier étage, puis dans une seconde, etc. Supposons qu'il y a « k » étages en série. Dans chacun d'eux, on considère que les machines sont regroupées en parallèles. Un groupe contient un

ensemble de machines susceptibles d'exécuter une même opération. Elles sont alors équivalentes dans leur fonctionnement mais pas forcément dans leurs performances. La durée d'une opération peut dépendre non seulement des ressources choisies dans le groupemais aussi du nombre de ressources affectées à l'opération (si la ressource est une machine, on peut parler alors de machines parallèles identiques, uniforme ou différentes). On supposera qu'il n'existe qu'un groupe de ressources par étage (où chaque produit subit un nombre d'opérations au plus égal à k, k étant le nombre d'étages).

Ce type de problème est appelé **flow shop hybride à k-étages**.

Remarques 1 :

- Dans un flow shop hybride, les gammes de fabrication sont toutes identiques comme dans tout flow shop et les machines sont regroupées par étages.

- De plus, si le nombre d'étages est égal à 1, alors on est en présence soit d'un problème à machines parallèles, soit d'un problème à une machine. En revanche s'il n'y a qu'une seule machine à chaque étage, alors le problème est équivalent à un problème classique de flow shop.[B]

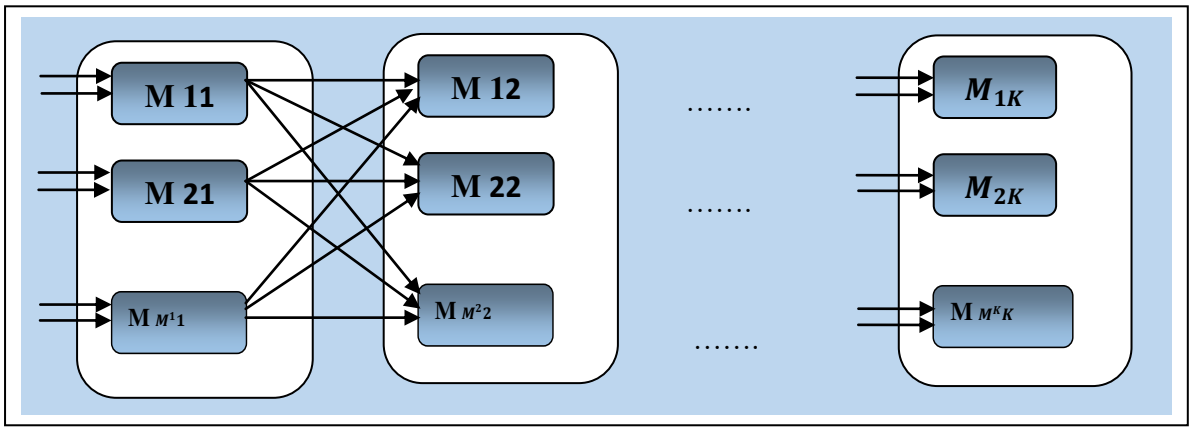


Figure 3. 1 - *La représentation de flow shop hybride à k étages.*

La Figure 3.1 présente un problème de flow shop hybride. On considère qu'il y a une "station d'entrée". Ensuite, il y a la succession des k-étages. Chaque étage « r » comporte $M^{(r)}$ machines. Enfin, l'atelier se termine par une "station de sortie". Entre les différents étages, des zones de stockages à capacités finies ou infinies peuvent èxister.[7]

3. UN ÈTAT DE LA CONNAISSANCE ET DE LA RECHERCHE SUR LE FLOW SHOP HYBRIDE À DEUX ÈTAGES

Les travaux précurseurs Vignier *(1999)* a cité notamment les travaux de Mac Naughton [15], Shen et Chen [16], Buten et Shen [17], Horn [18], Paul [19], Langston [20], Gupta [21], Koulamas et Smith [22] et Sriskandarajah et Sethi [23].

3.1 LES TRAVAUX DES CONTINUATEURS DE LA PERIODE 1990-1999

Les références citées sont Sherali [24], Gupta et Tunc [25], Deal et Hunsucker [26] , Lee et LIN [27], Gupta et Tunc [28], Lee et Vairaktarakis [29], Gupta, Hariri et Potts [30], Proust et Grunenberger[31], Chen [32], Li[33], Vignier et al. [34], Guinet et al.[35], Koulamas [36], Houari et M'Hallah [37] et Riane et al. [38].

3.2 LES RECHERCHES ET TRAVAUX CONTEMPORAINS (2000-2008)

Par travaux contemporains, nous sous entendons les travaux effectués dans le dernier octonat.

De récents travaux ont traité les problèmes de flow-shop hybrides à 2-étages de domaines variés. Ils ont proposé des méthodes exactes de résolution pour des problèmes de petites tailles. Les méthodes exactes utilisées étaient de type Séparation et évaluation (Branch and Bound) et à solveurs pour les modèles de programmation linéaire en nombres entiers.

- Glass et al. [39] considèrent le problème des machines parallèles quelconques. Ils fournissent une heuristique, analysent sa performance et étudient la complexité du problème de machines parallèles.
- Hall [40] a décrit une notation et un schéma de classification des problèmes d'ordonnancements à machines parallèles. Il fournit des résultats sur la complexité de certains problèmes. Pour les deux fonctions objectif, longueur d'ordonnancement et coût d'encours, si le nombre de machines est égal à deux ou quelconque, les temps d'exécutions de tâches unitaires ou arbitraires, les problèmes sont montrés polynomiaux. Si les temps d'exécutions des tâches sur le serveur sont quelconques et non unitaires sur les machines, la complexité est encore un problème ouvert. Il propose la règle SPT/FAM, une méthode polynomiale en O (n log(n)) de résolution du problème P2, S $|s_i = 1|\sum C_i$. Elle consiste a trié les travaux selon le plus petit temps d'exécution pour le serveur et les affecte à la première machine disponible.
- Kravchenko et Werner [41] proposent une heuristique pour minimiser la somme des dates de fin des travaux dans le cas du temps de chargement des tâches unitaires appelés « setup ». En tout étape de l'algorithme, on choisit une tâche, de plus petit temps d'exécution et qui ne génère pas un conflit parmi les tâches non encore ordonnancées. Si un telle tâche n'existe pas, on choisira arbitrairement une tâche pour exécution.
- On appelle « machine batch », une machine qui peut exécuter simultanément un lot de tâches. Ling-Huey [42] considère un flow shop hybride à deux étages avec une machine batch au premier étage et une seule machine au second étage où les temps d'attente des tâches au second étage ne dépassent pas une valeur, borne supérieure donnée. Il a proposé

une heuristique et un modèle en nombres entiers. Par une expérimentation numérique, il a étudié et justifié l'efficacité de cette heuristique.

- Oguz et al. [43] ont proposé des heuristiques pour la résolution du flow shop hybride à deux étages afin de minimiser la longueur de l'ordonnancement. Les heuristiques sont gloutonnes. Les tâches sont ordonnancées selon une liste donnée, formée par des règles de priorités entre les tâches définies dans la bibliographie. Quelques bornes inférieures sont fournies pour être utilisées dans les études de performance. L'analyse en moyenne de ces heuristiques est faite par une étude expérimentale en générant aléatoirement des instances de problèmes. Les résultats suggèrent quelle sont efficaces.
- Les machines sont supposées ne pas être disponibles en tout instant du à une panne impromptue ou à leur maintenance préventive. Wei Wang [44] a traité le flow shop hybride à deux étages avec une disponibilité des machines limitées afin de minimiser la longueur de l'ordonnancement. Il a étudié ce problème en supposant que les intervalles du temps d'indisponibilité appelés des « creux » sont connus en avance. Il a montré que ce problème est NP-dur dans le sens fort même s'il n'y a qu'une seule machine à un étage et une seule période creuse pour la dite machine. Il a aussi étudié la notion d'approximabilité de ce modèle et a montré APX –difficulté de ce problème avec des « creux » sur tout le second étage ou sur tout le premier étage avec au moins un « creux » . Ils fournissent deux algorithmes avec des ratios très proches du plus mauvais cas dans le cas d'une seule machine au premier étage et présentant qu'un seul « creux ». Ils fournissent encore un algorithme avec un ratio pour le cas des périodes de d'indisponibilités dans un centre informatique à un serveur.
- JINXING XIE et al. [45] ont étudié le problème de flow shop hybride à deux étages à machines parallèles à chaque étage sous la contrainte no-wait entre les étages afin de minimiser la longueur d'ordonnancement. Ils ont développé une heuristique de type MDA (minimum deviation algorithm). Par une expérimentation numérique en moyenne, ils ont montré l'efficacité de cette heuristique comparativement à d'autres algorithmes d'approximations connus. Un grand nombre de simulations ont été effectuées sous divers conditions d'ateliers et les résultats ont montré que les solutions fournies sont proches de l'optimum sous la majorité des conditions.[7]

4. NOTATIONS ET FORMULATIONS MATHÈMATIQUE

4.1 NOTES

le problème hybride flow shop hybride à l'étude est noté FH2,(1, Pm)|nowait, $p_{i,1} = 1| \sum C_i$, le premier champ α indique un atelier flow shop hybride à deux étages avec une seule machine au premier étage, m processeurs parallèles identiques à la deuxième étage. Les champs β et γ sont les plus classiques dans la littérature sur l'ordonnancement.[8][11]

On considère un ensemble J de n emplois $\{i\} 1 \leq i \leq n$ à programmer sur deux étapes. Chaque travail i ($1 \leq i \leq n$) est composé de deux opérations : l'opération $o_{i,1}$ traitée à la première étage et l'opération $o_{i,2}$ traitées à la deuxième étage. Nous supposons que la préemption n'est pas autorisée et que chaque machine peut traiter une seule opération à la fois.

On note $p_{i,1}$ et $p_{i,2}$ les temps de traitement de l'opération $o_{i,1}$ et opération $o_{i,2}$, respectivement , nous supposons que $p_{i,1} =1$, $\forall i$, $1 \leq i \leq n$, $p_{[\ell],j}$ est le temps de traitement à l'étape j de l'opération en position ℓ à la première étage ($1 \leq j \leq 2$; $1 \leq \ell \leq n$). Dans ce chapitre, nous considérons que les temps de traitement à la deuxième étage sont des entiers positifs. C_i fait référence à l'achèvement temps de travail i et $C_{i,j}$ désigne le temps d'achèvement de l'opération $o_{i,j}$ avec $1 \leq j \leq 2$ ($C_{i,2} = C_i$). Nous on note T_k ($1 \leq k \leq m$) la date avant laquelle il n'est pas possible d'effectuer un travail sur la machine M_k à la deuxième étage. Comme les temps de traitement des opérations sont unaires à la première étage, nous avoir $T_k = k$, $\forall$ k.

Le temps de traitement le plus court (SPT) est un ordre non décroissant de la règle des temps de traitement de la deuxième étage et la première machine disponible (FAM) planifie l'opération en cours sur la première machine disponible. SPT/FAM désigne une méthode de résolution O(n log(n)) [9] qui trie les travaux selon la règle SPT en temps O(n log(n)) puis les affecter selon la règle FAM en temps O(n).

La notation Pm, S|s_i|γ introduit par Hall et al.[9] représente les problèmes d'ordonnancement avec un serveur *S* et *m* machines parallèles. Comme mentionné ci-dessous, dans ces problèmes, la configuration du travail que nous devons être traité par le serveur et une machine parallèle. Dans la suite, nous désignons par PS-problèmes les problèmes de machines parallèles avec un seul serveur sous cette contrainte et FH2 en deux étapes problèmes de flow shop hybride pour lesquels cette contrainte n'est pas valide.

4.2 MODÈLE DE PROGRAMMATION LINÈAIRE ENTIER

Les données du problème sont : n , m et $p_{i,j}$,$1 \leq i \leq n$, $1 \leq j \leq 2$. H est une valeur élevée arbitraire. Les variables sont : $t_{i,j}$, l'heure de début de fonctionnement $o_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq 2$; $x_{i,k}$ est égal à 1 si l'opération $o_{i,2}$ est affectée à la machine M_k et 0 sinon, $1 \leq i \leq n$, $1 \leq k \leq m$;

$y_{i,\ell,j}$ est égal à 1 si le travail i précède le travail ℓ à l'étape j , $1 \leq i \leq n$, $1 \leq \ell \leq n$, $i \neq \ell$, $1 \leq j \leq 2$.

La fonction objectif est de minimiser $\sum_{i=1}^{n}(t_{i,2} + p_{i,2})$

Minimiser $\sum_{i=1}^{n}(ti, 2 + pi, 2)$ (1)

Sujet à $\sum_{k=1}^{m} xi, k = 1 , i = 1,2, \ldots \ldots, n,$ (2)

$t_{i,2} = t_{i,1} + p_{i,1}$, i = 1, ……..,n, (3)

$t_{\ell,1} \geq t_{i,1} + p_{i,1} - H \times (1 - y_{i,\ell,1})$, i = 1,…..,n, ℓ = 1,….,n, $i \neq \ell$, (4)

$t_{i,1} \geq t_{\ell,1} + p_{\ell,1} - H \times (1 - y_{i,\ell,1})$, i = 1,…..,n, ℓ = 1,….,n, $i \neq \ell$, (5)

$t_{\ell,2} \geq t_{i,2} + p_{i,2} - H \times (1 - y_{i,\ell,2}) - H \times (2 - x_{i,k} - x_{\ell,k})$,

i = 1,…..,n, ℓ = 1,….,n, $i \neq \ell$, k = 1,……,m, (6)

$t_{i,2} \geq t_{\ell,2} + p_{\ell,2} - H \times (1 - y_{i,\ell,2}) - H \times (2 - x_{i,k} - x_{\ell,k})$,

i = 1,…..,n, ℓ = 1,….,n, $i \neq \ell$, k = 1,……,m, (7)

$t_{i,j} \geq 0$, i = 1,…..,n, j = 1,2, (8)

$x_{i,k} \in \{ 0,1 \}$, i = 1,…..,n, k = 1,……,m, (9)

$x_{i,\ell,j} \in \{ 0,1 \}$, i = 1,…..,n, ℓ = 1,….,n, $i \neq \ell$, j = 1,2, (10)

Les contraintes (2) garantissent que chaque travail est affecté à exactement une machine à la deuxième étape.

Les contraintes de routage et la contrainte de non-attente sont formulées en (3). Contraintes (4) et (5)(respectivement (6) et (7)) sont les contraintes disjonctives au premier étape (respectivement au second étape). Ce modèle contient $nm + 2n(n - 1)$ variables booléennes et 2n variables positives et $4n + (m + 2)n(n - 1)$ contraintes.

5. CAS PARTICULIÈRS

Le problème considéré est noté FH2,(1, P2)|nowait, $p_{i,1} = 1|\sum C_i$. Pour P2, S $|s_i = 1|\sum C_i$ problème, Hall et al.[9] montrent que l'algorithme de liste SPT/FAM renvoie une valeur optimale solution. Malheureusement, lorsque la contrainte multiprocesseur est relâchée, c'est-à-dire lorsque l'activité de chargement peut être exécuté sur le serveur, sans avoir besoin de la machine exécutante, cet algorithme n'est pas optimale. Nous détaillons dans cette section deux cas particuliers du problème FH2,(1, P2)| nowait, $p_{i,1} = 1|\sum C_i$: $p_{i,2} \leq 1$, $\forall i$,$1 \leq i \leq n$ et $p_{i,2} > 1$, $\forall i$, $1 \leq i \leq n$ et on montre l'optimalité de l'algorithme SPT/FAM pour les deux cas. Pour les deux cas suivants, on note que les résultats restent valables lorsque les temps de traitement à la deuxième étape sont des nombres réels.

a. PROBLÈME AVEC pi,2 ≤ 1, ∀i , i= 1,……,n

Si on considère le problème avec $p_{i,2} \leq 1$ $\forall i$, $1 \leq i \leq n$, on a $\forall i$, $1 \leq i \leq n$ $C_i = C_{i,1} + p_{i,2}$. Ainsi $\sum_{i=1}^{n} C_{i,2}$ est équivalent à $\sum_{i=1}^{n} C_{i,1}$, qui est une constante car aucun temps mort n'est introduit sur le machine au premier étape. Cette constante est égale à $n(n + 1)/2$. Ainsi, tout ordonnancement semi-actif est une solution optimale.

b. PROBLÈME AVEC pi,2 > 1, ∀i , i= 1,……,n

Nous considérons que $p_{i,2} > 1$ pour tous les emplois et nous proposons un algorithme de liste basé sur la règle SPT/FAM. φ désigne la séquence SPT.

Théorème 1. Problème FH2,(1, P2)|nowait, $p_{i,1} = 1$, $p_{i,2} > 1|\sum C_i$ peut être résolu de façon optimale en Temps O(n log(n)) par l'algorithme SPT/FAM.

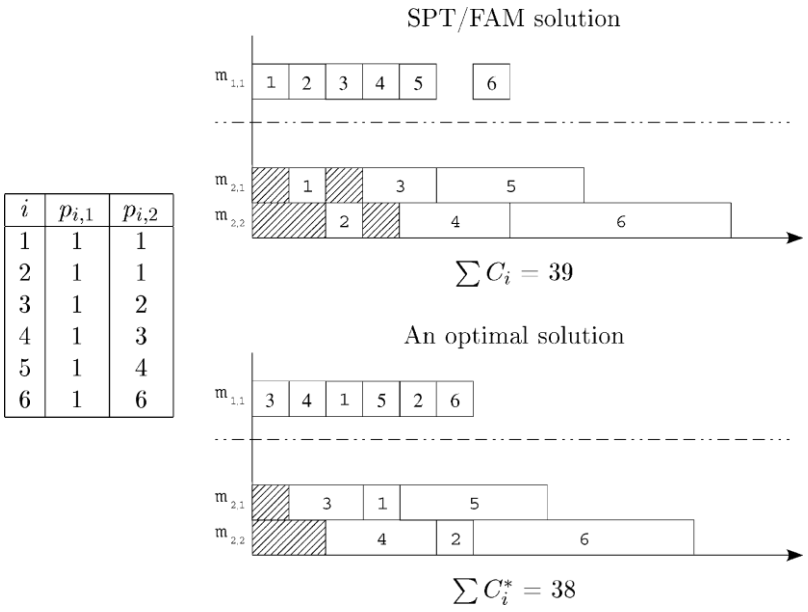
Preuve. Nous avons vu que la seconde opération du premier job ne peut démarrer avant l'instant 1 et du deuxième job avant l'instant 2. Ainsi, sans perte de généralité, nous posons $T_1 = 1$ et $T_2 = 2$. Nous avons donc considérer le problème comme un problème d'ordonnancement à deux machines parallèles où M_1 est disponible à l'instant $T_1 = 1$ et M_2 au temps $T_2 = 2$.

M_1 est le FAM, donc φ(1) est attribué à M_1. Parce $P_{\varphi(1),2} > 1$, M_2 devient le FAM. Puis, φ(2) est affecté à M_2 et parce que $P_{\varphi(1),2} \leq P_{\varphi(2),2}$, et $T_1 < T_2$, M_1 devient le FAM. Et donc activé, les tâches sont attribuées alternativement sur M_1 et M_2. Il s'ensuit que les temps d'exécution de deux les emplois ne peuvent pas être égaux, et parce que $p_{i,2} > 1$ $\forall i$,$1 \leq i \leq n$, la contrainte de nowait avec la première étage ne peut pas conduire à un temps mort sur M_1 ou sur M_2. Par conséquent, la solution renvoyée par SPT/FAM à la deuxième étage est équivalente à la solution optimale du problème

P2‖$\sum C_i$ où machine M_1 est disponible au temps $T_1 = 1$ et M_2 au temps $T_2 = 2$,[10] . Ainsi, il génère également un calendrier optimal pour le problème FH2,(1,P2)|nowait, $p_{i,1} = 1$, $p_{i,2} > 1|\sum C_i$.

Corollaire 2. Le théorème 1 ne peut pas être appliqué lorsqu'au moins une opéra temps de traitement à la deuxième étage inférieure ou égale à un.

Preuve. La preuve du théorème 1 n'est pas vérifiée si un travail est tel que $p_{i,2} \leq 1$ car dans ce cas, des temps morts peuvent être introduits sur M_1 et M_2 et le résultat n'est alors plus valide. Dans la figure 3.2, nous présenter un exemple où SPT/FAM ne renvoie pas la solution optimale.



i	$p_{i,1}$	$p_{i,2}$
1	1	1
2	1	1
3	1	2
4	1	3
5	1	4
6	1	6

Figure 3. 2 - *Exemple ou SPT/FAM n'est pas optimal.*

6. RÈDUCTION

Dans cette section, nous présentons une réduction polynomiale qui montre qu'un atelier de flow shop hybride à deux étages problème avec une seule machine au premier étage, m machines parallèles au deuxième étage, un no-wait contrainte entre les deux étages et des temps de traitement égaux à la première étage, est au moins aussi difficile comme le problème de la machine parallèle m avec un seul serveur et des temps d'installation égaux. Cette réduction est applicable au critère de durée totale d'exécution.

Théorème 3. Pour α ∈ {m , o}, les temps d'établissement s_i tous les temps de traitement égaux et arbitraires p_i, le problème Pα, S1|s_i = s|$\sum C_i$ se réduit polynomialement à FH2,(1, Pα) |$p_{i,1}$= s, nowait| $\sum$Ci

Preuve. Considérons les problèmes de décision suivants P et FH2.

Problème : Pα

Exemple : n tâches, $\{p_i\}_{1 \leq i \leq n}$ les temps de traitement, *s* et *B* deux entiers positifs.

Question : Peut-on trouver un ordonnancement réalisable au problème P α, S1|s_i= s| − avec une

complétion totale temps inférieur ou égal à B?
Problème : FH2
Exemple : ṅ travaux, $p_{i,1}$= p, ∀i , 1 ≤ i ≤ ṅ et $\{p_{i,2}\}_{1\leq i \leq \dot{n}}$ les temps de traitement à la seconde étage et un entier positif B'.

Question : Pouvons-nous trouver un ordonnancement réalisable au problème FH2,(1, Pα)|$p_{i,1}$ = p, nowait| − avec un temps de réalisation total inférieur ou égal à B' ?
Il est clair que le problème de décision FH2 ∈ NP car on peut vérifier une réponse positive dans Temps polynomial.

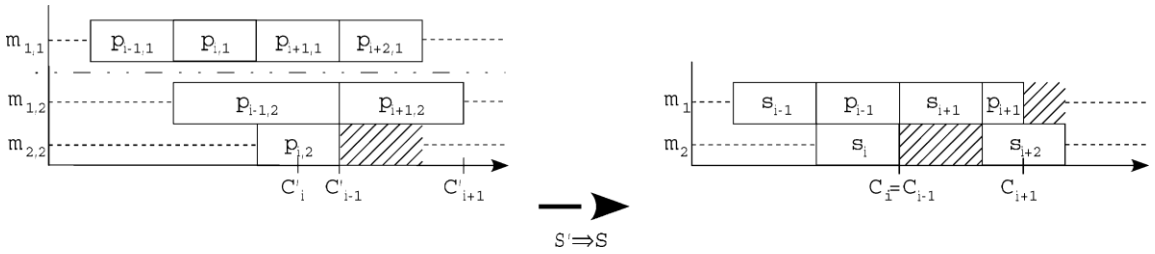


Figure 3. 3 - *ordonnancement S construits à partir d'ordonnancement S'.*

On construit une instance I du problème FH2 par la transformation polynomiale suivante :

$$n' = n;\ p_{i,1} = p = s,\ \forall i ,\ 1 \leq i \leq n;\ p_{i,2} = p_i + s_i,\ \forall i ,\ 1 \leq i \leq n;\ B' = B + n * s.$$

Supposons que la réponse à Pα soit "oui", et soit S un ordonnancement réalisable pour le problème Pα . Alors nous avons $\sum_{i=1}^{n} C_i\,(S) \leq B$. Nous construisons à partir de S une solution réalisable de FH2 comme suit : la configuration les activités de Pα sont affectées à la machine à la première étage, et l'affectation des opérations aux machines n'a pas changé. Cette solution est appelée S′, et C'_i indique le temps d'achèvement du travail i en S′. En S, la différence entre les heures de début d'une opération j et de son opération précédente i sur une machine est toujours supérieur ou égal à $p_i + s$. Ensuite, même avec la contrainte de no-wait, il est possible dans S′ d'affecter aux opérations de la première étage, les instants de début des activités de configuration dans S. Ensuite, à cause de contrainte de no-wait, nous avons $C_i' = C_i + s$; ∀i , 1 ≤ i ≤ n. Ainsi $\sum_{i=1}^{n} C_i^{'} = \sum_{i=1}^{n} C_i + ns$. Parce que $\sum_{i=1}^{n} C_i(S) \leq B$, $\sum_{i=1}^{n} C_i^{'}\,(S') \leq B + ns \Rightarrow \sum_{i=1}^{n} C_1^{'}(S') \leq B'$ et la réponse à FH2 est 'oui'.

Supposons maintenant que la réponse à FH2 soit « oui », et notons S′ une solution réalisable au problème FH2. Alors nous avons $\sum_{i=1}^{n} C_i^{'}(S') \leq B'$ pour cette solution S′. Parce que $p_{i,2} = p_i + s$, ∀i, 1 ≤ i ≤ n en S′, la différence entre l'heure de début d'une opération j et son opération précédente i sur une machine est toujours supérieur ou égal à $p_i + s$. Ainsi dans S, après avoir réduit le temps de

traitement de i à p_i, il est toujours possible d'introduire une activité de configuration avec un temps de traitement s avant j. Ainsi en S, le départ les temps des opérations sont les mêmes qu'en S′, et puis

$C_i = C_i' - s$, $\forall i, 1 \leq i \leq n$ (voir Fig 3.3). $\sum_{i=1}^{n} C_i' \leq B' \Rightarrow \sum_{i=1}^{n} C_i + ns \leq B + ns \Rightarrow \sum_{i=1}^{n} C_i$ B puis la réponse à Pα est "oui". Ainsi, le problème Pα se réduit polynomialement au problème FH2 : Pα ⍺ FH2.

Donc, parce que le P2, S1|s_i=s|$\sum C_i$ problème est NP-difficile [9] , on en déduit que le FH2, (1, P2)|p_i,1 = p, no-wait| $\sum C_i$ Le problème est NP-difficile. De plus, résoudre le FH2,(1,P2) | p_i,1 = p, no-wait| $\sum C_i$ problème avec $p_{i,2}$ nombres réels est équivalent à résoudre le FH2,(1, P2)|$p_{i,1}$ = p, no-wait| $\sum C_i$ où $p_{i,2}$ sont des entiers positifs. Comme ce problème est NP-difficile, le FH2,(1,P2) |$p_{i,1}$ = 1, no-wait| $\sum C_i$ problème avec $p_{i,2}$ nombres réels est NP-difficile.

7.ALGORITHME GUIRCHOUN ET MARTINEAU

Nous considérons l'algorithme 3.1. divise les emplois en deux ensembles et trie l'un d'eux selon la règle SPT en un temps O(n log(n)) puis attribue les travaux selon à la règle FAM en temps O(n). Ainsi, sa complexité est en temps O(n log(n)). Γs , Γ_1 et Γ_2 font référence au heure d'achèvement du dernier travail sur le serveur, respectivement sur M_1 et M_2. (Γs ,Γ_1 , Γ_2) est appelé le profile de l'ordonnancement partiel.

Algorithme 3.1 Algorithme GUIRCHOUN et MARTINEAU

Début

$\Gamma s = 0, \Gamma_1 = 1, \Gamma_2 = 2, \Gamma_m = m$.

1: **Déterminer** les ensembles A et B

2:**Tant que** (A ∪ B) ≠ ∅ **Faire**

3: Soit M_k la machine telle que $\Gamma_k = \min \{ \Gamma_k / k = 1,\ldots,m \}$

4: **Si** ((∃ compatible de A) ou (B ≠ ∅)) **Alors**

5: **Placer** le travail i de A sur M_k

6: A ← A \ { i }

7: $\Gamma_k \leftarrow \Gamma_k + p_{i2}$

8: **Sinon**

9: **Placer** le travail i de B de plus petit $p_{i,2}$ sur M_k

10: B← B \ { i }

11: $\Gamma_k \leftarrow \Gamma_k + p_{i2}$

12: **Fin Si**

13: **Fin Tant que**

14: **Fin**

On définit des ensembles A et lister B par A = {i | $p_{i,2} = 1$} et B = {i | $p_{i,2} > 2$}, où B est trié dans l'ordre $p_{i,2}$ non décroissant. L'idée est de placer judicieusement les jobs avec $p_{i,2} < 2$ dans afin de combler les temps morts à la première étape, créée par le $p_{i,2}$ le plus long. Faire comme ça de manière itérative réduit le temps total de réalisation. En effet, $\sum_{i=1}^{n} C_{i,2} = \sum_{i=1}^{n} C_{i,1} + \sum_{i=1}^{n} p_{i,2}$ à cause de nowait contrainte et $\sum_{i=1}^{n} C_{i,1} = n(n+1)/ + \sum_{i=1}^{n}(n-i+1)\,\delta_i$, où δ_i est le temps d'inactivité entre le travail i − 1 et le travail i sur le serveur à la première étage. Minimisant ainsi $\sum_{i=1}^{n} C_{i,2}$ équivaut à minimiser $\sum_{i=1}^{n}(n-i+1)\,\delta_i$.

Dans la suite, on note ϭ la liste des temps de réalisation des travaux à la seconde étape, triés dans l'ordre non décroissant : $\vartheta_{[\ell]}$ désigne le ℓ le temps de réalisation. Par extension, le travail en position ℓ dans ϭ désigne le travail avec ℓ l'heure d'achèvement dans ϭ.

Théorème 5. L'algorithme 3.1 renvoie une solution optimale au FH2,(1,P2) |nowait, $p_{i,l}$=1|∑C_i problème.

Preuve. Dans l'ensemble B, les travaux sont triés dans l'ordre non décroissant de leurs temps de traitement à la deuxième étage. Soit Définissons α = |{i | $p_{i,2} = 2$}|.

Selon l'algorithme 3.1, lors des α premières étapes, tous les jobs i en ϭ avec $p_{i,2} = 2$ sont ordonnancés sur M_l et sur M_2 alternativement. Ainsi, à une étape arbitraire υ ≤ α , l'ordonnancement partiel a un profile (υ, υ + 1 , υ + 2) ou (υ , υ+ 2 , υ + 1) et le serveur (à la première étage) exécute des tâches sans temps d'inactivité dans [0 ,υ]. Après l'étape α, les tâches en A ne sont pas ordonnancés. A l'étape (α+ 1), l'algorithme 3.1 ordonnance la première tâche ℓ de B avec $p_{\ell,2} > 2$, dans l'intervalle de temps [α + 1, α +1+ $p_{\ell,2}$]. A l'étape (α + 2), les ordonnancements de l'algorithme 3.1 d'un travail de A en [α+ 2 ,α + 3] et itère avec les travaux de A jusqu'à ce que l'ordonnancement partiel ait un profile de type (ε, ε+1,ε + 2) ou (ε, ε+2,ε + 1) avec ε = α + $p_{\ell,2}$ − 1. Ensuite, l'algorithme 3.1 programme le prochain travail de B et le processus itère jusqu'à ce que

l'ensemble A ou l'ensemble B soit vide. Supposons que l'ensemble B est vide. Puis, L'algorithme 3.1 planifie les travaux de A consécutivement sur la même machine. Parce qu'il n'y a pas de temps mort sur le serveur et parce que $\sum_{i=1}^{n} C_{i,2} = \sum_{i=1}^{n} C_{i,1} + \sum_{i=1}^{n} p_{i,2}$, l'ordonnancement est optimal. Supposons ensemble A est vide. A ce moment, disons à l'itération ε, le profil d'ordonnancement courant est (ε, ε+ 1, ε′), avec ε′ ≥ ε + 2 (voir Fig 3.4). Ensuite, les tâches restantes de B sont planifiées selon l'algorithme SPT/FAM. On note par γ le nombre de tâches non planifiées à l'étape ε, c'est-à-dire γ = n − ε. γ est soit impair ou pair et on pose γ = 2k + h avec h∈ {0 , 1}. Nous avons $\sum_{i=1}^{n} c_i + \sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i]} + \sum_{i=\varepsilon+h+1}^{n} \mathcal{C}_{[i]}$.

• Considérons les premiers travaux ε + h . $\sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i]} = \sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i],1} + \sum_{i=1}^{\varepsilon+h} p_{[i],2}$.

La valeur $\sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i],1}$ ne peut pas être réduit puisque le serveur travaille sans temps mort en [0, ε + h].

Ainsi, $\sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i],1} = p_{[1],1} + p_{[2],1} + \ldots . + p[\varepsilon + h]_{,1} = \frac{(\varepsilon+h)(\varepsilon+h+1)}{2} = k \Rightarrow \sum_{i=1}^{\varepsilon+h} \mathcal{C}_{[i]} = k + \sum_{i=1}^{\varepsilon+h} p_{[i],2}$

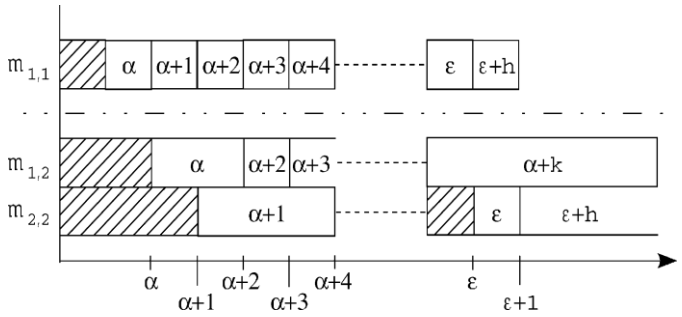


Figure 3.4 *- ordonnancement S construits à partir d'ordonnancement S′.*

La valeur $\sum_{i=1}^{\varepsilon+h} p_{[i],2}$ ne peut pas être réduit puisque cette somme contient le ε+h plus petit $p_{i,2}$, donc

$\mathcal{C}_{[n]} + \mathcal{C}_{[n-1]} = T_1 + T_2 + \sum_{i=1}^{n} p_{[i],2}$.

$\mathcal{C}_{[n]} = \mathcal{C}_{[n-2]} + p_{[i],2}$.

$\mathcal{C}_{[n-1]} = \mathcal{C}_{[n-3]} + p_{[n-1],2}$.

$\Rightarrow \mathcal{C}_{[n-2]} + \mathcal{C}_{[n-3]} = T_1 + T_2 + \sum_{i=1}^{n-2} p_{[i],2}$.

$\Rightarrow \mathcal{C}_{[\ell]} + \mathcal{C}_{[\ell-1]} = T_1 + T_2 + \sum_{i=1}^{\ell} p_{[i],2}$, $\forall\, \ell$, n − 2k + 2 $\leq \ell \leq$ n. (11)

$T_1 + T_2 + \sum_{i=1}^{\ell} p_{[i],2}$ est une borne inférieure pour $\mathcal{C}_{[\ell]} + \mathcal{C}_{[\ell-1]}$, $\forall\, \ell, 1 \leq \ell \leq$ n , Donc, on en Déduit que:

$\mathcal{C}[\varepsilon + h + 1] + \mathcal{C}[\varepsilon + h + 2] \geq T_1 + T_2 + \sum_{i=1}^{\varepsilon+h+1} p_{[i],2}$, $\mathcal{C}_{[\ell]} + \mathcal{C}_{[\ell-1]} \geq T_1 + T_2 + \sum_{i=1}^{n} p_{[i],2}$

Nous fixons: LB=$\frac{n-(\varepsilon+h)}{2} * (T_1 + T_2) + \sum_{W=0}^{½(n-(\varepsilon+h+1))} \sum_{i=1}^{\varepsilon+h+2w+1} p_{[i],2}$.

A cause de l équation (11), l'ordonnancement obtenu par l'algorithme 3.1 pour les travaux en position ε+ h + 1 à n est tel que $\sum_{i=\varepsilon+h+1}^{n} \theta_{[\ell]} = \mathrm{LB}$, et donc optimal.

Pour tout ordonnancement σ, $\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{\varepsilon+h} C_i(\sigma) + \sum_{i=\varepsilon+h+1}^{n} C_i(\sigma)$ Pour l'ordonnancement σ^a obtenu par Algorithme 3.1, $\sum_{i=1}^{n} C_i(\sigma^a) = \sum_{i=1}^{\varepsilon+h} C_i(\sigma^a) + \sum_{i=\varepsilon+h+1}^{n} C_i(\sigma^a)$. Nous avons montré que $\forall\sigma$, $\sum_{i=1}^{\varepsilon+h} C_i(\sigma) \geq \sum_{i=1_i}^{\varepsilon+h} C_i(\sigma^a)$. et , $\forall\sigma, \sum_{i=\varepsilon+h+1}^{n} C_i(\sigma) \geq \sum_{i=\varepsilon+h+1}^{n} C_i(\sigma^a)$. Ainsi , $\forall\sigma, \sum_{i=1}^{n} C_i(\sigma) \geq \sum_{i=1}^{n} C_i(\sigma^a)$ .Et donc, la valeur obtenue de $\sum_{i=1}^{n} C_i(\sigma^a)$ est optimal.

8.CONCLUSION

Dans ce chapitre, nous considérons un problème d'ordonnancement de machines parallèles avec un serveur dans un ordinateur système, modélisé par un problème d'ordonnancement d'atelier hybride à deux étages avec une contrainte de no-wait.

Nous montrons que le problème d'atelier de flow shop hybride à deux étages est au moins aussi difficile comme le problème d'ordonnancement des machines parallèles avec un seul serveur. On en déduit que le problème à l'étude avec des temps de traitement réels est NP-difficile. Cependant, avec des temps de traitement entiers, nous proposons un algorithme O(n log(n)) qui résout le problème de manière optimale. Enfin, nous montrons qu'un solution optimale au problème de flow shop hybride à deux étages avec la contrainte de no-wait est une solution optimale solution au problème sans la contrainte de no-wait.

CHAPITRE 04

« CONCEPTION ET IMPLIMENTATION DU PROBLÈME »

1. INTRODUCTION

Dans ce chapitre on s'est penché sur un cas pratique , On remarque que ce type d'ordonnancement est un ordonnancement de type flow shop hybride à deux machines et dix Jobs ($F\ 2 \parallel C_{max}$), notre but est d'organiser l'ensemble des 10 tâches sur les 2 machines de telle sorte à optimiser et à minimiser la durée de fin de projet, pour cela nous avons choisir l'algorithme Guirchoun et Martineau le plus utilisé pour résoudre ce type de problème d'ordonnancement .

2. LONGAGE DE PROGRAMMATION ET ENVIRENEMENT DE DEVLOPEMENT

Nous utilisons le langage Pascal pour implémenter notre modèle car il est un langage de programmation impératif qui, se caractérise par une syntaxe claire, rigoureuse et facilitant la structuration des programmes . Notre travail écrit dans l'environnement de développement Embarcadero RAD Studio sur Windows 10 Professionnel . Et nous avons utilisée comme élément matériel un ordinateur TOSHIBA qui possède les propriétés suivants :

- Un processeur Intel (R) Core(TM) i5-4310U CPU @ 2.00 GHz 2.60 GHz.
- Une mémoire vive de 8 Go .
- Système d'exploitation 64 bits .

2.1 LE LANGAGE PASCAL

Le langage de programmation Pascal (dont le nom vient du mathématicien français Blaise Pascal) a été inventé par Niklaus Wirth dans les années 1970 avec l'aide d'un de ses étudiants, Urs Amman. Il a été conçu pour servir à l'enseignement de la programmation de manière rigoureuse mais simple. Ce langage est l'un de ceux qui ont servi à enseigner la programmation structurée. Le goto ou saut n'importe où dans le programme (dit « branchement ») est fortement déconseillé dans ce langage, le programme est un assemblage de procédures et de fonctions, dans lesquelles on peut utiliser des blocs conditionnels (if, case) et répétitifs (while, for, repeat) ayant chacun une entrée et une sortie afin de faciliter les contrôles, ce qui aboutit à des mises au point rapides et sûres.[C]

2.2 L'ONVERONNEMENT EMBARCADERO RAD STUDIO (DELPHI)

En 1995, pour contrecarrer Microsoft et la programmation visuelle du Visual Basic, Borland sort Delphi. Contrairement à VB qui produit du p-code, Delphi produit du code machine, plus rapide.

On voit également apparaître la bibliothèque VCL servant d'interface aux bibliothèques système (en) de

Windows, facilitant grandement le développement.

Au début des années 2000, Borland produit Kylix, l'équivalent de Delphi pour le monde Linux, qui n'aura pas un grand succès. Au début des années 2010, Embarcadero, qui a repris les activités d'outils de développement de Borland, produit Delphi XE, dont les versions récentes sont compatibles avec Windows, OS X et iOS. Lazarus, est un logiciel libre de développement intégré RAD légèrement différent, permettant de compiler sur différents systèmes d'exploitation tel que Windows, GNU/Linux, OS X, Unix, OS/2, ReactOS, Haiku et plateformes telles que x86, x86-64, ARM, SPARC, PowerPC, IA-64.[D]

3. LE FLOW SHOP HYBRIDE À DEUX ÈTAGES

Dans un environnement informatique composé d'un « serveur » et de « M » machines parallèles identiques appelées processeur , « N » taches sont à exécuter sans interruption . Avant son exécution , un tache i (i = 1,…,n) doit être chargée sur le serveur . Ayant termine un chargement , le serveur est libre d'en effectuer un autre et le chargement d'une tache doit être suivi de son exécution .

Ce problème est appelé un flow shop hybride à deux étages et est note P2, S1| p_i, s_i |C_{max} .Dans la littérature , il est prouvé NP-difficile .

Dans cet environnement , nous considérons que le temps de chargement d'une tache sont tous unitaires . De plus, les temps de transfert ou de communication entre le serveur et les machines parallèles sont supposes négligeables .Le but est de déterminer un ordonnancement réalisable qui minimise la somme des dates de fin des travaux appelé « flow time ». La récupération des données nécessaires à l'exécution d'une tache est réalisée par la machine serveur . Durant ce chargement , il n'est pas nécessaire que la machine parallèle soit disponible.

En effet, toutes les machines parallèles ont à leur disposition un coprocesseur de communication qui leur permet de recevoir les informations du serveur tout en continuant leurs opérations. Ce système informatique s'interprète comme un flow shop hybride à deus étages . Une seule machine « serveur » est disponible au première étage , les « M » machines parallèles sont au second étage .

Nous supposons qu'aucune aire de stockage n'existe entre les deux étages .Le flow shop hybride est important.

4. LES MÈTHODES UTILISÈES POUR LA RESOLUTION NOTRE PROBLEME

Pour résoudre notre problème nous avons utilise l'algorithme de Guirchoun et Martineau , l'algorithme proposé donne une solution optimale en supposant que les temps d'exécution au

deuxième étage sont des entiers strictement positifs. Les temps d'exécution au premier étage sont unitaires.

5. L'ARCHITECTURE GÈNÈRALE DE LA RÈALISATION

Nous avons présenté dans le schéma suivant (Figure 4.1) la procédure que nous avons créées pour réaliser notre résolution.

Ci-dessous ,nous fournirons une explication détaillée du procédure.

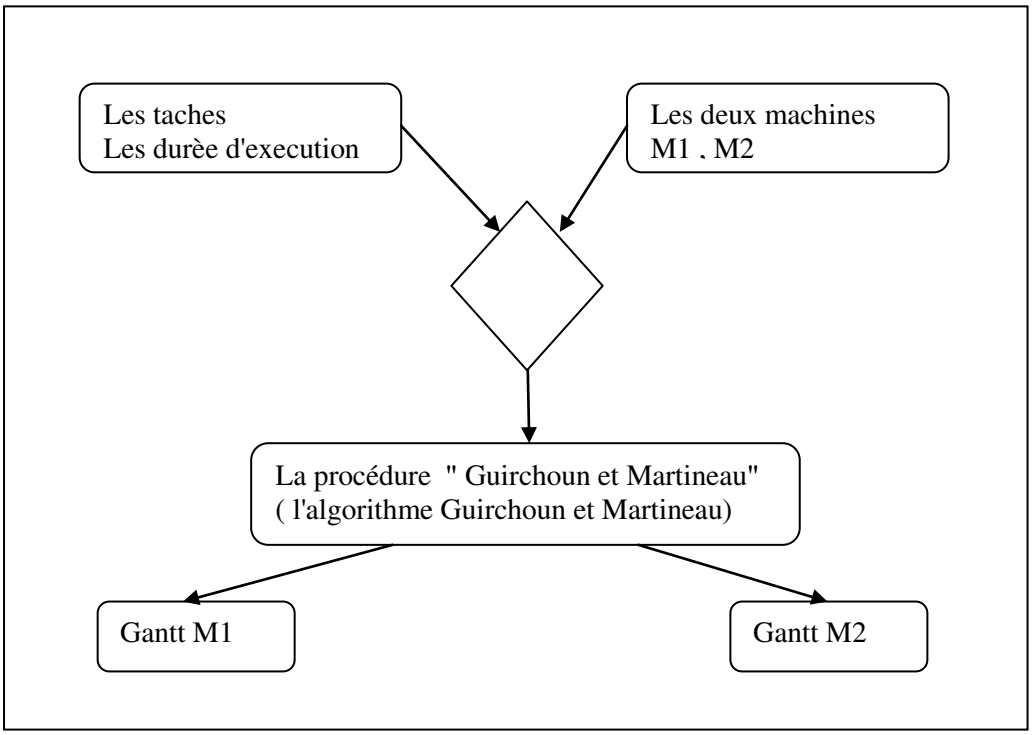


Figure 4.1 - *Le schéma de réalisation de problème* .

Nous avons deux machines ,Nous entrons le nombre de tâches à remplir dans le tableau :

- La procédure "Guirchoun et Martineau" : appliquons l'algorithme de Guirchoun et Martineau au tableau pour nous déterminer les ensembles A et B et nous testons la compatibilité et à partir de celui-ci produit le diagramme de Gantt pour la Machine 1 et le diagramme de Gantt pour la Machine 2 afin d'obtenir le Maksepan. .et le diagramme de Gantt est une matrice , la durée de tache c'est l'itération de sa nombre.

6. ILLUSTRATION DE L'APPLICATION

Notre application se compose de trois fenêtres d'affichage, la première représente la page d'accueil (Figure 4.2), la deuxième montre le menu de l'application (Figure 4.3), la troisième qui permet de saisir le nombres de machines et le nombre de taches et leur temps d'exécutions dans les machines (Figure 4.4), et la quatrième représente l'interface de l'implémentation de problème(Figure 4.5).

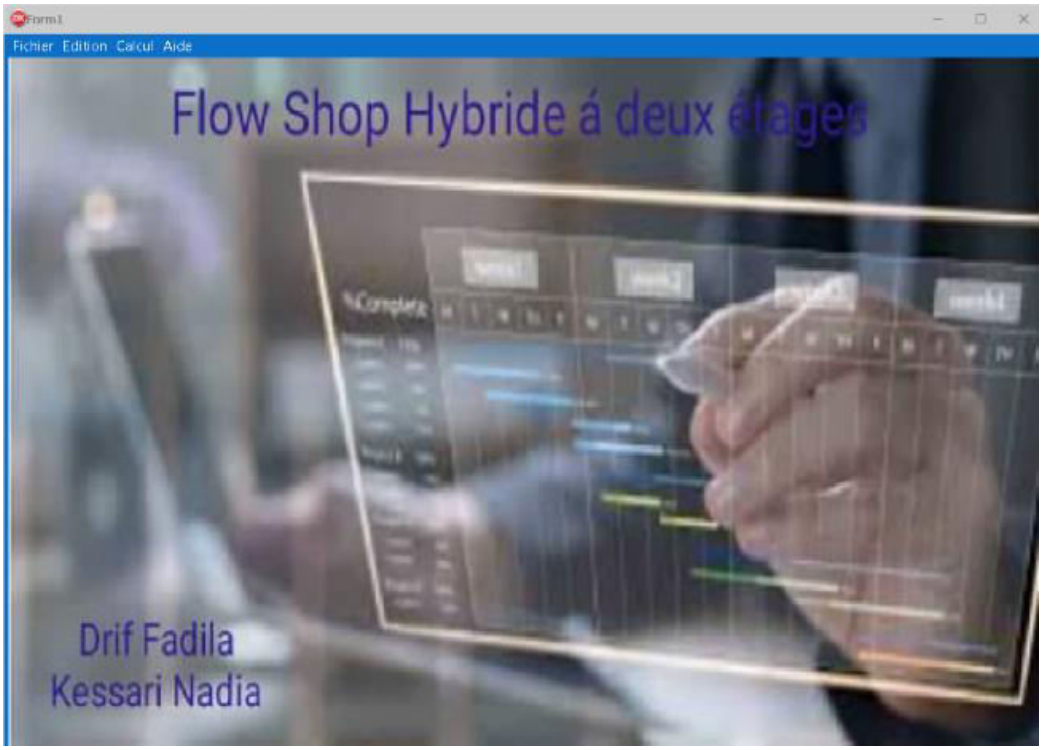


Figure 4.2 - *La page d'entrée de l'application.*

Figure 4.3 - *le menu de l'application.*

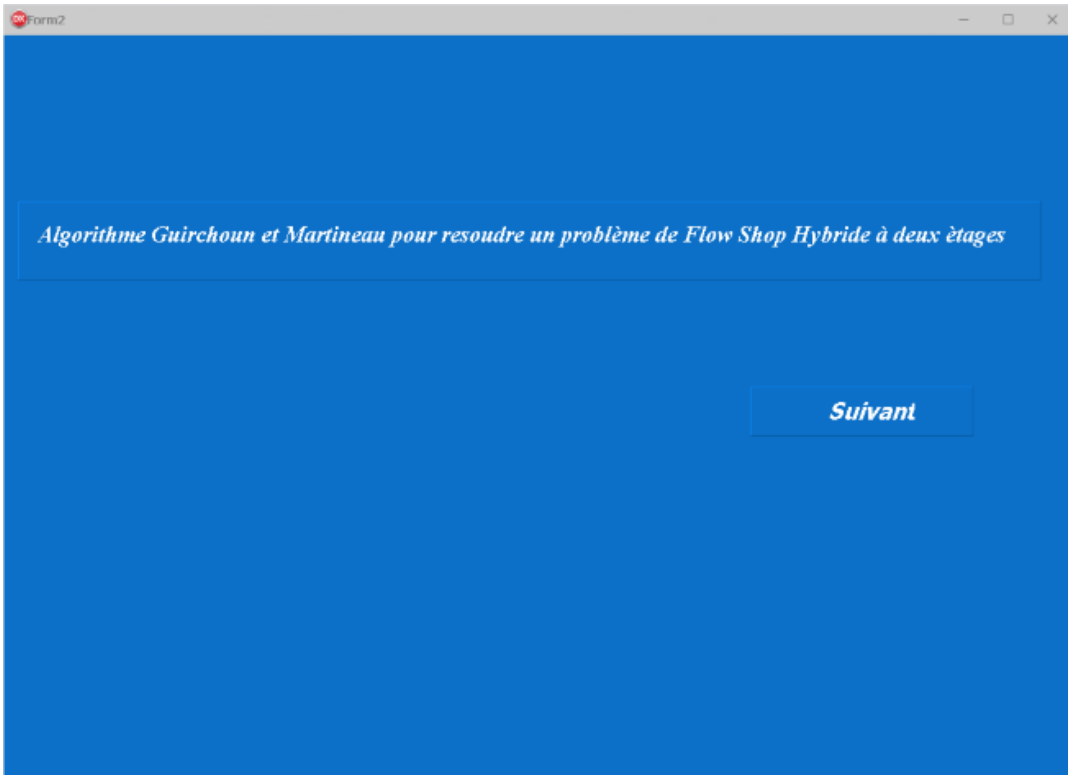


Figure 4.4 - *L'algorithme utilisée.*

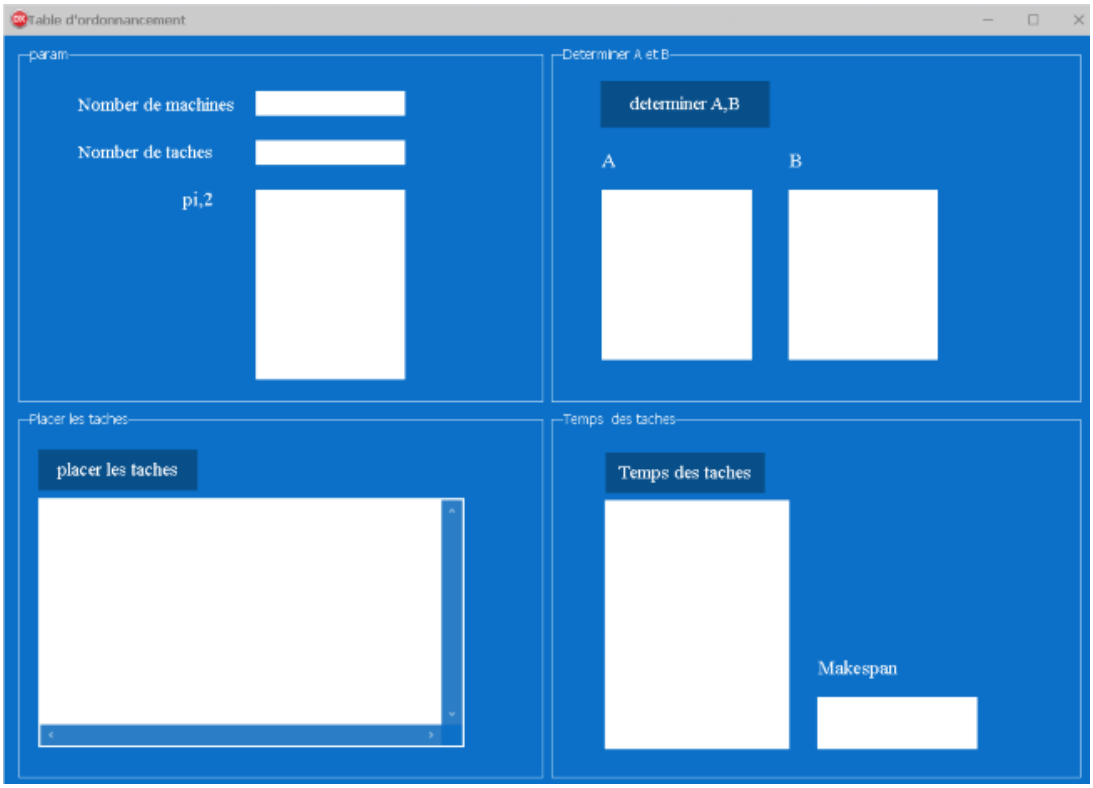


Figure 4.5 - *L'interface de l'implémentation*

7. QUELQUE EXEMPLE SUR L'IMPLÈMENTATION

Exemple 4.1 :

Soit à exécuter 10 taches dans un système informatique formé d'un serveur et 4 machines parallèles. Les temps d'exécution des taches sont donnes par le tableau ci-dessous:

Le temps de chargement de chaque tache sur le serveur $p_{i,1}$ est de « une » unité de temps.

I	1	2	3	4	5	6	7	8	9	10
$p_{i,1}$	1	1	1	1	1	1	1	1	1	1
$p_{i,2}$	1	2	3	3	4	5	6	8	9	9

Tableau 4.1 *– Temps d'exécution des taches.*

Les sous ensembles A et B sont formés respectivement de tâches dont $p_{i,2} < 4$, $p_{i,2} \geq 4$, 4 étant le nombre des machines. Ils seront alors : A = {1, 2, 3, 4} et B = {5, 6, 7, 8,9, 10}.

L'application de l'algorithme sur l'instance donne l'ordonnancement suivant.

Itération 1 :

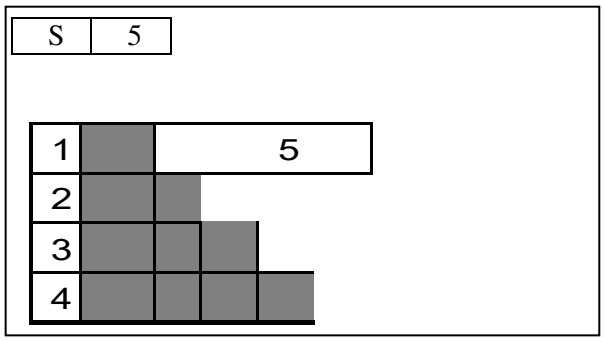


Figure 4.6- *Itération 1.*

On exécute la tache 5 sur la machine 1

Itération 2 :

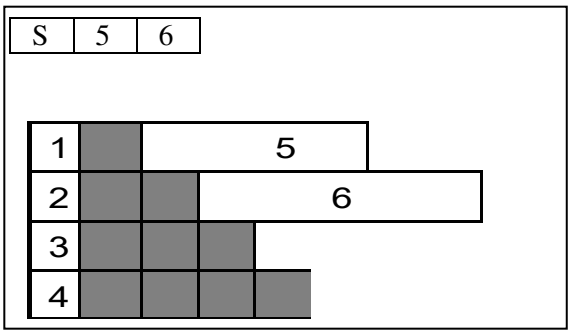


Figure 4.7- *Itération 2.*

On exécute la tache 6 sur la machine 2

Les itérations se succèdent . On omettra volontairement de les reproduire . Nous donnons l'avant dernière et la dernière itération .

Itération 9 :

On exécute la tache 10 sur la machine 2.

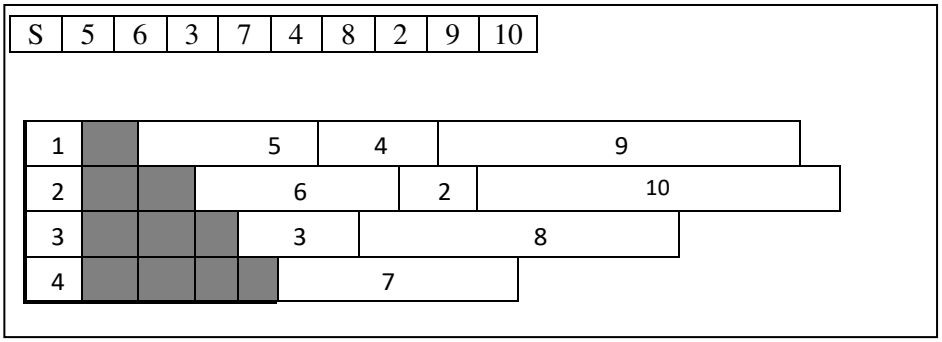


Figure 4.8 – *Itération9.*

Itération 10 :

On exécute la tache 1 sur la machine 4.

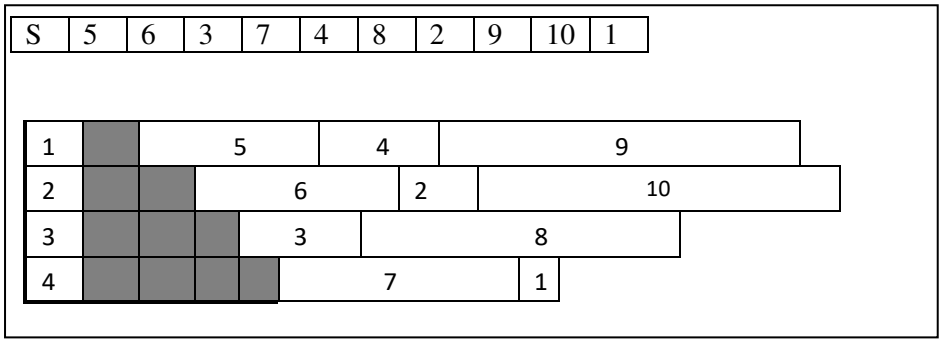


***Figure 4.9** – Ordonnancement final est optimal.*

Ainsi se lit directement l'ordonnancement optimal de Makespan 105 unités de temps. Les temps de fin d'exécution sont : $C_1 = 11$, $C_2 = 9$, $C_3 = 6$, $C_4 = 8$, $C_5 = 5$, $C_6 = 7$, $C_7 = 10$,$C_8 = 14$, $C_9 = 17$ et $C_{10} = 18$.

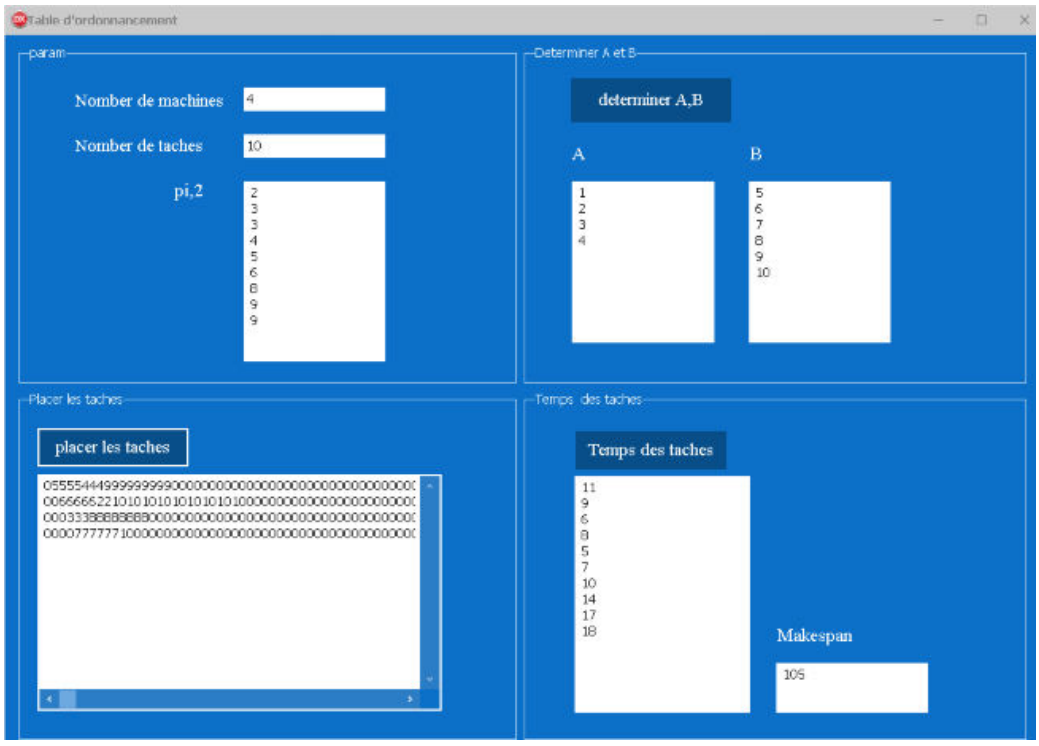


***Figure 4.10** – L'implémentation du l'exemple 4.1.*

Exemple 4.2 :

Soit à exécuter 8 taches dans un système informatique formé d'un serveur et 2 machines parallèles. Les temps d'exécution des taches sont donnes par le tableau ci-dessous:

Le temps de chargement de chaque tache sur le serveur $p_{i,1}$ est de « une » unité de temps.

I	1	2	3	4	5	6	7	8
$p_{i,1}$	1	1	1	1	1	1	1	1
$p_{i,2}$	1	1	1	2	3	4	6	6

Tableau 4.2 – *Temps d'exécution des taches .*

Les sous ensembles A et B sont formés respectivement de tâches dont $p_{i,2} < 2$, $p_{i,2} \geq 2$, 2 étant le nombre des machines. Ils seront alors : A = {1, 2, 3} et B = {4, 5, 6, 7, 8 }.

L'application de l'algorithme sur l'instance donne l'ordonnancement suivant:

Itération 1:

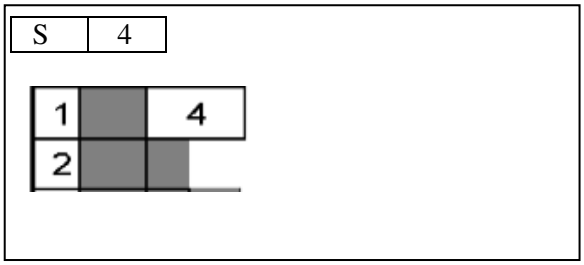


Figure 4.11 – *Itération 1.*

On exécute la tache 4 sur la machine 1.

Itération 2:

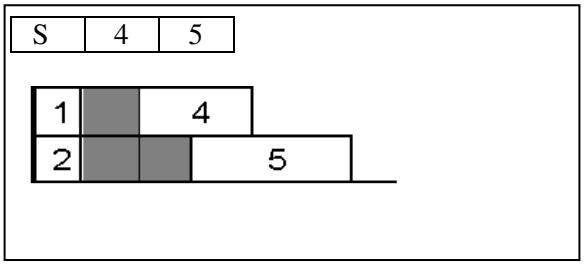


Figure 4.12 – *Itération 2.*

On exécute la tache 5 sur la machine 2.

Les itérations se succèdent . On omettra volontairement de les reproduire . Nous donnons l'avant dernière et la dernière itération .

Itération 7:

On exécute la tache 7 sur la machine 2.

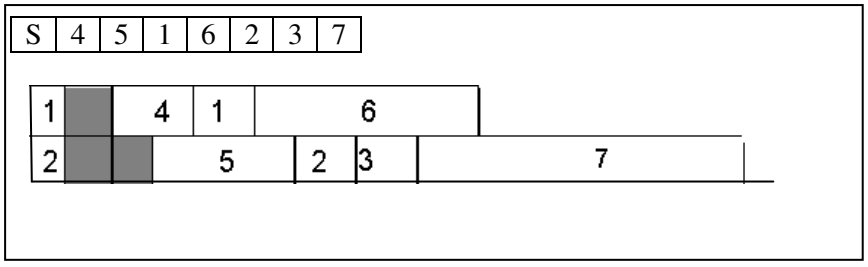


Figure 4.13 – *Itération 7.*

Itération 8:

On exécute la tache 8 sur la machine 1.

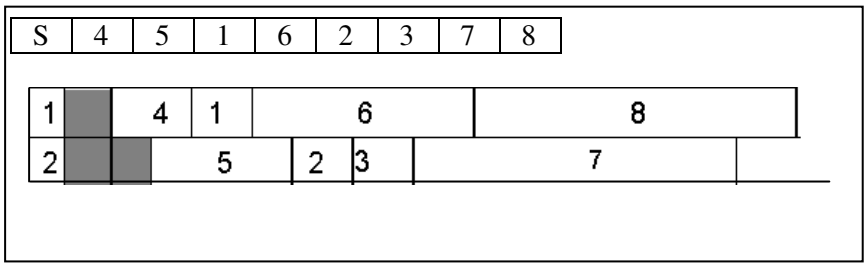


Figure 4.14– *Ordonnancement final est optimal.*

Ainsi se lit directement l'ordonnancement optimal de Makespan 60 unités de temps. Les temps de fin d'exécution sont : $C_1 = 4$, $C_2 = 6$, $C_3 = 7$, $C_4 = 3$, $C_5 = 5$, $C_6 = 8$, $C_7 = 13$ et $C_8 = 14$.

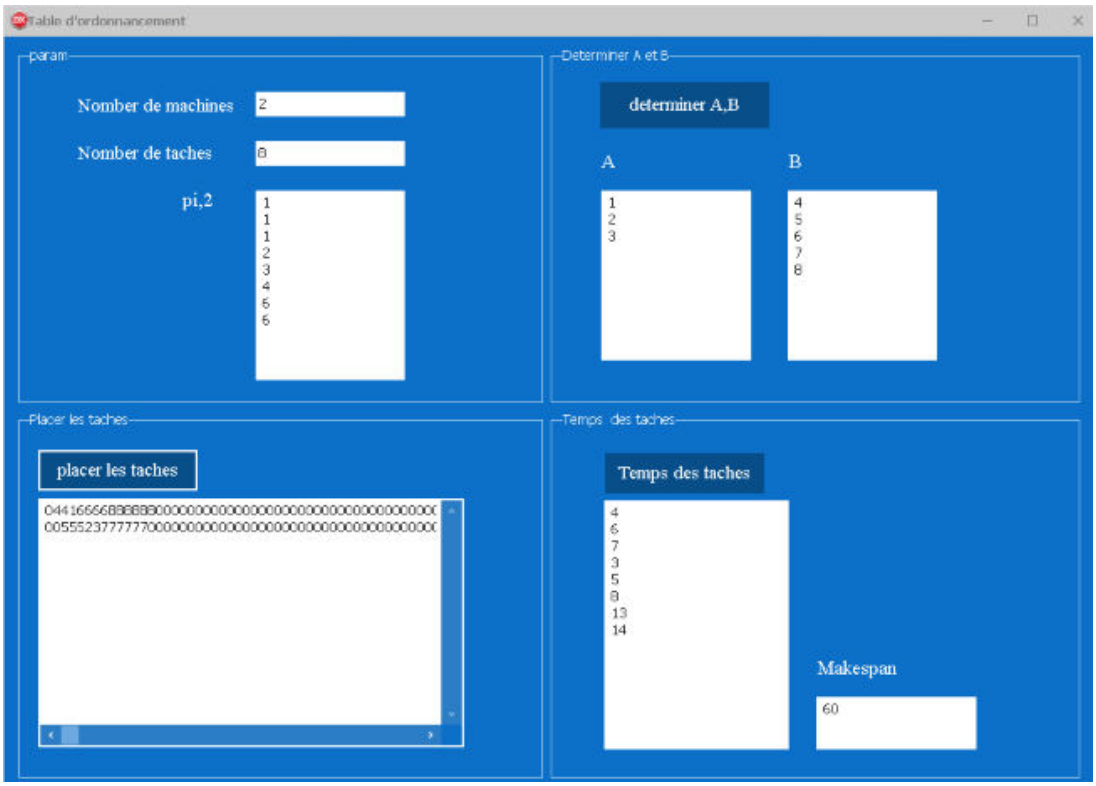


Figure 4.15 – *L'implémentation du l'exemple 4.2.*

8. Conclusion

Nous avons présenté dans ce chapitre l'application développée, le langage et la méthode utilisée pour résoudre le problème que nous avons couvert dans ce mémoire.

Ensuite, on a présenté l'implémentation de notre application et nous avons considéré des exemples avec les résultats obtenus.

Conclusion générale

L'environnement de production Flow-Shop est d'une grande importance pour l'ingénierie, représentantprès d'un quart des systèmes de production, ce qui nous a incité à considérer cet environnement. Dans cette perspective, notre travail est centré sur les problèmes d'ordonnancement de flow shop
hybride pour réduire le temps total de fin d'exécution des jobs (Makespan). A notre connaissance, peuont déjà abordé ce problème dans la littérature .

Dans ce travail, nous avons traité un problème de l'ordonnancement des ateliers de type Flow Shop Hybride . Nous avons adopté une méthodologie hiérarchique qui consiste à décomposer le problème initial en deux sous problèmes plus simples, ce qui permet de diminuer la complexité problématique. Donc, la résolution se fait en deux phases : phase d'assignement des machines et phase de séquencementdes opérations au niveau de chaque machine .

Selon notre recherche bibliographique, nous avons également remarqué que la majorité des travaux proposent soit des résultats de complexité , soit des algorithmes d'approximation . Cependant, il n'y aque des résultats de calcul sporadiques sur ce problème. Ceci nous a encore motivé à implémenter des méthodes exactes et des méthodes approchés afin de fournir des résultats de calcul .

Dans la première partie de cette thèse, nous avons fourni quelques généralités sur l'ordonnancement .Ensuite, nous avons présenté les différentes techniques utilisées pour la résolution des problèmes d'optimisation combinatoire en général, et les problèmes d'ordonnancement en particulier à savoir les méthodes exactes et les méthodes approchées. Pour chaque méthode, l'idée générale a été rapportée, plus de détails ont été donnés que pour les méthodes que nous utilisons dans notre étude .

La résolution du problème d'ordonnancement dans un atelier Flow-Shop hybride à deux etages avec afait l'objet de la deuxième partie de cette thèse .

Nous avons décidé d'implémenter un algorithme Guirchoun et Martineau aux résultats obtenus par lesproblèmes d'ordonnancement de type flow shop hybride et plus particulièrement par le problème de machines parallèles identiques .

L'algorithme proposé par Guirchoun et Martineau à travers lequel les résultats obtenus montrent une bonne maîtrise du processus d'optimisation lors des différentes itérations . L'algorithme utilisé permetde rechercher le meilleur ordre d'exécution des opérations au niveau de chaque machine.

Références bibliographiques

[1] Hela BOUKEF BEN OTHMAN , « l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques Optimisation par algorithmes génétiques et essaims particulaires», 2009. Thèse de doctorat.

[2] Sana ELBAHLOUL ,« Flow Shop à deux machines avec des temps de latence : approche exacte et heuristique » , 2008 .

[3] Abderrahman CHERGUI , Abd Naceur DAHMANI ,Abdellah ADDA ABBOU «Ordonnancement d'un flow-shop par métaheuristique hybride » , 2017.

[4] Toufik BENTRCIA,« Ordonnancement des systèmes de production sous contraintes technologiques ou contextuelles : Modélisation , étude de complexité et approches intégrées de résolution », 2017. Thèse doctorat.

[5] Mounir BELKHOUS, Samia BOURAHLA, «Modélisation d'un processus de fabrication et résolution d'un problème d'ordonnancement de type Flow-Shop. Cas «ENIEM».».2017.

[6] Imane LARIBI, «Résolution de problèmes d'ordonnancement de type Flow-Shop de permutation en présence de contraintes de ressources non-renouvelables», 2018. Thèse doctorat.

[7] Reda SAKRI, « Le flow shop hybride à deux étages », 2008.

[8] Graham RL, Lawler EL , Lenstra JK, Kan AHGR, « Optimization and approximation in deterministe sequencing and scheduling theory : a survey. Annals of descrete Mathematics »,1979.

[9] Hall NG, Potts CN, Sriskandarajah C, « Parallel machine scheduling with a common server. Discrete Applied Mathematics », 2000.

[10] Kaspi M, Montreuil B, « On the scheduling of identical parallel processes with arbitrary initial processor available time », 1988.

[11] Vignier A, Billaut J-C , Proust C, « Les flowshop hybrides : ètat de l'art .RAIRO-RO », 1999.

[12] Bourdeaud'huy T, Korbaa O, « Un modèle mathematique pour la resolution du probleme d'ordonnancement cyclique avec minimisation de l'en-cours », 2006

[13] Lopez P, Esquirol P , « L'ordonnancement » , Economia , 1999.

[14] Fatma TANGOUR TOUMI, « Ordonnancement dynamique dans les industries agroalimentaires » , Thèse de doctorat en Automatique et informatique Industrielle, 2007.

[15] R. MACNAUGHTON. « Scheduling with deadlines and loss functions», Management Science,(6), pp. 1-12, 19.59.

[16] V. Y. SHEN et Y. E. CHEN. « A scheduling strategy for the ftowshop problem in a system with two classes of processors », ln Conference on Information and Systems Science, pp. 645-649, 1972.

[17] R. E. BUTEN et V.Y. SHEN. « A scheduling model for computer systems with two classes of processors», ln Sagomore Computer Conference on Paraud Processing, 1973.

[18] A. HORN. « Some simple scheduling algorithms»,Naval Research Logistics Quarterly, (21): pp. 177-185, 1974.

[19] R. J. PAUL. « A production scheduling problem in the glass-container industry »,Operations Research, 21.(2):pp.290-302, 1979.

[20] A.LANGSTON. « Interstage transportation planning in the deterministic ftowshop environment», Operations Research, 35(4):pp. 556-564, 1987.

[21] W. SZWARC et J. N. D. GUPTA. « A flow-shop problem with sequence dependant additive setup time», Naval Research Logistics Quarterly (23), pp. 619-627;1987

[22] Koulamas CP et Smith ML. « Look-ahead scheduling for minimizing machine interference», International Journal of Production Research;pp.26-39,1988.

[23] C. SRISKANDARAJAH et S. P. SETHI. « Scheduling algorithms for flexible flow-shops: Worst and average case performance», European Journal of OperationsResearch,(43): pp.143-160, 1989.

[24] D. SHERALI , S. C. SARIN, et M. S. KODIALAM. « Models and algorithm for a two- stage production process» ,Production Planning and Control,(1):pp.27-39, 1990.

[25] J. N. D. GUPTA et E. A. TUNC. « Schedules for a two-stage hybrid flowshop with parallel machines at the second stage», International Journal of Production Research, 29(7):pp.1489-1502, 1991.

[26] D. L. SANTOS, J. L. HUNSUCKER, et D. E. DEAL. « Global lower bounds for flow shop with multiple processors», European Journal of Operations Research, (80):pp.112-120,1995.

[27] C. LEE, T. C. E. CHENG, et B. M. T. LIN. « Minimizing the makespan in the 3- machine assembly flowshop scheduling problem»,Management Science, 39(5):pp. 616-62.5:1993.

[28] Gupta et Tunc. « Scheduling a two-stage hybrid ftowshop with separable setup and removal times», European Journal of Operations Research, (77):pp.415-428, 1994.

[29] C. Y. LEE et G. L. VAIRAKTARAKIS. « Minimizing makespan in hybrid flowshop»,Operations Research Letters, 16(3): pp.149-158, 1994.

[30] J.N.D. GUPTA, A.M.A. HARIRI, et C.N. POTTS. « Scheduling a two-stage hybrid flowshop With parallel machines at the first stage», Mathematics of Industrial Systems (Annals of operations research*)*, 1995.

[31] C.PROUST, E.GRUNENBERGER. « Planification de production dans un contexte de flow-shop hybride à deux étages: conception et interprogrammation d'ARIANE 2000 » ,*RAPA,* 8(5),pp 715-734, 1995.

[32] B. CHEN.« Analysis of classes of heuristics for scheduling a two-stage flowshop With parallel machines at one stage», Journal of Operational Research Society, 46(2):pp.234-244, 1995.

[33] S. LI. , « A hybrid two-stage flowshop With part family, batch production, major and minor setups», European Journal of Operations Research, 1996.

[34] A. VIGNIER, J-C. BILLAUT, et C. PROUST. « Les problèmes de flow-shop hybride: état de l'art», Rapport Interne 155, LI/E3i/Univ. de Tours, mai 1995. 100p.

[35] A. GUINET, V. BOTTA, et M. SOLOMON. « Minimisation du plus grand retard ou de la plus grande date de fin dans les problèmes d'ordonnancement de type ftowshop hybride», Journées d'études: « Affectation et Ordonnancement», Tours (France), CNRS .1 GdR Automatique / pôle SED / GT3. pp. 95-111. septembre 1995.

[36] Koulamas CP. « Scheduling two parallel semiautomatic machines to minimize machine interference»,Computers and Operations Research 1996;23(10):pp. 945-956.

[37] Houari M'HALLAH. « Heuristic algorithm for the two-satge hybrid problem»,Operations Research Letters*,* 1997, vol 21, pp. 43-53.

[38] F. RIANE. C. RAC and A. ARLIBA. « Hybrid Auto-adaptable Simulated Annealing based Heuristic», Computers & Industrial Engineering,(37) ,pp. 277-280. (1999)

[39] C.A. GLASS, Y.M. SHFRANSKY et V.A. STRUSEVICH. « Scheduling for Parallel Dedicated Machines with a Single Server», Naval Research Logistics,vol. 47, pp. 304-328.

[40] Nicholas G. Hall; Chris N. Pott et Chelliah Sriskandarajah. « Parallel machine scheduling with a common server» , Discrete Applied Mathematics *(*102) pp. 223- 243, 2000

[41] Brucker P, Dhaenens-Flipo C, Knust S, Kravchenko SA et F, Werner. « Complexity results for parallel machine problems with a single server», Journal of Scheduling 2002*(5)*, pp.429–57,2000.

[42] Ling-huey SU. « A hybrid two-stage flowshop with limited waiting time constraints », Computers & Industrial Engineering 44 (2003) 409-424.

[43] C . OGUZ, M. FIKRET ERCAN, T.C.EDWIN CHENG, Y,F,FUNG. « Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop », European Journal of Operational Research 149 (2003), pp 390-403.

[44] Wei WANG et John L.HUNSUCKER . « Makespan distributions in flow shops with multiple processors » , Journal of the Chinese Institute of Industrial Engineers,Vol.21,No.3, Pp 242-250 (2004).

[45] Jinxing X., Wenxun X, Zhixin L. AND Jiefang D. « Minimum Deviation Algorithm for Two-Stage No-Wait Flowshops with Parallel Machines», Computers and Mathematics with Applications,47 (2004) 1857-1863

[A] https://fr.m.wikipedia.org/wiki/Ordonnancement_d%27atelier.

[B] https://www.researchgate.net/figure/flow-shop-hybride-a-k-etages_fig5_48907984.

[C] https://fr.wikibooks.org/wiki/Programmation_Pascal/Introduction.

[D] https://www.techno-science.net/glossaire-definition/Pascal-langage-page-3.html.

تلخيص

في نظام الكمبيوتر , يسمى الخادم الذي يرسل البيانات إلى الأجهزة خادم الشبكة , أثناء نشاط التحميل,ليس من الضروري أن تكون آلة الإنتاج متاحة : يمكنها معالجة مهمة أخرى . في الواقع , تحتوي الآلات على معالج اتصال يسمح لها بتلقي المعلومات على الخادم في أي وقت . و بالتالي , في نظام الكمبيوتر , يمكن اعتبار نشاط التحميل بمثابة عمل يتطلب فقط تشغيل الخادم .

تتعلق المشكلة التي تم تداولها في هذه الرسالة بنظام كمبيوتر نعتبر انه يمكن اعتباره , متجر تدفق هجين من طابقين , مع آلة واحدة في الطابق الأول , و هو الخادم , و الآلات المتوازية في الطابق الثاني , مع قيد عدم الانتظار بين المرحلتين , و من المفترض أن تكون أوقات المعالجة على الخادم موحدة و الهدف هو تقليل إجمالي وقت الإكمال.

في هذه الرسالة يتم تقديم رسم تخطيطي سريع لتنظيم ورشة العمل و تشغيلها . مقدمة موجزة نذكر فيها المفاهيم الأساسية و عرض مشاكل الجدولة . نقدم خوارزمية Guirchoun و Martineau .

. أمثلة توضيحية لهذه الخوارزمية غير مقيدة . يتم تجميع التطبيقات التي نقوم بها في برنامج تعليمي بواجهة سهلة الاستخدام .

الكلمات المفتاحية : متجر التدفق الهجين , تقليل إجمالي وقت الإكمال , آلة متوازية مع الخادم .

ABSTRACT

In a computer system, the server that sends data to machines is called a network server. During the loading activity, the producing machine does not need to be available: it can process another job. Indeed, the machines have a communication coprocessor which allows them to receive information on the server at any time. Thus, in a computer system, the loading activity can be considered as work that only requires the server to be executed.

The problem addressed in this thesis concerns a computer system that we consider can be seen as a two-story hybrid flow shop, with a single machine on the first floor, which is the server, and the m parallel machines on the second floor, with a no-wait constraint between the two stages, and the processing times on the server are supposed to be unique and the goal is to reduce the total completion time.

In this dissertation, a quick diagram of workshop organization and operation is given. A brief introduction, in which we recall the notions
basic and presenting the scheduling problems is made. We present the algorithm of Guirchoun and Martineau. Illustrative examples of this algorithm are unrolled. The implementations carried out by us are grouped in an educational software with a user-friendly interface.

Keywords: hybrid flow shop, Makespan, parallel machine with server.

RÈSUMÈ

Dans un système informatique, le serveur qui envoie des données aux machines estappelé un serveur de réseau. Pendant l'activité de chargement, il n'est pas nécessaire que la machine réalisatrice soit disponible : elle peut traiter un autre travail. En effet,les machines disposent d'un coprocesseur de communication qui leur permet de recevoir informations sur le serveur à tout moment. Ainsi, dans un système informatique, l'activité de chargement peut être considérée comme un travail qui ne nécessite que le serveur pour être exécuté.

Le problème abordé dans ce mémoire concerne un système informatique que nous considérons peut être vu comme un atelier de flow shop hybride à deux étages, avec une seule machine au premier étage, qui est le serveur et le m machines parallèles audeuxième étage, avec une contrainte de no-wait entre les deux étages, et les temps detraitement sur le serveur sont censés être uniques et le but est Pour réduire le temps total de réalisation.

Dans ce mémoire, un schéma rapide d'organisation et de fonctionnement d'atelier est donné. Une brève introduction, dans laquelle nous rappelons les notions de base et en présentant les problèmes d'ordonnancement est faite. Nous présentons l'algorithme de Guirchoun et Martineau. Des exemples illustratifs de ce algorithme sont déroulés. Les implémentations réalisées par nos soins sont regroupées dans unlogiciel pédagogique à interface conviviale.

Mots-clés : flow shop hybride, Makespan, machine parallèle avec serveur.