ROYAUME DU MAROC
Université Mohammed V - RABAT -
جامعة محمد الخامس – الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre :3694

THESE

En vue de l'obtention du : ***DOCTORAT***

Centre de Recherche : *Centre de Recherches Mathématiques et Applications de Rabat.*
Structure de Recherche *: Laboratoire de Mathématiques, Informatique et Applications-Sécurité de l'Information (LABMIA-SI).*
Discipline : *Mathématiques Appliquées.*
Spécialité : *Optimisation et Recherche opérationnelle.*

Présentée et soutenue le 22/10/2022 par :

Hanna ZINI

Approches métaheuristiques pour la résolution des problèmes d'ordonnancement dans les systèmes de production de type flow shop hybride

JURY

Hafida BENAZZA	PES, Université Mohammed V, Faculté des sciences de Rabat	Présidente
Sidi Mohamed DOUIRI	PH, Université Mohammed V, Faculté des sciences de Rabat	Rapporteur/examinateur
Mustapha ESGHIR	PH, Université Mohammed V, Faculté des sciences de Rabat	Rapporteur/examinateur
Halima LAKHBAB	PH, Université HASSAN II, Faculté des sciences AIN CHOK	Rapportrice/examinatrice
Imad EL HARRAKI	PH, Ecole nationale supérieure des mines de Rabat	Examinateur
Souad EL BERNOUSSI	PES, Université Mohammed V, Faculté des sciences de Rabat	Directrice de Thèse

Année Universitaire : 2021-2022

Dédicace

À mes chers parents.
À ma bien aimée grand-mère.
À mes chers frères et sœurs.
À ma chère Alia.

Hanna

Remerciements

Les travaux de cette thèse sont effectués au sein du Laboratoire de Mathématiques, Informatique et Applications, Sécurité de l'Information (LABMIA-SI), à la facullté des sciences de Rabat, Université Mohammed V.

Je tiens à adresser mes remerciements d'abord à Madame **Souad EL BERNOUSSI**, Professeur à la faculté des Sciences de Rabat d'avoir accepté de diriger ma thèse, je la remercie profondément pour son suivi, sa disponibilité, ses conseils et ses idées précieuses. Je suis très reconnaissante pour son soutien permanent et sa patience et pour tout le temps qu'elle m'a accordé.

Je tiens à exprimer ma gratitude à Madame **Hafida BENAZZA**, Professeur à la faculté des Sciences de Rabat qui m'a fait l'honneur d'avoir accepté de présider le jury de ma soutenance.

Je remercie vivement Messieurs **Mustapha ESGHIR** et **Sidi Mohamed DOUIRI**, Professeurs à la faculté des Sciences de Rabat d'avoir accepté de rapporter mon travail et de participer à ce jury.

Je tiens à remerier sincèrement Madame **Halima LAKHBAB**, Professeur à la faculté des Sciences Ain Chock d'avoir accepté de rapporter et d'examiner cette thèse et de son intérêt de mon travail et de son aide.

Mes remerciements s'adressent aussi à Monsieur **Imad EL HARRAKI**, Professeur à l'école nationale supérieure des mines de Rabat pour avoir accepté d'évaluer mon travail et de faire

partie de ce jury.

Mes profonds remerciements à mes parents pour leur amour, leurs encouragements et leur soutien constant, et à mes frères et sœurs pour leur présence à mes côtés.

Je tiens à remercier en particulier ma grand-mère pour son amour infini.

Résumé

La théorie d'ordonnancement est un axe central de l'optimisation combinatoire. Elle concerne les problèmes d'allocation dans le temps d'un ensemble de ressources pour réaliser un ensemble de tâches, en optimisant un ou plusieurs critères. L'ordonnancement est lié à la plupart des secteurs économiques, car il est nécessaire pour assurer le bon déroulement de la production.

Cette thèse est consacrée à la résolution des problèmes d'ordonnancement dans les systèmes de production de type flow shop hybride et flow shop hybride avec tâches multiprocesseurs, avec l'objectif de minimiser le temps total d'ordonnancement (le makespan). Ces deux problèmes sont connus comme étant des problèmes NP-difficiles.

Devant les limites rencontrées par les méthodes exactes pour résoudre des problèmes complexes, les métaheuristiques constituent une alternative importante qui permet de fournir de bonnes solutions dans des temps de calcul raisonnables. Nous nous intéressons dans cette thèse à développer des approches de résolution hybrides, basées sur l'optimisation par essaims particulaires et la recherche d'harmonie pour résoudre les problèmes étudiés.

Des Différentes expérimentations ont été menées sur des problèmes benchmarks connus pour valider les approches proposées.

Mots-clefs : Flow Shop Hybride, Ordonnancement, Tâches multiprocesseurs, Optimisation par essaims particulaires, Recherche d'harmonie, Makespan.

Abstract

Scheduling theory is a central axis of combinatorial optimization. It concerns the problems of allocating over time of a set of resources for performing a set of tasks with the aim of optimizing one or more criteria. Scheduling is linked to the most economic sectors because it is necessary to ensure the smooth running of production.

This thesis is devoted to solving the hybrid flow shop and the hybrid flow shop with multiprocessor tasks scheduling problems in production systems, with the objective of minimizing the maximum completion time (the makespan). These two problems are known to be NP-hard.

Faced with the limits encountered by exact methods for solving complex problems, metaheuristics constitute an important alternative that allows to provide good solutions in reasonable computation times.
In this thesis, we are interested in developing hybrid resolution approaches, based on particle swarm optimization and harmony search algorithms for the studied problems.

Various experiments were carried out on known benchmark problems to validate the proposed approaches.

Key Words : Hybrid Flow Shop, Scheduling, Multiprocessor tasks, Particle Swarm Optimization, Harmony Search, Makespan.

Liste des Symboles

C_{max}	Makespan
HFS	Hybrid Flow Shop
$HFSMT$	Hybrid Flow Shop with Multiprocessor Tasks
$FIFO$	First In First Out
SPT	Smallest Processing Time
LPT	Longest processing time
PSO	Particle Swarm Optimization
$DPSO$	Discret Particle Swarm Optimization
LB	Lower Bound
RCC	Right Corner Crossover
PD	Percentage Deviation
APD	Average Percentage Deviation
$IAIS$	Immunoglobulin-based Artificial Immune System Algorithm
AIS	Artificial Immune System Algorithm
QIA	Quantum-inspired Immune Algorithm
ACO	Ant colony optimization
HS	Harmony Search
$HMCR$	Harmony Memory Consideration Rate
PAR	Pitch Adjustment Rate
HMS	Harmony Memory Size
OBL	Opposition Based Learning
ARI	Relative Rate Improvment
GA	Genetic Algorithm
EGA	Effecient Genetic Algorithm
PGA	Parallel Greedy Search
$MGLS$	Memetic Global and Local Search
$ACSNDP$	Ant Colony System with a Novel Non-DaemonActions Procedure

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

La gestion des systèmes de production constitue un outil indispensable qui conditionne le bon déroulement des projets dans les entreprises. Dans ce processus, l'ordonnancement est une classe importante qui s'occupe de la réalisation des décisions stratégiques de l'entreprise. L'ordonnancement vise à programmer dans le temps l'exécution d'un ensemble de tâches, en respectant les contraintes de production et de ressources, et en optimisant certains critères. Il est devenu un champ d'investigation important que toute entreprise cherche à maximiser son profit.

Depuis plusieurs années, les recherches dans la théorie de l'ordonnancement ont suscité un intérêt intensif aux problèmes rencontrés dans les environnements réels de production. Ces recherches ont présenté de nombreux modèles qui visent à optimiser les problèmes d'ordonnancement pour répondre aux exigences de plus en plus fortes du marché en termes de qualité, du coût et du temps.
Les problèmes d'ordonnancement sont considérés comme étant des problèmes complexes de l'optimisation combinatoire. Généralement, la résolution d'un problème d'optimisation nécessite d'abord la définition des variables, des paramètres, des contraintes et de la fonction objectif à optimiser. Ensuite, le choix d'une méthode de résolution doit prendre en compte la qualité des solutions fournies et la complexité temporelle.
Devant les limites rencontrées par les méthodes exactes pour résoudre des problèmes d'ordonnancement, l'enjeu actuel est de trouver des techniques d'optimisation qui visent à résoudre efficacement ces problèmes. Les métaheuristiques constituent une alternative appropriée, qui permet d'obtenir de bonnes solutions dans des temps de calcul raisonnables. Contrairement aux méthodes heuristiques qui sont particulièrement conçues pour résoudre des problèmes spécifiques, les métaheuristiques sont des méthodes approximatives générales, elles possèdent une conception fondamentale permettant de les adapter à différentes classes de problèmes. Les métaheuristiques se décomposent généralement en deux grandes classes, celles qui mani-

pulent une seule solution, et celles qui font évoluer un ensemble de solutions en parallèle dans l'espace de recherche. Elles ont prouvé leur succès pour résoudre de nombreux problèmes d'ordonnancement. Il est intéressant de mentionner qu'aucune métaheuristique n'est meilleure qu'une autre, et le choix de la métaheuristique la plus adaptée à un problème donné dépend fortement des caractéristiques du problème tels que ces paramètres et sa fonction objectif.

Les problèmes d'ordonnancement d'atelier cherchent à ordonnancer un ensemble de jobs (travaux) sur différentes machines, où chaque machine ne peut réaliser qu'une seule tâche (entité d'un job) à la fois, dans le but de satisfaire un ou plusieurs objectifs. Ces problèmes se décomposent en plusieurs catégories selon l'ordre d'enchaînement des opérations et le nombre de machines et leur disposition.
Le problème d'ordonnancement d'ateliers de type flow shop hybride (HFS) est l'un des problèmes fréquemment rencontrés dans les systèmes de production. Ce problème consiste à ordonnancer un ensemble de jobs à travers un ensemble d'étages en séries, où chaque étage contient un ensemble de machines parallèles. Les jobs peuvent représenter des concepts variés tels que des produits, des programmes informatiques, et des commandes. De ce fait, le problème trouve son application dans de nombreux domaines industriels notamment le textile, l'industrie pharmaceutique, alimentaire et automobile, la fabrication des fils électriques...etc. Dû à ses diverses applications et à sa complexité, le problème du flow shop hybride a fait l'objet de plusieurs études.

Dans le cadre de cette thèse, nous contribuons à la résolution des problèmes d'ordonnancement d'atelier de type flow shop hybride et flow shop hybride avec tâches multiprocesseurs, en considérant le temps d'ordonnancement total (le makespan) comme critère d'optimisation. L'objectif est de proposer des approches de résolution efficaces et relativement simples à implémenter qui sont capables de supporter la nature NP-difficiles des problèmes traités. Ces approches permettent d'hybrider des métaheuristiques avec des stratégies de recherche locale dans le but d'ajuster l'équilibre entre les principes de la diversification (l'exploration de l'espace de recherche) et l'intensification (l'exploitation des informations accumulée pendant la recherche), et de cette façon trouver rapidement des régions prometteuses de l'espace de recherche contenant de bonnes solutions.

Cette thèse est organisée en quatre chapitres. Les deux premiers chapitres décrivent un état de l'art, les concepts et les techniques que nous abordons dans ce travail. Les deux derniers chapitres présentent nos contributions pour la résolution des problèmes traités.

- Le chapitre 1 donne une présentation générale aux problèmes d'ordonnancement d'atelier dans les systèmes de production. Il décrit les différentes notions essentielles liées aux

problèmes d'ordonnancement et ceux nécessaires pour traiter notre problème. Dans ce chapitre, nous abordons aussi l'optimisation combinatoire, la théorie de la complexité et un état de l'art sur le problème traité.

- Le chapitre 2 introduit les techniques d'optimisation : exactes, heuristiques et métaheuristiques. Nous décrivons plus spécifiquement les méthodes développées dans cette thèse. Aussi, nous rappelons des revus de littérature sur leur utilisation surtout pour résoudre des problèmes d'ordonnancement. Nous définissons aussi la notion de l'hybridation des méthodes et ses principales formes.
- Le chapitre 3 est consacré à une contribution à la résolution du problème de flow shop hybride avec l'objectif de minimiser le makespan. Nous donnons dans ce chapitre une description détaillée du problème et sa modélisation mathématique. Nous développons un algorithme à base de l'optimisation par essaims particulaires. Nous adaptons cette dernière qui est conçue essentiellement pour résoudre des problèmes de l'optimisation continue, dans le but de surpasser la nature discrète du problème traité. L'algorithme proposé utilise une nouvelle technique pour mettre à jour les particules pendant l'évolution de l'essaim. À ce niveau, un opérateur de recherche locale est incorporée pour constituer la quatrième composante du processus de la mise à jour de particules, qui contient principalement trois composantes : la composante d'inertie, la composante cognitive et la composante sociale. Nous proposons aussi une méthode de calcul du makespan C_{max} basée sur la combinaison de plusieurs règles de priorité. Nous présentons ensuite, les détails de l'application de l'algorithme proposé et ses étapes ainsi que le choix de ces paramètres et les résultats expérimentaux obtenus.
- Le chapitre 4 est dédié à l'étude du problème du flow shop hybride avec tâches multiprocesseurs. Ce dernier est une extension du problème du flow shop hybride qui permet aux jobs de demander un ou plusieurs processeurs simultanément à chaque étage. Pour résoudre ce problème, nous proposons un algorithme de recherche d'harmonie qui est inspiré de la performance musicale. Nous proposons de nouvelles règles d'improvisation pour générer de nouvelles solutions à chaque itération. L'objectif de ce chapitre est de valider la performance de l'algorithme de la recherche d'harmonie pour résoudre le problème du flow shop hybride avec tâches multiprocesseurs. Au début, nous définissons le problème traité. Nous présentons ensuite les différentes étapes de l'algorithme proposé. Finalement, nous donnons les résultats numériques trouvés et leurs interprétations.

Nous finissons cette thèse par une conclusion générale et quelques perspectives de recherche.

Chapitre 1

Ordonnancement d'ateliers : Contexte et état de l'art

1.1 Optimisation combinatoire

L'optimisation combinatoire est une discipline fondamentale de la recherche opérationnelle en mathématiques appliquées et en informatique, qui regroupe une vaste classe de problèmes ayant des applications dans l'industrie, la planification, la gestion, le transport, ...etc.

Généralement, un problème d'optimisation se définit comme la recherche d'un minimum (minimisation) ou d'un maximum (maximisation) d'une fonction donnée dans un espace de recherche S (un ensemble de solutions possibles). La résolution d'un problème d'optimisation dans le cas de minimisation par exemple consiste à trouver la meilleure solution s^* qui vérifie :

$$f(s^*) \leq f(s), \forall s \in S \tag{1.1}$$

$f : S \to R$ est la fonction à minimiser appelée aussi fonction objectif, la solution s^* est l'optimum global, mais il existe aussi la notion de l'optimum local qui est également un optimum mais seulement sur un sous ensemble de l'espace de recherche.

Dans ce contexte, Si l'espace de recherche S est muni d'une relation de voisinage V. Un optimum local est défini comme une solution $s^* \in S$ telle que :

$$\forall s \in V(s^*), f(s^*) \leq f(s) \tag{1.2}$$

La figure 1.1 illustre l'optimum global et local d'une fonction.
L'optimisation se décompose essentiellement en deux sous disciplines : l'optimisation continue

et l'optimisation combinatoire. Si l'espace de recherche S est continu, on parle d'optimisation continue. Si S est discret (i.e. énumérable), on parle donc d'optimisation combinatoire.
À signaler que l'espace de recherche S est déterminé souvent par un système d'égalités et d'inégalités qui expriment les caractéristiques des solutions du problème.

La plupart des problèmes d'optimisation combinatoire peuvent être formulés comme des programmes mathématiques en nombres entiers ou mixtes sous la forme suivante :

$$\min / \max \quad f(s) \tag{1.3}$$
$$s/c \quad s \in Z^p \times R^{n-p} \tag{1.4}$$

s/c signifie "sous contraintes". Notons que le problème d'optimisation fait partie de la programmation en nombres entiers si $p = n$, sinon il est un problème d'optimisation mixte en nombres entiers.

En raison de la nature discrète des problèmes combinatoires, le nombre de solutions possibles s'accroît exponentiellement avec la taille du problème, ce qui rend le temps de résolution excessivement long. De ce fait, les problèmes de l'optimisation combinatoire sont généralement difficiles à résoudre. Dans la suite, on introduit la notion de complexité qui permet d'identifier une classification des problèmes selon la complexité de leur résolution.

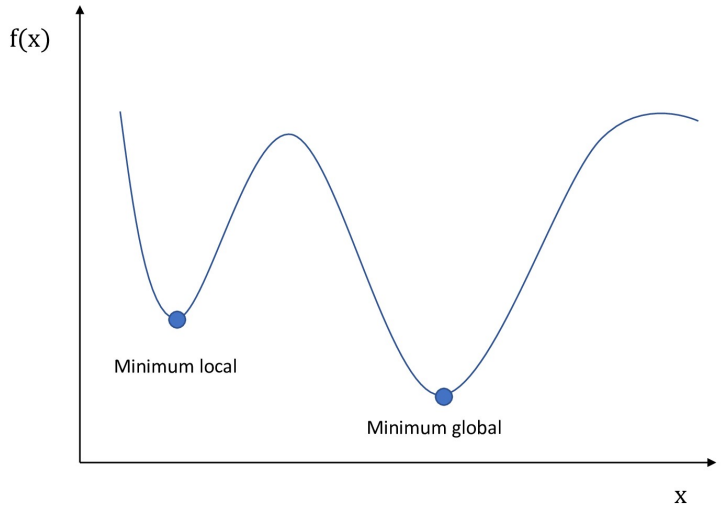


figure. 1.1 – Minimum local et global d'une fonction

1.2 Notions sur la théorie de complexité

La complexité d'un problème est une estimation du nombre d'opérations élémentaires à exécuter pour résoudre une instance du problème, en fait cette estimation est déterminée dans le cas pire, ce qui revient à dire que la complexité du problème est décidée en considérant son instance la plus difficile. En tenant compte de la complexité de résolution, on peut distinguer différentes classes de problèmes. Tout d'abord, on donne quelques définitions utiles :

Définition 1.2.1. *Soit $f : \mathbb{N} \to \mathbb{R}_+$ une fonction donnée, $O(f(n))$ est l'ensemble des fonctions $g(n)$ telles qu'il existe un entier positif n_0 et une constante $c > 0$, tel que pour tout $n \geq n_0$ on a $\quad g(n) \leq c.f(n)$*

Définition 1.2.2. *La complexité d'un algorithme est dite polynomiale si elle est en $O(n^k)$ pour un entier k.*

Définition 1.2.3. *Un problème de décision est un problème pour lequel une solution est soit "oui" soit "non".*

Définition 1.2.4. *On dit qu'un problème de décision P_1 se réduit polynomialement à un autre problème de décision P_2, s'il existe un algorithme polynomial construisant les données d'une instance de P_2 à partir des données d'une instance quelconque de P_1, de telle sorte que la réponse à P_1 est "oui" si et seulement si la réponse à P_2 est "oui".*

Les classes de complexité sont nombreuses, mais on introduit seulement ceux qui permettent de caractériser les problèmes de l'optimisation combinatoire.

— La classe P (Polynomial) contient les problèmes qui peuvent être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes efficacement résolubles.

— la classe NP (Non deterministic Polynomial) contient les problèmes qui peuvent être résolus sur une machine non déterministe (qui peut être vue comme une machine capable d'effectuer plusieurs exécutions en parallèle) par un algorithme de complexité polynomiale, cela signifie que la résolution des problèmes de la classe NP peut nécessiter l'exécution d'un grand nombre de possibilités (possiblement exponentiel), mais chaque possibilité peut être testée à l'aide d'un algorithme polynomial. La classe P est inclus dans la classe NP, mais l'inclusion inverse n'a jamais été prouvée, et la question de déterminer si $P \neq NP$ est une célèbre conjecture émise en 1971 par Stephen Cook [1], il s'agit donc d'un problème fondamental de l'informatique théorique. La classe NP contient des problèmes plus difficiles que la classe P. De la même façon, on peut trouver dans la classe NP des problèmes encore plus difficiles.

— la classe NP-complet (NP-complete) contient les problèmes les plus difficile de la classe NP, i.e., un problème est NP complet s'il est au moins aussi difficile que tout autre problème de la classe NP.

— la classe NP-difficile (NP-Hard) : un problème NP-difficile n'appartient pas nécessairement à la classe NP, et il est au moins aussi dur que tous les problèmes de la classe NP.

La figure 1.2 montre la relation entre les classes P, NP, NP-complet et NP-difficile.

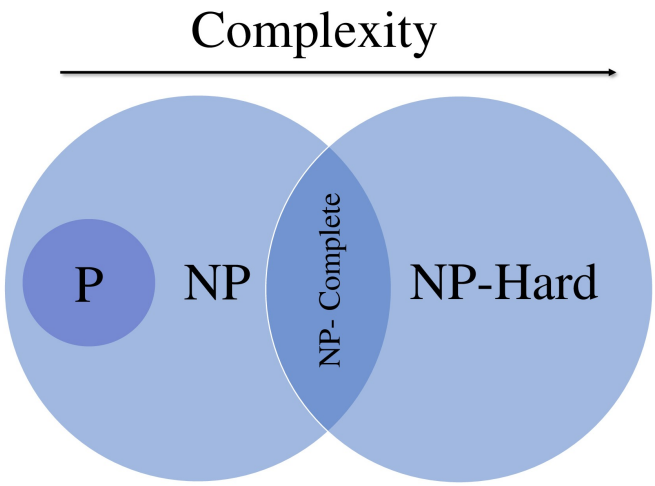


figure. 1.2 – Représentation des classes P, NP, NP-Complete et NP-Hard

1.3 Problèmes d'ordonnancement d'ateliers

1.3.1 La gestion de production

La production est un processus de transformation d'un ensemble de ressources (machines et matières) pour créer de biens ou de services. Elle s'effectue dans une chaîne d'opérations (appelée souvent chaîne de production) en utilisant les composants du système de production. Ce dernier représente l'ensemble de ressources (équipements, machines, moyens humains...) qui permettent le bon déroulement du processus de fabrication, dans le but d'obtenir des produits finis destinés à la consommation.

La concurrence industrielle a conduit les entreprises à améliorer leurs performances et leurs techniques en vue de répondre aux besoins (exigences) croissants du marché en terme de productivité et de réduction du coût et du temps. La gestion de production est un outil indispensable qui permet de mieux exploiter les moyens disponibles, et d'organiser les différents processus de fabrication pour réaliser cet objectif. Elle occupe un vaste champ d'application dans les environnements économiques allant de l'approvisionnement des matières jusqu'à la distribution des produits. La gestion de production s'effectue à l'aide d'un ensemble de dé-

cisions qui sont habituellement classées en trois catégories [106] : les décisions stratégiques, tactiques et opérationnelles.

- **Les décisions stratégiques :**
  Les décisions stratégiques représentent la politique à long terme de l'entreprise. Elles traduisent la stratégie générale de l'entreprise, et portent principalement sur la gestion des ressources durables pour qu'elles soient toujours suffisantes en vue d'assurer la pérennité de l'entreprise [2]. On note que ces ressources peuvent être des machines, des hommes, des moyens informatiques ou des données techniques.
- **Les décisions tactiques**
  Les décisions tactiques correspondent à la politique à moyen terme de l'entreprise, et elles garantissent la liaison entre le niveau stratégique et le niveau opérationnel. Ces décisions sont destinées à obtenir la bonne exploitation des ressources disponibles. On peut citer la planification de la production comme exemple de décisions tactiques.
- **Les décisions opérationnelle**
  Les décisions opérationnelles déterminent la politique à court terme de l'entreprise. Il s'agit à ce niveau d'assurer la flexibilité nécessaire au bon déroulement de la production. On trouve comme exemple de décisions opérationnelles : la gestion des stocks et l'ordonnancement de la production. La gestion des stocks garantit la mise à disposition des produits et des composants nécessaires aux ordres de fabrication, alors que l'ordonnancement consiste à organiser les activités et à programmer une exécution optimale des différentes tâches. L'ordonnancement représente une classe importante de ces décisions. Il s'occupe de la réalisation des décisions prise aux niveaux supérieurs en mettant en œuvre un ensemble d'actions qui traduisent les décisions prédéterminées en instructions d'exécution détaillées destinées à piloter et contrôler à court terme l'activité des postes de travail dans l'atelier.

Les trois niveaux de décisions sont illustrés dans la figure 1.3.
Dans ce travail, on se place au niveau opérationnel, et on s'intéresse uniquement à l'ordonnancement. Nous abordons dans la partie suivante une présentation des problèmes d'ordonnancement d'ateliers.

1.3.2 Présentation des problèmes d'ordonnancement

L'ordonnancement consiste à organiser dans le temps l'exécution d'un ensemble de tâches (ou opérations), en respectant les contraintes de production et de ressources et en cherchant à optimiser certains critères [3,4]. Il est lié à différents domaines de recherche. Par exemple, en gestion de projet, l'ordonnancement permet de définir le plan de projet et de programmer les dates de mise en œuvre des activités qui composent le projet. En informatique, l'ordonnancement sert à choisir l'ordre d'exécution des processus sur les processeurs, et ainsi utiliser

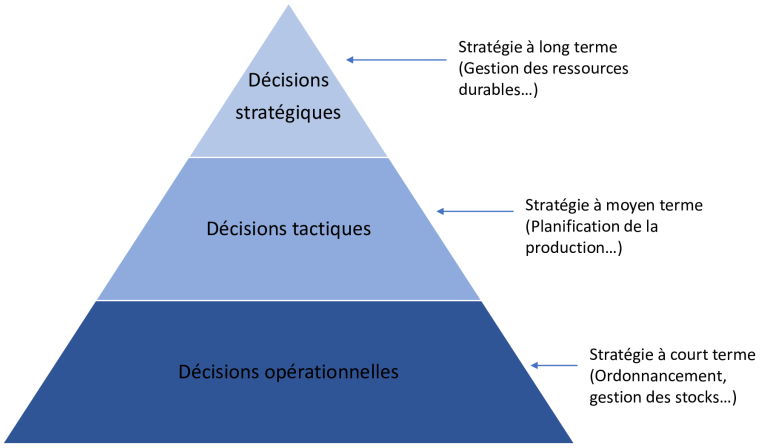


figure. 1.3 – Les niveaux de décisions de la gestion de production

au mieux les processeurs par l'utilisateur. En production, l'ordonnancement détermine les séquences d'opérations à réaliser sur les différentes machines de l'atelier. En administration, la gestion des ressources humaines et les emplois du temps se modélisent comme des problèmes d'ordonnancement.

L'ordonnancement occupe une place particulière dans la plupart des domaines et devient de plus en plus un champ important d'investigation et d'exploration tant par les différents problèmes traités, que par les méthodes de résolution utilisées.
Comme mentionné précédemment, l'ordonnancement a en général une fonction à court terme, même s'il est parfois utilisé à moyen terme [4]. Sa durée est donc relativement courte.
Pour décrire explicitement un ordonnancement, on définit les quatre éléments qui le constituent : les tâches, les contraintes, les ressources et les objectifs.

1.3.2.1 Tâches

Une tâche ou opération est une entité élémentaire de travail localisée dans le temps par une date de début (starting time) et/ou de fin (ending time), dont la réalisation nécessite une durée d'exécution et utilise certains moyens ou ressources avec une certaine quantité.

Une tâche i peut être caractérisée par différentes informations selon le problème étudié et son objectif. Parmi ces informations, on mentionne la durée d'exécution de la tâche (processing time) notée p_i, la date de début t_i , la date de fin c_i, la date de disponibilité r_i (ou la date de début au plus tôt) qui désigne que l'exécution de la tâche ne peut pas commencer avant cette date, et la date d'échéance d_i (ou la date de fin au plus tard) qui signifie que l'exécution de la tâche doit être achevée avant cette date. La figure 1.1 montre quelques caractéristiques temporelles d'une tâche.

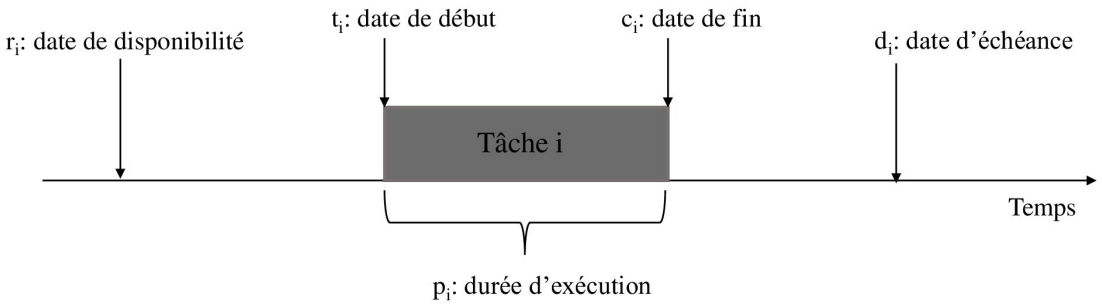


figure. 1.4 – Caractéristiques d'une tâche

Les tâches peuvent généralement être préemptives ou non préemptives. Quant à une tâche préemptive, elle peut être interrompue puis reprise à tout moment sans aucun coût, tandis qu'une tâche non préemptive, une fois commencée, elle ne peut jamais s'arrêter avant son achèvement. Le terme travail ou "job" est utilisé pour désigner un ensemble de tâches.

1.3.2.2 Ressources

Les ressources sont les moyens matériels, techniques ou humains désignés pour réaliser les tâches. Généralement, on peut distinguer deux principaux types de ressources selon leurs disponibilités :

— Les ressources renouvelables qui peuvent être réutilisées en même quantité après avoir été allouées à une tâche (les machines, les outils de production, les hommes...)
— Les ressources non renouvelables ou consommables dont leur disponibilité décroît après leur utilisation, c'est le cas par exemple de matière première et de budget.

Parmi les ressources renouvelables, il existe deux catégories principales de ressources : disjonctives et cumulatives. Les ressources disjonctives ne peuvent exécuter qu'une tâche à la fois, tandis que les ressources cumulatives peuvent exécuter plusieurs tâches simultanément.

1.3.2.3 Contraintes

Les contraintes expriment les restrictions sur les valeurs que peuvent prendre conjointement les variables de décision [4]. Elles indiquent les limites et les conditions à respecter afin d'assurer la réalisation de l'ordonnancement. On peut distinguer trois types fondamentales de contraintes :

— Les contraintes temporelles : les contraintes temporelles correspondent aux impératifs temporels relatifs aux dates limites des tâches ou de l'ensemble du projet. On peut citer essentiellement la date de disponibilité et la date d'échéance.
— Les contraintes de précédence : Ce sont les contraintes d'enchaînement ou de succession imposées par la gamme de production, elles représentent les positionnements relatifs que doivent respecter certaines tâches par rapport à d'autres.
— Les contraintes de ressources, ce type de contraintes décrit la disponibilité et la quantité de ressources. Elles distinguent entre les différents types de ressources, ainsi on peut définir deux types de contraintes de ressources : les contraintes cumulatives (réaliser une seule tâche sur la même ressource) et les contraintes disjonctives (réaliser plusieurs tâches sur la même ressource).

On note que ces contraintes peuvent être strictes ou non, selon la nature du problème. Les contraintes strictes caractérisent l'ensemble de contraintes qu'on doit respecter, cependant les contraintes non strictes ou de préférence peuvent n'être pas respectées.

1.3.2.4 Objectif

En vue de répondre au mieux aux attentes de production, les problèmes d'ordonnancement cherchent à satisfaire un ou plusieurs critères. Les objectifs ou les critères désignent alors les mesures utilisées pour évaluer la performance d'un ordonnancement. Généralement, les objectifs peuvent être divisés en plusieurs catégories [2], nous citons ici les plus connues :

- Les objectifs liés au temps : on trouve notamment le temps total d'exécution (ou la durée totale de l'ordonnancement), les dates d'achèvement des jobs, les différents types de retards par rapport aux livraison,...etc.
- Les objectifs liés aux ressources : pour ce type, on peut trouver le nombre de ressources nécessaires pour réaliser un ensemble de tâches, la charge de chaque ressource,...etc.
- Les objectifs liés aux coûts : par exemple, les coûts de production, les coûts de transport, et les coûts de stockage.

Pour estimer l'efficacité d'une stratégie à atteindre les objectifs déterminés, chaque entreprise adopte des critères appropriés qui reflètent sa politique et son environnement manufacturier. Généralement en ordonnancement, on parle d'une optimisation mono-objectif, si la fonction

objectif renvoie un seul critère, si non, c'est à dire si la fonction objectif renvoie plusieurs critères simultanément, il s'agit dans ce cas d'une optimisation multi-objectif.

1.3.2.5 Classification de l'ordonnancement

Les problèmes d'ordonnancement sont très variés, ce qui justifie l'introduction d'une classification qui permet de les distinguer. En 1979, Graham et al.[5] ont introduit une classification des problèmes d'ordonnancement définie par un triplet $\alpha|\beta|\gamma$, en se basant sur la spécification de trois éléments : tâches, ressources et critères. Cette notation à trois champs est reprise par Lageweg et al. [6], et Blazewicz et al [7], et ultérieurement par toute la communauté scientifique du domaine. Le champ α désigne l'environnement machines, il contient deux sous champs α_1 et α_2 où α_1 indique le type d'ateliers de production et α_2 détermine le nombre de machines considérées. Le tableau 1.1 montre quelques configurations possibles du champ α.

Le champ β décrit les caractéristiques des tâches et les contraintes imposées par l'environnement de fabrication et par les ressources, il se compose de plusieurs sous champs, le tableau 1.2 indique certaines inscriptions de ce champ.

Table. 1.1 – Configurations possibles du champ α

Champ α		
Sous Champs	Notation	Description
α_1	1	Problème à une seule machine
	P	Problème à machines parallèles identiques
	F	Flow shop
	J	Job shop
	FH	Flow shop hybride
	JF	Job shop flexible
	OG	Open shop généralisé
α_2	m	Nombre de machines

Le champ γ précise les critères d'optimisation (les objectifs) du problème d'ordonnancement, il contient souvent une seule entrée. On trouve dans la littérature de nombreux critères, nous présentons ici les plus utilisés :

- C_{max} : la durée totale d'ordonnancement (makespan) ($C_{max}=\max_{j\in\{1,\ldots,n\}} C_j$) où C_j est la date de fin d'exécution du job j), le makespan est égal à la date de fin de la tâche la plus tardive.

Table. 1.2 – Configurations possibles du champ β

Champ β		
Sous Champs	Notation	Description
β_1	$pmtn$	Préemption autorisée
β_2	res	Ressources suplémentaires
β_3	$prec$	Contraintes de précédence
β_4	r_j	Les dates de disponibilité (Dates au plus tôt) des tâches
β_5	p_j	Durées d'exécution des tâches
β_6	d_j	Dates d'échéance (Date au plus tard) des tâches
β_7	nwt	Contraintes de non attente
β_8	$size_{ij}$	tâches multiprocesseurs

- $\mathrm{T_{max}} = \max_{j\in\{1,\ldots,n\}} T_j$: le plus grand retard absolu, avec $T_j = \max(C_j - d_j, 0)$ et d_j est la date au plus tard de la tâche j.
- $\sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \mathrm{C_j}$: la somme des dates de fin d'exécution des tâches.
- $\sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \omega_\mathrm{j}\mathrm{C_j}$: la somme pondérée des dates de fin d'exécution des tâches.
- $\overline{\mathrm{T}} = \sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \mathrm{T_j}$: la somme des retards (le retard total).
- $\sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \omega_\mathrm{j}\mathrm{T_j}$: la somme pondérée des retards.
- $\mathrm{L_{max}} = \max_{j\in\{1,\ldots,n\}}\{C_j - d_j\}$ Le plus grand retard algébrique.
- $\sum \mathrm{F_j}$: le temps de séjour total (total flow time) où F_j est le temps de séjour du travail j (le temps que passe le travail j dans le système).
- $\sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \mathrm{U_j} = \sum_{j\in\{1,\ldots,n\}} |\{J_j/C_j > d_j\}|$ Le nombre de jobs en retard.
- $\sum_{\mathrm{j}\in\{1,\ldots,\mathrm{n}\}} \omega_\mathrm{j}\mathrm{U_j} = \sum_{j\in\{1,\ldots,n\}} \omega_j|\{J_j/C_j > d_j\}|$ Le nombre pondéré de jobs en retard.

Ces critères sont appelés critères réguliers, car la valeur à minimiser est une fonction croissante des dates de fin des opérations.

1.3.2.6 Représentation graphique

La représentation graphique d'un problème d'ordonnancement est un outil performant, qui permet de mettre en évidence le problème étudié et de présenter une aide visuelle facilitant sa manipulation. Cet outil a connu une grande évolution avec le développement de la théorie des graphes et son rôle fondamental pour les mathématiques appliquées, et surtout avec l'apparition des réseaux de Petri. Nous abordons ici deux graphes très utilisés pour représenter des problèmes d'ordonnancement et aider à leurs résolution : le graphe PERT et le graphe des potentiels.

le graphe PERT : La méthode PERT (Program Evolution and Review Technology) a été mise au point en 1958 à la demande de la marine américaine pour contrôler le développement d'un projet de construction des sous-marins Polaris. La méthode s'appuie sur la méthode du chemin critique CPM (Critical Path method), qui vise à identifier le chemin critique, c'est à dire le chemin qui correspond à l'ensemble de tâches pour lequel tout retard en cours de traitement aura un impact sur le délai de finalisation d'ordonnancement.
Le graphe PERT permet d'identifier les connexions entre les différentes tâches de l'ordonnancement compte tenu des contraintes de précédence. Il est composé d'un ensemble de nœuds reliés entre eux par des arcs, où chaque nœud correspond à l'instant de début ou de fin d'une ou de plusieurs tâches. Une tâche est représentée par un arc, sur lequel on indique le temps d'exécution de la tâche. Si la valeur de l'arc est égale à 0, il s'agit dans ce cas d'une tâche fictive qui sert à représenter une contrainte de précédence. Notons que la longueur d'un arc n'est pas proportionnelle au temps d'exécution de la tâche représentée.

Le graphe des potentiels : La méthode des potentiels appelée aussi méthode des potentiels métra (MPM) a été développée par Bernard Roy vers la fin des années 50. La méthode est semblable à la méthode de PERT, mais elle offre plus de simplicité. Le graphe des potentiels comporte un ensemble de sommets et d'arcs. Les sommets représentent les tâches à réaliser, auxquels deux sommets fictifs appelés "source" et "puits" sont ajoutés pour indiquer le début et la fin de l'ordonnancement. Les arcs entre les sommets peuvent être conjonctifs ou disjonctifs. Les arcs conjonctifs connectent deux tâches consécutives d'un même travail (contraintes de précédence), et dans ce cas chaque arc porte une valeur numérique indiquant la durée de la tâche située à son extrémité initiale. Tandis que les arcs disjonctifs connectent deux tâches appartenant à des travaux différents qui utilisent la même machine (contraintes de ressources) [3].

1.3.3 Les types d'ateliers

Il s'agit généralement dans un problème d'ordonnancement d'ateliers d'usiner ou d'assembler un job (un travail) sur différentes machines, où chaque machine ne peut réaliser qu'une seule tâche à la fois (ressource disjonctive). La réalisation d'un produit doit respecter une séquence d'opérations déterminée par la gamme de fabrication. Les problèmes d'ordonnancement d'ateliers se décomposent en deux principaux modèles selon le nombre de machines et l'ordre d'enchaînement des opérations : les modèles à une opération (machine unique et machines parallèles), et les modèles à plusieurs opérations (flow shop, job shop, et open shop)

Machine unique C'est un modèle simple où l'ensemble des tâches à réaliser s'exécute sur une seule machine.

Machines parallèles Cet atelier est constitué de plusieurs machines organisées en parallèle qui effectuent la même tâche. De ce fait, chaque job peut être réalisé par n'importe quelle machine. Alors une fois une machine i devient disponible, on lui affecte un job j. Les machines sont classées suivant leurs vitesses en trois catégories :

- Identiques : Où toutes les machines ont la même vitesse de traitement.
- Uniformes : Si les machines ont des vitesses de traitement différentes mais linéaires.
- Non reliées ou indépendantes (Non identiques) : Où les vitesses des machines sont complètement indépendantes les unes des autres.

Flow-shop Dans ce type d'ateliers, la gamme de fabrication est constituée de plusieurs machines en série, l'ordre de passage de tous les travaux sur les différentes machines est le même. Ce type d'ateliers est dit atelier à cheminement unique. Plus précisément, il s'agit de n jobs qui doivent s'exécuter suivant le même ordre sur les m machines de l'atelier.

Job-shop Les opérations dans un atelier de type job-shop sont exécutées dans un ordre déterminé selon le job à réaliser, en d'autres termes chaque job possède son propre ordre de passage sur les machines. Ainsi, ce type d'ateliers est nommé atelier à cheminements multiples. Il est à noter que les opérations peuvent passer plusieurs fois sur la même machine pour effectuer les tâches requises.

Open-shop Dans un atelier open shop, le séquencement des opérations est libre, c'est à dire qu'il n'existe aucune restriction sur l'ordre de passage des opérations sur les différentes machines. Alors les opérations peuvent être exécutées dans n'importe quel ordre, c'est pourquoi cet atelier est dit atelier à cheminement libre.

Afin d'aborder des problèmes qui se rencontrent souvent dans les systèmes de production réels, la littérature a proposé plusieurs extensions basées sur les modèles principaux, comme le problème du flow-shop hybride (hybrid flow shop) qui combine les propriétés du problème flow shop et des machines parallèles. Cet atelier est constitué de plusieurs étages ou stations en

séries, et chaque étage contient un ensemble de machines parallèles. Tous les jobs traversent les étages dans le même ordre, de façon que chaque job peut être exécuté par une seule machine à chaque étage.

Nous pouvons citer également le job shop flexible (flexible job shop) qui généralise le problème job shop, il se caractérise par le fait qu'une opération peut être exécutée par n'importe quelle machine d'un ensemble donné de machines. Aussi le problème open shop généralisé étend le problème open shop, de façon que plusieurs machines peuvent réaliser une même opération.

Ces ateliers offrent une grande flexibilité par rapport aux modèles classiques, et nécessitent la résolution d'un problème d'affectation des opérations aux machines, en plus d'un problème de séquencement des différentes opérations sur les machines. Dans cette thèse, nous nous intéressons aux problèmes d'ordonnancement de type flow shop hybride.

1.4 Flow shop hybride

1.4.1 Présentation du problème

Dans un environnement de fabrication, les articles produits peuvent être décomposés en un certain nombre de tâches ou opérations. Chaque tâche nécessite un temps de traitement et une variété de matériaux, machines et équipements. Ainsi, un travail passe généralement par plusieurs étapes dans une série de postes de travail ou de groupes de machines. Il existe de nombreuses contraintes de temps, de précédence et de ressources ce qui augmente les restrictions sur le traitement des tâches qui doivent être gérées avec toutes les limitations respectées. Le flow shop hybride se rencontre dès qu'un atelier se compose de plusieurs étages, contenant chacun un ensemble de machines et où tous les jobs à ordonnancer ont la même gamme de fabrication.

Théoriquement, le flow shop hybride (HFS) ou encore le flow shop flexible est une extension basée sur les deux modèles flow shop et machines parallèles. En fait, si le nombre d'étages est égal à 1, alors on est dans le cas d'un problème à machines parallèles, en revanche, s'il n'y a qu'une seule machine à chaque étage, on se ramène au problème de flow shop classique. Il s'agit dans un environnement flow shop hybride d'ordonnancer un ensemble de n jobs dans un atelier composé de plusieurs étages, où chaque étage k comporte un ensemble de m_k machines parallèles et identiques. Chaque job est constitué de k opérations ou tâches, où chacune d'elles doit être exécutée sur une seule machine dans chaque étage. De même, chaque machine ne peut traiter qu'une seule opération à la fois. Aussi, un job ne peut commencer sur cette machine qu'après la fin du job en cours. Dans le modèle HFS, le problème d'affectation des jobs aux machines sur les différents étages s'ajoute au problème d'ordonnancement des jobs.

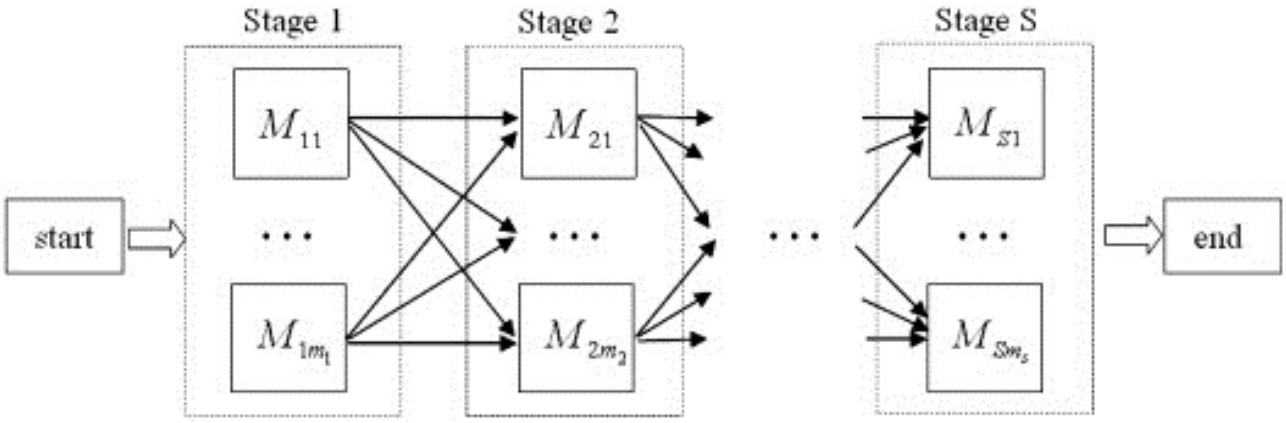


figure. 1.5 – La structure du flow shop hybride

Tous les jobs ont la même gamme, ce qui signifie que l'ordre de passage des opérations sur les k étages est le même pour tous les jobs. Le temps d'exécution de la tâche j à l'étage i notée p_{ji} est connu et fixé, mais les temps d'exécution peuvent être différents d'un job à l'autre, et d'un étage à l'autre pour le même job. La préemption n'est pas autorisée, ce qui revient à dire qu'on ne peut pas interrompre une tâche une fois commencée.

Nous rappelons que Graham et al. [5] ont proposé la notation $\alpha|\beta|\gamma$ pour distinguer entre les problèmes d'ordonnancement. Selon cette classification, le flow shop hybride est représenté comme suit : $FHm, ((PM^{(l)})_{l=1}^{m})\|C_{max}$ où :

- FH : permet d'identifier le problème du flow shop hybride
- m : indique le nombre d'étages
- P : indique le type de machines (Parallel machines).
- M^l : indique le nombre de machines à l'étage l.
- C_{max} : indique l'objectif à minimiser.

1.4.2 Complexité problématique

La complexité d'un problème d'ordonnancement constitue un outil important qui permet de déterminer la classe de difficulté de ce problème et choisir la méthode de résolution appropriée. Les premiers travaux sur la complexité du problème HFS sont réalisés par Gupta [8]. L'auteur a traité le problème du flow shop hybride à deux étages, et il a démontré que le problème est NP-complet.

Théorème 1.4.1. *pour* $\max(m_1, m_2) > 1$*, le flow shop hybride à deux étage est NP-complet même si le nombre de machines à l'un des deux étages est égal à un.*

Gupta a montré aussi que le problème est NP-difficile dans le cas de la minimisation de la plus grande date de fin (Makespan).
Les résultats présentés dans [9] ont montré que le problème du flow shop hybride est NP-difficile au sens fort. Rappelons que le problème HFS combine les caractéristiques des problèmes flow shop et machines parallèles. Le tableau 1.3 donne la complexité des différents problèmes d'ordonnacement selon la fonction objectif. Dans ce tableau P représente la classe P, NP* désigne la classe NP difficile au sens faible, et NP indique la classe NP difficile au sens fort.

Table. 1.3 – Classes de complexité des problèmes d'ordonnancement d'ateliers monocritères

Problème	C_{max}	L_{max}	$\sum C_j$	$\sum \omega_j C_j$	$\sum T_j$	$\sum \omega T_j$	$\sum U_j$	$\sum \omega U_j$
Une seule machine	P	P	P	P	NP*	NP	P	NP*
Machines parallèles	NP	NP	P	NP	NP	NP	NP	NP
Flow shop	NP	NP	NP	NP	NP	NP	NP	NP
Flow shop hybride	NP	NP	NP	NP	NP	NP	NP	NP
Job shop flexible	NP	NP	NP	NP	NP	NP	NP	NP
Open shop généralisé	NP	NP	NP	NP	NP	NP	NP	NP

1.4.3 Etat de l'art sur le problème du flow shop hybride

Le problème du flow shop hybride est l'un des problèmes les plus difficiles en optimisation combinatoire, et également les plus étudiés en raison de son intérêt pratique. Il est très courant dans le domaine industriel tels que l'industrie manufacturière, alimentaire, chimique et textiles, de ce fait, plusieurs systèmes de production peuvent être modélisés comme un flow shop hybride. Gupta [8] a montré la complexité du problème et également son intérêt théorique, ouvrant ainsi le champ des études de ce type de problèmes d'ordonnancement. Depuis, de nombreuses méthodes exactes, heuristiques et métaheuristiques ont été proposées pour résoudre le HFS, avec comme objectif la minimisation du makespan, ou du temps de séjour total (flow time total), ou du retard total,...etc.
La méthode de séparation et évaluation (Branch and Bound, BB) est la méthode exacte largement utilisée pour résoudre le problème du flow shop hybride. Plusieurs recherches traitent le cas particulier à deux étages, qui reste NP-difficile si au moins l'un des deux étages contient

deux machines [8]. Par exemple, Lee et Kim [10] ont proposé une procédure de séparation et évaluation pour résoudre le problème du flow shop hybride à deux étages, avec une seule machine au premier étage et plusieurs machines au deuxième étage, l'objectif et de minimiser le retard total $\overline{T}$ $(FH2, (1^{(1)}, PM^2) \| \overline{T})$. Les auteurs ont développé deux bornes inférieures pour les utiliser dans l'algorithme de (BB). Des instances générées aléatoirement sont utilisées pour évaluer La performance de la méthode. Aussi, Allaoui et artiba [11] ont développé une méthode de séparation et évaluation pour le même problème avec des contraintes de disponibilité des machines. Leur travail utilise le concept de branch and bound présenté dans l'article de Brah et Hunsucker [14].

Dans [12], les auteurs ont présenté une procédure de BB pour le problème du flow shop hybride à deux étages avec une nouvelle borne inférieure. De plus, les auteurs ont proposé des heuristiques basées sur des règles de priorité et sur l'heuristique "Shifting Bottleneck". Choi et Lee [13] s'adressent au même problème, mais avec l'objectif de minimiser le nombre de travaux en retard ($\sum \mathrm{U_j}$) représenté par :

$$\sum_{i=1}^{i=n} \delta(T_i) \tag{1.5}$$

avec $T_i = \max(C_{i2} - d_i, 0)$, et $\delta(a) = \begin{cases} 1 & \text{si a>0} \\ 0 & \text{sinon} \end{cases}$

C_{i2} est la date de fin du job i au deuxième étage, et d_i est sa date d'échéance. Les auteurs ont proposé un algorithme BB avec des nouvelles méthodes pour obtenir une borne inférieure et des propriétés de dominance qui peuvent réduire l'espace de recherche. De plus, ils ont utilisé des heuristiques à deux phases basées sur l'approche de backward (backward scheduling approach).

Pour le problème à k étages, plusieurs travaux ont également utilisé la méthode de Branch and Bound, nous citons par exemple le travail de Brah et Hunsucker [14], qui a présenté une méthode BB qui repose sur l'énumération des séquences d'opérations associées aux machines pour chaque étage. Cette énumération est accomplie en générant un arbre qui contient deux types de nœuds, des nœuds ronds et des nœuds carrés. Ainsi si on rencontre un nœud rond, l'affectation d'une opération s'opère à la suite des opérations déjà affectées sur la machine courante, tandis que pour un nœud carré, on affecte l'opération sur une nouvelle machine qui devient à son tour la machine courante. Dans ce travail, les auteurs ont développé deux classes de bornes inférieures, la première borne est liée à la machine alors que la deuxième est liée au job. À chaque nœud, les bornes sont calculées, ce qui diminuent considérablement le nombre de nœuds explorés dans la recherche arborescente.

Rajendran et Chaudhuri [15] ont présenté aussi un algorithme BB pour le même problème. Ils ont considéré $\beta = prmu$, ce qui veut dire que les jobs doivent être exécutés dans le même ordre sur tous les étages. Divers problèmes tests ont été générés et résolus par la procédure d'évaluation et séparation proposée.
Dans leur travail [16], Carlier et Néron ont proposé une méthode de séparation et évaluation qui se base sur des relaxations successives du problème initial à des problèmes à m machines, chacun d'eux correspond à un étage. Les bornes sont donc dérivées de celles utilisées pour le problème à m machines. Le schéma de séparation utilisée est une généralisation de la notion d'arbitrage présentée pour les problèmes disjonctifs. Aussi Néron et al.[44] ont présenté une méthode BB améliorée pour le problème, basée sur un raisonnement énergétique et des opérations globales pouvant améliorer l'efficacité des procédures de branchement et de liaison.
L'étude de Fattahi et al. [17] a considéré le problème du flow shop hybride avec opérations d'assemblage, où les produits sont fabriqués dans les deux premiers étages avant de passer par une étape d'assemblage sur un troisième étage. Les auteurs ont proposé un algorithme de Branch and Bound hiérarchique, ils ont développé des bornes inférieures et supérieures pour aider l'algorithme à considérer les interdépendances entre les étapes de fabrication et d'assemblage.

Les méthodes approchées offrent une alternative intéressante qui permet de trouver des solutions approchées aux problèmes de grande taille. Différentes heuristiques et métaheuristiques ont été proposées pour résoudre le problème du flow shop hybride. Parmi les heuristiques proposées, nous citons le travail de Gupta [8] qui a étudié la minimisation du makespan dans un problème du flow shop hybride à deux étages, pour un cas particulier où il n'y a qu'une seule machine au deuxième étage. L'auteur a proposé une méthode heuristique qui décompose le problème initial en deux problèmes : un problème de séquencement et un problème d'affectation où l'auteur a utilisé une règle d'affectation des jobs aux différentes machines, qui permet de minimiser le temps d'inactivité sur le deuxième étage. De même, Gupta et al. [19] ont proposé des heuristiques constructives pour résoudre le même problème avec des machines parallèles au premier étage.
Lee et Vairaktarakis [18] ont présenté une heuristique pour le flow shop hybride à deux étages avec un nombre quelconque de machines à chaque étage. Les auteurs ont développé cinq bornes inférieures qui sont utilisées pour montrer que le saut relatif de l'heuristique par rapport à l'optimalité est petit. Ensuite, ils ont étendu leur méthode au cas de plus de deux étages.

Le travail de Guinet al. [20] a proposé une résolution du problème par une heuristique basée sur une décomposition à deux phases : séquencement et affectation étage par étage. Brah et

Loo [21] ont proposé également une heuristique basée sur le séquencement et l'affectation des machines pour minimiser le makespan et le temps de séjour moyen.

Dans [22], les auteurs ont conçu deux heuristiques basées sur le recuit simulé (SA) et la recherche tabou (TS) pour résoudre le même problème. Leur approche améliore les résultats de Lee et Vairaktarakis [18].

Le travail de Yang [23] a considéré le problème de minimisation du makespan dans un flow shop hybride à deux étages avec machines dédiées. L'auteur a développé quatre heuristiques basées sur la règle de Johnson et la règle d'ordonnancement glouton, puis il a comparé ses performances dans diverses conditions en examinant les effets des divers paramètres.

Arboleda et al. [24] ont étudié le problème du flow shop hybride à k étages, ils ont proposé une heuristique fondée sur l'identification et l'exploitation de l'étage bottleneck (l'étage qui a la plus faible cadence dans le flux de production). Leur approche est inspirée de la théorie des contraintes qui part du postulat que le déséquilibre est inévitable, alors il vaudrait mieux l'admettre et l'exploiter pour améliorer la performance du système. Les résultats expérimentaux présentés par les auteurs montrent la compétitivité de leur méthode par rapport aux deux autres heuristiques de la littérature basées également sur la procédure de shifting bottleneck.

Dans le même contexte, Chen et Chen [25] ont développé deux heuristiques basées sur la notion de l'étape bottleneck, pour un problème similaire mais avec des machines non reliées. L'objectif du problème est de minimiser le retard total. Sept règles de priorité et un algorithme de recherche tabou de base sont examinés à des fins de comparaison.

Les métaheuristiques présentent la possibilité de s'adapter à toutes sortes de problèmes, alors elles deviennent de plus en plus un outil de résolution privilégié. En général, l'application des métaheuristiques aux problèmes d'ordonnancement est très populaire. De là, nous trouvons dans la littérature un nombre important de travaux qui utilisent des approches à base de métaheuristiques pour résoudre le problème du flow shop hybride.

La plupart de ces travaux ont considéré la minimisation du makespan, nous citons par exemple le travail de Engin et Döyen [26], où les auteurs ont proposé un système immunitaire artificiel (AIS), qui est une technique intelligente de résolution inspirée de l'immunologie théorique. Dans leur étude comparative, ils ont considéré les benchmarks présenté par Carlier et Néron [16]. Alaykýran et al. [27] ont utilisé le même benchmark pour valider l'algorithme de colonie de fourmis (ACO) proposé.

Kahraman et al.[28] ont présenté un algorithme génétique (GA) pour résoudre le problème, ils ont comparé leur résultats avec ceux de [26].
Niu et al. [29] ont étudié le même problème avec l'objectif de minimiser le makespan. Les auteurs ont présenté un algorithme immunitaire quantique (Quantum Immune Algorithm, QIA), qui combine les principes de l'algorithme immunitaire (IA) avec les principes de la recherche quantique (QA). Les résultats de la simulation montrent que l'algorithme QIA améliore considérablement les performances de l'algorithme IA, et produit également des résultats plus efficaces et plus stables que l'algorithme QA. Le benchmark de carlier est utilisé pour montrer la performence de l'approche proposée. Les auteurs ont proposé aussi un algorithme QIA amélioré pour le même problème, mais avec l'objectif de minimiser le temps de séjour moyen ($\overline{F} = \frac{\sum F_j}{n}$) [30].

Liao et al. [31] ont proposé un algorithme hybride à base de l'optimisation par essaims particulaires et le recuit simulé. Ils ont utilisé l'heuristique Shifting Bottleneck pour mieux exploiter l'étape du goulot d'étranglement. Chung et Liao [32] ont proposé un algorithme de système immunitaire artificiel à base des immunoglobulines (IAIS). L'algorithme vient d'améliorer la méthode AIS. Les comparaisons ont considérées des algorithmes basés sur le système immunitaire et d'autres algorithmes de la littérature. L'algorithme IAIS a donné des résultats compétitifs et a pu améliorer les solutions trouvées sur les problèmes tests [16].
Le travail [33] a présenté une recherche tabou parallèle pour résoudre le problème. Les auteurs ont implémenté deux versions principales de la recherche tabou (TS) : le TS séquentiel pur et le TS parallélisé, ce dernier utilise un accélérateur parallèle pour accélérer le calcul.
Marichelvama et al. [34] ont proposé un algorithme de recherche coucou (Cuckoo Search algorithm, CS) pour la résolution du problème. Notons que CS est une méthode d'optimisation globale basée sur la façon dont les coucous pondent leurs œufs dans les nids de l'hôte.
Dans [35], les auteurs ont proposé un algorithme de colonie d'abeilles artificielles (Artificial bee colony algorithm, ABC). L'algorithme est basé sur le comportement intelligent de recherche de nourriture des abeilles. D'abord, un schéma d'initialisation basé sur l'heuristique NEH [70] est conçu dans le but de construire une population initiale diversifiée. Ensuite, un algorithme à évolution différentielle et une recherche à voisinage variable sont appliqués pour générer de nouvelles solutions pour les abeilles employées, les abeilles observatrices et les abeilles scoutes. Pour évaluer sa performance, la méthode est comparée à plusieurs algorithmes de la littérature basés sur les mêmes benchmarks. De même, Pan et al. [36] ont présenté un algorithme de colonie d'abeilles artificielles pour résoudre le problème.

Parmi les études récentes, nous citons par exemple le travail de Siqueira et al. [37] qui a proposé un algorithme de recherche à voisinage variable (VNS) multi-objective. L'étude a considéré l'évaluation de deux critères : la minimisation du makespan et la minimisation de

Table. 1.4 – Quelques importants travaux de l'état de l'art

Auteurs	Année	Problème	Commentaire
Gupta	1988	$FH2, (P2^{(1)}, 1^2) \| C_{max}$	Heuristiques
Brah et Hunsucker	1991	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	BB
Rajendran et Chaudhuri	1992	$FHm, (PM^{(l)})_{l=1}^{m} \| prmu \| C_{max}$	BB
Lee et Vairaktarakis	1994	$FH2, (PM^{(l)})_{l=1}^{2} \| C_{max}$	Heuristique
Guinet et al.	1996	$FH2, (PM^{(l)})_{l=1}^{m} \| C_{max}$	Heuristique + Règles de priorité
Gupta et al.	1997	$FH2, (PM^{(1)}), 1^2 \| C_{max}$	Heuristique constructive
Haouari et M'Hallah	1997	$FH2, (PM^{(1)})_1^2 \| C_{max}$	Heuristique à base de SA et TS
Brah et Loo	1999	$FHm, (PM^{(l)})_{l=1}^{m} \| \{C_{max}, \overline{F}\}$	Heuristiques
Vignier et al.	1999	$FH \star\star\star \mid \star \mid \star$	État de l'art
Carlier et Néron	2000	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	BB
Lee et Kim	2004	$FH2, (1^{(1)}, PM^2) \| \overline{T}$	BB
Engin et Döyen	2004	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	AIS
Haouari et al.	2006	$FH2, (PM^{(l)})_{l=1}^{m} \| C_{max}$	BB
Alaykýran et al.	2007	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	ACO
Arboleda et al.	2008	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	Heuristique
Kahraman et al.	2008	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	GA
Arboleda et al.	2008	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	heuristique (étage Bottleneck)
Niu et al.	2009	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	QIA
Choi et Lee	2009	$FH2, (PM^{(l)})_{l=1}^{m} \| \sum U_j$	BB
Ruiz et Rodríguez	2010	$FH \star\star\star \mid \star \mid \star$	Revue de littérature
Liao et al.	2012	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	PSO+SA
Niu et al.	2013	$FHm, (PM^{(l)})_{l=1}^{m} \| \overline{F}$	QIA amélioré
Bożejko	2013	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	TS parallèle
Chung et Liao	2013	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	IAIS
Pan et al.	2014	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	ABC
Marichelvama et al.	2014	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	CS
Cui et al.	2015	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	ABC
Siqueira et al.	2018	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	VNS
Fernandez-Viagas et al.	2019	$FHm, (PM^{(l)})_{l=1}^{m} \| C_{max}$	Représentations de solutions
Lang et al.	2021	$FH2, (PM^{(l)})_{l=1}^{m} \| \{C_{max}, \overline{T}\}$	NEAT

la somme pondérée du retard. Nous pouvons également citer l'étude [38], qui a abordé le problème du flow shop hybride avec la minimisation du makespan. Les auteurs ont analysé les différentes représentations de solutions utilisées dans la littérature pour le problème, et ils ont posé la question sur la qualité des ordonnancements qui sont atteints par ces représentations. Récemment, le travail [39] a étudié l'application de l'algorithme NEAT (NeuroEvolution of augmenting topologies) sur le problème du flow shop hybride à deux étage en considérant la minimisation du retard total $\overline{T}$ et le makespan. NEAT est un algorithme génétique utilisé pour la génération des réseaux de neurones artificiels. La performance de la méthode est comparée à d'autres méthodes tels que l'algorithme GA, l'algorithme TS et le recuit simulé. Nous notons selon l'état de l'art que l'étude du problème HFS est un domaine de recherche très actif.

Pour un état de l'art détaillé sur les problèmes d'ordonnancement de type flow shop hybride, nous référons le lecteur aux articles [52,61].

1.4.4 Critère d'optimalité

Dans cette thèse, nous nous intéressons à la minimisation du makespan :

$$\mathrm{C}_{\max} = \max_{j \in \{1,\ldots,n\}} C_j$$

Ce dernier est l'un des critères les plus étudiés pour évaluer la performance d'un ordonnancement, et les plus utilisés en pratique dans les systèmes de production. Il est particulièrement intéressant puisque les ordonnancements au plus tôt constituent des sous-ensembles dominants [1] pour de nombreux critères réguliers [40]. De plus, la minimisation de ce critère permet d'améliorer le rendement et réduire le temps moyen d'inactivité des machines [53].

1.5 Conclusion

Dans ce chapitre, nous avons donné un aperçu sur l'optimisation combinatoire et la théorie de complexité. Puis, nous avons parlé de la gestion de production, en montrant l'importance de l'ordonnancement dans les systèmes de production. Nous avons présenté les caractéristiques principales d'un ordonnancement, ainsi que les définitions et les classifications des problèmes d'ordonnancement d'atelier. Nous avons aussi montré la complexité du problème traité. Un état de l'art du problème traité est donné à la fin du chapitre.

1. Un ensemble de solutions d'un problème d'optimisation est dit dominant s'il contient au moins une solution optimale

Chapitre 2

Méthodes d'optimisation

Les problèmes d'ordonnancement appartiennent à une classe de problèmes mathématiques NP-complets marquée par l'explosion combinatoire. Les méthodes de résolution de ces problèmes sont essentiellement celles de la discipline de la recherche opérationnelle. Cette dernière regroupe différentes méthodes et techniques pour la résolution des problèmes d'optimisation combinatoire. Généralement, on considère deux voies de résolution des problèmes d'ordonnancement :

- Une résolution exacte selon une stratégie connue qui mène à trouver une solution optimale du problème, mais qui nécessite souvent des temps de calcul importants.
- Une résolution approchée qui permet d'obtenir des solutions approchées sans garantie d'optimalité, mais en un temps raisonnable.

Dans ce chapitre, nous allons introduire ces deux voies, nous commençons par présenter trois méthodes exactes de la littérature. Pour les méthodes approchées, nous détaillons plus particulièrement celles qui seront utilisées dans notre étude.

2.1 Les méthodes exactes

Une méthode exacte est une méthode qui permet d'assurer la résolution optimale d'un problème d'optimisation combinatoire, c'est à dire de générer son optimum. Elle consiste à explorer l'espace de recherche du problème d'une manière implicite afin de trouver une solution optimale. Les méthodes exactes demandent un temps de calcul croissant exponentiellement avec la taille du problème, donc elles sont incapables de résoudre des problèmes de grandes tailles en un temps acceptable. De là, elles s'utilisent généralement pour résoudre des problèmes de petites tailles.

Parmi les méthodes exactes utilisées pour résoudre des problèmes d'ordonnancement d'ateliers, nous citons particulièrement les méthodes : Branch and Bound, la programmation dynamique et la programmation linéaire. Nous abordons dans ce qui suit ces trois méthodes.

Branch and Bound

La procédure par séparation et évaluation également appelée branch and bound (BB) est l'une des méthodes les plus utilisées pour la résolution optimale des problèmes d'optimisation combinatoire, y compris les problèmes d'ordonnancement. Elle a été proposée initialement par Land et Doig en 1960 [45]. Cette méthode est basée sur une énumération intelligente de l'ensemble des solutions afin d'éviter la numération des mauvaises solutions. Elle comprend la construction d'un arbre de recherche dont les nœds représentent des sous-ensembles de solutions de plus en plus petits, cet arbre sera exploré de façon à éliminer les branches qui ne contiennent pas des solutions intéressantes.
La méthode de branch and bound repose sur deux principes : la séparation et l'évaluation.

— La séparation consiste en la réduction progressive de l'ensemble de solutions en le décomposant en sous-ensembles plus petits. Ces sous-ensembles définissent à leur tour des sous problèmes de façon à ce que la résolution des sous problèmes mène à la résolution du problème originel.

— L'évaluation consiste à évaluer les solutions des sous problèmes associés aux sous-ensembles engendrés par la séparation, pour réduire le nombre des solutions explorées et supprimer les sous-ensembles qui ne contiennent pas la solution optimale. Pendant sa progression, l'algorithme de branch and bound alterne à chaque itération entre ces deux principes.

La méthode de Branch and Bound est largement utilisée pour résoudre des problèmes d'ordonnancement [97,95,100,87]. La section 1.4.2 (chapitre1) a présenté un état de l'art sur l'utilisation de BB et les méthodes approchées pour résoudre les problèmes du flow shop hybride. On note que la méthode donne de bons résultats si la taille du problème est petite, sinon il sera difficile de résoudre le problème dans un temps raisonnable.

Programmation dynamique

La programmation dynamique est une technique d'optimisation dont ses origines remontent aux travaux de Bellman [71]. Elle est basée sur l'idée que dans une séquence optimale, chaque sous séquence doit aussi être optimale. La méthode est exploitable pour tout problème pouvant être décomposé en sous-problèmes liés entre eux par une relation de récurrence, de façon que la valeur optimale de la fonction objectif à une étape donnée est décrite en fonction de sa valeur à l'étape précédente. De ce fait, le problème de base peut être résolu en combinant

les solutions des sous problèmes mémorisés au préalable. Comme exemple d'utilisation de la programmation dynamique pour le problème du flow shop hyride, on peut citer le travail [50]. Comme la plupart des méthodes exactes, cette approche atteint ses limites à mesure que la taille du problème devient grande.

Programmation linéaire

La programmation linéaire (PL) est un outil classique de recherche opérationnelle basé sur la modélisation. Son utilisation demande que le problème posé puisse se ramener à l'optimisation d'une fonction objectif linéaire, en respectant des contraintes qui sont décrites par des fonctions linéaires. Il y a trois types de programmes linéaires : les programmes linéaires continus, les programmes linéaires en nombres entiers et les programmes linéaires en nombres mixtes. On parle d'un programme linéaire continu si les variables sont des nombres réels. Par contre si les variables sont des nombres entiers, il s'agit d'un programme linéaire en nombres entiers. Dans le cas où les variables peuvent prendre à la fois des valeurs réelles et des valeurs entières, on parle d'un programme linéaire mixte.
Notons que les programmes linéaires en nombres entiers sont les plus difficiles à résoudre. L'intérêt de la programmation linéaire réside non seulement dans le fait qu'un grand nombre de problèmes réels se modélisent en programmes linéaires, mais aussi dans l'existence de plusieurs logiciels et langages de modélisation (LP-Solve, Cplex,...) qui permettent de résoudre ces modèles linéaires.

2.2 Les méthodes approchées

Les méthodes exactes sont limitées face aux problèmes d'optimisation de grande taille, ce qui conduit les chercheurs à développer des méthodes approchées en vue de traiter rapidement les problèmes de taille importante. Les méthodes approchées sont considérées comme une alternative intéressante aux méthodes exactes dont l'objectif n'est certainement pas de trouver une solution optimale mais plutôt d'obtenir des solutions de qualité en un temps d'exécution raisonnable. Pour mesurer la performance de ces méthodes, on estime souvent le pourcentage de l'erreur entre la valeur de la solution trouvée et la valeur de l'optimum s'il est calculable, sinon on peut comparer les résultats trouvés avec des bornes inférieures.
On peut classer les méthodes approchées en deux grandes catégories : les heuristiques et les métaheuristiques. Une heuristique est généralement destinée pour résoudre un problème spécifique en se basant sur sa propre structure. Tandis qu'une métaheuristique possède une conception fondamentale qui permet de l'adapter à différentes classes de problèmes [96].

2.2.1 Les heuristiques

Les heuristiques sont donc des méthodes approchées conçues spécifiquement pour un problème d'optimisation donné, afin de trouver une solution que l'on espère proche de l'optimum. D'ailleurs, l'usage de ces techniques permet d'accélérer l'obtention de solutions même si leur qualité n'est pas garantie.
Les heuristiques peuvent être classées en plusieurs types notamment les heuristiques constructives qui permettent de construire la solution étape par étape, et c'est par ajout d'éléments à la solution partielle jusqu'à ce qu'une solution complète soit construite. Elles sont utilisées souvent comme des auxiliaires dans d'autres méthodes surtout pour calculer des solutions initiales. Le fonctionnement d'une heuristique constructive est illustré dans l'algorithme 1.

Algorithme 1 Heuristique constructive

Données : Solution non complète
Résultats : S la solution complète.
Début
  Tant que (S n'est pas une solution complète)
    choisir un élément e à ajouter
    $S \leftarrow S + e$
  Fin Tant que
  Retourner S
Fin

Parmi les types des heuristiques, nous trouvons aussi les heuristiques itératives (Iterative heuristics) ou les heuristique de recherche locale qui partent d'une solution initiale, et tentent de l'améliorer tout en explorant un voisinage. L'algorithme 2 démontre la structure générale d'une heuristique de recherche locale.

Algorithme 2 Heuristique de recherche locale

Données : S une solution de départ pour l'heuristique
Tant que le critère d'arrêt n'est pas satisfait
  Générer un voisin S' de S
  if S' est meilleure que la solution courante S **then**
    $S \leftarrow S'$
  end if
Fin Tant que
Retourner S

Les règles de priorité (dispatching rules) sont aussi parmi les heuristiques couramment utili-

sées, elles sont connues également sous le nom d'algorithmes d'ordonnancement de liste [52]. Elles sont en général des règles qui définissent la priorité entre les travaux en attente de traitement et décident ainsi le travail à traiter sur une machine disponible. La littérature comprend un grand nombre de règles de priorité comme par exemple FIFO (First In First Out) qui donne La priorité à l'opération qui est arrivée en premier dans la file d'attente, SPT (Shortest processing time) où la priorité est donnée à l'opération dont le temps opératoire est le plus petit, et LPT (Largest processing time) qui contrairement à SPT donne la priorité à l'opération avec un temps opératoire plus grand.

Parmi les heuristiques largement présentes dans la littérature des problèmes d'ordonnancement d'ateliers de type flow shop et flow shop hybride, nous citons à titre d'exemple l'heuristique NEH [70] et les heuristiques de Gupta [8,42].

2.2.2 Les métaheuristiques

Une métaheuristique est définie comme un framework algorithmique de haut niveau qui combine intelligemment plusieurs concepts et stratégies et offre une perspective pour développer des algorithmes d'optimisation heuristique [43]. Elle peut être adapter à la résolution de différents problèmes. Les métaheuristiques sont des méthodes approximatives générales qui peuvent s'appliquer à une large classe de problèmes d'optimisation, elles visent principalement à mieux résoudre les problèmes dits d'optimisation difficile. Elles utilisent souvent des concepts d'intensification et de diversification. En ce qui concerne l'intensification, elle consiste à explorer les régions prometteuses de l'espace de solutions, tandis que la diversification réoriente la recherche vers les régions rarement visitées. Il existe un problème lié à la façon de gérer l'emploi de l'intensification et la diversification, mais en général un équilibre tactique entre les deux concepts doit être maintenu au cours du processus de recherche afin d'atteindre rapidement les régions de l'espace de recherche contenant des bonnes solutions. Les métaheuristiques reposent généralement sur un ensemble commun de principes [96] :

- Elles sont souvent des approches stochastiques ce qui permet de faire face à l'explosion combinatoire (le fait que le temps de calcul augmente exponentiellement avec la taille du problème).
- Elles sont généralement d'origine discrète.
- Elles sont inspirées par des analogies avec la physique comme le recuit simulé, avec la biologie comme par exemple les algorithmes évolutionnaires, ou avec l'éthologie (colonies de fourmis, essaims particulaires...)..
- Le travail de réglage de leurs paramètres est généralement difficile.

Les métaheuristiques peuvent être classées en deux familles selon le nombre de solutions qu'elles manipulent : les métaheuristiques à solution unique qui se basent sur la progression itérative d'une seule solution et les métaheuristiques à population de solutions qui mani-

pulent parallèlement une population de solutions. Nous détaillons ces deux familles dans les paragraphes suivantes.

2.2.2.1 Les métaheuristiques à solution unique

Les métaheuristiques à solution unique connues aussi sous le nom de méthodes de recherche locale ou méthodes de trajectoire, consistent à faire évoluer une solution unique d'une manière itérative dans l'espace de recherche. Elles partent d'une solution initiale, puis en définissant une relation de voisinage qui est une application qui associe à chaque solution un ensemble de solutions appelées voisins, elles expoloitent à chaque itération un voisinage de la solution courante et sélectionnent un voisin. La recherche se fait en privilégiant le choix des meilleurs voisins. Le processus se termine si on ne peut plus améliorer la solution courante ou si le critère d'arrêt est atteint. L'idée est donc de structurer l'espace de recherche en terme de voisinage, en identifiant les règles pour se déplacer d'une solution à l'autre. Ainsi des trajectoire ont été construites dans l'espace de recherche, dans le but de se diriger vers des solutions optimales. Les méthodes de recherche locale sont très nombreuses, nous présentons par la suite deux exemples typiques de méthodes de recherche locale : la recherche tabou et le recuit simulé.

la recherche tabou

La recherche tabou (Tabu search, TS) est une stratégie de recherche locale présentée initialement par Fred Glover en 1986 [57], et détaillée plus tard dans ses deux articles [58,59]. Elle consiste principalement à utiliser la notion de mémoire dans les stratégies d'exploration de voisinage. La méthode part d'une solution initiale, puis elle explore à chaque itération un voisinage de la solution courante en exécutant un ensemble d'opérateurs de voisinage (mouvements d'insertion, d'inversion...). Malheureusement, les méthodes de recherche locales sont sujettes à être bloquées dans des zones d'optimalité similaires, la recherche tabou introduit alors des mécanismes qui visent à s'échapper des optima locaux et à explorer de nouvelles zones de l'espace de recherche.
Pour éviter de retourner à des solutions recemment visitées, la recherche tabou stocke leurs informations dans une liste d'interdiction appelée liste tabou. En d'autres termes, le but de classer une solution comme interdite, c'est à dire tabou est d'empêcher le cyclage. Ainsi la recherche d'un meilleur voisin de la solution courante dans un voisinage se fait en ignorant les solutions de la liste tabou. Il faut souligner que la taille de cette liste est un paramètre qui a beaucoup d'impact sur la performance de la méthode.

Algorithme 3 Pseudo code de la recherche tabou

Données : s_0 une solution initiale, $L = \emptyset$ la liste tabou initiale est vide.
Début
  Résultas : s : la meilleure solution trouvée
  $s \leftarrow s_0$
  Tant que le critère d'arrêt n'est pas satisfait
    Générer un ensemble de solutions voisines de s
    Choisir s' le meilleur voisin non interdit (qui n'appartient pas à la liste tabou).
    Mettre à jour la liste tabou
    $s \leftarrow s'$
  Fin Tant que
Fin

Recuit simulé

L'idée du recuit simulé (Simulated Annealing, SA) est inspirée par une analogie avec le procédé métallurgique du recuit. Ce dernier s'appuie sur la réalisation des cycles contrôlés de refroidissement et de réchauffage d'un matériau en vue de minimiser son énergie. En fait, à une température élevée, les particules du matériau se trouvent dans un état de désordre total. Lorsque la température diminue lentement en marquant des différents paliers de températures, un équilibre thermique est atteint à chaque palier. Les particules deviennent de plus en plus stables en s'approchant de la température nulle. On note qu'un refroidissement vite peut causer de nombreux défauts microscopiques et c'est par analogie correspond à un optimum local en optimisation.

L'adaptation de cette technique pour résoudre des problèmes d'optimisation a donné naissance à la méthode du recuit simulé. La méthode dont les origines remontent à la procédure de Metropolis (1953) [82] a été proposée par Kirkpatrick et al. en 1983 [76], comme une méthode itérative qui évite les optimums locaux.
Le recuit simulé utilise la règle d'acceptation de Metropolis : Étant donné un système physique à une température T évoluant vers son équilibre thermique (énergie minimale), une transformation qui provoque une augmentation de l'énergie est acceptée avec la probabilité $e^{\frac{-\Delta E}{T}}$. C'est pourquoi le recuit simulé autorise non seulement des solutions qui améliorent la fonction objectif qui est analogue à l'énergie, mais aussi des solutions qui la dégradent avec la probabilité $e^{\frac{-\Delta E}{T}}$), ce qui permet d'éviter la convergence vers un optimum local.

L'un des principaux avantages de l'algorithme du recuit simulé est la facilité d'implémentation, aussi la méthode peut être adaptée à la plupart des méthodes d'optimisation. En

revanche, sa grande faiblesse réside dans le nombre important des paramètres.

2.2.2.2 Les métaheuristiques à population de solutions

Les métaheuristiques à population de solutions et contrairement aux métaheuristiques à solution unique travaillent sur un ensemble de solutions en parallèle, et tentent de l'améliorer itérativement. Ces méthodes consistent à utiliser régulièrement les propriétés collectives d'un ensemble de solutions distinguables, appelé population, dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche [102].Elles sont des méthodes de recherche globale qui explorent des différentes régions de l'espace de recherche ce qui permet de trouver des solutions de bonne qualité dans des temps de calcul raisonnables. Les métaheuristiques se différent les unes des autres par les stratégies d'évolution de la population d'une itération à l'autre.
Nous introduisons par la suite les métaheuristiques utilisées dans la cadre de cette thèse.

Optimisation par essaims particulaires

L'optimisation par essaims particulaires (Particle Swarm Optimization, PSO) est une métaheuristique issue d'une simulation du comportement social d'animaux évoluant en essaim, tels que les oiseaux migrateurs, les bancs de poissons et les abeilles. Elle a été introduite par Kennedy et Eberhart en 1995 [63] pour résoudre des problèmes difficiles de l'optimisation continue. Les auteurs se sont inspirés du comportement collectif au sein de groupes d'animaux, à la lumière de la théorie de la socio-psychologie sur le partage de l'information et les prises de décisions dans ces groupes.

L'optimisation par essaims particulaires s'apparente aux algorithmes génétiques dans le sens où elle repose sur l'évolution d'une population de solutions afin de trouver la meilleure solution possible à un problème donné. Mais à la différence des algorithmes génétiques s'appuyant sur les mécanismes de la sélection naturelle, qui permettent d'éliminer les individus moins performants, PSO adopte une stratégie de collaboration entre les particules pour faire évoluer l'essaim.

La méthode est basée sur une population d'individus appelés particules qui se déplacent dans l'espace de recherche, suivant un ensemble de règles de changement de direction et de regroupement, afin de se positionner sur des solutions optimales. Initialement, les paritcules sont distribuées d'une manière aléatoire dans l'espace de recherche. Chaque particule possède une vitesse (appelée vélocité) qui sera ajustée dynamiquement en fonction de la performance historique de la particule ainsi que la meilleure performance de leurs voisins dans l'espace de recherche. La performance historique de la particule appelée aussi son expérience désigne

que sa meilleure position trouvée jusqu'à présent est stockée dans sa mémoire. En d'autres termes, PSO dirige les particules de l'essaim vers les zones prometteuses de l'espace de recherche en réponse aux mémoires individuelles et collectives diffusées à travers cet essaim.

Les particules qui correspondent aux solutions du problème traité, se déplacent dans l'espace de recherche vers une meilleure position en effectuant la mise à jour de leurs vitesses et leurs positions courantes. A chaque itération, le déplacement des particules se fait en tenant compte de trois composantes :

- **Une composante d'inertie** : où la particule tend à suivre la direction induite par sa vitesse courante.
- **Une composante cognitive** : où la particule tend à se diriger vers sa meilleure performance (sa meilleure position visitée).
- **Une composante sociale** : où la particule tend à se diriger vers la meilleure performance de ses voisins (la meilleure position atteinte par ses voisins).

Chaque particule s'évolue en fonction de sa vitesse, sa meilleure performance et la meilleure performance de ses voisins. La stratégie de déplacement d'une particule est illustrée dans la figure 2.1.

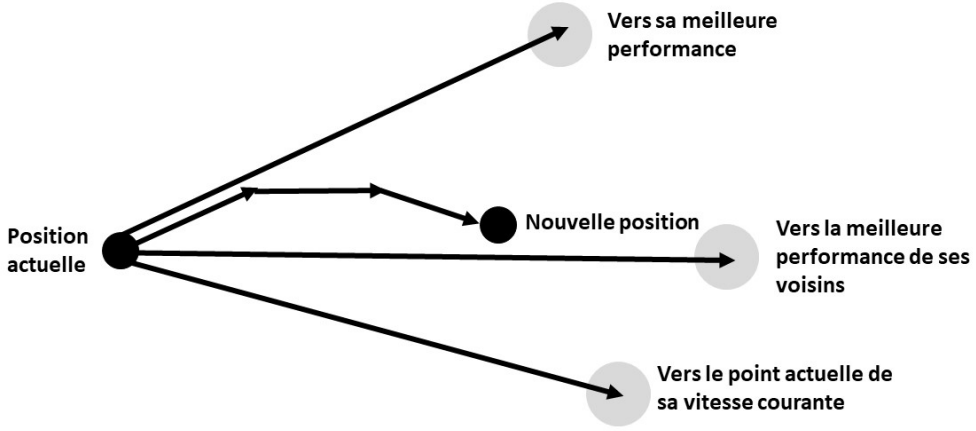


figure. 2.1 – Déplacement d'une particule

On considère dans un espace de recherche de dimension d une particule i, qui est représentée

par son vecteur position :

$$X_i = (x_{i1}, x_{i2}, ..., x_{id})$$

et sa vitesse :

$$V_i = (v_{i1}, v_{i2}, ..., v_{id})$$

Il convient de mentionner ici que dans la version globale de PSO, toutes les particules de l'essaim sont considérées comme voisines de la particule i, d'où la meilleure performance de ses voisines est égale à la meilleure performance de tout l'essaim.
Le vecteur $Pb_i = (pb_{i1}, pb_{i2}, ..., pb_{id})$ désigne la meilleure position atteinte par la particule i, et $Pb_g = (pb_{g1}, pb_{g2}, ..., pb_{gd})$ représente la meilleure position atteinte par tout l'essaim.

La vélocité et la position de la particule i à l'itération $t+1$ sont calculées à partir de leurs valeurs à l'itération t par les équations suivantes :

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (Pb_i(t) - X_i(t)) + c_2 r_2 (Pb_g(t) - X_i(t)) \tag{2.1}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{2.2}$$

Où $V_i(t)$ et $X_i(t)$ sont respectivement la vitesse et la position de la particule i à l'itération t, c_1 et c_2 sont deux constantes appelées coefficients d'accélération, ω désigne le coefficient d'inertie, et r_1 et r_2 sont deux nombres aléatoires uniformément générés dans $[0, 1]$. Nous rappelons que les deux constantes c_1 et c_2 permettent de pondérer l'orientation de la particule à choisir son expérience individuelle (aspect cognitif) ou son sens collectif (aspect social), tandis que ω exprime sa tendance à suivre son propre instinct (aspect égoïste). La manipulation de ces coefficients permet de contrôler l'intensification et la diversification de la méthode, en vue de garder un équilibre entre les deux concepts durant la recherche.

Comme mentionné précédemment, l'algorithme est développé originellement pour résoudre des problèmes d'optimisation continue, mais plusieurs auteurs ont proposé des versions discrètes de PSO afin de l'adapter à résoudre des problèmes de l'optimisation combinatoire y compris des problèmes d'ordonnancement. Ces versions sont basées essentiellement sur la façon de représenter les solutions du problème traité, ainsi que les expressions de la vitesse et la position. Cela signifie que le passage du continu au discret consiste à modifier les équations de vitesse et de position qui représentent l'évolution des particules.

La méthode PSO est utilisée avec succès pour résoudre de nombreux problèmes d'ordonnancement [31,73,74].

Algorithme 4 Algorithme de l'optimisation par essaim particulaire

Début
- **Initialiser** l'essaim et les paramètres de l'algorithme.
- **Initialiser** les vitesses et les positions des particules.
- **Evaluer** la fonction objectif pour chaque particule i
- **Initialiser** Pb_i
- **Calculer** Pb_g
- **Tant que** Critère d'arrêt non satisfait
  - **for** i allant de 1 à n **do**
    - **Calculer** la vitesse de i selon l'équation 2.1
    - **Calculer** la position de i selon l'équation 2.2
    - **Evaluer** la fonction objectif pour chaque particule i
    - **if** La fonction objectif de i est meilleure que la fonction objectif de Pb_i **then**
      - Metter à jour Pb_i
    - **end if**
    - **if** La fonction objectif de i est meilleure que la fonction objectif de Pb_g **then**
      - Mettre à jour Pb_g
    - **end if**
  - **end for**
- **Fin Tant que**
- Retourner Pb_g la meilleure solution trouvée au cours de la recherche.

Fin

La recherche d'harmonie

La recherche d'harmonie (Harmony Search, HS) est l'une des plus récentes métaheuristiques basées sur une population, elle est présentée pour la première fois en 2001 par Geem [68] pour résoudre des problèmes d'optimisation continue. La méthode s'inspire du processus naturel de la performance musicale, qui se produit lorsqu'un musicien cherche à trouver un état parfait d'harmonie.

L'harmonie en musique et le fait que différents sons s'accordent (ce qui ajoute une richesse à la mélodie). La recherche d'harmonie imite la performance musicale qui consiste à trouver une harmonie parfaite déterminée par une norme esthétique, qui est à son tour déterminée par l'ensemble des sons joués par des instruments joints. D'une manière analogue, le processus d'optimisation cherche à trouver une solution optimale globale déterminée par une fonction objectif. Et comme les sons peuvent être améliorés par la pratique, répétition après répétition pour une meilleure estimation esthétique, une solution peut être améliorée itération par

itération.

En réalité, si par exemple trois musiciens jouent trois instruments différents, l'intérêt dans le cas d'une improvisation réside dans la coordination entre les trois musiciens pour obtenir une harmonie plaisante. Cette harmonie sera conservée dans la mémoire de chaque musicien, et la possibilité d'obtenir une harmonie parfaite augmente la prochaine fois. De même dans l'optimisation, on génère à chaque itération une solution qui sera conservée si sa qualité est bonne. L'analogie entre le processus d'optimisation et la performance musicale est illustrée dans la figure 2.2 et le tableau 2.1.

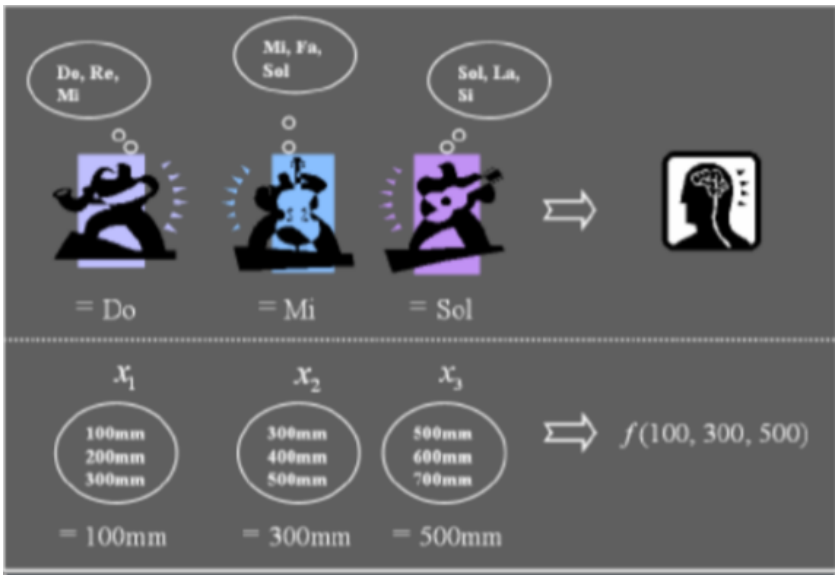


figure. 2.2 – Improvisation musicale et Optimisation [68]

Dans l'algorithme HS, Chaque solution est appelée "harmonie", et est représentée par un vecteur réel de taille n. Une population initiale d'harmonies est générée aléatoirement et stockée dans une matrice appelée mémoire d'harmonie (Harmony Memory, HM) de taille HMS (Hamony memory size). Puis une nouvelle solution (harmonie) candidate est générée à partir de toutes les solutions de HM. Si cette nouvelle solution est meilleure que la pire solution de HM, elle la remplace, sinon elle sera éliminée. L'improvisation d'une nouvelle harmonie utilise trois règles : la considération de la mémoire d'harmonie (Harmony memory consideration), l'ajustement du ton (Pitch adjustment) et la sélection aléatoire (Random selection). En fait, ces trois règles traduisent les trois possibilités qui s'offrent au musicien lorsqu'il joue une mélodie :

- **La considération de la mémoire d'harmonie** correspond à jouer une mélodie connue.

- **L'ajustement du ton** correspond à jouer une mélodie connue, mais en ajoutant des modifications.
- **La sélection aléatoire** correspond à composer une nouvelle mélodie.

Notons que ces règles sont dirigées par deux paramètres principaux : $HMCR$ (Harmony memory consideration rate) qui désigne le taux de considération de la mémoire d'harmonie, ce paramètre définit la probabilité d'utiliser les informations stockées dans la mémoire d'harmonie. Le deuxième paramètre PAR (Pitch adjustment rate) signifie le taux d'ajustement du ton, il indique la probabilité de muter l'harmonie. Ces deux paramètres contrôlent la diversification et l'intensification de l'algorithme HS et affectent sa vitesse de convergence.

Table. 2.1 – Analogie entre la musique et l'optimisation

Processus musical	Processus d'optimisation
Harmonie musicale	Solution
Improvisation musicale	Itération
Instrument musical	Variable de décision
Qualité de l'harmonie	Fonction objectif

Les étapes de l'algorithme HS sont illustrées dans la figure 2.3, et peuvent être résumées comme suit :

Etape 1 Initialiser la mémoire d'harmonie (HM). HM initiale se compose d'une popoulation de solutions générées aléatoirement. Pour un problème à n dimensions, une mémoire d'harmonie HM de taille HMS (Harmony memory size) peut être représentée de la manière suivante :

$$HM = \begin{pmatrix} x_1^1 & x_2^1 & . & . & . & x_n^1 \\ x_1^2 & x_2^2 & . & . & . & x_n^2 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_1^{HMS} & x_2^{HMS} & . & . & . & x_n^{HMS} \end{pmatrix}$$

où $x_i = (x_1^i, x_2^i, ..., x_n^i)$, $i \in \{1, 2, ..., HMS\}$ est une solution candidate. Les solutions de HM sont générées dans la version standard de HM en utilisant l'équation suivante :

$$x_i(j) = LB(j) + (UB(j) - LB(j)) \times rand \tag{2.3}$$

pour $j = 1, ..., n$ et $i = 1, ..., HMS$. LB et UB sont la borne inférieure et la borne supérieure de $x(j)$, et rand est un nombre aléatoire entre 0 et 1.

Etape 2 Improviser une nouvelle harmonie à partir de HM, en appliquant les trois règles : la considération de la mémoire d'harmonie, l'ajustement du ton et la sélection aléatoire. Premièrement, avec la probabilité de $HMCR$ la variable de décision $x_{new}(j)$ est produite par la considération de la mémoire, sinon $x_{new}(j)$ est obtenue par une sélection aléatoire avec la probabilité de $(1-HMCR)$. La considération de mémoire et la sélection aléatoire sont données respectivement par les équations (2.4) et (2.5) :

$$x_{new}(j) = x_a(j), \quad a \in \{1, 2, ..., HMS\} \tag{2.4}$$

$$x_i(j) = LB(j) + (UB(j) - LB(j)) \times rand() \tag{2.5}$$

Deuxièment, si $x_{new}(j)$ est générée par la considération de la mémoire, un adjustement du ton est appliquée avec une probabilité de PAR comme un déplacement vers une valeur voisine dans une plage de valeurs possibles à l'aide de l'équation suivante :

$$x_{new}(j) = x_{new}(j) \bar{+} BW \times rand() \tag{2.6}$$

où BW est la distance-bande passante, et rand() un nombre aléatoire généré uniformement dans l'intervalle $[0, 1]$.

Etape 3 Si la nouvelle harmonie est meilleure que la mauvaise harmonie de HM, la nouvelle harmonie est ajoutée à HM et la mauvaise harmonie est supprimée de HM.

Etape 4 Si le critère d'arrêt n'est pas satisfait, retourner à l'étape 2.

En raison de ses excellentes caractéristiques telles que la mise en œuvre simple, la flexibilité et la grande performance [41], HS a attiré de plus en plus l'attention de chercheurs, et plusieurs variantes ont été proposées pour l'adapter aux différents problèmes, et aussi pour améliorer sa performance. De là, les applications de HS ont couvert plusieurs domaines, tels que l'industrie, l'ingénierie, la robotique, la santé et l'énergie [69].

Dans ce contexte plusieurs auteurs ont développé des algorithmes HS pour résoudre des problèmes d'ordonnancement. Ling Wang et al. [55] ont introduit un algorithme de recherche d'harmonie hybride pour le problème de flow shop de permutation avec blocage, où un schéma d'initialisation basé sur l'heuristique Nawaz-Enscore-Ham (NEH) a été présenté pour construire une mémoire d'harmonie avec un certain niveau de qualité et diversité, aussi une nouvelle règle d'ajustement a été développée pour bien hériter les bonnes structures des meilleures harmonies trouvées durant les improvisations. Gao et al.[56] ont proposé un algorithme de recherche d'harmonie pour résoudre le problème de flow shop sans attente avec l'objectif de minimiser le temps total d'exécution (Makespan). D'abord, par un mécanisme de regroupement dynamique, la mémoired'harmonie est divisée en plusieurs groupes pour partager réciproquement l'information. Ensuite, une recherche locale à voisinage variable est

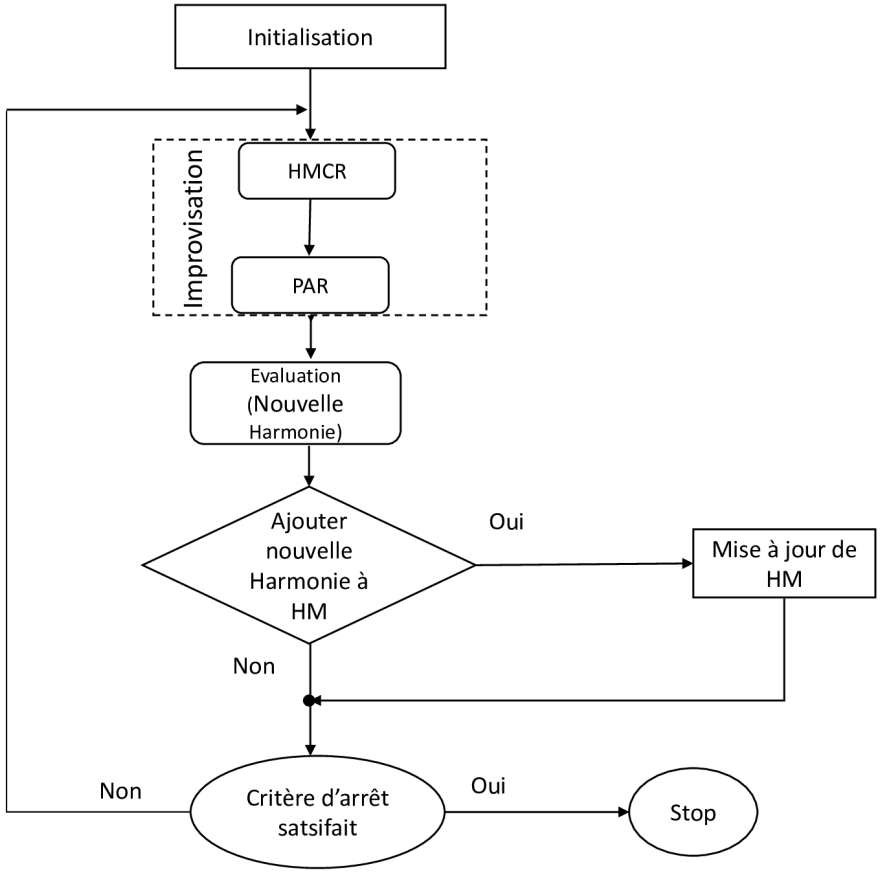


figure. 2.3 – Organigramme de HS

intégrée dans la recherche d'harmonie pour garder l'équilibre entre l'exploration globale et l'exploration locale. Ils ont testé leur approche sur des jeux-tests de la littérature.

Les algorithmes génétiques

Les algorithmes génétiques (Genetic algorithms, GA) sont des techniques de recherche basées sur les mécanismes de l'évolution naturelle et de la génétique. Ils ont été introduits en 1975 par Holland [51], tandis que leur application à la résolution des problèmes d'optimisation n'a été formalisée qu'en 1989 par goldberg [54].

En imitant les phénomènes de l'évolution biologique dans lesquels les populations évoluent

selon les concepts de la sélection naturelle et la capacité d'adaptation des individus, les algorithmes génétiques visent à faire évoluer une population de solutions progressivement sur plusieurs générations.

De ce fait, les algorithmes GA consistent à générer un ensemble de solutions appelées aussi individus ou chromosomes d'une manière aléatoire pour former la population initiale. Notons que chaque chromosome est composé de plusieurs gènes dont chacun représente un élémént de la solution (selon le codage utilisé). Chaque solution de la population doit être évaluée pour déterminer sa performance à l'aide d'une fonction d'évaluation appelée fitness. Dans l'étape suivante, un mécanisme de sélection est utilisé pour choisir les parents les plus adaptés de la population. Ensuite, un opérateur de croisement est appliqué aux parents choisis dans le but de produire de nouveaux individus (enfants). L'opérateur de mutation permet par la suite de modifier le code génétique des nouveaux individus. Les opérateurs de sélection, croisement et mutation sont appliqués selon des probabilités. Enfin, une méthode de remplacement est appliquée pour choisir les individus de la génération suivante.
Ce processus se répète pendant chaque génération de l'algorithme. La structure générale d'un algorithme génétique est illustrée par l'algorithme 5.

Algorithme 5 Structure générale d'un algorithme génétique

Début
  Initialiser les paramètres de l'algorithme.
  Générer la population initiale.
  Tant que Critère d'arrêt non satisfait
    Évaluer les individus
    Choisir les parents en utilisant un mécanisme de sélection.
    Appliquer l'opérateur de croisement avec une probabilité de croisement.
    Appliquer l'opérateur de mutation avec une probabilité de mutation .
    Remplacement de la population.
  Fin Tant que
  Retourner le meilleur individu de la population.
Fin

Nous détaillons dans la partie suivante les différents opérateurs génétiques, et surtout les opérateurs de croisement et de mutation qui seront utilisés par la suite dans les méthodes de résolution proposées dans cette thèse.

-Opérateurs de sélection
La sélection permet de déterminer les individus les plus privilégiés (en fonction de leur valeur

de fitness) de se présenter aux opérateurs de croisement et de mutation. Il existe plusieurs types d'opérateurs de sélection, notamment la sélection par la roulette où on associe à chaque chromosome une probabilté de sélection proportionnelle à sa valeur de fitness. Donc plus les individus sont meilleurs (selon leur valeur de fitness) plus leur chance d'être sélectionnés est élevée. Nous citons aussi la sélection par rang qui permet de classer la population par fitness, chaque individu de la population se voit accorder un rang de telle façon que plus l'individu est bon, plus son rang est élevé. La sélection par rang est similaire à la sélection par roulette, la différence est que les proportions sont en relation avec le rang plutôt qu'avec la fonction d'évaluation [105].

-Opérateurs de croisement

L'opérateur de croisement consiste à produire un ou plusieurs individus (enfants) à partir de deux individus parents sélectionnés. Il permet aux individus enfants d'hériter des caractéristiques de leurs parents. Le croisement est appliqué avec une probabilité P_c appelée probabilité de croisement.

Dans la littérature, plusieurs opérateurs de croisement ont été developpés. Ils dépendent essentiellement du codage de solutions et de la nature du problème traité.

Parmi les opérateurs de croisement les plus connus, nous citons à titre d'exemple le croisement à un point et le croisement à deux point.

- Le croisement à un point cet opérateur consiste à choisir une position arbitraire α ($1 \leq \alpha \leq n$) dans les deux chromosomes parents P_1 et P_2 de taille n, ce qui permet de les fragmenter en deux parties à la même position. Ensuite, les éléments de P_1 et P_2 se trouvant avant le point de croisement sont copiés respectivement dans la structure des enfants $E1$ et $E2$. Les éléments manquants de $E1$ et $E2$ sont remplis respectivment à partir de P_2 et P_1.
- Le croisement à deux point consiste à choisir deux positions fixes sur les parents, ensuite échanger la partie entre les deux positions pour créer les enfants.

-Opérateurs de mutation

L'opérateur de mutation est un opérateur unaire qui permet de modifier un ou plusieurs gènes d'un individu choisi. Il est appliqué avec une faible probabilité P_m appelée probabilité de mutation. Cet opérateur favorise l'exploration efficace de l'espace de solutions. Il existe dans la littérature plusieurs types d'opérateurs de mutation, comme par exemple l'opérateur d'insertion qui choisit d'une manière aléatoire un gène et l'insère dans une autre position du chromosome, et l'opérateur d'échange (swap mutation), qui consiste à choisir deux gènes au hasard et à les permuter.

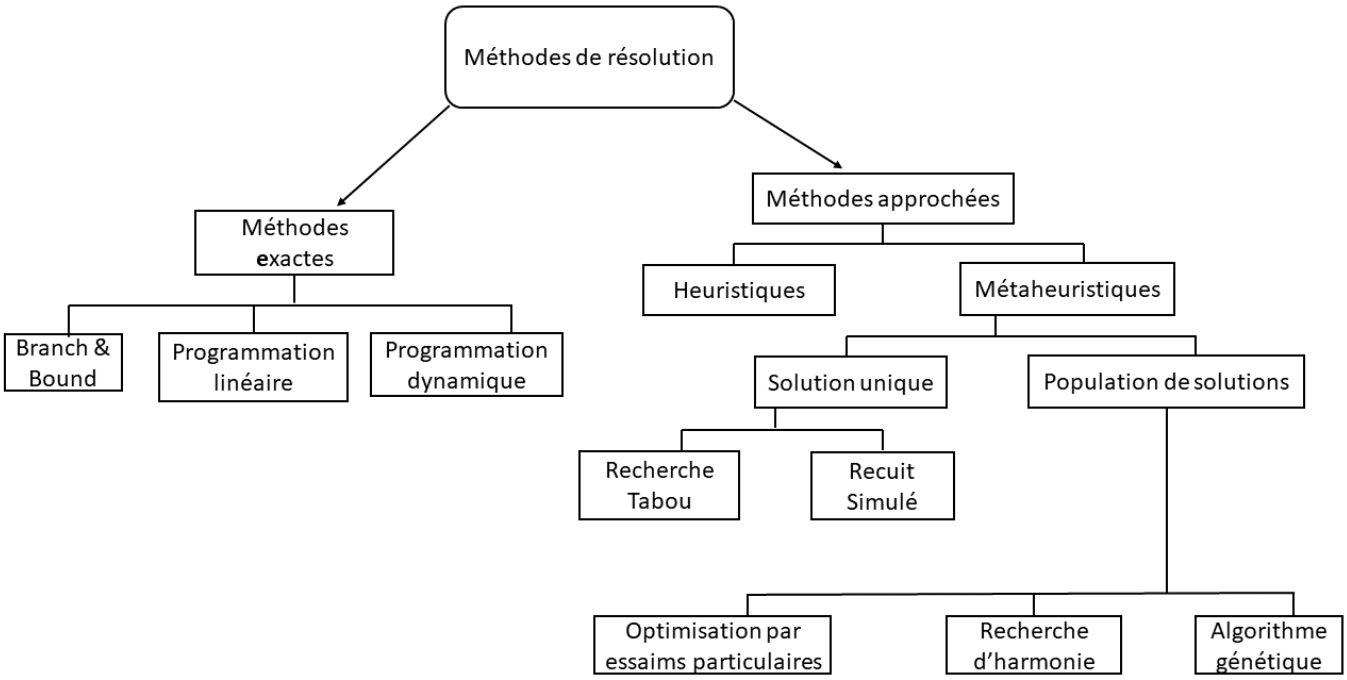


figure. 2.4 – Classification des méthodes d'optimisation

2.3 Méthodes hybrides

Comparativement aux méthodes exactes, les métaheuristiques permettent de trouver des solutions de bonne qualité dans des temps de calcul raisonnables, mais il est certain que chaque méthode présente des avantages et des inconvénients. Nous avons vu précédemment que les métaheuristiques reposent sur l'utilisation des concepts d'intensification et de diversification, et qu'un équilibre entre les deux concepts doit être maintenu durant la recherche. Pourtant, une méthode est rarement aussi efficace pour exploiter l'espace de recherche que pour l'explorer. La tendance actuelle consiste à combiner plusieurs méthodes dans le but de concevoir des méthodes hybrides qui profitent de la complémentarité de deux ou plusieurs approches. La méthode hybride est une méthode de recherche constituée d'au moins deux méthodes de recherche [72]. Le rôle de l'hybridation réside donc dans l'exploitation de deux ou plusieurs principes de recherche pour tirer profit de leurs points forts, et améliorer ainsi la performance et le comportement des algorithmes proposés.
Les principales formes d'hybridation peuvent être classifiées selon leur architecture en trois catégories [72].

hybridation séquentielle : consiste à exécuter séquentiellement différentes méthodes de recherche de telles manière que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante. C'est une technique d'hybridation simple qui ne nécessite pas de modification des méthodes utilisées.

hybridation parallèle synchrone : consiste à utiliser une méthode de recherche particulière dans un opérateur d'une autre. Cette technique est plus complexe à mettre en oeuvre que la précédente, et il faut tenir compte des fortes interactions entre les méthodes incorporées pour ne pas aboutir à une méthode hybride moins efficace.

hybridation parallèle asynchrone : consiste à faire évoluer parallèlement différentes méthodes de recherche, ce qui permet une bonne coopération des méthodes de recherche utilisées au travers d'un coordinateur. Cette technique offre l'avantage d'exploiter les résultats de plusieurs algorithmes de recherche qui travaillent ensemble et s'échangent des informations.

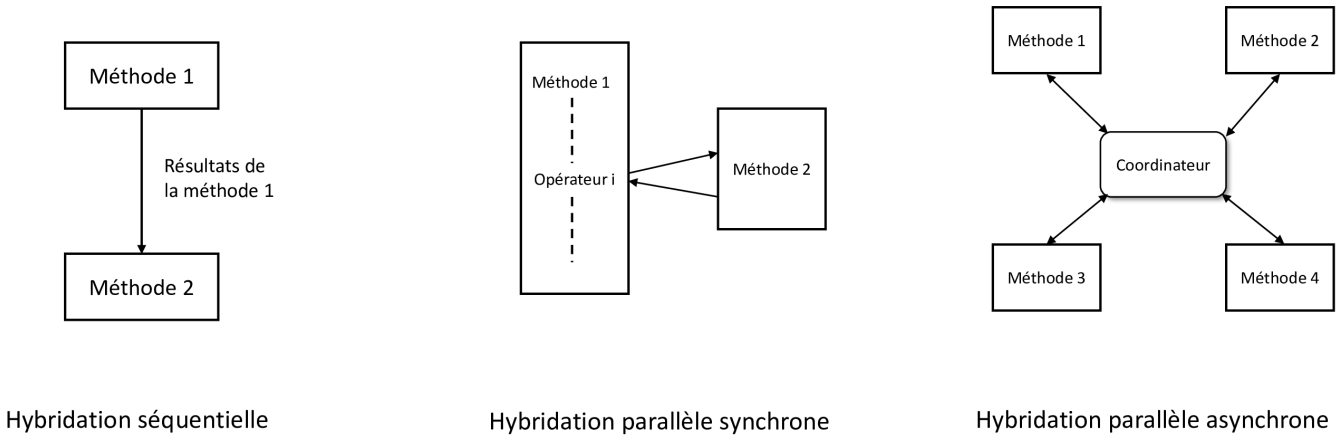


figure. 2.5 – Techniques d'hybridation

La figure 2.5 illustre ces trois types d'hybridation. Notons qu'il est possible de combiner plusieurs stratégies d'hybridation au sein d'une seule méthode. Dans le cadre de notre tavail, nous nous limitons à cette classification. Mais, il existe d'autres classifications des formes d'hybridation, nous citons à titre d'exemple la taxonomie des métaheuristiques hybrides présentée par Talbi [83], qui comporte deux aspects : une classification hiérarchique qui permet d'identifier la structure de l'hybridation, et une classification générale qui spécifie les détails

des algorithmes impliqués dans l'hybridation. Dans leur classification, Puchinger et Raidi [80] ont présenté une classification axée sur l'hybridation de méthodes exactes et approximatives. Cette classification est divisée en deux catégories : une hybridation collaborative et une hybridation intégrative. Il s'agit d'une hybridation collaborative lorsque les algorithmes impliqués échangent de l'information de façon séquentielle, parallèle ou entrelacée. Cependant, les algorithmes à hybridation intégrative font en sorte qu'une technique est une composante incorporée à une autre technique.

2.4 Conclusion

Ce chapitre a introduit les méthodes utilisées pour résoudre des problèmes d'ordonnancement d'ateliers. Ces méthodes peuvent se diviser en deux catégories : Méthodes exactes et méthodes approchées. Ces dernières sont également réparties en méthodes heuristiques et métaheuristiques. Nous avons présenté des différentes méthodes d'optimisation dans ce chapitre, mais nous avons détaillé principalement les méthodes utilisées dans ce travail. Nous avons abordé aussi la notion des méthodes hybrides en présentant les formes principales de l'hybridation.

Chapitre 3

Optimisation par essaims particulaires pour la résolution de problèmes de type flow shop hybride

3.1 Introduction

Dans ce chapitre, nous proposons un algorithme à base de l'optimisation par essaims particulaires pour résoudre le problème du flow shop hybride (HFS). Puisque l'optimisation par essaims particulaires a été conçue originellement pour résoudre des problèmes de l'optimisation continue, alors l'algorithme proposé est adapté pour résoudre le problème HFS en modifiant la stratégie de la mise à jour des particules. De plus, une recherche locale est incorporée dans la mise à jour de chaque particule pour renforcer l'intensification de la recherche.

Nous rappelons que le problème HFS est défini par un ensemble $J = \{J_1, J_2, ..., J_n\}$ de n jobs à ordonnancer dans le même ordre sur s étages en série. Chaque étage k ($k \in \{1, 2, .., s\}$) est constitué de m_k machines parallèles identiques. Chaque job j comporte un ensemble d'opérations O_{ji} ($i \in \{1, 2, ..s\}$) où O_{ji} est l'opération du job j sur l'étage i. Les jobs traversent les différents étages de façon que chaque job doit être exécuté par une seule machine à chaque étage. L'objectif est de trouver la séquence de jobs qui minimise la durée totale d'ordonnancement (le makespan).

Ce chapitre est structuré comme suit : nous présentons dans la deuxième section (3.2) une modélisation mathématique qui décrit les différents paramètres, variables et contraintes de

notre problème. Ensuite, les différentes étapes de l'algorithme proposé sont détaillées dans la section (3.3). Les résultats expérimentaux et les comparaisons sont présentés à la section (3.5).

3.2 Modélisation

La modélisation constitue une étape importante de la résolution d'un problème, il permet de déterminer d'une manière simplifiée les idées et les hypothèses fondamentales du problème. Généralement, on peut distinguer deux méthodes pour modéliser un problème d'ordonnancement :

- La modélisation mathématique qui aide à formuler l'ensemble de données, la fonction objectif, les variables et les contraintes du problème sous forme des équations et/ou des inéquations mathématiques.

- La modélisation graphique qui utilise des outils graphique pour visualiser les différents composants du problème.

Par la suite, nous proposons une modélisations mathématique du problème du flow shop hybride sous forme d'un programmes mathématique en nombres entiers.

3.2.1 Le modèle mathématique

Dans cette partie, nous présentons un modèle mathématique en nombres entiers pour le problème du flow shop hybride avec l'objectif de minimiser le makespan. Le modèle est basé sur [28]. Nous commençons par introduire les paramètres et les variables utilisés pour modéliser le problème.

Les paramètres

n **:** Nombre de jobs

s **:** le nombre d'étages.

M_i **:** le nombre de machines parallèles à l'étage i.

b_{ji} **:** date de début du job j à l'étage i.

C_{ji} **:** date de fin de traitement du job j à l'étage i.

p_{ji} **:** temps d'exéction de job j à l'étage i.

O_{ji} **:** Opération du job j l'étage i

j **:** indice de job, où $j = 1, ..., n$.

i **:** indice d'étage $i = 1, ..., s$.

k **:** indice de machine, où $k = 1, ..., M_i$.

L **:** une constante suffisamment grande.

Les variables

$\mathrm{C_{max}} = \max_{j \in \{1,...,n\}} C_j$: date de fin d'exécution de tous les jobs.

b_{ji} : date de début de traitement du job j à l'étage i.

$\mathrm{X_{jik}} = \begin{cases} 1 & \text{Si job j est affecté à la machine k à l'étage i} \\ 0 & \text{Sinon} \end{cases}$

$\mathrm{Y_{jrik}} = \begin{cases} 1 & \text{Si job j précède job r à la machine k à l'étage i} \\ 0 & \text{Sinon} \end{cases}$

Le modèle mathématique

En utilisant les notations précédentes, le problème du flow shop hybride peut être présenté sous forme d'un programme mathématique en nombres entiers de la façon suivante :

$$\text{Minimiser} \quad C_{max}$$

sous les contraintes

$$C_{max} \geq C_j \qquad j = 1, ..., n \tag{3.1}$$

$$C_j = C_{js} \qquad j = 1, ..., n \tag{3.2}$$

$$C_{ji} = b_{ji} + p_{ji} \qquad j = 1, ..., n\,;\; i = 1, ..., s \tag{3.3}$$

$$\sum_{j=1}^{n} X_{jik} = 1 \qquad k = 1, ..., m_i\,;\; i = 1, ..., s \tag{3.4}$$

$$\sum_{k=1}^{m_i} X_{jik} = 1 \qquad j = 1, ..., n\,;\; i = 1, ..., s \tag{3.5}$$

$$C_{ji} \leq b_{j(i+1)} \qquad j = 1, ..., n\,;\; i = 1, ..., s-1 \tag{3.6}$$

$$C_{xi} \leq b_{yi} + L(1 - Y_{xyik}) \qquad \text{pour tout couple de jobs } (x, y)\,;\; i = 1, ..., s\,;\; k = 1, ..., m_i \tag{3.7}$$

$$Y_{xyik} + Y_{yxik} \leq 1 \qquad \text{pour tout couple de jobs } (x, y)\,;\; i = 1, ..., s\,;\; k = 1, ..., m_i \tag{3.8}$$

$$X_{jik} \quad \text{et} \quad Y_{jrik} \quad \text{sont des variables binaires} \tag{3.9}$$

$$C_{max} \geq 0 \quad ; \quad b_{ji} \geq 0 \quad j = 1, ..., n\,;\; i = 1, ..., s \tag{3.10}$$

La contrainte (3.1) indique que C_{max} doit être supérieur ou égal à la date de fin de traitement de tous les jobs
La contrainte (3.2) indique que la date de fin de traitement d'un job i est égale à sa date de fin au dernier étage.
La contrainte (3.3) définit la date de fin de traitement du job j à l'étage i.
Les contraintes (3.4) et (3.5) indiquent que chaque job doit être traité exactement par une machine à chaque étage, et qu'une machine ne peut traiter qu'un seul job à la fois.
La contrainte (3.6) exprime les précédences entre les opérations d'un même job, elle assure que le traitement du job j ne peut débuter à l'étage $i+1$ (c.à.d le traitement de l'opération $O_{j(i+1)}$) que lorsque son traitement à l'étage i (c.à.d le traitement de l'opération O_{ji}) est terminé.
Les contraintes (3.7) et (3.8) expriment les précédences entre les jobs. La contrainte (3.7) assure que la date de début de traitement d'un job affecté à une machine doit être supérieure à la date de fin de traitement du job qui le précède sur la même machine.
Pour chaque machine et chaque étage, la contrainte (3.8) décrit la relation prédécesseur/successeur entre deux jobs, il exige qu'un job ne peut pas être un prédécesseur et un successeur à la fois.
La contrainte (3.9) déclare que les variables X_{jik} et Y_{jrik} sont binaires.
La contrainte (3.10) indique que C_{max} et b_{ij} sont toujours positifs.

3.2.2 Diagramme de Gantt

Le diagramme de Gantt est une représentation graphique très utilisée en ordonnancement et en gestion de projets qui permet de visualiser dans le temps les différentes opérations à réaliser, le concept a été diffusé par l'ingénieur américain Henry Laurence Gantt vers 1910. Ce diagramme offre une organisation des ateliers, il renseigne sur la durée d'une tâche, et sur ses dates de début et de fin. Cet outil est couramment utilisé pour représenter la solution d'un problème d'ordonnancement.
Pour représenter un ordonnancement, le diagramme de Gantt se compose de deux axes orthogonaux : l'axe horizontal représente les unités de temps et l'axe vertical correspond aux différentes machines. Ainsi, une tâche est représentée par une barre horizontale dont la longueur indique la durée d'exécution de la tâche.

Pour illustrer cette représentaion, nous considérons un exemple d'un atelier flow shop hybride qui contient deux étages, où chacun d'eux contient deux machines parallèles. Il s'agit d'ordonnancer 3 jobs, le tableau 3.1 indique les temps de traiement (processing time) de chaque job à chaque étage :

Table. 3.1 – Temps de traitement pour un exemple de 3 jobs et 2 étages

job	Processing time	
	étage 1	étage 2
J_1	20	32
J_2	11	6
J_3	39	16

Nous utilisons le diagramme de gantt pour représenter une solution réalisable du problème flow shop hybride considéré. En plus de dates de début et de fin et les temps d'exécution des opérations, ce diagramme indique l'occupation des machines ainsi que leur temps d'inactivité à chaque étage (voir figure 3.1).

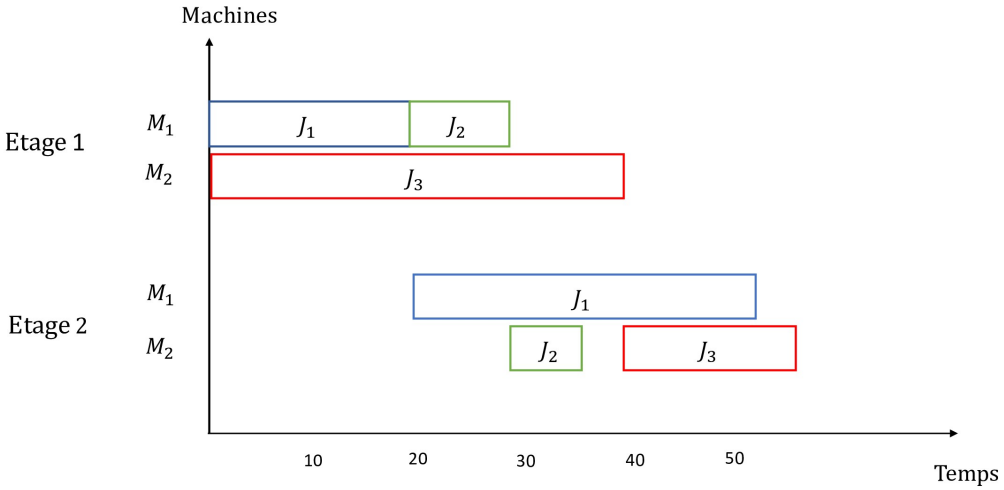


figure. 3.1 – Le diagramme de gantt d'une solution d'un problème HFS

3.3 Les étapes de l'algorithme DPSO

3.3.1 Introduction

La méthode d'optimisation par essaims particulaires est l'une des métaheuristiques les plus utilisées pour trouver de bonnes solutions dans des temps de calcul raisonnables aux problèmes complexes. La méthode a été appliquée sur plusieurs types de problèmes d'optimisation, et elle a prouvé son efficacité. Nous rappelons que l'algorithme d'optimisation par essaims particulaires est présenté en détail dans le deuxième chapitre de cette thèse.

L'optimisation par essaims particulaires fait partie des méthodes évolutionnaires inspirées de la biologie, c'est une technique itérative qui fait évoluer une population de solutions (essaim) dans l'espace de recherche dans le but de se rapprocher de l'optimum. Les solutions sont appelées particules où chaque particule est caractérisée par une vitesse, une position et la valeur de la fonction objectif de cette position.
L'idée principale de l'algorithme consiste à déplacer chaque particule selon sa vitesse, sa mémoires personnelle (la meilleure position personnelle), et la mémoire collective de l'essaim (la meilleure position atteinte par tout l'essaim). Si la nouvelle particule est meilleure que la particule courante, alors sa mémoire personnelle est mise à jour. De même, si la mémoire collective de l'essaim est améliorée durant la recherche, alors elle est mise à jour. Ce processus

se répète tant que le critère d'arrêt n'est pas atteint.

Puisque l'algorithme d'optimisation par essaims particulaires a été conçu pour résoudre les problèmes d'optimisation continue, alors nous proposons une approche discrète de l'algorithme notée DPSO afin de l'adapter à notre problème [78].
Nous détaillons à la suite les étapes de l'algorithme d'optimisation par essaim particulaire proposé (DPSO).

3.3.2 Représentation des solutions

Nous rappelons que notre problème consiste à trouver la séquence de jobs qui minimise le makespan. Pour cette raison, nous avons choisi une représentation par permutation. Cette dernière permet de représenter une solution (une particule) sous forme d'une permutation de n éléments où chaque élément correspond à un job. La position de chaque job dans la permutation désigne son ordre de passage sur le premier étage. Le tableau 3.2 donne une illustration de cette représentation pour une permutation de 10 jobs où le j^{eme} élément dans la permutation indique le job situé à la position j.

Table. 3.2 – Représentation d'une solution ($n = 10$)

Dimension j	1	2	3	4	5	6	7	8	9	10
Permutation de job	J_9	J_5	J_4	J_{10}	J_6	J_2	J_8	J_3	J_1	J_7

3.3.3 Essaim initial

La génération de l'essaim initial est une étape importante qui permet de générer une population initiale de particules qui sert ensuite de base pour créer les générations futures.
Le principe d'initialisation de cet essaim en PSO est le suivant : choisir aléatoirement les positions des particules dans l'espace de recherche. Alors, nous choisissons d'utiliser un essaim de particules générées d'une manière aléatoire, cet essaim forme une matrice qui contient N particules pouvant être générées en utilisant la fonction randperm sous Matlab.

3.3.4 Mise à jour de la position et de la vitesse

Nous détaillons dans cette partie l'étape qui vise à faire évoluer l'essaim. Nous utilisons des opérateurs d'évolution empruntés aux algorithmes génétiques : croisement et mutation pour

mettre à jour la position de la particule dans l'algorithme proposé.

Rappelons que l'état de particule dans l'espace de recherche est déterminé par sa position X_i^t et sa vitesse V_i^t. On note, P_i^t la meilleure position personnelle atteinte par la particule jusqu'à l'itération t, et G^t la meilleure position globale, qui représente la meilleure position trouvée par toutes les particules de l'essaim jusqu'à l'itération t.

En se basant sur le modèle présenté par Pan et al.[60], la position de la particule à l'itération t peut être représentée par l'équation :

$$X_i^t = c_2 \otimes F_3(c_1 \otimes F_2(\omega \otimes F_1(X_i^{t-1}), P_i^{t-1}), G^{t-1}) \tag{3.11}$$

Cette équation comporte trois composantes :

- La première composante $V_i^t = \omega \otimes \mathrm{F_1(X_i^{t-1})}$ représente la vitesse de la particule, où F_1 désigne l'opérateur de mutation appliqué avec une probabilité de ω. En d'autres termes, un nombre aléatoire uniforme r_1 est généré entre 0 et 1. Si r_1 est inférieur à ω alors l'opérateur de mutation est appliqué pour générer une nouvelle vitesse. Si non, on garde la vitesse courante.

- La deuxième composante $\mathrm{B_i^t} = c_1 \otimes \mathrm{F_2(V_i^t, P_i^{t-1})}$ est la composante cognitive, qui représente l'expérience personnelle de la particule, F_2 désigne l'opérateur de croisement appliqué avec une probabilité de c_1.

- La troisième composante $\mathrm{X_i^t} = c_2 \otimes \mathrm{F_3(B_i^t, G^{t-1})}$ correspond à la partie sociale de la particule, où F_3 est l'opérateur de croisement appliqué avec une probabilité de c_2.

De plus, nous ajoutons à l'équation (3.11) une nouvelle composante représentant l'application d'une stratégie de recherche locale. Afin de renforcer l'intensification dans les régions explorés de l'espace de recherche, la nouvelle composante essaye de trouver la meilleure particule dans un voisinage. Notons que cette recherche locale est appliquée avec une probabilité de c_3.

Donc, la position de la particule est représentée par l'équation suivante :

$$X_i^t = c_3 \otimes F_4(c_2 \otimes F_3(c_1 \otimes F_2(\omega \otimes F_1(X_i^{t-1}), P_i^{t-1}), G^{t-1})) \tag{3.12}$$

Les opérateurs de mutation et de croisement, ainsi que la stratégie locale utilisée seront détaillés dans les sections suivantes.

3.3.5 Opérateur de mutation

Au début de la phase de mise à jour de particules, un opérateur de mutation est appliqué au sein de l'essaim avec une certaine probabilité. D'une manière générale, la mutation est un opérateur unaire qui permet de créer une nouvelle solution à partir d'une solution existante en modifiant certaines de ces caractéristiques.

Le fait de définir la solution comme une permutation nous permet d'utiliser une mutation basée sur une structure de voisinage discrète associée à des mouvements d'inversion pour passer d'une solution à une autre. La mutation par inversion revient à permuter les jobs situés entre deux positions choisies d'une manière aléatoire. La procédure de la mutation par inversion est présentée par l'algorithme 6.

Algorithme 6 Mutation d'inversion

Données : J une séquence de jobs de taille n, ω probabilité de mutation, $r \in [0, 1]$,
Résultat : J' la séquence de jobs mutée

Début
  Générer aléatoirement un nombre $r \in [0, 1]$
  if $r <= \omega$ **then**
    Choisir aléatoirement deux positions dans la permutation des jobs.
    Inverser la section entre les deux positions choisies.
  end if
  Mettre à jour la séquence mutée J'
Fin

Nous considérons une permutation de jobs de taille $n = 8$, le graphe 3.2 illustre un exemple de mutation par inversion où les jobs situés entre $J1$ et $J2$ sont inversés.

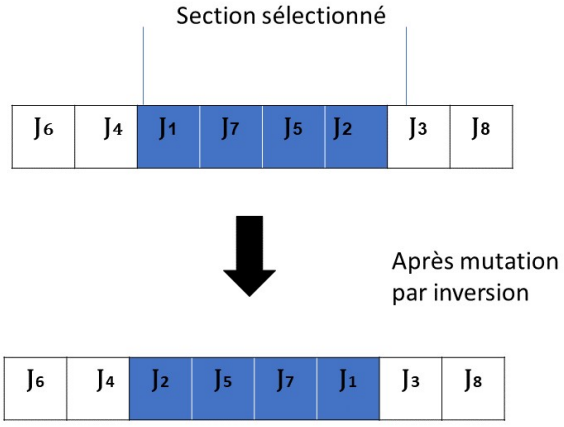


figure. 3.2 – Mutation par inversion

3.3.6 Opérateurs de croisement

L'opérateur de croisement consiste à produire un ou plusieurs individus (descendants) à partir de deux individus appelés parents, c'est un opérateur dérivé de la génétique et de l'évolution naturelle. Deux méthodes de croisement ont été utilisées dans notre algorithme, le croisement RCC (Right corner crossover) et le croisement uniforme.

- **Le croisement RCC** (Right Corner Crossover) [60] commence par choisir au hasard deux positions dans le premier parent. Ensuite, le bloc déterminé par ces deux positions est déplacé vers le coin droit du descendant (nouvelle solution). Les jobs restants sont ajoutés à partir du deuxième parent. Ce croisement est utilisé dans notre méthode pour générer la deuxième composante B_i^t (voir section 3.3.4) qui désigne la partie cognitive de la particule. Notons que ce type de croisement peut produire un descendant différent même à partir de deux parents identiques.

- **Le croisement uniforme** consiste à construire les nouveaux individus en leur affectant aléatoirement une combinaison de caractéristiques existantes dans les parents. Cet opérateur est appliqué après le croisement RCC pour produire la troisième composante définissant la partie sociale de la particule.

Les différentes étapes de ces deux procédures sont données dans les algorithmes 7 et 8.

Algorithme 7 Croisement RCC

Données : P_1 et P_2 individus parents (séquences de jobs) de taille n, c_2 probabilité de croisement, $r \in [0, 1]$,
Résultat : E individu enfant (séquence de jobs)

Début
  Générer un nombre aléatoire $r \in [0, 1]$.
  if $r <= c_2$ **then**
    Choisir aléatoirement deux position dans la séquence du premier parent.
    Déplacer Le bloc déterminé par les deux positions vers le coin droit de l'enfant.
    Ajouter Les jobs restants à partir du deuxième parent en ne reprenant que les jobs non encore déplacés.
  end if
  Mettre à jour l'enfant E.

Fin

Algorithme 8 Croisement uniforme

Données :P_1 et P_2 individus parents (séquences de jobs) de taille n, c_2 probabilité de croisement, $r \in [0, 1]$,
Résultat : E individu enfant(séquence de jobs)

Début
  Générer un nombre aléatoire $r \in [0, 1]$
  if $r <= c_2$ **then**
    Générer aléatoirement un masque binaire de taille n
    Transmettre les éléments du premier parent à l'enfant si la valeur du bit du masque est égale à 1, et les éléments du deuxième parent si la valeur du bit du masque est égale à 0, tout en respectant leur ordre dans le parent et en ne prenant que les éléments non encore transmis.
  end if
  Mettre à jour l'enfant E.

Fin

On considère le même exemple que précédemment, le croisement RCC est illustré par la figure 3.3.

Le block entre les deux jobs J_2 et J_1 dans le premier parent P_1 est déplacé vers le côté droit de l'enfant E. À partir de P_2, les jobs restants sont ajoutés, en considérant seulement les jobs non déplacés.

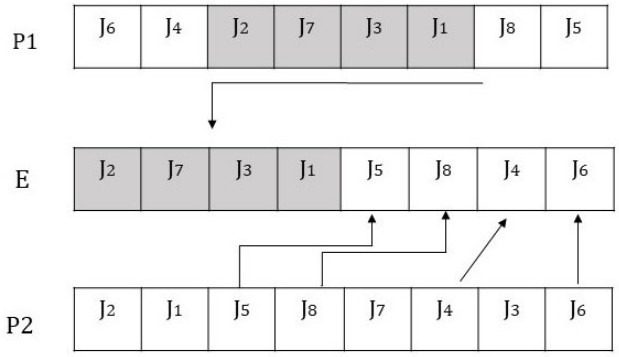


figure. 3.3 – Illustration du croisement RCC

Un exemple de croisement uniforme pour un problème à 8 jobs est représenté par la figure 3.4.
La valeur du premier bit du masque binaire est égale à 1, alors le job J_6 du premier parent P_1 est transmis à l'enfant E. Ensuite la valeur du deuxième bit du masque est égale à 0, alors le job J_2 de P_2 est transmis à l'enfant E. Cette procédure se continue, mais en prenant en compte l'ordre des jobs dans chaque parent.

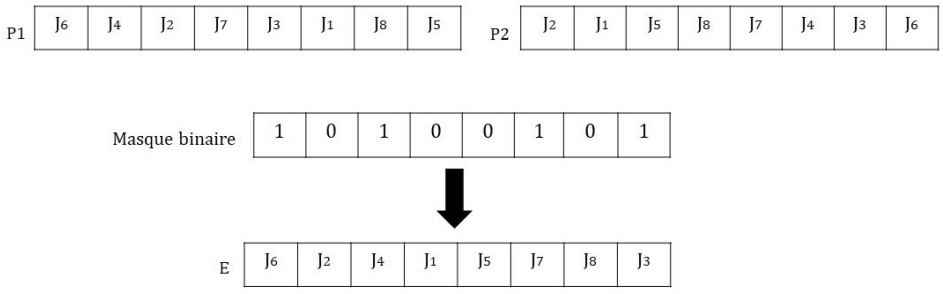


figure. 3.4 – Illustration du croisement uniforme

3.3.7 Recherche locale

La dernière étape du processus de mise à jour des particules réside dans l'application d'une approche de recherche locale. L'idée de base est de chercher des améliorations de la solution courante, en examinant des solutions voisines définies à partir de deux structures de voisinage différentes.

Le voisinage V d'une solution X est défini par une structure de voisinage qui détermine les modifications à appliquer à la solution courante pour créer son voisin. L'application de ces modifications appelées aussi mouvements permet de produire un voisin $X' \in V$.
Nous proposons deux structures de voisinage adaptées à notre problème :

- **Le voisinge par insertion** (insertion neighborhood) associé à des mouvements d'insertion.
- **Le voisinage par échange** (interchange neighborhood) associé à des mouvements d'échange.

Plus formellement, la recherche locale utilisée s'appuie sur deux types de mouvements : des mouvements d'insertion (insert moves) qui consistent à supprimer le travail actuellement en position i et l'insérer dans une autre position j, et des mouvements d'échange (interchange moves) qui consistent à échanger deux jobs choisis aléatoirement dans la séquence de jobs pour obtenir une nouvelle séquence. Ces deux types de mouvements sont illustrés dans les algorithmes 9 et 10.

Algorithme 9 Mouvement d'insertion

Données : J une séquence de jobs de taille n
Résultat : J' la séquence de jobs résultante

Début
Choisir aléatoirement deux positions i et j dans la permutation de jobs.
Supprimer le job en position i, et l'insérer dans la position j.
Décaler à la droite les autres jobs.
Mettre à jour la séquence J'
Fin

On considère la permutation de jobs présentée dans la figure 3.2, des exemples de mouvements d'insertion et d'échange sont donnés dans les figures 3.5 et 3.6 respectivement.

Algorithme 10 Mouvement d'échange

Données :J une séquence de jobs de taille n
Résultat : J' la séquence de jobs résultante
Début
  Choisir aléatoirement deux positions i et j dans la permutation de jobs.
  Échanger les deux job.
  Mettre à jour la séquence J'
Fin

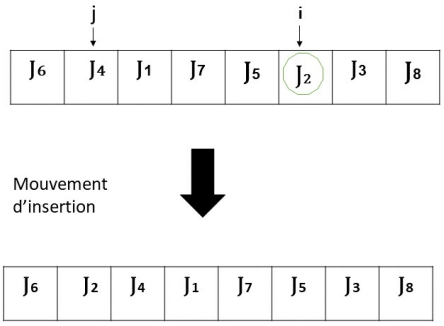


figure. 3.5 – mouvement d'insertion

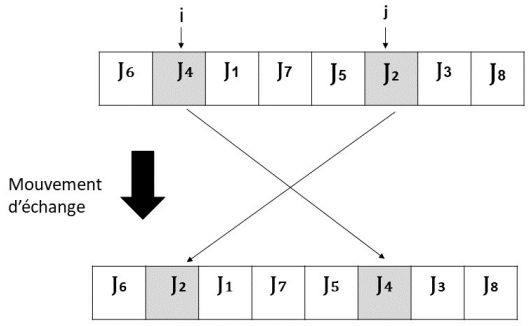


figure. 3.6 – mouvement d'échange

La procédure de la recherche locale proposée fonctionne en explorant les deux voisinages consécutivement, en commençant d'abord par le voisinage par insertion, puis le voisinage par échange.
Cette approche locale vise à assurer l'intensification dans les régions atteintes de l'espace de recherche et à améliorer la convergence de l'algorithme DPSO. Elle combine deux structures de voisinage différentes ce qui permet une exploration efficace de l'espace de recherche. Le critère d'arrêt est un temps fixé à $0.01ms$.
Le pseudo code de la recherche locale proposé est donné dans l'algorithme 8. Notons que "Insert-LS" désigne la recherche locale basée sur le voisinage d'insertion, et "Interchange-LS" représente la recherche locale basée sur le voisinage par échange.

Algorithme 11 Recherche locale

Données : X_0 **Solution initiale**
Résultat : X **Meilleure solution trouvée**
Début
  $X \leftarrow X_0$
  Tant que Critère d'arrêt n'est pas satisfait
    $X' = Insert - LS(X)$
    $X' = Interchange - LS(X')$
    if X' **est meilleure que** X **then**
      $X \leftarrow X'$
    end if
  Fin Tant que
  Retourner X
Fin

3.3.8 Fonction objectif

La fonction objectif consiste à mesurer la qualité des solutions fournies par la procédure de résolution. L'objectif considéré pour notre problème est la minimisation du temps total d'exécution ou le makespan.
Pour calculer le makespan, nous partons de l'idée que la minimisation de ce dernier revient à minimiser les temps d'exécution partiels. Nous avons utilisé une technique basée sur la combinaison de plusieurs règles de priorité. Cette technique utilise trois règles de priorité (priority dispatching rules) : $FIFO$ (First In First Out), SPT (Smallest Processing Time) et LPT (Longuest Processing Time). La description de ces règles est donnée dans le tableau 3.3.

Table. 3.3 – Description de FIFO, SPT et LPT

Règle de priorité	Description
$FIFO$	$\min C_{j(i-1)}$
SPT	$\min p_{ji}$
LPT	$\max p_{ji}$

La méthode proposée combine la liste classique dans laquelle les tâches sont organisées selon la règle $FIFO$, et une liste modifiée utilisant les règles FIFO, FIFO+SPT et FIFO+LPT en parallèle, pour déterminer la séquence de jobs à ordonnancer à chaque étage à partir du 2^{eme} étage.

À chaque étage, la liste modifiée forme des sous listes de tâches suivant les règles $FIFO$, $FIFO + SPT$ et $FIFO + LPT$ respectivement. Les dates de fin des tâches sont calculées, et la sous liste qui minimise la date de fin partielle sur l'étage considéré est adoptée.

Finalement, la plus petite valeur du makespan fournie par ces deux listes est retenue.

Notons que l'affectation des jobs sur les machines s'effectue à l'aide de la règle FAM (First Available Machine), qui choisit la première machine libre parmi les machines disponibles (la machine qui se libère le plus tôt).

Remarque 3.3.1. *Les règles de priorité combinées $FIFO+SPT$ et $FIFO+LPT$ désignent que plusieurs caractéristiques du job sont considérées simultanément. Par exemple, $FIFO + SPT$ s'obtient à partir d'une combinaison linéaire du temps d'achèvement du job et le temps de traitement de sa prochaine opération.*

3.3.9 critère d'arrêt

Le critère d'arrêt est un paramètre important pour une métaheuristique. Il est nécessaire pour mettre fin à l'exécution de l'algorithme.
Il est défini généralement par un certain temps de calcul à ne pas dépasser ou un nombre maximal d'itération à réaliser.
Dans notre étude, l'algorithme proposé est limité par un temps de calcul maximal de 1600 secondes ou jusqu'à ce que la borne inférieure (LB) soit atteinte. Dans le cas où la borne inférieure n'a pas été trouvée dans le temps de calcul limite, la recherche s'est arrêtée et la meilleure solution trouvée est acceptée comme solution finale.

3.4 Benchmarks

Pour tester les performances des approches de résolution, nous utilisons souvent des problèmes types construits par un ou plusieurs auteurs appelés Benchmarks.
Nous avons testé notre algorithme sur un jeu de 77 instances (Benchmarks) proposées par Carlier et Néron [16]. Ce benchmark est le plus utilisé pour les problèmes d'ordonnancement de type flow shop hybride,(voir par exemple [26,27,28,31,32]).

Il convient de mentionner que les 77 instances de Carlier et Néron sont divisées en 24 instances difficiles et 53 instances faciles [5]. Les problèmes avec une configuration machines a et b et les instances de type j10c10c sont identifiés en tant qu'instances faciles, tandis que les problèmes avec une configuration machine c et d sont regroupés en tant que problèmes difficiles. Ce benchmark présente quatre types de problèmes selon le nombre de travaux et le nombre d'étages :

— Instances avec 10 travaux et 5 étages.
— Instances avec 10 travaux et 10 étages.
— Instances avec 15 travaux et 5 étages.
— Instances avec 15 travaux et 10 étages.

Le nombre de machines varie entre 1 machine et 3 machines par étage. Une configuration de machine est une liste de chiffres représentant le nombre de machines disponibles dans chaque étage, elle joue un rôle essentiel dans l'évaluation de la complexité des problèmes (instances). Il existe quatre configurations de machines a, b, c et d, correspondent à l'étape de goulot d'étranglement (Bottleneck), c'est à dire l'étape de production ayant la plus faible cadence dans un atelier (limitation de la capacité de production).

la signification des lettres de configuration de la machine [11] est la suivante :

a : Il y a une machine à l'étage du milieu (goulot d'étranglement), et trois machines aux autres étages.

b : Il y a une machine au premier étage (goulot d'étranglement), et trois machines aux autres étages.

c : Il y a deux machines à l'étage du milieu (goulot d'étranglement), et trois machines aux autres étages.

d : Il y a trois machines sur tous les étages.

Chaque instance est représentée par trois lettres : j qui désigne le nombre de jobs, c qui

indique le nombre de centres ou étages et (a, b, c ou d) correspondant à la configuration de machines sur les étages. Par exemple, la notation $j10c5b3$ signifie l'instance où il y a 10 travaux, 5 étages (Les lettres j et c sont les abréviations de "job" et de "center" respectivement) et la lettre b définit la configuration de machine, où il y a trois machines à chaque étage sauf le premier étage qui est goulot d'étranglement avec une seule machine.

Pour toutes les instances, les temps de traitement ont été générés aléatoirement. La figure 3.7 montre un exemple d'une instance de 5 étages et 10 jobs et de configuration machine c, où il y a deux machines à l'étage du milieu et trois machines aux autres étages.

```
nbjobs - 1
nbcentre - 1
p[j1, c1] p[j2,c1]....p[jn,c1] nbmachine[c1]
p[j1, c2] p[j2,c2]....p[jn,c2] nbmachine[c2]
.
.
.
p[j1, ck] p[j2,ck]....p[jn,ck] nbmachine[ck]



avec  nbjobs            : le nombre de jobs
      nbcentre      : le nombre de centres
      p[ji,ck]      : la durée  d'exécution du job ji dans le centre ck
       nbmachine[ck] : le nombre de machines disponibles pour le centre ck

Exemple:
4
9
11  4 13 14 14  3 15  7  4 14  3
15  7  7 12  3  6 11  4  4  9  3
 4  9 11  5 12  5 11  6 15 12  2
10 11 13  7  9 12  8  9  3 10  3
 8  3 15 12 12 15 15  7  4  4  3


est un exemple de 5 centres , 10 jobs

première ligne -> premier centre
la durée d'exécution du job 1 dans le centre 1 est 11
la durée  d'exécution du job 7 dans le centre 1 est 15
le centre 1 a 3 machines disponibles
le centre 3 a 2 machines disponibles
```

figure. 3.7 – Exemple descriptif d'une instance

3.5 Expérimentations

Nous présentons dans cette partie les résultats expérimentaux de l'application de l'algorithme proposé DPSO pour résoudre le problème d'ordonnancement d'atelier de type flow shop hybride. Nous commençons d'abord par un réglage de paramètres dans le but de déterminer les meilleures valeurs des paramètres utilisés dans notre algorithme. Ensuite, nous examinons l'influence de l'incorporation de la recherche locale dans l'algorithme DPSO. Nous étudions la performance de l'algorithme d'optimisation particulaire DPSO en utilisant un jeu de problèmes Benchmarks, où 10 essais indépendants sont effectués pour chaque instance. Les résultats de comparaisons avec d'autres algorithmes de la littérature sont présentés.

3.5.1 Réglage des paramètres

En plus de la taille de la population P_s (Population size), il y a quatre paramètres dans notre algorithme : la probabilité de croisement RCC (c_1), la probabilité de croisement uniforme (c_2), la probabilité de mutation (ω), et la probabilité de la recherche locale (c_3).
Compte tenu des tests préliminaires et des valeurs de référence de la littérature [17], la taille de la population est fixée à 20 particules. Pour les deux probabilités de croisement c_1 et c_2, nous posons initialement $c_1 = c_2 = 0.8$, et nous faisons varier ω et c_3 en utilisant l'ensemble de valeurs {0.1, 0.2,...,0.9}. Trois instances difficiles de Benchmarks (instances $j10c5c1$, $j10c5d1$ et $j15c5c5$) sont considérées dans les tests réalisés.

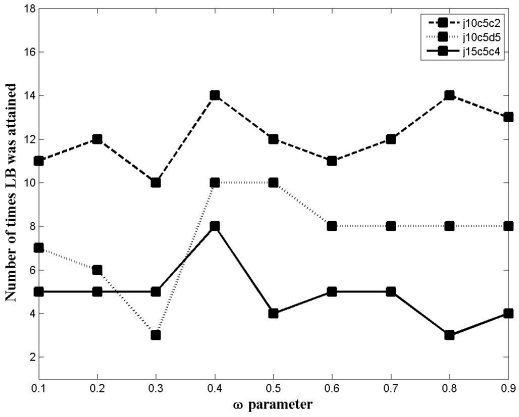


figure. 3.8 – Paramètre ω

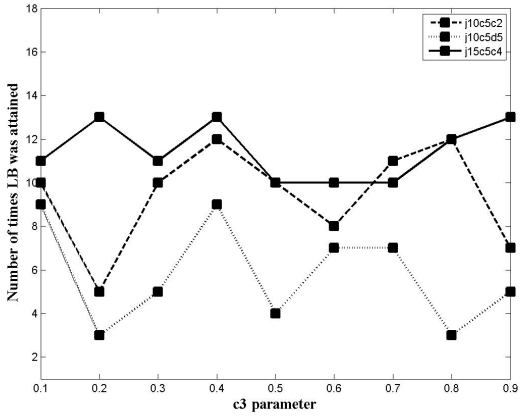


figure. 3.9 – Paramètre c_3

Les figures 3.8 et 3.9 illustrent pour chaque paramètre, le nombre de fois où la borne inférieure (LB) a été atteinte. La valeur correspondante à un nombre de fois élevé est retenue, alors

$\omega = 0.4$ et $c_3 = 0.4$. Après avoir fixé les deux paramètres ω et c_3, nous cherchons cette fois à varier les deux paramètres c_1 et c_2 pour définir les valeurs définitives à adopter. À ce propos, nous avons effectué d'autres tests. Les résultats du tableau 3.4 (instance $j10c5c3$ avec $LB = 71$) montrent que les valeurs déjà choisies pour c_1 et c_2 sont les plus appropriées. Le taux de réussite indique le nombre de tests réussis parmi les 10 essais.
Il faut souligner que nous avons pris $c_1 = c_2$ dans le but d'équilibrer l'utilisation des deux types de croisement, autrement dit donner une importance similaire aux deux composantes de la position de particule que représente chaque croisement.

Table. 3.4 – Résultats des tests préliminaires des paramètres c_1 et c_2

Essai	c_1	c_2	c_1	c_2	c_1	c_2
	0.7	0.7	0.8	0.8	0.9	0.9
1	73		71		71	
2	71		71		73	
3	73		72		71	
4	71		71		72	
5	71		71		71	
6	71		71		71	
7	72		71		71	
8	71		72		73	
9	71		71		73	
10	71		71		71	
Taux de réussite	$\frac{7}{10}$		$\frac{8}{10}$		$\frac{6}{10}$	

3.5.2 Résultats numériques

Nous nous intéressons premièrement dans le cadre de notre étude expérimentale à examiner l'impact de l'incorporation de la recherche locale dans l'algorithme proposé, en comparant l'algorithme DPSO avec l'algorithme DPSO sans recherche locale (DPSO-SRL).
La figure 3.10 illustre les valeurs du makespan en fonction du temps (en secondes) pour les deux algorithmes DPSO et DPSO-SRL, pour des instances aléatoires sous différentes configurations. Nous pouvons remarquer clairement que DPSO fonctionne plus rapidement que DPSO-SRL. Pour toutes les configurations, la valeur du makespan de l'algorithme DPSO décroît rapidement, ce qui démontre l'efficacité de l'incorporation de la recherche locale.

Pour la clarté des figures, les temps de calculs présentés sont inférieurs à 10 secondes. La supériorité de l'algorithme DPSO provient du compromis exploration-exploitation, réalisé par la combinaison de la recherche globale et locale. Cet équilibre maintenu entre les concepts de l'intensification (Opérateur de mutation et recherche locale) et la diversification (Opérateurs de croisement) aide à assurer la convergence dans des temps raisonnables.

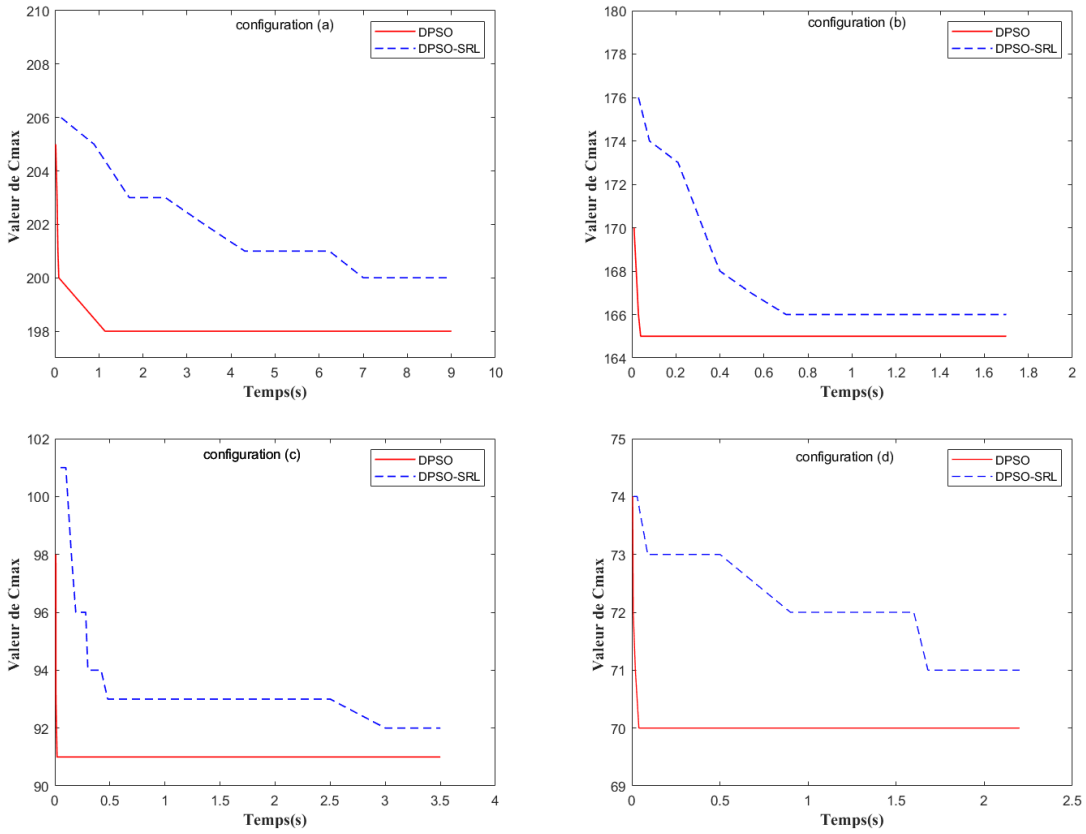


figure. 3.10 – Convergence de l'algorithme DPSO sur toutes les configurations

Nous nous intéressons aussi à déterminer l'effet de l'utilisation de la méthode des règles de priorité combinées sur les valeurs du makespan.

La figure 3.11 illustrent pour des différentes instances, les valeurs du makespan en utilisant les règles FIFO, SPT, LPT et règles combinées respectivement.

Les résultats montrent que l'utilisation des règles de priorité combinées donne des meilleures valeurs du makespan par rapport aux règles de priorité FIFO ou SPT ou LPT. Ceci peut être expliqué par le fait que la gestion des files d'attentes en utilisant des règles combinées

supporte au mieux la nature dynamique de l'atelier et permet de réduire le temps d'attente.

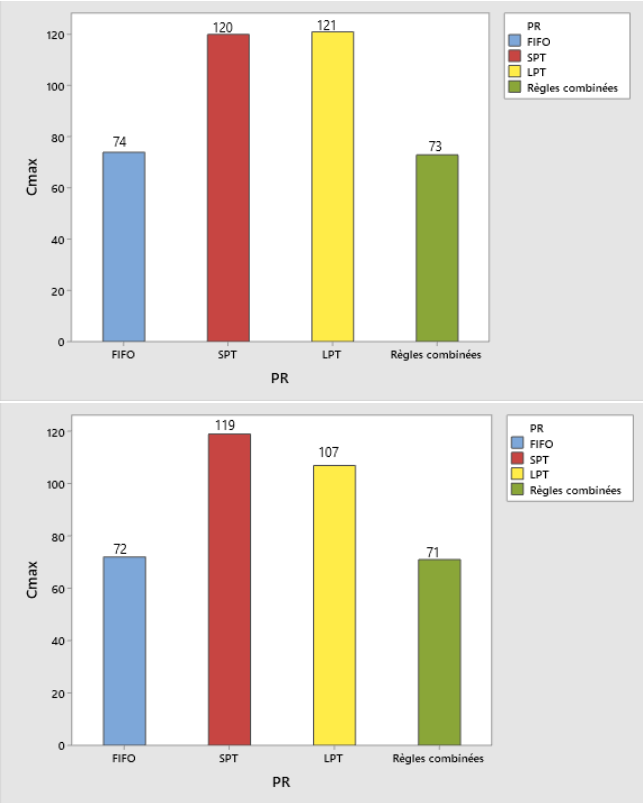


figure. 3.11 – Effet des règles de priorité sur C_{max}

Pour évaluer la performance de l'algorithme proposé DPSO, quatre méthodes parmi les plus répandues dans la littérature ont été utilisées. La première méthode est notée IAIS (Immunoglobulin-based Artificial Immune System algorithm) [32], la deuxième méthode est notée AIS (Artificial Immune System Algorithm) [26], et se sont tous les deux des algorithmes métaheuristiques basés sur le système immunitaire artificiel. La troisième méthode est notée QIA (Quantum-inspired Immune Algorithm) [29], et la quatrième méthode ACO (Ant colony optimization) est un algorithme de colonie de fourmis développé par Alaykýran et al. [27] (pour plus de détails, voyez section 1.4.3). Dans ces méthodes, les auteurs ont utilisé les problèmes Benchmarks de Carlier et Néron [16], et ils ont limité leur algorithmes par un temps maximal de calcul $T = 1600s$. Si la solution optimale n'est pas trouvée au bout de ce temps, l'algorithme s'arrête et la meilleure solution obtenue est acceptée comme solution finale.

Les résultats obtenus ont été comparés à une borne inférieure globale (LB) [16], le pourcentage de déviation est calculé par la formule suivante :

$$\%\text{déviation} = \frac{C_{max} - LB}{LB} \times 100 \tag{3.13}$$

Rappelons que l'algorithme DPSO, ainsi que tous les algorithmes proposés dans cette thèse sont codés en Langage MATLAB et exécutés sur un PC intel core i3, 2.1 GHz.

3.5.2.1 Résultats pour les instances difficiles

Le tableau 3.5 montre les résultats obtenus par les différents algorithmes pour les instances difficiles, où C_{max} représente la valeur du makespan trouvée pour chaque algorithme, et CPU indique le temps de calcul en secondes. Pour tous les algorithmes, le pourcentage de déviation pour chaque instance est calculé. Pour les algorithmes IAIS, AIS, ACO et QIA, les résultats de calcul ont été obtenus à partir de leurs articles originaux.
À partir du tableau 3.5, nous pouvons remarquer que les valeurs du pourcentage de déviation (%déviation) résultantes de DPSO sont meilleures ou égales aux valeurs obtenues par les autres algorithmes.

Les performances de tous les algorithmes pour les problèmes difficiles sont résumées dans le tableau 3.6, où "Nb de pbs" représente le nombre de problèmes considérés par chaque algorithme, "% Pbs résolus" représente le pourcentage de problèmes difficile résolus optimalement, et $\overline{\%deviation}$ représente le pourcentage de déviation moyen obtenu par chaque algorithme.

Nous constatons à partir de ce tableau que l'algorithme DPSO peut résoudre 18 problèmes parmi 24, ce qui représente 75% des problèmes difficiles. Quant aux algorithmes IAIS et AIS, ils peuvent résoudre 16 problèmes (66.7%). L'algorithme ACO peut résoudre 12 problèmes difficiles (50%), alors que QIA ne peut résoudre aucun problème difficile. En outre, l'algorithme DPSO fournit un pourcentage de déviation moyen de 2.85%, contre 3.02%, 3.28%, 4.10% et 5.04% pour les algorithmes IAIS, AIS, ACO et QIA respectivement.

Quant au temps CPU, les algorithmes ont été exécutés sur des machines différentes, alors il ne serait pas utile de les comparer directement. La méthode de comparaison proposée par Pan et al. [60] est utilisée. Cette dernière est basée sur un facteur de correction calculé à partir des vitesses de processeurs.

Table. 3.6 – Comparaison de performance des différents algorithmes sur les problèmes difficiles

Méthode	Problèmes difficiles		
	Nb de pbs	% Pbs résolus	$\overline{\%deviation}$
DPSO	24	75	2.85
IAIS	24	66.7	3.02
QIA	12	0	5.04
ACO	18	50	4.10
AIS	24	66.7	3.13

L'algorithme IAIS a été exécuté sur un PC intel core 2, 1.83 GHz, et DPSO a été exécuté sur un PC intel core i3, 2.1 GHz, le facteur de correction est égal à

$$1.83/2.1 = 0.87$$

Ainsi, en employant ce facteur de correction pour le problème j10c5d5 par exemple, le temps CPU obtenu par IAIS est

$$3.76 \times 0.87 = 3.27s$$

Il est nettement plus grand que le temps CPU obtenu par DPSO ($0.13s$).
En fait, c'est le cas pour tous les problèmes résolus sauf les problèmes j10c5d1 et j10c5d6. On note que le problème j15c5d1 n'est pas inclus dans cette comparaison puisque le temps d'exécution obtenu par les deux algorithmes pour ce problème est égal à $0.00s$.

En résumé, DPSO fonctionne plus rapidement pour 15 problèmes. Ces problèmes constituent 62.5% du nombre total de problèmes difficiles. Alors que le nombre de problèmes dans lesquels IAIS fonctionne plus rapidement est 2 (8.33% du nombre total de problèmes difficiles).
Il faut souligner que IAIS est utilisé dans cette comparaison en raison de son superiorité par rapport aux trois autres algorithmes AIS, ACO et QIA.

3.5.2.2 Résultats pour les instances faciles

Pour les instances faciles, nous avons comparé les résultats obtenus par DPSO avec les algorithmes AIS et ACO (puisque les résultats pour IAIS ne sont pas disponibles).

Les tableaux 3.7 et 3.8 illustrent les résultats obtenus par les algorithmes DPSO, AIS et ACO pour les instances faciles avec $c = 5$ et $c = 10$ respectivement. Pour toutes les instances, la valeur de C_{max}, le temps CPU et le pourcentage de déviation sont calculés.

Table. 3.7 – Comparaison des résultats : problèmes benchmarks faciles avec $c = 5$

Problème	DPSO		AIS		ACO		LB	% déviation		
	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU		DPSO	AIS	ACO
$j10c5a2$	88	0.02	88	1	88	–	88	0	0	0
$j10c5a3$	117	0.04	117	1	117	–	117	0	0	0
$j10c5a4$	121	0.12	121	1	121	–	121	0	0	0
$j10c5a5$	122	0.11	122	1	122	–	122	0	0	0
$j10c5a6$	110	0.01	110	4	110	–	110	0	0	0
$j10c5b1$	130	0.15	130	1	130	–	130	0	0	0
$j10c5b2$	107	0.01	107	1	107	–	107	0	0	0
$j10c5b3$	109	0.13	109	1	109	–	109	0	0	0
$j10c5b4$	122	0.04	122	2	124	–	122	0	0	1.64
$j10c5b5$	153	0.04	153	1	153	–	153	0	0	0
$j10c5b6$	115	0.04	115	2	115	–	115	0	0	0
$j15c5a1$	178	0.13	178	1	178	–	178	0	0	0
$j15c5a2$	165	0.5	165	1	165	–	165	0	0	0
$j15c5a3$	130	0.14	130	1	132	–	130	0	0	1.54
$j15c5a4$	156	0.14	156	2	156	–	156	0	0	0
$j15c5a5$	164	0.2	164	1	166	–	164	0	0	1.22
$j15c5a6$	178	0.01	178	1	178	–	178	0	0	0
$j15c5b1$	170	0.01	170	1	170	–	170	0	0	0
$j15c5b2$	152	0.01	152	1	152	–	152	0	0	0
$j15c5b3$	157	0.02	157	1	157	–	157	0	0	0
$j15c5b4$	147	0.2	147	1	149	–	147	0	0	1.36
$j15c5b5$	166	0.4	166	2	166	–	166	0	0	0
$j15c5b6$	175	0.01	175	1	176	–	175	0	0	0.57

Table. 3.8 – Comparaison des résultats : problèmes benchmarks faciles avec $c = 10$

Problème	DPSO		AIS		ACO		LB	% déviation		
	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU		DPSO	AIS	ACO
$j10c10a1$	139	0.05	139	1	–	–	139	0	0	–
$j10c10a2$	158	0.04	158	18	–	–	158	0	0	–
$j10c10a3$	148	0.04	148	1	–	–	148	0	0	–
$j10c10a4$	149	0.04	149	2	–	–	149	0	0	–
$j10c10a5$	148	0.04	148	1	–	–	148	0	0	–
$j10c10a6$	146	0.04	146	4	–	–	146	0	0	–
$j10c10b1$	163	0.05	163	1	163	–	163	0	0	0
$j10c10b2$	157	0.04	157	1	157	–	157	0	0	0
$j10c10b3$	169	0.04	169	1	169	–	169	0	0	0
$j10c10b4$	159	0.04	159	1	159	–	159	0	0	0
$j10c10b5$	165	0.04	165	1	165	–	165	0	0	0
$j10c10b6$	165	0.04	165	1	165	–	165	0	0	0
$j10c10c1$	115	a	115	a	118	–	113	1.77	1.77	4.42
$j10c10c2$	118	a	119	a	117	–	116	1.72	2.59	0.86
$j10c10c3$	108	a	116	a	108	–	98	10.2	18.37	10.2
$j10c10c4$	112	a	120	a	112	–	103	8.74	16.50	8.74
$j10c10c5$	125	a	126	a	126	–	121	3.31	4.13	4.13
$j10c10c6$	105	a	106	a	102	–	97	8.25	9.28	5.15
$j15c10a1$	236	0.13	236	1	236	–	236	0	0	0
$j15c10a2$	200	0.4	200	30	200	–	200	0	0	0
$j15c10a3$	198	0.14	198	4	198	–	198	0	0	0
$j15c10a4$	225	0.14	225	12	225	–	225	0	0	0
$j15c10a5$	182	0.1	182	2	182	–	182	0	0	0
$j15c10a6$	200	0.01	200	2	200	–	200	0	0	0
$j15c10b1$	222	0.02	222	3	222	–	222	0	0	0
$j15c10b2$	187	0.5	187	1	188	–	187	0	0	0.54
$j15c10b3$	222	0.02	222	1	224	–	222	0	0	0.9
$j15c10b4$	221	0.03	221	1	221	–	221	0	0	0
$j15c10b5$	200	0.3	200	1	–	–	200	0	0	–
$j15c10b6$	219	0.02	219	1	–	–	219	0	0	–

a. l'algorithme n'a pas pu atteindre la valeur LB en 1600s

Table. 3.9 – Comparaison de performance des diffférents algorithmes sur les problèmes faciles

Méthode	Problèmes faciles		
	Nb de pbs	% Pbs résolus	$\overline{\%déviation}$
DPSO	53	88.7	0.64
AIS	53	88.7	0.99
ACO	45	60.38	0.92

À partir des tableaux 3.7 et 3.8, nous observons que les algorithmes DPSO et AIS donnent des résultats similaires en terme de makespan et ceci pour 47 problèmes, ce qui veut dire qu'ils peuvent résoudre 88.7% des problèmes faciles. Alors que ACO ne peut résoudre que 32 problèmes (60.38%).
Le pourcentage des problèmes résolus pour chaque algorithme est donné dans le tableau 3.9.

Par ailleurs, ce tableau montre que DPSO améliore la qualité des solutions pour plusieurs problèmes. Sur ce point, le pourcentage de déviation moyen ($\overline{\%deviation}$) fourni par DPSO est 0.64% contre 0.99% pour AIS et 0.92% pour ACO. Il faut noter que pour les algorithmes ACO et QIA, les pourcentages de déviation moyens sont calculés sur le nombre de problèmes considérés par chaque algorithme.

En ce qui concerne le temps de calcul, les résultats indiquent que DPSO donne un temps de calcul significativement inférieur à celui obtenu par AIS pour toutes les instances, même après l'application du facteur de correction. À noter que AIS est exécuté sur un PC Intel P4, 1.7 GHz.
Le pseudo code de la méthode DPSO est donné par l'algorithme 12.

Algorithme 12 Algorithme DPSO

Données : ω Probabilité de mutation, c_1 et c_2 Probabilités de croisement, c_3 Probabilité de recherche locale, P_s Taille de l'essaim.
Résultat : Meilleure solution trouvée.

Début
  t=0
  for i allant de 1 à P_s **do**
    Générer la particule X_i^t
    Évaluer X_i^t
    $P_i^t = X_i^t$
  end for
  Trouver $\mathrm{G^t}$ la meilleure particule de l'essaim
  Tant que Critère d'arrêt n'est pas atteint
    t=t+1
    for i allant de 1 à P_s **do**
      Générer une probabilité $r_1 \in [0, 1]$
      if $r_1 <= \omega$ **then**
        $V_i^t = \omega \otimes \mathrm{F_1(X_i^{t-1})}$
      end if
      Générer une probabilité $r_2 \in [0, 1]$
      if $r_2 <= c_1$ **then**
        $\mathrm{B_i^t} = c_1 \otimes \mathrm{F_2(V_i^t, P_i^{t-1})}$
      end if
      Générer une probabilité $r_3 \in [0, 1]$
      if $r_3 <= c_2$ **then**
        $\mathrm{X_i^t} = c_2 \otimes \mathrm{F_3(B_i^t, G^{t-1})}$
      end if
      Générer une probabilité $r_4 \in [0, 1]$
      if $r_4 <= c_3$ **then**
        Effectuer la recherche locale pour la particule $\mathrm{X_i^t}$
      end if
      Évaluer X_i^t
      Mettre à jour P_i^t

    end for
    Mettre à jour $\mathrm{G^t}$
  Fin Tant que
Fin

3.6 Conclusion

Dans ce chapitre, nous avons proposé une méthode métaheuristique pour résoudre le problème d'ordonnancement d'atelier de type flow shop hybride avec l'objectif de minimiser le makespan.

En premier, nous avons commencé par la présentation d'un modèle mathématique en nombres entiers qui permet de déterminer les paramètres, les variables et les contraintes du problème et facilite sa compréhension. Ensuite, nous avons développé une méthode à base de l'algorithme de l'optimisation par essaims particulaires qui est conçu initialement pour résoudre des problèmes de l'optimisation continue. De ce fait, nous avons proposé une version discrète de PSO, caractérisée par l'utilisation d'opérateurs évolutionnaires (mutation et croisement) pour mettre à jour les particules à chaque itération. Le processus de la mise à jour comporte aussi un opérateur de recherche locale qui vise à chercher des améliorations de la solution courante en explorant deux structures de voisinage différentes. Nous avons détaillé ensuite les différentes étapes de l'algorithme proposé en expliquant le fonctionnement des différents opérateurs. L'application de chaque opérateur est déterminée par une certaine probabilité fixée à l'aide d'un réglage paramétrique.

Pour calculer le makespan, nous avons proposé une méthode qui combine les règles de priorité FIFO, SPT et LPT pour ordonnancer les jobs à chaque étage à partir du 2^{eme} étage.

Les résultats expérimentaux effectués montrent que l'algorithme proposé DPSO donne de bons résultats que se soit en terme de makespan ou en terme du temps de calcul.

Chapitre 4

Algorithme de recherche d'harmonie pour le flow shop hybride avec tâches multiprocesseurs

4.1 Introduction

Le problème d'atelier de type flow shop hybride avec tâches multiprocesseurs (HFSMT) est l'un des problèmes complexes de l'ordonnancement. Il s'agit d'une extension du problème d'ordonnancement d'atelier de type flow shop hybride (HFS) étudié dans le chapitre précédent, qui relaxe la supposition selon laquelle chaque tâche au sein d'un job nécessite exactement un processeur, et permet aux tâches d'exiger un ou plusieurs processeurs simultanément. Ainsi, à chaque étage un ensemble de machines est affecté au job j pour assurer son exécution, la consommation de ressources se diffère alors d'un job à l'autre.

Le problème HFSMT a une grande importance théorique puisqu'il est NP difficile au sens fort. En outre, les domaines d'application de HFSMT sont nombreux et variés. Notamment la productions de moteurs d'avion, les industries électroniques, la fabrication textile et les systèmes de vision en temps réel. Pour ces raisons, le problème a attiré l'attention de nombreux chercheurs qui ont proposé des différentes méthodes heuristiques et métaheuristiques pour faire face à sa complexité.

Au cours des dernières années, l'algorithme de la recherche d'harmonie (HS) a été appliqué avec succès à divers problèmes d'optimisation [66,67,69,75], et il a été prouvé une grande performance. De surcroît, l'algorithme HS est caractérisé par la simple implémentation et la flexibilité.

Dans ce chapitre, nous proposons une métaheuristique à base de la recherche d'harmonie pour résoudre le problème HFSMT avec la minimisation du makespan. Nous présentons tout d'abord la démarche de l'algorithme HS tout en montrant ses différentes étapes, ensuite

nous donnons les résultats numériques ainsi que les comparaisons avec d'autres approches de résolution de la littérature.

4.2 Description du problème

Le problème est défini par un ensemble $J = \{J_1, J_2, ..., J_n\}$ de n jobs à traiter séquentiellement sur un ensemble de k étages en série. Chaque étage i se compose de m_i machines parallèles identiques (processeurs). À l'étage i, le job j requiert $size_{ij}$ processeurs simultanément. En d'autres termes, le job j nécessite $size_{ij}$ machines au niveau de l'étage i pour son traitement pendant un temps d'exécution p_{ji}. L'objectif est de minimiser la durée totale de l'ordonnancement, c'est-à-dire le makespan (C_{max}).

Les suppositions suivantes sont considérées :

1. Le nombre de jobs et leurs temps d'exécution sont connus et fixés.
2. Une machine ne peut traiter qu'une seule tâche à la fois.
3. Tous les jobs et toutes les machines sont disponibles au temps zéro.
4. La préemption n'est pas autorisée.
5. Les temps de trajet entre les étages sont inclus dans les temps d'exécution.

En se basant sur la notation standard à trois champs [5], le problème HFSMT est représenté comme suit : $FHm, (PM^{(l)})_{l=1}^{m} \| size_{ij} \| C_{max}$, où $size_{ij}$ identifie le problème avec tâches multiprocesseurs et C_{max} indique l'objectif à minimiser.

4.3 État de l'art de HFSMT

En raison de sa complexité, le problème HFSMT n'a pas pu être résolu par des algorithmes exactes, en particulier pour les problèmes à grande échelle. Alors, plusieurs algorithmes heuristiques et métaheuristiques sont proposées pour résoudre ce problème. Oğuz et al. [104] ont développé plusieurs heuristiques pour minimiser le makespan dans un atelier flow shop hybride à deux étages avec des tâches multiprocesseurs. Aussi, l'étude [103] a présenté des heuristiques constructives pour résoudre le même problème. Ensuite Oğuz et al. [101] ont proposé une recherche tabou basée sur des techniques heuristiques pour un problème HFSMT à plusieurs étage. Dans le même contexte, Lahimer et al [91] ont développé une heuristique de recherche à divergence (discrepancy search heuristic), ils ont proposé aussi une borne inférieure basée sur le concept des fonctions duales réalisables. Kahraman et al. [98] ont introduit un algorithme glouton parallèle (parallel greedy algorithm), qui est un processus de recherche

itérative. Les résultats de leur approche sont comparés avec les études [94,101].
Plusieurs études ont proposé des algorithmes génétiques pour résoudre le problème, nous citons par exemple les travaux de Şerifoğlu et Ulusoy [89], Oğuz et Ercan [94] et Engin et al. [99]. Dans [88], les auteurs ont développé un algorithme de recuit simulé (SA), avec trois méthodes d'ordonnancement pour calculer le makespan. Chou [86] a développé une optimisation par essaims particulaires (PSO) avec plusieurs règles de priorité. Ces dernières sont employées pour la détermination de la séquence de tâches et pour l'affectation des tâches aux machines à chaque étage. Le travail de Wei Lin et al. [85] a proposé un algorithme hybride de colonies d'abeilles artificielles (ABC) avec une planification bidirectionnelle pour le problème HFSMT. Dans le même cadre, nous pouvons citer aussi les travaux [62,90,92,84,65,64].
Dans ce travail, nous proposons un algorithme à base de la recherche d'harmonie (HS) pour résoudre le problème.

4.4 L'algorithme de recherche d'harmonie HS

L'algorithme de recherche d'harmonie (HS) repose généralement sur un ensemble de solutions (harmonies) générées aléatoirement au début de l'algorithme et stockées dans la mémoire d'harmonie (HM). À chaque itération, une nouvelle harmonie candidate est générée à partir de toutes les harmonies de HM, qui est considérée comme le noyau de l'algorithme HS. Autrement dit, HS utilise toute la population courante pour créer une nouvelle harmonie. La nouvelle harmonie est engendrée en appliquant les trois règles d'improvisation : la considération de la mémoire d'harmonie, l'ajustement du ton et la sélection aléatoire. Dans ce qui suit, nous détaillons les étapes de l'algorithme proposé HS.

4.4.1 Représentation de solution

La représentation de solution constitue une étape cruciale dans l'algorithme de recherche d'harmonie. Puisque dans le cadre du problème de flow shop hybride avec tâches multiprocesseurs, nous cherchons à trouver la séquence de jobs qui minimise le makespan. Alors nous gardons la représentation par permutation présentée dans le chapitre précédent. Nous considérons dans l'algorithme proposé, un ensemble d'harmonies HM (mémoire d'harmonie) de taille HMS (harmony memory size), pour chaque harmonie nous associons une séquence de n jobs $X_i = \{X_{i1}, X_{i2}, ..., X_{in}\}$ et une fonction objectif représentée par la valeur du makespan C_{max} de cette séquence.
HM est initialisée en utilisant une méthode basée sur l'apprentissage par opposition que l'on détaille dans la section suivante.

4.4.2 Initialisation de mémoire d'harmonie HM

L'objectif de l'étape d'initialisation est d'initialiser les paramètres du problème et de créer la mémoire d'harmonie. Cette dernière est souvent générée d'une manière aléatoire dans la version de base de l'algorithme de recherche d'harmonie. Dans l'algorithme HS proposé, nous utilisons la méthode d'apprentissage basée sur l'opposition (opposition based learning, OBL) pour générer une mémoire d'harmonie diversifiée.

4.4.2.1 La méthode OBL

Le concept de l'opposition peut être trouvé dans de nombreux domaines. Notamment, la particule et l'antiparticule en physique, le bien et le mal dans le domaine religieux, la majorité et les partis d'opposition en politique et le principe de forces complémentaires et contraires "YIN" et "YANG" dans la philosophie chinoise. En d'autres termes, le monde est structuré par des oppositions complémentaires, et dans différents cas, les idées, les conceptions, les entités et les situations peuvent être facilement appréhendées via leur opposés. La méthode opposition based learning (OBL) est une nouvelle approche de l'intelligence artificielle introduite par Tizhoosh en 2005 [49]. L'idée de la méthode OBL est de considérer à la fois les solutions candidates actuelles et leurs contraires dans le but d'améliorer les directions de la recherche. Soit $x \in [a, b]$ un nombre réel, le nombre opposé de x est défini par :

$$\hat{x} = a + b - x$$

D'une manière similaire, étant donné $X = (x_1, x_2, ..., x_n)$ un point dans un espace à n dimensions, où $x_i \in [a_i, b_i]$ est un nombre réel, et $i \in \{1, 2, ..., n\}$. Le point opposé de X est le point defini par : $\hat{X} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$ avec $\hat{x}_i = a_i + b_i - x_i$

Pour illustrer la méthode OBL, supposons que $f(.)$ est la fonction objectif utilisée pour évaluer les solutions candidates. Soient X une solution candidate et $\hat{X}$ son opposé. Si $f(X) \geq f(\hat{X})$, i.e, $\hat{X}$ a une meilleure évaluation que X, alors X est remplacé par $\hat{X}$, sinon la solution X est conservée. En résumé, la solution et son opposé sont évalués simultanément, puis l'apprentissage se poursuit avec la solution la plus apte.

4.4.2.2 Schéma de l'initialisation OBL

La méthode OBL est utilisée pour initialiser la mémoire d'harmonie (HM). Au début, une population de solutions (harmonies) notée *pop* est générée d'une manière aléatoire. Si $\pi_i = (\pi_{i1}, \pi_{i2}, ..., \pi_{in})$ désigne la i^{eme} harmonie de la population pop, où π_{ik} $(k \in \{1, 2, ...n\})$

figure. 4.1 – exemple d'opposition : Yin et Yang

représente le job affecté dans la position k de l'harmonie. Alors la population pop peut être représentée comme suit :

$$pop = \begin{pmatrix} \pi_{11} & \pi_{12} & \cdot & \cdot & \cdot & \pi_{1n} \\ \pi_{21} & \pi_{22} & \cdot & \cdot & \cdot & \pi_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \pi_{HMS1} & \pi_{HMS2} & \cdot & \cdot & \cdot & \pi_{HMSn} \end{pmatrix}$$ Ensuite, la population opposée (opop) est calcu-

lée, tout en prenant en considération que l'harmonie ne pourrait pas avoir un job manquant ou répétitif.

$$opop = \begin{pmatrix} \hat{\pi}_{11} & \hat{\pi}_{12} & \cdot & \cdot & \cdot & \hat{\pi}_{1n} \\ \hat{\pi_{21}} & \hat{\pi_{22}} & \cdot & \cdot & \cdot & \hat{\pi}_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hat{\pi}_{HMS1} & \hat{\pi}_{HMS2} & \cdot & \cdot & \cdot & \hat{\pi}_{HMSn} \end{pmatrix}$$

Les solutions les plus appropriées sont sélectionnées parmi $pop \bigcup opop$ pour construire la mémoire d'harmonie HM, et les pires solutions sont éliminées. La procédure de l'initialisation OBL est donnée dans l'algorithme 10.

Algorithme 13 L'initialisation OBL

Données : HMS la taille de la mémoire d'harmonie
Résultat : HM la mémoire d'harmonie
Début
  Générer aléatoirement une poulation d'harmonies *pop* de taille HMS
  Calculer la population opposée *opop*
  Évaluer les harmonies des deux poulations *pop* et *opop*
  Choisir les meilleures HMS harmonies à partir de $pop \bigcup opop$.
  Stocker les harmonies choisies dans HM
  Éliminier les harmonies restantes
Fin

4.4.3 Improvisation d'une nouvelle harmonie

L'improvisation est l'étape la plus importante de l'algorithme HS, où une nouvelle harmonie est générée en utilisant trois règles : la considération de la mémoire, l'ajustement du ton et la sélection aléatoire. Puisque les solutions du problème sont représentées comme des permutations de n jobs, l'improvisation donnée par la recherche d'harmonie standard n'est pas adaptée pour produire de nouvelles harmonies. Pour cette raison, nous présentons une improvisation basée sur des opérateurs de croisement et de mutation. Les étapes de l'improvisation proposée sont présentées par la suite.

4.4.3.1 Règle de considération de la mémoire

Pour cette règle, un nombre aléatoire r_1 ($r_1 \in [0, 1]$) est généré. Si $r_1 \leq HMCR$ (le taux de considération de la mémoire d'harmonie), alors une nouvelle harmonie est générée à partir de toutes les harmonies de HM. La considération de la mémoire proposée est basée sur le croisement RCC présenté dans le chapitre précédent. Cette règle fonctionne en commençant par établir le croisement de la meilleure harmonie X_{best} dans HM avec les harmonies de HM successivement. Si l'harmonie résultante du croisement est meilleure que X_{best}, alors X_{best} est mise à jour et la procédure se continue. Cette règle permet de créer une nouvelle harmonie en utilisant toutes les harmonies de HM, tout en gardant la meilleure expérience représentée par X_{best}. La règle utilise la meilleure harmonie dans HM jusqu'à ce qu'une meilleure harmonie soit créée. L'algorithme 10 illustre la règle de considération de la mémoire.

Algorithme 14 Considération de la mémoire

Données : HM la mémoire d'harmonie, HMS la taille de HM
Résultat : X la nouvelle harmonie

Début
  Déterminer X_{best} la meilleure harmonie de HM
  for j allant de 1 à HMS **do**
    $X_j \longleftarrow j^{th}$ harmonie dans HM
    Etablir le croisement RCC
    $X \longleftarrow X \otimes (X_j \otimes X_{best})$
    if X est meilleure que X_{best} **then**
      Mettre à jour X_{best}
    end if
  end for
  Mettre à jour X

Fin

4.4.3.2 Règle d'ajustement du ton

La règle d'ajustement du ton est appliquée lorsque l'harmonie est déterminée à partir de HM en utilisant la règle de considération de la mémoire. Chaque harmonie choisie à partir de HM est ajustée avec une probabilité PAR (le taux d'ajustement du ton) pour une exploration approfondie du voisinage. Cette règle vise alors à favoriser l'exploration et à éviter les optimums locaux.
Une procédure basée sur la mutation par insertion est utilisée comme règle d'ajustement du ton, où divers mouvements d'insertion sont appliqués pour trouver une harmonie voisine ajustée. Rappelons que La mutation par insertion consiste à supprimer le job actuellement en position i et à l'insérer dans une autre position j choisie aléatoirement.

4.4.3.3 Règle de la sélection aléatoire

Si $r_1 \geq HMCR$, alors une sélection aléatoire est effectuée. Autrement dit, la nouvelle harmonie est générée aléatoirement avec la probabilité $(1 - HMCR)$.

4.4.4 Mise à jour de HM

Si la nouvelle harmonie obtenue à l'étape de l'improvisation est meilleure que la pire harmonie dans HM, alors HM est mise à jour en supprimant la pire harmonie et en la remplaçant par cette nouvelle harmonie.

4.4.5 Stratégie de recherche locale

La recherche locale proposée est appliquée à la fin de l'algorithme HS dans le but d'intensifier la recherche dans le voisinage de la solution trouvée.
Nous utilisons la stratégie de recherche locale décrite au chapitre 3. Étant donné que l'ordre de voisinages explorés peut influencer la qualité de la solution, ce dernier est choisi au hasard. Au début de la recherche, un nombre aléatoire $k \in \{0.1\}$ est généré, si $k = 0$, la recherche locale s'applique en explorant le voisinage par insertion puis le voisinage par échange, si non l'ordre de l'exploration est inversé.

4.4.6 Condition d'arrêt

Dans cette étude, l'arrêt de l'algorithme HS est conditionné par un nombre maximal d'itérations noté N_{max} ou un temps maximal de calcul.

4.5 Paramétrage de l'algorithme HS

4.5.1 Description des paramètres

Puisque les valeurs des paramètres ont une grande influence sur les performances de l'algorithme, alors le choix des valeurs adéquates des différents paramètres constitue une étape importante dans la mise en œuvre de l'algorithme proposé. Dans l'algorithme HS, les valeurs des paramètres suivants doivent être choisies : la taille de la mémoire d'harmonie (HMS), le taux de considération de la mémoire ($HMCR$), le taux d'ajustement du ton (PAR), et le nombre maximal d'itérations (N_{max}). Mais $HMCR$ et PAR sont les paramètres prépondérants dans l'algorithme HS.
La règle de considération de la mémoire utilise la mémoire d'harmonie (HM) pour créer de nouvelles harmonies. Cette règle est appliquée avec une certaine probabilité appelé le taux de considération de la mémoire ($HMCR$). Ce dernier permet une utilisation efficace de HM et assure que les meilleures harmonies de HM soient considérées pour créer de nouvelles harmonies. Si la valeur de ce paramètre est trop petit, alors la sélection à partir de HM sera limitée ce qui va ralentir la mise à jour de HM et également la convergence de l'algorithme.

C'est pourquoi, la valeur de $HMCR$ est généralement prise dans l'intervalle [0.7,0.95] [75]. La règle d'ajustement du ton permet d'ajuster les harmonies stockées dans HM. Essentiellement, c'est un processus de raffinement de solutions locales. Elle est similaire à l'opérateur de mutation dans les algorithmes génétiques. De même, un taux d'ajustement du ton (PAR) faible peut ralentir la convergence, car il va limiter l'exploration à un petit sous-espace de l'espace de recherche total. Aussi, un taux très élevé peut entraîner la dispersion de solutions autour d'un certain optimum potentiel [75].

4.5.2 Méthode de Taguchi

Dans le but de trouver les meilleures valeurs des paramètres de l'algorithme HS, nous utilisons une méthode de la famille des méthodes de plan d'expériences (design of experiments, DOE) appelée la méthode de Taguchi.
La méthode de Taguchi est une méthode statistique inventée par Gen'ichi Taguchi dans les années 60 pour réaliser des plans d'expériences. Mais avant de détailler cette approche, nous devons définir les concepts suivants :

1. **La réponse** est la grandeur (la grandeur d'intérêt) que l'on mesure pour connaître l'effet des facteurs sur le système. Par exemple, dans notre problème la réponse est la valeur du makespan.
2. **Les Facteurs** ou les variables d'entrées sont les variables qui influencent la réponse. Ainsi, la réponse dépend de valeurs des facteurs. Par exemple, les facteurs dans notre problème sont les paramètres de l'algorithme HS.
3. **Niveau** d'un facteur est une des valeurs que peut prendre ce facteur dans le plan d'expériences. Pendant la réalisation des expériences, chaque facteur a un nombre de niveaux fixes.
4. **Le plan d'expériences** est une technique qui mesure les effets des facteurs sur la réponse, puis elle détermine en un nombre minimal d'expériences la combinaison des facteurs qui optimise la réponse. Il existe de nombreux plans d'expériences adaptés aux différentes expérimentations rencontrées. En plus des plans de Taguchi, nous pouvons citer les plans factoriels complets, les plans factoriels fractionnaires et les plans de Plackett-Burmans.

La méthode de Taguchi vient d'enrichir les méthodes de plans d'expériences, elle apporte des améliorations considérables aux plans factoriels complets et fractionnaires [48]. Cette méthode permet de simplifier le protocole expérimental en réduisant le nombre des expériences, tout en conservant une bonne précision des résultats.
On peut résumer les étapes d'application de la méthode de Taguchi comme suit :

1. Identifier le problème et les objectifs.

2. Déterminer les facteurs et leurs niveaux.
3. Choisir une table (matrice) de Taguchi convenable.
4. Construire le plan d'expériences correspondant à la table de Taguchi.
5. Réaliser les expériences.
6. Analyser et interpréter les résultats.

Rappelons que les paramètres de notre algorithme sont :

$HMCR$: le taux de considération de la mémoire.

PAR : le taux d'ajustement du ton.

HMS : la taille de la mémoire d'harmonie.

N_{max} : le nombre maximal d'itérations.

Pour les facteurs $HMCR$, HMS et N_{max}, trois niveaux ont été choisis pour chacun d'eux, et neuf niveaux ont été choisis pour le facteur PAR. Le tableau 4.1 montre les niveaux de ces facteurs.

Table. 4.1 – Niveaux des facteurs $HMCR$, PAR, HMS et N_{max}

Facteurs	Niveaux								
$HMCR$	0.85	0.90	0.95						
PAR	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
HMS	15	20	25						
N_{max}	500	1000	1500						

En examinant les tables de Taguchi sur le logiciel Minitab 19, nous trouvons que ces tables ne supportent pas un facteur à 9 niveaux. Alors nous utilisons le logiciel Ellistat 6.7 dans cette partie de notre étude statistique. Nous commençons par étudier les effets des deux facteurs $HMCR$ et PAR, et on fixe les facteurs HMS et N_{max} aux niveaux respectifs 20 et 1000. Puis, nous étudions les effets des facteurs HMS et N_{max}, en fixant $HMCR$ et PAR aux niveaux respectifs 0.9 et 0.7 ajustés à l'aide de Ellistat. Les figures 4.2 et 4.3 montrent le meilleur niveau pour chaque facteur.

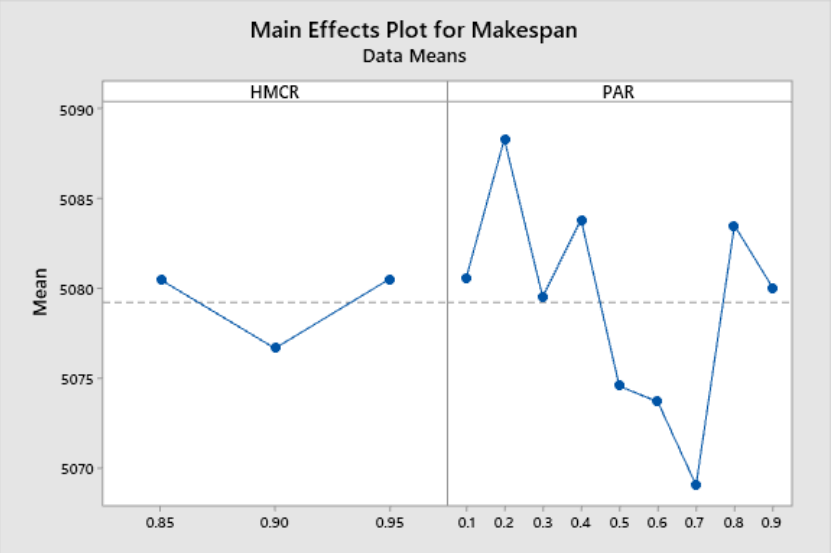


figure. 4.2 – Effets de $HMCR$ et PAR sur le Makespan

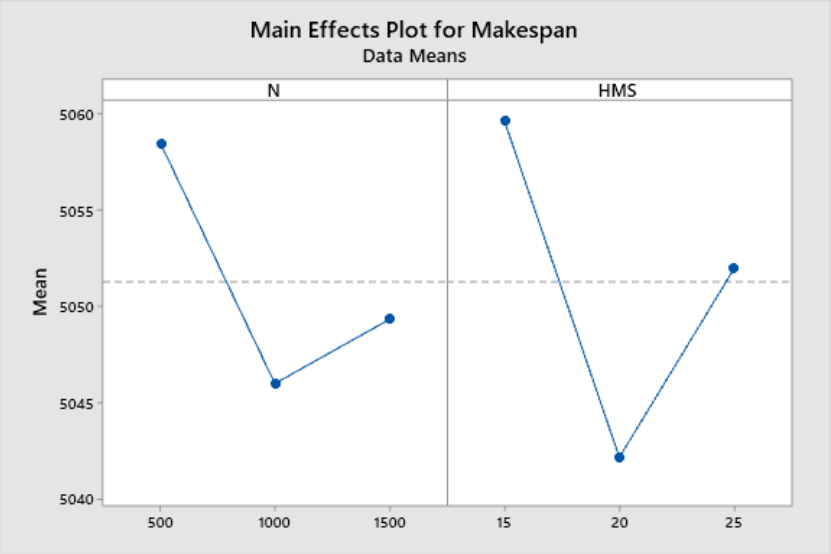


figure. 4.3 – Effets de HMS et N sur le Makespan

Au vu de ces résultats, nous réalisons un autre plan de Taguchi sur Minitab 19. Parmi les niveaux de PAR, nous considérons les 6 niveaux suivants : 0.3, 0.4, 0.5, 0.6, 0.7 et 0.8. Nous

choisissons la table $L18$ (matrice des expériences) qui est la plus adaptée à notre problème, puis nous réalisons les expériences basées sur les différentes combinaisons données par cette matrice. Les réponses sont transformées en rapport signal-bruit S/N (Signal to noise), et les résultats ont été évalués en s'appuyant sur l'analyse de ce rapport calculé par la formule suivante :

$$S/N = -10\log[\frac{1}{n}.\sum_{i=1}^{n} y_i^2]$$

Où y_i est la réponse obtenue à la i^{eme} répétition de l'expérience et n représente le nombre de répétitions.

Response Table for Signal to Noise Ratios

Smaller is better

Level	PAR	HMCR	Namx	HMS
1	-74,12	-74,12	-74,13	-74,12
2	-74,12	-74,09	-74,11	-74,11
3	-74,14	-74,14	-74,11	-74,12
4	-74,14			
5	-74,09			
6	-74,09			
Delta	0,06	0,04	0,02	0,01
Rank	1	2	3	4

figure. 4.4 – Tableau de Réponses

Dans le cas où l'optimum est un minimum, on cherche à maximiser le rapport S/N [47]. Puisque notre problème vise à minimiser le makespan, alors les meilleures valeurs des facteurs sont celles qui correspondent à un rapport S/N élevé. La figure 4.5 et le tableau 4.4 générés par Minitab illustrent les réponses pour les rapports S/N. Alors selon la figure 4.5, les meilleures valeurs des facteurs sont comme suit : $HMCR = 0.9$, $PAR = 0.7$, $N_{max} = 1000$ et $HMS = 20$.

D'après la figure 4.5, nous pouvons constater aussi que $HMCR$ et PAR sont les facteurs les plus importants de l'algorithme HS.

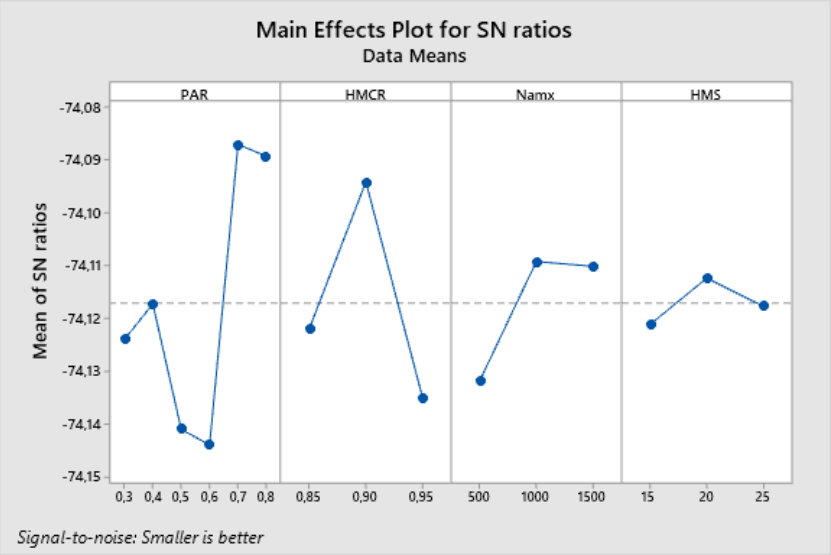


figure. 4.5 – Graphe de Réponses

4.6 Résultats expérimentaux

Dans cette partie, nous présentons les résultats numériques obtenus par l'application de l'algorithme proposé HS pour résoudre le problème d'ordonnancement d'atelier de type flow shop hybride avec tâches multiprocesseurs (HFSMT). Ainsi que les résultats de comparaison avec d'autres algorithmes de la littérature.

4.6.1 Types d'instances

Dans ce travail, les algorithmes sont testés sur des problèmes benchmarks d'oğuz [93,94]. Les 90 instances considérées dans cette études expérimentale sont des combinaisons de n (nombre de jobs) et de k (nombre d'étages) avec $n \in \{20, 50, 100\}$ et $k \in \{2, 5, 8\}$. Pour chaque combinaison de n et k, 10 instances ont été générées. Le nom de chaque problème est décodé en plus de type de problème, en tant que nombre de jobs, nombre d'étages et l'indice du problème. Par exemple $P20S2T1$ désigne la première instance de type Pcomposé de 100 jobs et 2 étages. Notons que la catégorie P désigne les instances qu'elles ont une configuration machine variable, c'est à dire les instances avec un nombre différent de processeurs (m_i) à chaque étage. Ce nombre est sélectionné d'une manière aléatoire dans l'ensemble $\{1, ..., 5\}$. Les temps d'exécution p_{ji} sont des nombres entiers aléatoires déterminés à partir de l'intervalle [1,100].

Á titre d'exemple, les temps d'exécution p_{ji} et les processeurs nécessaires aux jobs $size_{ij}$ (avec $j \in \{1, 2, ..., 20\}$ et $i \in \{1, 2\}$) pour le problème $P20S2T1$ sont donnés dans le tableau 4.2.

Table. 4.2 – Données de l'instance $P20S2T1$

j	Etage 1		Etage 2	
	$size_{1j}$	p_{j1}	$size_{2j}$	p_{j2}
1	1	74	3	71
2	2	53	3	58
3	3	57	3	16
4	3	68	2	79
5	3	17	1	35
6	2	61	3	82
7	2	7	1	10
8	1	83	2	31
9	1	40	1	82
10	1	38	2	46
11	1	65	3	73
12	2	26	1	50
13	1	86	1	26
14	3	69	2	4
15	1	6	1	59
16	3	41	1	98
17	2	38	1	31
18	3	25	3	77
19	1	19	1	27
20	1	10	11	64
Nombre de machines	3		3	

4.6.2 Résultats numériques

Nous nous intéressons au début à examiner l'effet de la méthode OBL sur la qualité de la mémoire d'harmonie initiale. La figure 4.6 montre un exemple d'une mémoire d'harmonie

donnée par l'initialisation OBL (OBL-HM) et une mémoire d'harmonie générée d'une manière aléatoire (Random-HM) pour la même instance.

$$OBL-HM=\begin{pmatrix}
19 & 12 & 10 & 6 & 4 & 11 & 2 & 1 & 18 & 3 & 8 & 15 & 20 & 9 & 13 & 5 & 17 & 16 & 14 & 7 & 765\\
16 & 5 & 7 & 19 & 9 & 2 & 20 & 15 & 13 & 1 & 8 & 12 & 14 & 17 & 6 & 10 & 11 & 18 & 3 & 4 & 766\\
2 & 10 & 18 & 1 & 6 & 17 & 14 & 8 & 3 & 7 & 4 & 11 & 19 & 5 & 13 & 9 & 15 & 16 & 12 & 20 & 770\\
5 & 3 & 6 & 19 & 16 & 8 & 9 & 15 & 17 & 2 & 10 & 18 & 1 & 11 & 12 & 14 & 4 & 13 & 20 & 7 & 777\\
18 & 20 & 9 & 12 & 17 & 13 & 4 & 16 & 5 & 2 & 19 & 11 & 3 & 14 & 7 & 1 & 8 & 10 & 15 & 6 & 781\\
19 & 18 & 20 & 4 & 6 & 15 & 11 & 8 & 3 & 13 & 5 & 9 & 16 & 12 & 1 & 2 & 7 & 17 & 14 & 10 & 794\\
18 & 5 & 20 & 6 & 15 & 19 & 16 & 2 & 17 & 8 & 3 & 10 & 13 & 11 & 9 & 7 & 14 & 12 & 4 & 1 & 800\\
7 & 11 & 18 & 13 & 10 & 20 & 17 & 5 & 16 & 19 & 3 & 15 & 9 & 4 & 14 & 2 & 6 & 8 & 12 & 1 & 804\\
18 & 1 & 17 & 14 & 20 & 15 & 13 & 10 & 9 & 12 & 2 & 7 & 3 & 4 & 11 & 5 & 6 & 16 & 19 & 8 & 810\\
2 & 9 & 11 & 15 & 17 & 10 & 19 & 20 & 3 & 18 & 13 & 6 & 1 & 12 & 8 & 16 & 4 & 5 & 7 & 14 & 814\\
4 & 11 & 14 & 9 & 8 & 7 & 15 & 20 & 18 & 10 & 16 & 19 & 2 & 6 & 1 & 13 & 12 & 17 & 5 & 3 & 817\\
2 & 3 & 1 & 17 & 15 & 6 & 10 & 13 & 18 & 8 & 16 & 12 & 5 & 9 & 20 & 19 & 14 & 4 & 7 & 11 & 822\\
3 & 16 & 1 & 15 & 6 & 2 & 5 & 19 & 4 & 13 & 18 & 11 & 8 & 10 & 12 & 14 & 7 & 9 & 17 & 20 & 826\\
18 & 6 & 8 & 3 & 10 & 17 & 4 & 15 & 19 & 1 & 9 & 20 & 5 & 13 & 11 & 14 & 7 & 12 & 2 & 16 & 827\\
16 & 18 & 15 & 2 & 5 & 13 & 12 & 6 & 4 & 19 & 11 & 3 & 20 & 10 & 9 & 7 & 17 & 8 & 1 & 14 & 836\\
3 & 20 & 4 & 7 & 1 & 6 & 8 & 11 & 12 & 9 & 19 & 14 & 18 & 17 & 10 & 16 & 15 & 5 & 2 & 13 & 843\\
14 & 18 & 4 & 2 & 17 & 19 & 12 & 3 & 5 & 1 & 13 & 9 & 8 & 6 & 16 & 10 & 7 & 20 & 11 & 15 & 847\\
9 & 8 & 12 & 18 & 11 & 1 & 4 & 17 & 3 & 5 & 6 & 20 & 7 & 15 & 13 & 10 & 2 & 14 & 19 & 16 & 851\\
10 & 18 & 4 & 15 & 6 & 9 & 3 & 1 & 20 & 7 & 11 & 8 & 14 & 16 & 5 & 13 & 2 & 19 & 12 & 17 & 852\\
5 & 16 & 14 & 2 & 12 & 19 & 1 & 6 & 8 & 20 & 13 & 9 & 7 & 4 & 15 & 11 & 10 & 3 & 18 & 17 & 858
\end{pmatrix}$$

$$Random-HM=\begin{pmatrix}
19 & 11 & 8 & 16 & 15 & 13 & 14 & 20 & 1 & 18 & 3 & 12 & 6 & 5 & 17 & 10 & 9 & 7 & 4 & 2 & 852\\
18 & 15 & 19 & 4 & 14 & 7 & 13 & 3 & 8 & 20 & 12 & 1 & 2 & 16 & 11 & 10 & 6 & 17 & 5 & 9 & 902\\
1 & 3 & 10 & 14 & 17 & 18 & 16 & 4 & 9 & 5 & 8 & 15 & 11 & 19 & 2 & 12 & 13 & 20 & 7 & 6 & 872\\
11 & 7 & 8 & 5 & 19 & 12 & 4 & 14 & 15 & 6 & 10 & 16 & 9 & 3 & 20 & 17 & 1 & 18 & 13 & 2 & 852\\
6 & 14 & 10 & 20 & 8 & 18 & 19 & 1 & 9 & 13 & 2 & 15 & 3 & 4 & 16 & 7 & 11 & 12 & 17 & 5 & 845\\
15 & 2 & 19 & 20 & 18 & 16 & 9 & 17 & 14 & 10 & 3 & 5 & 12 & 11 & 1 & 4 & 6 & 13 & 7 & 8 & 778\\
15 & 20 & 11 & 13 & 8 & 6 & 7 & 5 & 14 & 4 & 16 & 19 & 3 & 1 & 2 & 18 & 12 & 17 & 9 & 10 & 786\\
17 & 1 & 2 & 19 & 10 & 14 & 16 & 6 & 12 & 4 & 3 & 5 & 7 & 11 & 9 & 20 & 13 & 15 & 8 & 18 & 799\\
3 & 8 & 11 & 15 & 6 & 12 & 7 & 1 & 20 & 17 & 5 & 16 & 19 & 9 & 2 & 10 & 14 & 18 & 13 & 4 & 826\\
13 & 5 & 1 & 9 & 2 & 19 & 17 & 20 & 15 & 10 & 4 & 6 & 18 & 11 & 14 & 16 & 8 & 3 & 12 & 7 & 934\\
1 & 3 & 17 & 11 & 2 & 15 & 16 & 8 & 12 & 18 & 13 & 5 & 9 & 6 & 4 & 7 & 14 & 20 & 10 & 19 & 885\\
7 & 4 & 2 & 10 & 1 & 13 & 9 & 20 & 17 & 15 & 3 & 16 & 5 & 14 & 6 & 11 & 19 & 12 & 8 & 18 & 874\\
13 & 2 & 5 & 10 & 7 & 17 & 15 & 18 & 14 & 1 & 20 & 12 & 19 & 11 & 9 & 8 & 16 & 6 & 3 & 4 & 860\\
2 & 20 & 14 & 12 & 3 & 6 & 18 & 19 & 8 & 17 & 5 & 4 & 7 & 9 & 11 & 15 & 13 & 16 & 10 & 1 & 884\\
16 & 11 & 12 & 3 & 13 & 14 & 8 & 2 & 18 & 15 & 17 & 19 & 5 & 7 & 6 & 1 & 4 & 9 & 20 & 10 & 913\\
9 & 2 & 20 & 12 & 13 & 10 & 16 & 14 & 15 & 5 & 8 & 11 & 4 & 17 & 3 & 1 & 19 & 7 & 6 & 18 & 822\\
15 & 11 & 6 & 20 & 9 & 10 & 16 & 1 & 5 & 8 & 7 & 3 & 2 & 12 & 18 & 14 & 19 & 13 & 4 & 17 & 785\\
2 & 20 & 14 & 11 & 17 & 4 & 19 & 18 & 9 & 5 & 12 & 1 & 13 & 8 & 16 & 6 & 3 & 7 & 10 & 15 & 854\\
14 & 13 & 1 & 18 & 10 & 19 & 4 & 8 & 20 & 12 & 5 & 6 & 2 & 16 & 17 & 15 & 3 & 9 & 7 & 11 & 917\\
9 & 2 & 20 & 8 & 15 & 14 & 13 & 7 & 5 & 18 & 12 & 19 & 17 & 3 & 11 & 4 & 10 & 1 & 6 & 16 & 965\\
5 & 16 & 14 & 2 & 12 & 19 & 1 & 6 & 8 & 20 & 13 & 9 & 7 & 4 & 15 & 11 & 10 & 3 & 18 & 17 & 858
\end{pmatrix}$$

figure. 4.6 – Initialisation OBL vs initialisation aléatoire

Puisque à chaque itération l'algorithme HS est mis à jour en supprimant la pire harmonie si la nouvelle est meilleure, alors l'initialisation OBL aide à accélérer le comportement de la recherche.

Afin de valider les résultats obtenus, l'algorithme HS est comparé avec trois approches de la littérature : l'algorithme GA (Genetic algorithm) proposé par oğuz et Ercan [94], l'algorithme PGA (parallel greedy algorithm) développé par Kahraman et al. [98], et l'algorithme EGA (efficient genetic algorithm) présenté par Engin et al. [99].

Le tableau 4.3 présente les résultats numériques des différents algorithmes, les performances des algorithmes sont mesurées en terme de qualité de solution représentée par la valeur du pourcentage de déviation (PD).
Pour estimer la valeur PD des différents algorithmes, la borne inférieure (LB) présentée par oğuz [101] est utilisée. Cette borne est basée sur les étages, en considérant une borne inférieure du problème de tâches multiprocesseurs avec un seul étage. La formule de la borne inférieure est présentée comme suit :

$$LB = \max_{i \in S}\{\min_{j \in J}\{\sum_{k=1}^{i-1} p_{ji}\} + \frac{1}{m_i}\sum_{j \in J} p_{ji} size_{ij} + \min_{j \in J}\{\sum_{k=i+1}^{s} p_{ji}\}\}$$

où J représente l'ensemble de jobs et S représente l'ensemble des étages.
Le pourcentage de déviation (PD) de la solution par rapport à la borne inférieure (LB) est défini comme suit :

$$PD = \frac{BestC_{max} - LB}{LB} \times 100 \tag{4.1}$$

Pour chaque algorithme, la valeur de (PD) pour chaque problème est donnée. Nous obtenons les résultats de calcul des algorithmes GA, EGA et PGA à partir de leurs articles originaux.

Table. 4.3 – Les résultats obtenus par les algorithmes HS, EGA, PGA et GA

Problème	PD				Problème	PD				Problème	PD			
	GA	PGA	EGA	HS		GA	PGA	EGA	HS		GA	PGA	EGA	HS
P20S2T01	1.997	1.690	2.611	1.997	P20S5T01	1.567	1.567	1.567	1.567	P20S8T01	8.163	9.448	9.599	**4.913**
P20S2T02	2.368	1.275	1.639	**0.911**	P20S5T02	7.165	6.858	7.165	**6.567**	P20S8T02	0.000	0.000	0.000	0.000
P20S2T03	0.000	0.000	0.000	0.000	P20S5T03	2.561	6.170	2.561	2.561	P20S8T03	2.573	4.117	2.573	**1.715**
P20S2T04	0.000	0.000	0.000	0.000	P20S5T04	2.268	0.113	0.000	0.000	P20S8T04	1.593	3.805	7.080	3.894
P20S2T05	0.000	0.000	0.000	0.000	P20S5T05	2.313	3.049	1.682	1.682	P20S8T05	3.152	2.641	2.300	2.300
P20S2T06	0.000	0.000	0.000	0.000	P20S5T06	2.650	2.479	1.880	1.880	P20S8T06	8.163	9.675	4.913	4.913
P20S2T07	0.341	0.341	0.341	0.341	P20S5T07	0.000	0.000	0.000	0.000	P20S8T07	23.795	27.487	28.821	24.205
P20S2T08	0.278	0.278	0.278	0.278	P20S5T08	3.519	3.519	3.128	3.128	P20S8T08	3.791	2.729	3.791	3.791
P20S2T09	0.000	0.000	0.000	0.000	P20S5T09	0.000	0.000	0.000	0.000	P20S8T09	0.000	0.000	0.000	0.000
P20S2T10	0.000	0.000	0.000	0.000	P20S5T10	7.165	7.165	6.858	6.858	P20S8T10	4.117	4.803	3.945	**1.715**
P50S2T01	1.753	1.088	0.605	**0.484**	P50S5T01	0.911	0.835	0.911	0.911	P50S8T01	2.709	6.578	5.288	**2.665**
P50S2T02	2.368	1.639	0.9107	**0.000**	P50S5T02	4.539	0.532	1.099	0.532	P50S8T02	2.750	2.750	0.000	0.000
P50S2T03	0.590	0.215	0.000	0.000	P50S5T03	0.998	0.998	0.998	0.998	P50S8T03	0.936	1.161	0.787	**0.487**
P50S2T04	1.238	1.423	0.681	0.681	P50S5T04	0.618	4.325	2.523	1.287	P50S8T04	4.760	3.917	2.696	**2.612**
P50S2T05	0.353	0.588	0.235	**0.059**	P50S5T05	2.577	0.000	0.000	0.000	P50S8T05	3.639	2.691	0.986	**0.872**
P50S2T06	0.000	0.000	0.000	0.000	P50S5T06	0.000	0.000	0.000	0.000	P50S8T06	1.875	1.641	0.313	**0.000**
P50S2T07	2.716	2.130	0.692	**0.532**	P50S5T07	1.100	0.477	0.513	**0.329**	P50S8T07	5.436	3.701	2.544	**2.120**
P50S2T08	0.000	0.000	0.000	0.000	P50S5T08	2.447	0.745	1.099	**0.532**	P50S8T08	2.434	1.917	1.881	**0.184**
P50S2T09	0.000	0.000	0.000	0.000	P50S5T09	5.096	0.668	0.000	0.000	P50S8T09	4.760	4.465	3.412	**3.328**
P50S2T10	0.262	0.052	0.052	0.052	P50S5T10	0.271	1.264	0.587	**0.225**	P50S8T10	5.371	4.726	3.187	**2.220**
P1HS2T01	0.534	0.507	0.721	**0.053**	P1HS5T01	3.493	0.056	0.000	0.000	P1HS8T01	2.877	1.509	0.000	0.000
P1HS2T02	0.000	0.000	0.000	0.000	P1HS5T02	1.543	0.000	0.000	0.000	P1HS8T02	3.568	0.801	0.157	0.157
P1HS2T03	0.814	0.603	0.693	**0.030**	P1HS5T03	3.819	2.272	0.674	**0.624**	P1HS8T03	1.987	1.472	0.644	**0.532**
P1HS2T04	0.000	0.000	0.000	0.000	P1HS5T04	1.425	1.548	1.130	**0.393**	P1HS8T04	2.149	0.498	0.103	0.103
P1HS2T05	0.000	0.000	0.000	0.000	P1HS5T05	2.347	0.000	0.000	0.000	P1HS8T05	1.944	0.731	0.019	**0.000**
P1HS2T06	0.000	0.000	0.000	0.000	P1HS5T06	3.591	4.488	2.924	**2.180**	P1HS8T06	3.422	2.907	1.104	**0.662**
P1HS2T07	0.926	0.538	0.657	**0.090**	P1HS5T07	0.300	0.000	0.000	0.000	P1HS8T07	2.426	1.203	0.019	**0.000**
P1HS2T08	0.020	0.000	0.000	0.000	P1HS5T08	0.673	0.000	0.000	0.000	P1HS8T08	7.294	8.451	5.584	5.936
P1HS2T09	0.105	0.394	0.683	**0.026**	P1HS5T09	1.379	0.000	0.000	0.000	P1HS8T09	1.951	1.334	0.020	**0.000**
P1HS2T10	0.097	0.242	0.097	**0.000**	P1HS5T10	3.248	3.389	2.994	**2.881**	P1HS8T10	3.838	0.985	0.185	0.185

D'après le tableau 4.3, l'algorithme HS peut résoudre optimalement (LB atteinte) 37 problèmes parmi 90 problèmes, tandis que les algorithmes EGA et PGA ne peuvent résoudre que 31 et 25 problèmes respectivement. L'algorithme GA peut résoudre uniquement 18 instances. Le nombre et le pourcentage de problèmes résolus (% problèmes résolus) pour chaque algorithme sont donnés dans le tableau 4.4 et la figure 4.7.

Table. 4.4 – Les performances de HS, EGA, PGA, et GA

Méthode	Nombre de Problèmes résolus	% Problèmes résolus
HS	37	41.11
EGA	31	34.44
PGA	25	27.78
GA	18	20.00

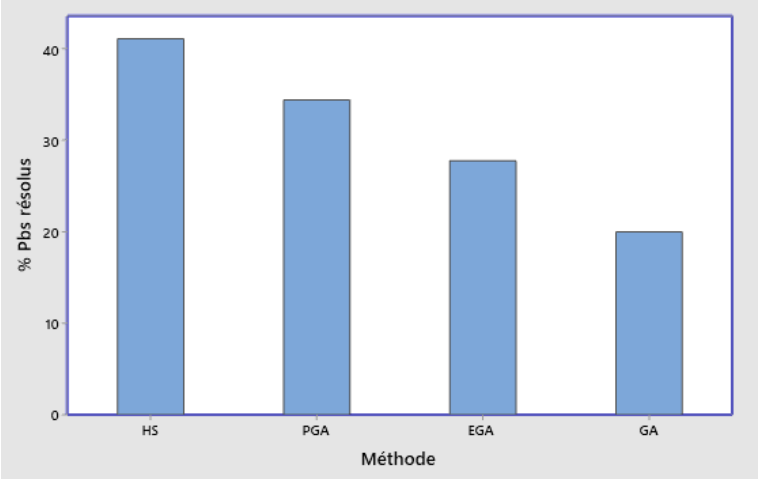


figure. 4.7 – Pourcentage des problèmes résolus pour les différents algorithmes

Comme le montre le tableau 4.3, HS fournit de meilleures solutions pour 35 instances ce qui représente 38.89% du nombre total des problèmes. Pour les deux algorithmes PGA et GA, ils trouvent chacun de meilleures solutions à 3 problèmes (3.33% du nombre total des problèmes), alors que EGA ne peut trouver qu'une seule meilleure solution sur tous les problèmes considérés (1.11%). Par ailleurs, les performances de tous les algorithmes sont similaires pour 22 problèmes.

La figure 4.8 montre les valeurs de PD pour les algorithmes HS, EGA, PGA et GA sur différents types d'instances. D'après les résultats obtenus, la méthode HS est plus performante que les autres méthodes. La qualité des solutions trouvées par HS est supérieure et ceci pour la majorité des instances.

Pour chaque catégorie d'instances, le tableau 4.5 donne la moyenne du pourcentage de déviation (APD) :

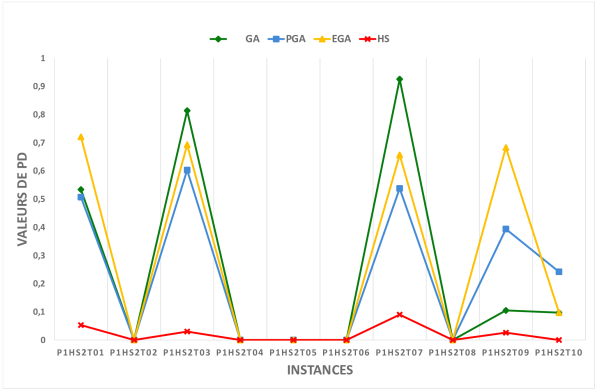
$$APD = [\sum_{i=1}^{10}(\frac{BestC_{max}(i) - LB(i)}{LB(i)}) \times 100\ ]/10 \tag{4.2}$$

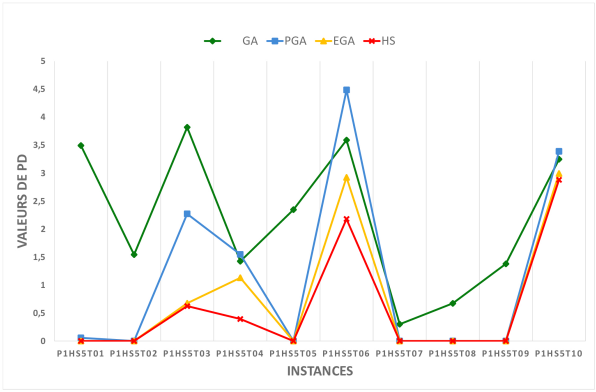
Où $BestC_{max}(i)$ et $LB(i)$ sont respectivement la meilleure valeur du makespan et la borne inférieure de l'instance i.

Table. 4.5 – Les résultats comparatifs de APD pour les différents algorithmes

s	n	APD				$CPU(s)$	
		GA	PGA	EGA	HS	HS	GA
2	20	0.50	0.36	0.48	**0.35**	0.22	26.10
2	50	0.93	0.71	0.32	**0.18**	2.68	178.22
2	100	0.25	0.23	0.29	**0.02**	18.70	567.14
Average		0.56	0.43	0.36	**0.18**		
5	20	0.92	3.09	2.48	2.42	0.60	47.56
5	50	1.86	0.98	0.77	**0.48**	6.25	367.13
5	100	2.18	1.18	0.77	**0.61**	47.17	1024.87
Average		2.32	1.75	1.34	**1.17**		
8	20	5.53	6.47	6.30	**4.74**	1.09	103.26
8	50	3.47	3.35	2.11	**1.45**	31.60	553.87
8	100	3.15	1.99	0.78	**0.76**	82.37	1899.03
Average		4.05	3.94	3.06	**2.32**		
Total Average		6.93	6.12	4.76	**3.67**	21.52	529.68

Pour les problèmes à 2 étages, les résultats du tableau 4.5 indiquent que les valeurs de APD pour l'algorithme HS sont significativement meilleures que celles pour PGA, EGA et GA. Ainsi, la moyenne de APD pour HS est de $0,18$ alors que les moyennes de APD pour EGA,





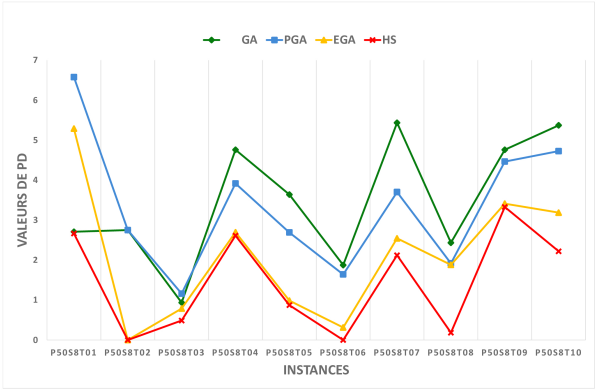


figure. 4.8 – Valeurs de PD pour les algorithmes HS, EGA, PGA et GA sur différentes types d'instances

PGA et GA sont de $0,36$, $0,43$ et $0,56$ respectivement.

En ce qui concerne les problèmes à 5 étages, l'algorithme HS fonctionne mieux que les autres algorithmes comparés. Selon le tableau 4.5, la moyenne de APD pour HS est de $1,17$ tandis que les valeurs moyennes de APD pour EGA, PGA et GA sont respectivement de $1,34$, $1,75$ et $2,32$.

L'algorithme HS fonctionne mieux également sur les problèmes à 8 étages, où la valeur moyenne de APD pour HS est de $2,32$ contre $3,06$ pour EGA, $3,94$ pour PGA et $4,05$ pour GA. Ainsi, si on considère tous les problèmes, le tableau 4.5 montre que la moyenne totale de déviation obtenue par HS (3.67) est bien meilleure que celles obtenues par les algorithmes EGA $(4,76)$, PGA $(6,12)$ et GA $(6,93)$.
Ces résultats démontre l'efficacité de la méthode proposée HS par rapport aux autres méthodes.

De plus, pour fournir une comparaison plus précise des résultats, on définit l'amélioration relative moyenne de l'algorithme HS par rapport aux autres méthodes (Method) de la façon suivante :

$$ARI = \frac{APD_{Method} - APD_{HS}}{APD_{Method}} \times 100 \tag{4.3}$$

Dans le tableau 4.6, ARI représente l'amélioration relative moyenne de l'algorithme HS par rapport aux résultats de l'algorithme EGA.

Ce tableau indique que la plupart des améliorations (les valeurs de ARI) par rapport à EGA sont importantes. Nous observons que pour les problèmes avec $n = 20$, le taux d'amélioration peut aller jusqu'à 27.08%. Ce taux est important également si $n = 50$, il varie entre 31.38% et 43.75%. Pour les problèmes avec $n = 100$, HS fournit de meilleures solutions avec un taux d'amélioration allant jusqu'à 93.10% par rapport à la méthode EGA.

En ce qui concerne les temps CPU, le tableau 4.7 donne les temps maximums en minutes alloués à la recherche pour chaque algorithme. À ce niveau, nous nous intéressons à comparer HS avec les algorithmes génétiques GA et EGA, mais comme les résultats pour EGA sont représentés seulement en terme du makespan, alors HS est comparé avec l'algorithme GA.

Le tableau 4.5 présente les valeurs de CPU pour les deux algorithmes HS et GA. Selon ce tableau les valeurs moyennes de CPU sont conditionnées par le nombre de jobs n et le nombre d'étage s. Si par exemple on fixe n, nous remarquons une augmentation dans les valeurs de

Table. 4.6 – Taux d'amélioration ARI de HS par rapport à EGA

$n \times s$	APD		ARI
	EGA	HS	
20×2	0.48	0.35	27.08
20×5	2.48	2.42	2.42
20×8	6.30	4.74	24.76
50×2	0.32	0.18	43.75
50×5	0.77	0.48	37.66
50×8	2.11	1.45	31.28
100×2	0.29	0.02	93.10
100×5	0.77	0.61	20.78
100×8	0.78	0.76	2.56

CPU lorsque s augmente, de même si on fixe s et on augmente n, les temps CPU augmentent. Alors les temps CPU dépendent de la difficulté des problèmes.

Puisque les algorithmes HS et GA sont exécutés sur des ordinateurs différents, alors nous adoptons la méthode de correction présentée dans le chapitre précédent. Les résultats affirment que l'algorithme HS donne des temps de calcul beaucoup plus petits que ceux obtenus par GA, et ceci pour toutes les instances considérées.

Table. 4.7 – Les temps CPU pour HS, GA, PGA, et EGA

Méthode	Configuration de l'ordinateur	CPU maximum (min)
HS	PC Intel core i3-2.1 GHz-3GB RAM	1.25
EGA	PC Pentium 4-3GHz-512MB RAM	25
PGA	PC Pentium 4-3GHz-512MB RAM	25
GA	PC Pentium 4-2GHz-256MB RAM	30

Afin de tester la performance de notre approche par rapport à des travaux publiés récemment, nous avons décidé de comparer l'approche proposée avec deux approches récentes de la littérature. Il s'agit de l'algorithme MGLS (Memetic global and local search)[65], et l'algorithme ACSNDP (Ant colony system with a novel Non-DaemonActions procedure)[84].

Nous allons d'abord présenter les résultats de comparaison de HS et MGLS sur les problèmes benchmarks considérés dans notre travail, ensuite nous présentons la comparaison entre HS et ACSNDP. Les résultats de calcul des algorithmes MGLS et ACSNDP sont obtenus à partir de leurs articles originaux.
Le tableau 4.8 présente les résultats obtenus par les deux algorithmes HS et MGLS, pour chaque instance le pourcentage de déviation PD est calculé. D'après ce tableau, l'algorithme HS peut résoudre 37 problèmes d'une manière optimale, tandis que l'algorithmes MGLS peut résoudre seulement 33 problèmes. Le nombre d'instances résolues par les deux algorithmes HS et MGLS pour les différents types de problèmes est illustré dans la figure 4.9.

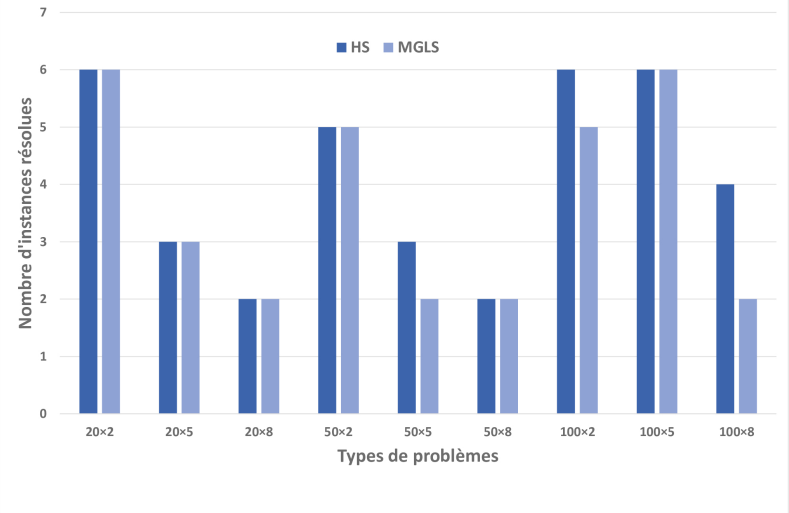


figure. 4.9 – Nombre de problèmes résolus par HS et MGLS

Selon le tableau 4.8, nous remarquons que l'algorithme HS donne de meilleurs résultats pour 34 instances (37.78%), et MGLS trouve de meilleurs résultats à 6 instances (6.67%), ce qui démontre que l'algorithme HS est plus efficace que l'algorithme MGLS. Les résultats obtenus par les deux algorithmes sont similaires pour les instances restantes.

Le tableau 4.9 représente les valeurs moyennes de déviation APD pour les deux algorithmes HS et MGLS sur les différents types de problèmes. La comparaison entre ces valeurs est illustrée dans la figure 4.10.

Table. 4.8 – Les résultats obtenus par les méthodes HS et MGLS

Problème	PD		Problème	PD		Problème	PD	
	MGLS	HS		MGLS	HS		MGLS	HS
P20S2T01	2.611	**1.997**	P20S5T01	1.567	1.567	P20S8T01	4.913	4.913
P20S2T02	1.639	**0.911**	P20S5T02	7.165	**6.567**	P20S8T02	0.000	0.000
P20S2T03	0.000	0.000	P20S5T03	2.561	2.561	P20S8T03	2.573	**1.715**
P20S2T04	0.000	0.000	P20S5T04	0.000	0.000	P20S8T04	3.628	3.894
P20S2T05	0.000	0.000	P20S5T05	1.893	**1.682**	P20S8T05	2.641	**2.300**
P20S2T06	0.000	0.000	P20S5T06	2.137	**1.880**	P20S8T06	5.291	**4.913**
P20S2T07	0.341	0.341	P20S5T07	0.000	0.000	P20S8T07	26.974	**24.205**
P20S2T08	0.278	0.278	P20S5T08	3.128	3.128	P20S8T08	3.791	3.791
P20S2T09	0.000	0.000	P20S5T09	0.000	0.000	P20S8T09	0.000	0.000
P20S2T10	0.000	0.000	P20S5T10	7.165	**6.858**	P20S8T10	4.031	**1.715**
P50S2T01	0.605	**0.484**	P50S5T01	0.911	0.911	P50S8T01	2.880	**2.665**
P50S2T02	0.000	0.000	P50S5T02	0.390	0.532	P50S8T02	0.000	0.000
P50S2T03	0.000	0.000	P50S5T03	0.998	0.998	P50S8T03	0.075	0.487
P50S2T04	0.681	0.681	P50S5T04	2.781	**1.287**	P50S8T04	2.148	2.612
P50S2T05	0.235	**0.059**	P50S5T05	0.000	0.000	P50S8T05	0.796	0.872
P50S2T06	0.000	0.000	P50S5T06	0.000	0.000	P50S8T06	0.000	0.000
P50S2T07	0.692	**0.532**	P50S5T07	0.330	0.329	P50S8T07	2.120	2.120
P50S2T08	0.000	0.000	P50S5T08	1.099	**0.532**	P50S8T08	0.221	**0.184**
P50S2T09	0.000	0.000	P50S5T09	0.209	**0.000**	P50S8T09	3.665	**3.328**
P50S2T10	0.052	0.052	P50S5T10	0.316	**0.225**	P50S8T10	2.363	**2.220**
P1HS2T01	0.107	**0.053**	P1HS5T01	0.000	0.000	P1HS8T01	0.664	**0.000**
P1HS2T02	0.000	0.000	P1HS5T02	0.000	0.000	P1HS8T02	0.157	0.157
P1HS2T03	0.814	**0.030**	P1HS5T03	1.248	**0.624**	P1HS8T03	0.405	0.532
P1HS2T04	0.000	0.000	P1HS5T04	1.376	**0.393**	P1HS8T04	0.103	0.103
P1HS2T05	0.000	0.000	P1HS5T05	0.000	0.000	P1HS8T05	0.000	0.000
P1HS2T06	0.000	0.000	P1HS5T06	3.078	**2.180**	P1HS8T06	1.270	**0.662**
P1HS2T07	0.837	**0.090**	P1HS5T07	0.000	0.000	P1HS8T07	0.000	0.000
P1HS2T08	0.000	0.000	P1HS5T08	0.000	0.000	P1HS8T08	7.193	**5.936**
P1HS2T09	0.499	**0.026**	P1HS5T09	0.000	0.000	P1HS8T09	0.319	**0.000**
P1HS2T10	0.000	0.000	P1HS5T10	3.417	**2.881**	P1HS8T10	0.185	0.185

Table. 4.9 – Les taux d'amélioration de HS par rapport à MGLS

$n \times s$	APD		ARI
	HS	MGLS	
20×2	0.35	0.48	27.08
20×5	2.42	2.56	5.47
20×8	4.74	5.38	11.89
50×2	0.18	0.22	18.18
50×5	0.48	0.70	31.43
50×8	1.45	1.43	-1.40
100×2	0.02	0.23	93.30
100×5	0.61	1.91	32.97
100×8	0.76	1.02	25.49

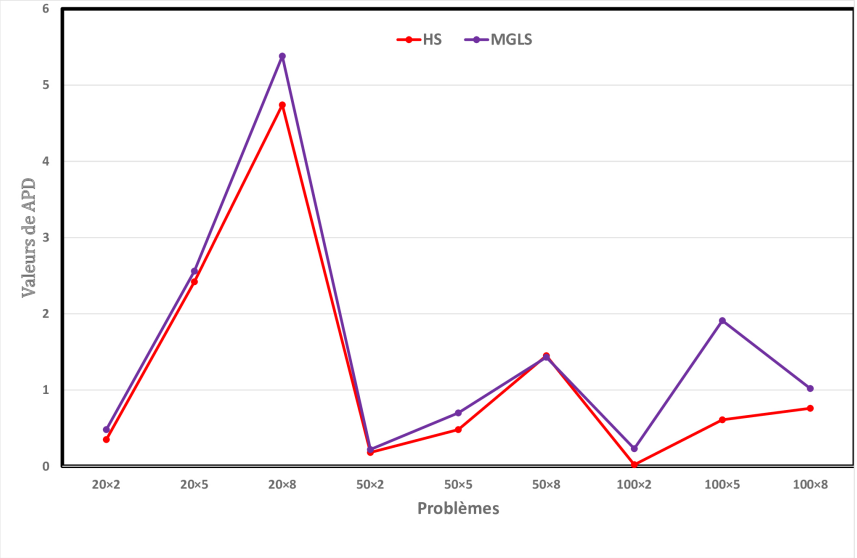


figure. 4.10 – Comparaison des valeurs APD pour HS et MGLS

En même temps, le tableau 4.9 contient les taux d'amélioration ARI de HS par rapport à

MGLS. Les résultats de ce tableau indiquent que pour les problèmes (20×2), (100×2), (50×5), (100×5) et (100×8), les taux d'amélioration ARI sont importants avec des valeurs allant jusqu'à 91.30%, ceci signifie que les résultats trouvés par notre approche améliorent considérablement ceux trouvés par l'approche MGLS. Les résultats obtenus par HS pour les problèmes (20×5), (20×8) et (50×2) sont aussi meilleurs avec un taux d'amélioration allant jusqu'à 18.18%. MGLS est meilleur (ARI négatif) pour les problèmes (50×8), mais le tableau 4.8 montre que HS améliore 4 instances, tandis que MGLS améliore 3 instances seulement parmi les 10 instances de ce type de problèmes.

Nous passons maintenant à comparer notre approche avec l'algorithme ACNDSP. Le tableau 4.10 illustre les valeurs moyennes de déviation APD pour les deux algorithmes HS et ACNDSP, et les taux d'amélioration ARI sur les différents types de problèmes. Selon ce tableau, nous observons que l'algorithme HS donne des valeurs de APD meilleures que celles données par ACNDSP, et ceci pour tous les problèmes. La comparaison entre les valeurs de APD de HS et ACNDSP est illustrée dans la figure 4.11.

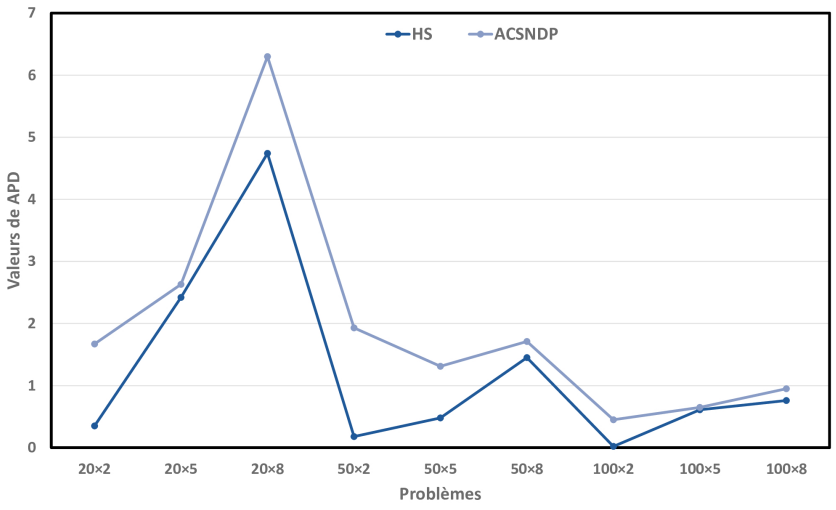


figure. 4.11 – Comparaison des valeurs APD pour HS et ACNDSP

À partir du tableau 4.10, nous remarquons que notre approche améliore les résultats trouvés par la méthode ACNDSP. Pour tous les jeux-tests considérés, les taux d'amélioration sont importants surtout pour les problèmes (20×2),(50×2),(50×8) et (100×2), avec des valeurs

allant jusqu'à 95.55%.
Selon les différentes expérimentations et comparaisons réalisées, nous pouvons affirmer l'efficacité de l'approche proposée pour résoudre les problèmes d'ordonnancement de type flow shop hybride avec tâches multiprocesseurs.

Table. 4.10 – Les taux d'amélioration de HS par rapport à ACSNDP

$n \times s$	APD		ARI
	ACSNDP	HS	
20×2	1.67	0.35	79.04
20×5	2.63	2.42	7.98
20×8	6.30	4.74	24.76
50×2	1.93	0.18	90.67
50×5	1.31	0.48	63.36
50×8	1.71	1.45	15.20
100×2	0.45	0.02	95.55
100×5	0.65	0.61	6.15
100×8	0.95	0.76	20.00

4.7 Conclusion

Ce chapitre a abordé la résolution du problème du flow shop hybride avec tâches multiprocesseurs en minimisant le makespan. Le problème HFSMT est NP-difficile au sens fort, il généralise le problème du flow shop hybride. Il est clair que dans le cas particulier où $size_{ij} = 1$ pour tous les jobs, le problème HFSMT se réduit à un problème HFS.
Puisque les méthodes approchées constituent des alternatives importantes qui permettent de fournir des bonnes solutions dans des temps raisonnables, alors nous avons proposé dans ce chapitre une métaheuristique à base de la recherche d'harmonie pour résoudre le problème. L'algorithme de la recherche d'harmonie imite la performance musicale qui consiste à trouver une harmonie parfaite déterminée par une norme esthétique, tout comme, le processus d'optimisation qui cherche à trouver une solution optimale globale.
Dans ce chapitre, nous avons commencé premièrement par donner un état de l'art sur le problème HFSMT. Puisque la recherche d'harmonie est à l'origine conçue pour résoudre des problèmes continus, alors nous l'avons adapté à notre problème dans le but de surmonter sa nature discrète. Nous avons utilisé la méthode OBL qui est basée sur le concept d'opposition

pour initialiser l'algorithme de recherche d'harmonie. Aussi, nous avons développé des règles d'improvisation basées sur des opérateurs évolutionnaires. Pour améliorer la performance de l'algorithme, nous avons appliqué une stratégie de recherche locale basée sur deux structures de voisinage différentes. La méthode de Taguchi a été utilisée pour déterminer les meilleures valeurs des paramètres. Les résultats obtenus montrent la supériorité de l'approche proposée.

Conclusion et perspectives

Le problème du flow shop hybride est d'un grand intérêt théorique en raison de sa complexité. De plus, l'environnement de production flow shop hybride se rencontre souvent dans les systèmes de production, ce qui révèle également son importance pratique.
L'objectif de ce travail est d'apporter des contributions à la résolution des problèmes d'ordonnancement dans les systèmes de production de type flow shop hybride.

Les travaux présentés dans cette thèse s'articulent sur deux préoccupations principales :

— Développer des approches efficaces d'optimisation pour résoudre le problème du flow shop hybride et le problème du flow shop hybride avec tâches multiprocesseurs en considérant la minimisation du makespan.
— Tester les performances des approches proposées en réalisant des comparaisons avec d'autres approches de la littérature.

Dans les deux premiers chapitres de cette thèse, nous avons fourni une présentation générale sur la gestion des systèmes de production et particulièrement les problèmes d'ordonnancement dans les systèmes de production. Ensuite nous avons présenté un état de l'art contenant une grande partie des travaux de la littérature qui ont été proposés pour résoudre le problème du flow shop hybride. Aussi, nous avons introduit les différentes techniques d'optimisation utilisées dans le domaine de l'optimisation combinatoire.

Nous avons développé dans ce travail des méthodes hybrides d'optimisation basées sur l'algorithme d'optimisation par essaims particulaires, la recherche d'harmonie et des stratégies de recherche locale pour résoudre les problèmes traités.
L'algorithme de la recherche d'harmonie présente certaines similarités avec l'algorithme PSO dans le sens qu'il manipule un ensemble de solutions qui interagissent entre elles pour trouver la meilleure solution possible du problème. Mais au niveau de l'évolution de la population, l'algorithme PSO adopte une stratégie de collaboration entre les particules, tandis que HS s'évolue en appliquant des règles d'improvisation.
Les algorithmes PSO et HS ont prouvé leurs performances pour des problèmes d'optimisation

continue, les versions discrètes proposées de PSO et HS sont destinées à garder les structures générales des deux algorithmes afin de profiter de leurs propriétés.
La résolution du problème du flow shop hybride a fait l'objet du troisième chapitre, l'algorithme d'optimisation par essaims particulaires proposé utilise des opérateurs évolutionnistes (mutation et croisement) avec une procédure de recherche locale pour explorer l'espace de recherche.
Dans le dernier chapitre, notre étude s'est portée sur la résolution du problème du flow shop hybride avec tâches multiprocesseurs. Nous avons développé un algorithme de recherche d'harmonie qui utilise également des techniques d'évolution spéciales.
Les expérimentations effectuées ont prouvé l'efficacité des approches proposées dans cette thèse.

Finalement, Plusieurs perspectives intéressantes restent à développer dans les futurs travaux que nous envisageons :

— Traiter d'autres variantes du problème, comme par exemple le problème du flow shop hybride réentrant où chaque tâche doit revisiter l'atelier de production un ou plusieurs fois, et la variante avec un espace d'attente limité où la gestion des files d'attente doit prendre en considération cette limite.
— Développer la notion du croisement multi-parents, qui nous a inspiré la règle de considération de la mémoire.
— Comparer les performances d'optimisation par essaims particulaires et la recherche d'harmonie sur des problèmes d'ordonnancement.

Les travaux présentés dans cette thèse sont basés sur des publications dans des revues internationales [78][81], et des communications dans des conférences internationales [77][79].

Bibliographie

[1] Stephen A. Cook. The complexity of theorem-proving procedures STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing May 1971, Pages 151-158.

[2] Agnès Letouzey. Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs. PhD thesis, Institut National Polytechnique de Toulouse, 2001.

[3] GOTHA. Les problèmes dordonnancement. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 27, no 1 (1993), pp. 77-150.

[4] P. Esquirol, P. Lopez. L'ordonnancement. Economica, Paris, (1999).

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. Annals of discrete mathematics, 5 (1979), pp. 287-326. The complexity of theorem-proving procedures STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing May 1971, pp. 151-158

[6] Maart. B.J. Lagweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Computer-Aided Complexity Classification of Deterministic Scheduling Problems. Report No. BW138, Matematisch Centrum, Amsterdam. 1981.

[7] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, J. Węglarz. Scheduling computer and manufacturing processes, Springer-Verlag, 1996.

[8] Jatinder N.D Gupta. Two stage, hybrid flowshop scheduling problem. Journal Operational Research Society. Vol. 39, No. 4 (1988), pp : 359-364.

[9] J.A. Hoogeveen, J.K. Lenstra, B. Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. European Journal of Operational Research 89 (1) (1996), pp : 172-175.

[10] G.-C. Lee, Y.-D. Kim. A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. International Journal of Production Research, 42 (22) (2004), pp. 4731-4743.

[11] H. Allaoui, A. Artiba. Scheduling two-stage hybrid flow shop with availability constraints. Computers and Operations Research 33 (2006), pp. 1399-1419.

[12] M. Haouari, L. Hidri, A. Gharbi. Optimal scheduling of a two-stage hybrid flow shop, Mathematical Methods of Operations Research 64 (1) (2006), pp. 107-124.

[13] H.S. Choi, D.H. Lee. Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops. Computers and Industrial Engineering 56 (1) (2009), pp. 113-120.

[14] S.A. Brah, J.L. Hunsucker. Branch and bound algorithm for the flow-shop with multiple processors. European Journal of Operational Research 51 (1) (1991), pp. 88-99.

[15] C. Rajendran, D. Chaudhuri. A multistage parallel-processor flowshop problem with minimum flowtime. European Journal of Operational Research 57 (1) (1992), pp. 111-122.

[16] J. Carlier, E. Néron. An exact method for solving the multi-processor flowshop. RAIRO Recherche Opérationnelle, 34 (1) (2000), pp. 1-25.

[17] P. Fattahi, S. M. H. Hosseini, F. Jolai, R. Tavakkoli-Moghaddam. A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. Applied Mathematical Modelling Volume 38, Issue 1, 1 January 2014, pp. 119-134.

[18] C.Y. Lee, G.L. Vairaktarakis. Minimizing makespan in hybrid flowshops. Operations Research Letters 16 (3) (1994), pp. 149-158.

[19] J. N. D. Gupta, A. M. A. Hariri, C. N. Potts. Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. Annals of Operations Research, 69 (1997), pp. 171-191.

[20] A.G.P. Guinet, M.M. Solomon, P.K. Kedia, A. Dussauchoy. A computational study of heuristics for two-stage flexible flowshops. International Journal of Production Research 34 (5) (1996), pp. 1399-1415.

[21] S.A. Brah, L.L. Loo. Heuristics for scheduling in a flow shop with multiple processors. European Journal of Operational Research 113 (1) (1999), pp. 113-122.

[22] M. Haouari, R. M'Hallah. Heuristic algorithms for the two-stage hybrid flowshop problem. ELSEVIER Operations Research Letters, 21 (1997), pp. 43-53.

[23] J. Yang. A two-stage hybrid flow shop with dedicated machines at the first stage. Computers and OperationsResearch 40(2013), pp. 2836-2843.

[24] Carlos D. Paternina-Arboleda, Jairo R. Montoya-Torres, Milton J. Acero-Dominguez, Maria C. Herrera-Hernandez. Scheduling jobs on a k-stage flexible flow-shop. Ann Oper Res 164 (2008), pp. 29-40.

[25] Chun-Lung Chen, Chuen-Lung Chen. Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. Computers and Industrial Engineering. Volume 56, Issue 4, May 2009, pp. 1393-1401.

[26] Orhan Engin, Alper Döyen. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Generation Computer Systems 20 (2004), pp. 1083-1095.

[27] K. Alaykýran, O. Engin, A. Döyen. Using ant colony optimization to solve hybrid flow shop scheduling problems,Int J Adv Manuf Technol,35 (2007), pp. 541-550.

[28] Cengiz Kahraman, Orhan Engin, İhsan Kaya, Mustafa Kerim Yilmaz. An application of effective genetic algorithm for solving hybrid flow shop scheduling problems. International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), pp. 134-147.

[29] Q. Niu, T. Zhou, S. Ma. A Quantum-Inspired Immune Algorithm for Hybrid Flow Shop with Makespan Criterion, Journal of Universal Computer Science, 15 (2009), pp. 765-785.

[30] Q. Niu, T. Zhou, M. Fei, B. Wang. An effcient quantum immune algorithm to minimize mean flow time for hybrid flow shop problems. Mathematics and Computers in Simulation, 84 (2012), pp. 1-25.

[31] Ching-Jong Liao, Evi Tjandradjaja, Tsui-Ping Chung. An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. Applied Soft Computing 12 (2012), pp. 1755-1764.

[32] T.P. Chung, C.J. Liao. An immunoglobulin-based artificial immune system for solving the hybrid flow shop problem. Applied Soft Computing, 13 (2013), pp. 3729-3736.

[33] Wojciech Bożejko, Jarosław Pempera, Czesław Smutnicki. Parallel tabu search algorithm for the hybrid flow shop problem. Computers and Industrial Engineering 65 (2013), pp. 466-474.

[34] M.K. Marichelvam, T. Prabaharan, X.S. Yang. Improved cuckoo search algorithm for hybrid flow shop schedulingproblems to minimize makespan. Applied Soft Computing 19 (2014), pp. 93-101.

[35] Zhe Cui, Xingsheng Gu. An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. Neurocomputing 148(2015), pp. 248-259.

[36] Quan-Ke Pan, Ling Wang, Jun-Qing Li, Jun-Hua Duan. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega 45(2014), pp. 42-56.

[37] E.C. de Siqueira, M.J.F. Souza, S.R. de Souza. A Multi-objective Variable Neighborhood Search algorithm for solving the hybrid flow fhop problem. Electronic Notes in Discrete Mathematics 66 (2018), pp. 87-94.

[38] Victor Fernandez-Viagas, Paz Perez-Gonzalez, Jose M. Framinan. Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. Computers and Operations Research 109 (2019), pp. 77-88.

[39] Sebastian Lang, Tobias Reggelin, Johann Schmidt, Marcel Müller, Abdulrahman Nahhas. NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem : A comparison of different solution strategies. Expert Systems With Applications 172 (2021) 114666.

[40] Arnaud Malapert. Techniques d'ordonnancement d'atelier et de fournées basées sur la programmation par contraintes. Université de Nantes, Thèse de doctorat.

[41] Yang, XS. Harmony Search as a Metaheuristic Algorithm. In : Geem, Z.W. (eds) Music-Inspired Harmony Search Algorithm. Studies in Computational Intelligence, vol 191. Springer, Berlin. (2009).

[42] J.N.D. Gupta, K. Kruger, V. Lauff, F. Werner, Y.N. Sotskov. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. Computers and Operations Research, Volume 29, Issue 10, September 2002, Pages 1417-1439.

[43] Sörensen, K., Sevaux, M., Glover, F. (2018). A History of Metaheuristics. Handbook of Heuristics, 791-808.

[44] E. Néron, Ph. Baptiste, J.N.D. Gupta. Solving hybrid flow shop problem using energetic reasoning and global operations. OMEGA, The International Journal of Management Science, vol. 29 (2001), pp. 501-511.

[45] A. H. Land, A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. Econometrica Vol. 28, No. 3 (Jul., 1960), pp. 497-520.

[46] Hamid Allaoui, Abdelhakim Artiba. Scheduling two-stage hybrid flow shop with availability constraints. Computers and Operations Research 33 (2006), pp. 1399-1419.

[47] Maurice Pillet. Introduction aux plans d'expériences par la méthode Taguchi. Editions d'Organisation (1994).

[48] Rachid Sabre. Plan d'expériences, Méthode de Taguchi. Techniques de l'ingénieur, 2007, F1006 (3), pp. 1-10.

[49] H.R. Tizhoosh. Opposition-based learning : a new scheme for machine intelligence, in Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation, Vienna, Austria, November (2005), pp. 695-701.

[50] Q. Wei, Y. Wu. Dynamic Programming Algorithms for Two-Machine Hybrid Flow-Shop Scheduling With a Given Job Sequence and Deadline, in IEEE Access, vol. 8, pp. 89964-89975, 2020.

[51] John H Holland. Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, 1975.

[52] Rubén Ruiz, José Antonio Vázquez-Rodríguez. The hybrid flow shop scheduling problem. European Journal of Operational Research 205 (2010), pp. 1-18.

[53] J. Carlier, P. Chrétienne. Problèmes d'ordonnancement : modélisation, complexité, algorithmes. Edition Masson. (1988)

[54] D.E. Goldberg. Genetic algorithms in search, optimization and machine Learning. Addison- Wesley, 1989.

[55] Ling Wang, Quan-Ke Pan, M. Fatih Tasgetiren. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. Computers and Industrial Engineering 61 (2011), pp. 76-83.

[56] K. Gao, S. Xie, H. Jiang, J. Li. Discrete Harmony Search Algorithm for the No Wait Flow Shop Scheduling Problem with Makespan Criterion. Advanced Intelligent Computing, ICIC 2011. Lecture Notes in Computer Science, vol 6838. pp. 592-599.

[57] F. Glover. Future paths for integer programming and links to artificial intelligence, Computers and Operations Research, VOL. 13, pp. 533-549, 1986.

[58] Fred Glover. Tabu search, part I, ORSA journal on computing 1 :3 (1989), pp. 190-206.

[59] Fred Glover. Tabu search, part II, ORSA journal on computing, Vol. 2, No. 1, winter 1990.

[60] Q.K. Pan, M.F. Tasgetiren, Y.C. Liang. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem, Computers and Operations Research 35 (2008), pp. 2807-2839.

[61] A. Vignier, J.-C. Billaut, C. Proust. Les problèmes d'ordonnancement de type flow-shop hybride : État de l'art, RAIRO Recherche Opérationnelle 33 (2) (1999), 117-183.

[62] Tseng Chao Tang, Liao Ching-Jong. A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks. International Journal of Production Research. Taylor and Francis, 0020-7543.

[63] J. Kennedy, R. Eberhart. Particle Swarm Optimization, Proceedings of IEEE International Conference on Neural Network, Piscataway, NY, (1995), pp. 1942-1948.

[64] M. R. Singh, S. S. Mahapatra. A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks. Int J Adv Manuf Technol, 62, 2012, pp. 267-277.

[65] Engin, B.E., Engin, O. A new memetic global and local search algorithm for solving hybrid flow shop with multiprocessor task scheduling problem. SN Appl. Sci. 2, 2059 (2020).

[66] Geem, Z.W. Optimal Scheduling of Multiple Dam System Using Harmony Search Algorithm. (eds) Computational and Ambient Intelligence. IWANN 2007. Lecture Notes in Computer Science, vol 4507. Springer.

[67] Kang Seok Lee, Zong Woo Geem. A new meta-heuristic algorithm for continuous engineering optimization : harmony search theory and practice. Comput. Methods Appl. Mech. Engrg. 194 (2005), pp. 3902-3933.

[68] Geem, Z.W., Kim, J.H., Loganathan, G.V. A new heuristic optimization algorithm : harmony search. Simulation 76(2) (2001), pp. 60-68.

[69] D. Manjarres, I.Landa-Torres, S.Gil-Lopez, J.DelSer, M.N.Bilbao, S. Salcedo-Sanz, Z.W.Geem. A survey on applications of the harmony search algorithm Engineering Applications of Artificial Intelligence, 26(2013), pp. 1818-1831.

[70] Nawaz, M., E.E. Enscore Jr., I. Ham. A Heuristic Algorithm for the m-machine n-Job Flow-shop Sequencing Problem, OMEGA, 11,(1983), pp. 91-95.

[71] R. Bellman. Some applications of the Theory of dynamic programming-a review. Journal of the Operational Research Society of America, vol. 2, n 3, pp. 275-288.

[72] David Duvivier. Étude de l'hybridation des méta-heuristiques, application à un problème d'ordonnancement de type jobshop. Université du Littoral Côte d'Opale. Thèse de doctorat (2000).

[73] Changsheng Zhang, Jigui Sun, Xingjun Zhu, Qingyun Yang. An improved particle swarm optimization algorithm for flowshop scheduling problem. Information Processing Letters 108 (2008), pp. 204-209.

[74] Bassem Jarboui, Saber Ibrahim, Patrick Siarry, Abdelwaheb Rebai. A combinatorial particle swarm optimisation for solving permutation flowshop problems. Information Processing Letters 108 (2008), pp. 204-209.

[75] Z.W. Geem. Music-Inspired Harmony Search Algorithm, Theory and applications. Springer-Verlag Berlin Heidelberg, Studies in Computational Intelligence. Volume 191, ISBN 978-3-642-00184-0, 2009.

[76] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi. Optimization by simulated annealing. Science, tome 220, n 4598, pages 671-680, 1983.

[77] H.Zini, S. ElBernoussi. Minimizing makespan in hybrid flow shop scheduling with multiprocessor task problems using a discrete harmony search, IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications, CIVEMSA 2017 Proceedings, art. no. 7995322, 177-180.

[78] H. Zini, S. ElBernoussi. A discrete particle swarm optimization with combined priority dispatching rules for hybrid flow shop scheduling problem. Applied Mathematical Sciences, Vol. 9, 2015, no. 24, 1175-1187.

[79] H. Zini, S. ElBernoussi. A modified harmony search for flow shop scheduling problem, 2016 3rd International Conference on Logistics Operations Management (GOL), 2016, pp. 1-5.

[80] Jakob Puchinger, Günther R Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. In International Work-Conference on the Interplay Between Natural and Artificial Computation, pages 41-53. Springer, 2005.

[81] H. Zini, S. ElBernoussi. An OBL Harmony Search for Hybrid Flow Shop Scheduling with Multiprocessor Tasks Problem. Journal of Advanced Manufacturing Systems Vol. 19, No. 04, pp. 663-674 (2020).

[82] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6) (1953), pp. 1087-1092.

[83] E.-G. Talbi. A Taxonomy of Hybrid Metaheuristics,Journal of Heuristics, 8 : 541-564, 2002.

[84] Mohamed Kurdi. Ant colony system with a novel Non-DaemonActions procedure for multiprocessor task scheduling in multistage hybrid flow shop. Swarm and Evolutionary Computation, Volume 44, February 2019, Pages 987-1002.

[85] Shih-Wei Lin, Kuo-Ching Ying, Chien-Yi Huang. Multiprocessor task scheduling in multistage hybrid flowshops : A hybrid artificial bee colony algorithm with bi-directional planning. Computers and Operations Research, Volume 40, Issue 5, May 2013, Pages 1186-1195.

[86] F-D. Chou, Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessortasks,Int. J.Production Economics, 141,2013, 137-145.

[87] J. Carlier, I. Rebaï. Two branch and bound algorithms for the permutation flow shop problem. Eur. J. Oper. Res. 90 ((996), pp. 238-251.

[88] H-M. Wang, F-D. Chou, F-C. Wu. A simulated annealing for hybrid ow shop scheduling with multiprocessor tasks to minimize makespan. Int J Adv Manuf Technol, 53 (2011), 761-776.

[89] F.S. Şerifoğlu, G. Ulusoy. Multiprocessor task scheduling in multistage hybrid flow-shops : a genetic algorithm approach. Journal of the Operational Research Society, 55, 2004, 504-512.

[90] Ying Kuo-Ching, Lin Shih-Wei. Multiprocessor task scheduling in multistage hybrid flow-shops : an ant colony system approach. International Journal of Production Research. Taylor and Francis, 0020-7543.

[91] A. Lahimer, P. Lopez, M. Haouari. Improved bounds for hybrid flow shop scheduling with multiprocessor tasks. Computers and Industrial Engineering, 66 (2013) 1106-1114.

[92] J. Rezaeiana, H. Seidgar, M. Kiani. Scheduling of a Hybrid Flow Shop with Multiprocessor Tasks by a Hybrid Approach Based on Genetic and Imperialist Competitive Algorithms. Journal of Optimization in Industrial Engineering, vol. 13, pp. 1-11, 2013.

[93] C. Oğuz. Data for hybrid flow shop scheduling with multiprocessor tasks Koç University, 2006, [online] Available : http ://home.ku.edu.tr/coguz/.

[94] C.Oğuz, and M. F. Ercan. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Journal of Scheduling, 8(4) : 323-351, 2005.

[95] O. Moursli, Y. Pochet. A branch-and-bound algorithm for the hybrid flowshop. Int. J. Production Economics 64 (2000), 113-125.

[96] Johann Dréo, Alain Pétrowski, Patrick Siarry, Eric Taillard. Métaheuristiques pour l'optimisation difficile. EYROLLES, pp.356, 2003, Algorithmes, 978-2-212-11368-6. (hal-00843020).

[97] G.B. McMahon, P.G. Burton. Flow-Shop Scheduling with the Branch-and-Bound Method. Operations Research, Vol. 15, No 3, 1967.

[98] Cengiz Kahraman, Orhan Engin, İhsan Kaya, R. Elif Öztürk. Multiprocessor task scheduling in multistage hybrid flow-shops : A parallel greedy algorithm approach. Applied Soft Computing 10 (2010) 1293-1300.

[99] Orhan Engin, Gülşad Ceran, Mustafa K. Yilmaz. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Applied Soft Computing 11 (2011) 3056-3065.

[100] C-S. Chung, J. Flynn, O. Kirca. A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems. International Journal of Production Economics 79, 185-196 (2002).

[101] Ceyda Oğu, Yakov Zinder, Van Ha Do, Adam Janiak, Maciej Lichtenstein. Hybrid flow-shop scheduling problems with multiprocessor task systems. European Journal of Operational Research 152 (2004) 115-131.

[102] Hela Boukef Ben Othman. Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques : optimisation par algorithmes génétiques et essaims particulaires. Université de Tunis El manar. Thèse de doctorat (2009).

[103] C. Oğuz, 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, Issue 2, (2003), pages 390-403.

[104] C. Oğuz, F.M. Ercan. Scheduling multiprocessor tasks in a two stage flow shop environment. Computers and Industrial Engineering 33 (1997) 269-272.

[105] Imen Ayachi Hajjem. Techniques avancées d'optimisation pour la résolution du problème de stockage de conteneurs dans un port. Université de Tunis El manar. Thèse de doctorat (2012).

[106] R.N. Anthony. Planning and control systems : a framework for analysis. Studies in management control. Division of Research, Graduate School of Business Administration, Harvard University, 1965.

Résumé

La théorie d'ordonnancement est un axe central de l'optimisation combinatoire. Elle concerne les problèmes d'allocation dans le temps d'un ensemble de ressources pour réaliser un ensemble de tâches, en optimisant un ou plusieurs critères. L'ordonnancement est lié à la plupart des secteurs économiques, car il est nécessaire pour assurer le bon déroulement de la production.

Cette thèse est consacrée à la résolution des problèmes d'ordonnancement dans les systèmes de production de type flow shop hybride et flow shop hybride avec tâches multiprocesseurs, avec l'objectif de minimiser le temps total d'ordonnancement (le makespan). Ces deux problèmes sont connus comme étant des problèmes NP-difficiles.

Devant les limites rencontrées par les méthodes exactes pour résoudre des problèmes complexes, les métaheuristiques constituent une alternative importante qui permet de fournir de bonnes solutions dans des temps de calcul raisonnables.
Nous nous intéressons dans cette thèse à développer des approches de résolution hybrides, basées sur l'optimisation par essaims particulaires et la recherche d'harmonie pour résoudre les problèmes étudiés.

Des Différentes expérimentations ont été menées sur des problèmes benchmarks connus pour valider les approches proposées.

Mots-clefs: Flow Shop Hybride, Ordonnancement, Tâches multiprocesseurs, Optimisation par essaims particulaires, Recherche d'harmonie, Makespan.

Abstract

Scheduling theory is a central axis of combinatorial optimization. It concerns the problems of allocating over time of a set of resources for performing a set of tasks, with the aim of optimizing one or more criteria. Scheduling is linked to the most economic sectors because it is necessary to ensure the smooth running of production.

This thesis is devoted to solving the hybrid flow shop and the hybrid flow shop with multiprocessor tasks scheduling problems in production systems, with the objective of minimizing the maximum completion time (the makespan). These two problems are known to be NP-hard.

Faced with the limits encountered by exact methods for solving complex problems, metaheuristics constitute an important alternative that allows to provide good solutions in reasonable computation times.
In this thesis, we are interested in developing hybrid resolution approaches, based on particle swarm optimization and harmony search algorithms for the studied problems.

Various experiments were carried out on known benchmark problems to validate the proposed approaches.

Key Words: Hybrid Flow Shop, Scheduling, Multiprocessor tasks, Particle Swarm Optimization, Harmony Search, Makespan.