جامعة أبو بكر بلقايد
كلية التكنولوجيا
UNIVERSITÉ DE TLEMCEN
Faculté de Technologie
Laboratoire de Productique

Thèse

Présentée pour l'obtention du grade de DOCTEUR
DE L'UNIVERSITE Tlemcen
par

Imane LARIBI

Résolution de problèmes d'ordonnancement de type Flow-Shop de permutation en présence de contraintes de ressources non-renouvelables

---

Spécialité : Productique

Soutenue le 12 décembre 2018 devant un jury composé de :

Président du jury	MCA	Meliani Sidi Mohammed	(Université de Tlemcen, Algérie)
Directeur de thèse	Prof	Sari Zaki	(Université de Tlemcen, Algérie)
Examinateur	Prof	Hafid Haffaf	(Université d'Oran, Algérie)
Examinateur	MCA	Latéfa Ghomri	(Université de Tlemcen, Algérie)
Examinateur	MCA	Mehdi Souier	(École Supérieure de Management de Tlemcen, Algérie)
Invité	Prof	Farouk Yalaoui	(Université de Technologie de Troyes, France)

Thèse effectuée au sein du **Laboratoire de productique**

de l'Université Tlemcen

Algérie

et

Le Laboratoire d'Optimisation des Systèmes Industriels (LOSI)

de l'Université Troyes

France

dans le cadre programme boursier PNE

Résumé

L'axe de recherche traité dans cette thèse recouvre un domaine très connu dans la recherche opérationnelle, il s'agit de l'ordonnancement d'atelier de production et plus particulièrement l'étude de problème Flow-Shop. Le Flow-Shop à une grande pertinence en ingénierie, représentant près du quart des systèmes de production, tels que l'industrie automobile, agro-alimentaires et textiles. Vu cette pertinence, ce problème a été largement étudié dans la littérature sous sa forme classique. Notre contribution consiste à intégrer certaines spécificités du monde industriel dans les problèmes classiques d'ordonnancement Flow-Shop. Ces spécificités consistent essentiellement à prendre en compte les contraintes de ressources non-renouvelables. Ce choix a été motivé par une réalité industrielle divergente avec le monde académique. Cette thèse est consacrée à la résolution des problèmes d'ordonnancement Flow-Shop avec contraintes de ressources non-renouvelables. Les travaux développés portent sur l'un des critères les plus étudiés dans la théorie d'ordonnancement à savoir la minimisation du makespan qui revient à maximiser la productivité. À notre connaissance, il n'existe pas de résultat pour ce problème dans la littérature. Comme première attaque systématique à ce problème, une formulation mathématique a été proposée pour décrire la problématique et résoudre de manière exacte le problème. Étant donné que le problème considéré est NP-difficile et que la résolution exacte est limitée aux problèmes de petites tailles en raison de contrainte de temps de calcul, des méthodes de résolution approchées basées sur un algorithme génétique et un algorithme d'essaim particulaire ont été proposées afin de fournir des solutions optimales ou proches de l'optimum dans un temps acceptable. De nombreuses expérimentations ont été menées sur des instances théoriques pour valider les approches proposées sur différentes configurations de disponibilité de ressources non-renouvelables.

Mots-clé :

Ordonnancement, Flow-Shop, ressources non-renouvelables, optimisation, modélisation mathématique, méthodes approchées.

Solving a permutation Flow-Shop scheduling problems subject to non-renewable resources constraints

Abstract

The research subject treated in this thesis covers a well-known area in operation research. It comes to scheduling in production workshop, especially, the Flow-Shop problem. Flow-shop has an extensive engineering relevance, representing nearly a quarter of manufacturing systems, such as automobile, agro-food and textile industries. In view of this relevance, this problem has been widely studied in literature in its classic form. Our contribution focuses on the integration of certain specificities of the industrial world into the classical Flow-Shop scheduling problems. These specificities consist essentially of taking into account the constraints of non-renewable resources. This choice was motivated by a divergent industrial reality with the academic world. This thesis is dedicated to solving Flow-Shop scheduling problems subject to non-renewable resource constraints. Developed work focuses on one of the most studied criteria in the scheduling theory, that is, the minimization of the makespan designed to maximize the productivity. To the best of our knowledge, there is no result for this problem in the literature.

As a first systematic attack on this problem, a mathematical formulation has been developed to describe the problem and to give an exact resolution. Since the considered problem is NP-hard and the exact resolution is limited to small size problems due to computation time constraint, approximating methods based on genetic algorithm and particle swarm optimization are developed in order to provide optimal or near-optimal solutions in a reasonable time. Numerous experiments have been conducted on theoretical instances to validate the proposed approaches on different configurations of resources availability.

Keywords :

Scheduling, Flow-Shop,non-renewable resource, optimization, mathematical modeling, approximate methods.

Remerciements

Mes travaux de thèse se sont déroulés au laboratoire de productique à l'université de Tlemcen et au laboratoire d'optimisation des systèmes industriels à l'université de Troyes dans le cadre du programme boursier PNE.

Mes remerciements s'adressent tout naturellement d'abord à mon directeur de thèse, Monsieur Zaki Sari, Professeur à l'université de Tlemcen d'avoir accepté de diriger ma thèse. Je souhaite lui exprimer ma profonde gratitude pour son accompagnement constant, ses conseils, son soutien, sa disponibilité, ses encouragements, son écoute et sa patience, mais aussi bien pour son enthousiasme et son humour qui ont été des clés pour réaliser ce travail.

Je tiens aussi à adresser mes remerciements les plus sincères et mon profond respect à Monsieur Farouk Yalaoui, Professeur à l'université de technologie de Troyes, pour m'avoir si bien accueillie au sein de son équipe durant touts les périodes passées au sein du laboratoire d'optimisation des systèmes industriels. Je le remercie infiniment pour m'avoir guidée durant toute l'élaboration de cette thèse avec le sérieux et la compétence qui le caractérise. Je le remercie aussi pour son écoute, ses corrections avisées de mes travaux, sa disponibilité, son encouragement et ces conseils qui m'ont permis d'acquérir une maturité suffisante pour continuer dans le chemin de la recherche et de l'enseignement.

Avec vous Messieurs Zaki Sari et Farouk Yalaoui, j'ai eu le privilège de bénéficier d'un excellent encadrement scientifique, ce qui a très largement contribué la réussite de cette thèse.

À Monsieur Sidi Mohammed Meliani, Maître de Conférences à l'université de Tlemcen, qui ma fait l'honneur d'accepter de présider ce jury de thèse et d'avoir consacré une partie de son temps pour examiner ce travail.

Je tiens à exprimer ma gratitude à Monsieur Hafid Haffaf, Professeur à l'université d'Oran, pour sa compréhension et pour l'acceptation de participer à ce jury.

Je suis particulièrement reconnaissante à Mademoiselle Latéfa Ghomri, Maître de Conférences à l'université de Tlemcen, pour sa gentillesse et l'intérêt qu'elle a porté à mon travail.

Je remercie également Monsieur Mehdi Souier, Maître de Conférences à École Supérieure de Management de Tlemcen, d'avoir accepter d'évaluer mes travaux et de m'honorer en faisant partie du jury.

Je tiens à exprimer ma grande gratitude à Jatinder Gupta, Professeur à l'Université d'Alabama à Huntville, USA, pour ces conseils précieux et l'intérêt qu'il a porté à mon travail de recherche.

Je tiens également à remercier les personnes, auprès de qui j'ai directement travaillé ou qui m'ont soutenu. Je ne cite pas les noms pour n'oublier personne, mais je voudrais ici les remercier tous très chaleureusement, leur souhaitant plein de bonheur et de réussite.

Enfin, je tiens à remercier très sincèrement tous les membres de ma famille et surtout ma mère, pour leur soutien et leurs encouragements constants durant mes années d'études. Leur présence de tous les instants a été pour moi un atout indispensable pour mener à bien ce travail.

Et maintenant Maman, promis, les études c'est fini El Hamdoulah !

Table des matières

Table des figures

Liste des tableaux

Introduction

L'environnement actuel des entreprises est caractérisé par des marchés soumis à une forte concurrence et sur lesquels les exigences et les attentes des clients deviennent de plus en plus fortes en termes de qualité, de coût et de délais de mise à disposition. Par conséquent, les entreprises doivent produire mieux, moins cher et/ou plus vite que leurs homologues afin d'améliorer leur compétitivité. Dans cette optique, la performance de l'entreprise se construit selon deux dimensions essentielles : une dimension technologique qui vise à développer les performances des produits mis sur le marché afin de satisfaire aux exigences de qualité et de réduction du coût de possession des produits et une dimension organisationnelle qui vise a développer la performance de l'entreprise en termes de durée des cycles de fabrication, respect des dates de livraison prévues, etc. Cette dernière joue un rôle très important, dans la mesure où les marchés sont de plus en plus variables et évolutifs et exigent des temps de réponse des entreprises de plus en plus courts. Pour satisfaire les besoins des clients, l'organisation repose sur la mise en œuvre d'un ensemble de fonctions, parmi lesquelles la fonction d'ordonnancement joue un rôle prépondérant.

La fonction d'ordonnancement consiste à organiser un ensemble d'activités au sein d'un système imposant certaines règles à respecter : il s'agit d'élaborer, à court terme (généralement quelques jours en production industrielle), un planning détaillé d'un ensemble de tâches à réaliser, en indiquant pour chaque tâche son instant de déclenchement sur une ressource exploitée tout en optimisant un ou plusieurs objectifs. Dans une entreprise, cette fonction est toujours confrontée à des problèmes de plus en plus complexes à résoudre, à cause du grand nombre de tâches qu'elle doit réaliser, en prenant en compte des contraintes temporelles et des contraintes portant sur l'utilisation et la disponibilité des ressources requises par les tâches ainsi les critères à optimiser. Donc, apporter des solutions efficaces et performantes à ces problèmes constitue sûrement un enjeu économique important.

L'ordonnancement s'identifie, généralement, à un problème d'optimisation combinatoire dans lequel la recherche de solutions efficaces et performantes nécessite la mise au point de méthodes de résolution efficace. L'enjeu économique considérable des problèmes d'ordonnancement a suscité depuis des décennies une recherche scientifique intensive pour développer des outils et des méthodologies efficaces de résolution qui sont classé en méthodes exactes permettant de résoudre des problèmes de petites tailles et des méthodes approchées pour des problèmes de grandes tailles.

Un ordonnancement efficace est donc un outil essentiel pour les entreprises afin d'atteindre des performances élevées. C'est pourquoi dans cette thèse nous nous intéressons à la résolution de ce problème à l'aide de différents techniques.

Au-delà des valeurs économiques, l'ordonnancement peut contribuer également à la société du point de vue de l'humanité. En fait, dans les secteurs professionnels fonctionnant 24 heures sur 24 (par exemple dans le domaine de la santé, l'industrie, la police, etc), Harrington [55] a précisé qu'un mauvais ordonnancement demande aux personnels de travailler plus et cela fait apparaitre des risques pour la santé (qualité et quantité de sommeil moins bonnes, fatigue accrue) et la sécurité (accident). En effet, le rôle et l'impact de l'ordonnancement sur le quotidien sont aussi les raisons de notre intérêt pour ce domaine de recherche. Mais dans le cadre de cette thèse, nous nous limiterons aux problèmes d'ordonnancement dans les systèmes de production.

Une grande partie des études des problèmes d'ordonnancement de production se placent dans un contexte où les machines sont considérées comme les seules ressources dans le système de production. Ce qui en réalité n'est pas toujours le cas. Dans la plupart des environnements de fabrication réels, le traitement des tâches sur les machines peut nécessiter des ressources supplémentaires tel-que : la matière première, des pièces, etc. Ces ressources sont connues sous le nom de ressources non-renouvelables ou consommables, c'est-à-dire que leurs consommations globales au cours du temps est limitée (une fois, elles sont utilisées par une tâche, elles ne sont plus disponibles pour les autres). Ces ressources sont produites dans des systèmes externes, ensuite elles sont stockées dans des zones de stockage (entrepôts ou magasins) où un stock initial peut éventuellement exister avant le lancement de la production.

En prenant en compte, la disponibilité des ressources non-renouvelables dans le processus de production implique non seulement que la machine doit être libre lors du traitement d'une tâche, mais aussi les ressources non-renouvelables requises doivent être disponibles en quantités suffisantes.

Les tâches pour qu'elles soient exécutées requièrent une certaine quantité déterminée a priori de la ressource non-renouvelable. Si la quantité demandée n'est pas suffisante alors la machine restera inactive jusqu'à la disponibilité de la quantité requises par la tâche. La présence de ces temps d'inactivité (*idle time*) sur les machines influe de manière significative sur le processus de production et tout ordonnancement réaliste se doit d'en tenir compte.

Les problèmes d'ordonnancement sous contraints de ressources non-renouvelables sont souvent rencontrés dans les environnements de production tels que : les industries du textile, d'automobile et l'industrie pharmaceutique. Dans l'atelier de peinture dans l'industrie automobile chez Audi, les carrosseries de voitures passent par trois stations pour l'application de la charge, de la couche de base et de la peinture. Après chaque étape du processus, les carrosseries peintes passent à travers un séchoir. Le processus complet de peinture pour un modèle Audi dure environ trois heures. Par exemple, dans la station de peinture, les carrosseries ont besoin d'une certaine quantité de peinture qui est une ressource consommable, c'est-à-dire, qu'après avoir peint la car-

rosserie, le stock de peinture est diminué par la quantité appliquée. Donc, si la quantité requise n'est pas suffisante alors le processus de peinture va être affecté.

Dans cette optique, l'objectif de cette thèse consiste à intégrer quelques spécificités du monde industriel dans les problèmes d'ordonnancement classiques. De telles spécificités consistent essentiellement dans la prise en compte de contraintes de ressources non-renouvelables. Ce choix a été motivé par une réalité industrielle divergent avec le monde académique. Dans ce cadre, les problèmes d'ordonnancement avec contraintes de ressources consommables sont suscité l'intérêt d'une communauté de chercheurs. Une diversité de problèmes à été traité, avec différents critères d'optimisation et utilisant différentes configurations d'atelier.

Dans cette thèse, nous nous intéressons plus particulièrement aux problèmes de type Flow-Shop en présence de contraintes de ressources non-renouvelables. L'objectif est de trouver une séquence appropriée de tâches en fonction de la disponibilité des ressources renouvelables et non-renouvelables, de manière à minimiser le temps d'exécution maximal (makespan). À notre connaissance, ce problème n'a pas été traité dans la littérature auparavant.

Dans cette optique, ce travail a pour objectif de contribuer à l'étude des problèmes d'ordonnancement avec contraintes de ressources consommables. Différentes méthodes de résolution sont développées afin de choisir laquelle est la plus efficace pour les problèmes d'ordonnancement d'atelier Flow-Shop en présence de contraintes de ressources non-renouvelables. Dans la fin de la thèse, nous avons obtenu des résultats satisfaisantes, mais il reste encore des travaux à faire plus tard.

Le manuscrit est organisé en deux parties :

1. La première partie regroupe les chapitres de 1 à 3, dans laquelle nous abordons l'ordonnancement de production, les techniques d'optimisation et un état de l'art sur le problème traité.
   - Le Chapitre 1 constitue une introduction générale aux problèmes d'ordonnancement dans le système de production, ce chapitre englobe diverses notations et définitions essentielles pour aborder le problème traité dans cette thèse. Il contient aussi quelques définitions de la complexité des problèmes d'optimisation combinatoires, dont les problèmes d'ordonnancement font partie.
   - Dans le Chapitre 2, nous décrivons à la fois les techniques générales de résolution de la recherche opérationnelle pour permettre une meilleure compréhension des méthodes développées dans cette thèse, ainsi qu'une revue de la littérature de l'utilisation de ces techniques pour la résolution des problèmes d'ordonnancement Flow-Shop.
   - Le Chapitre 3 est consacré à un état de l'art couvrant les travaux de recherche menés sur les problèmes statiques et déterministes d'ordonnancement sous contraintes de ressources non-renouvelables. Cet état de l'art nous permet de bien positionner notre problématique.
2. La deuxième partie regroupe les chapitres de 4 à 6, dans laquelle nous présentons nos prin-

cipales contributions à l'étude du problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

— Le chapitre 4 est dédié à l'optimisation par méthodes exactes de l'ordonnancement d'atelier de type Flow-Shop sous contraintes de ressources non-renouvelables. Nous présentons dans ce chapitre une description détaillée de la problématique, ainsi que la modélisation mathématique sous forme de programme linéaire en nombres entiers, laquelle comporte les contraintes et l'objectif relatifs au fonctionnement de notre problème. Cette modélisation nous permet d'une part de résoudre de manière optimale les instances de petites tailles et d'autre part d'évaluer la qualité des méthodes approchées développées dans les chapitres suivants. Nous construisons aussi dans ce chapitre un benchmark qui constitue l'échantillon de test pour nos expérimentations.

— Le chapitre 5 et 6 sont consacré à la résolution de la problématique par des méthodes approchées, ces derniers représentent une alternative appropriée aux méthodes exactes pour résoudre des problèmes complexes de tailles importantes. Dans le chapitre 5, nous proposons un algorithme génétique pour la résolution de problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables afin de minimiser le makespan. Nous exposons dans ce chapitre, les détails de l'implémentation de cet algorithme ainsi que la technique utilisée pour ajuster les paramètres de ce dernier.

— Dans le chapitre 6, nous développons un algorithme d'optimisation par essaims particulaires. Cet algorithme à été initialement conçu pour l'optimisation continue, mais qui récemment fait l'objet de quelques adaptations pour résoudre les problèmes d'optimisation discrets, en particulier les problèmes Flow-Shop et a donné de bons résultats. L'objectif de ce chapitre consiste à valider le potentiel de cet algorithme pour résoudre le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

Nous finissons ce mémoire par une conclusion et quelques perspectives.

Première partie

Ordonnancement de la production, optimisation et état de l'art

Chapitre 1

Ordonnancement de production

Ce chapitre constitue une introduction générale aux problèmes d'ordonnancement dans la gestion de production. Il présente diverses notions essentielles pour la compréhension de l'ensemble des idées présentées dans cette thèse. La première section définit, de manière concise, la gestion de production. Ceci permet de situer les problèmes d'ordonnancement de la production présentés dans la seconde section. Cette dernière, décrit le problème d'ordonnancement notamment la définition, les différentes notions qui interviennent dans cette définition, les classes et les typologies des problèmes d'ordonnancement ainsi que les outils de modélisation et de représentation associés. Dans ce chapitre, nous exposons également quelques concepts sur la théorie de complexité afin de mieux comprendre la difficulté de la résolution de certains problèmes.

1.1 La gestion de production

Face aux défis de la mondialisation et de la concurrence, les entreprises industrielles sont obligées de réadapter leur système de production en vue d'augmenter la qualité de leur produit, de mieux gérer leurs ressources, de diminuer leur coût de revient et d'augmenter leur flexibilité afin de rester compétitif.

La gestion de production apparaît donc comme le guide indispensable de l'entreprise industrielle pour la réalisation de cet objectif. En effet, une telle gestion doit organiser le fonctionnement du système de production et de mieux gérer ses différents composants.

Les systèmes de production peuvent être vus comme un ensemble de ressources divers (matériels, humains, etc.) qui interagissent et interfèrent dans le but de produire des biens ou services.

Ces derniers peuvent être des systèmes très complexes et difficiles à gérer au vu de toutes leurs composantes fonctionnelles (fabrication, achat, distribution, maintenance...) [81]. À cet égard, plusieurs approches ont été envisagées dans le but de mieux comprendre leur fonctionnement et de mieux les appréhender. L'application de la théorie des systèmes aux systèmes de production suggère une décomposition de ces derniers en trois sous-systèmes : le sous-système physique

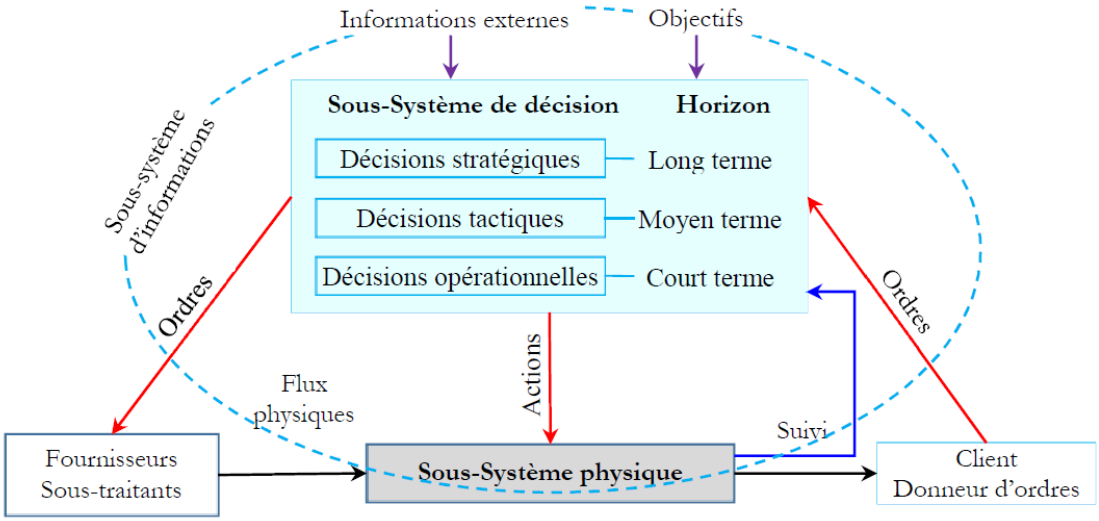


FIGURE 1.1 : Décomposition d'un système de production [5]

de production, le sous-système de décision et le sous-système d'information qui s'intègrent afin d'assurer la pérennité et la compétitivité de l'entreprise industrielle comme illustrée dans la figure 1.1.

- Le sous-système physique de production englobe tout les ressources humaines et physiques nécessaires pour la transformation des matières premières en produits finis.
- Le sous-système de décision contrôle le système physique de production à travers l'organisation des différentes activités en prenant des décisions basées sur les données transmises par le sous-système d'information.
- Le sous-système d'information intervient à plusieurs endroits, entre les sous-systèmes de décision et de production et à l'intérieur même du sous-système de décision, pour la gestion des informations utilisées lors de prises de décision, et du sous-système physique de production, pour la création et le stockage d'informations de suivi par exemple [81]. Donc, son rôle peut se résumer a la collecte, le stockage, le traitement et la transmission d'informations.

Les deux sous-systèmes décisionnels et informationnels traitent les fonctions rattachées directement à la production à savoir, la gestion de stock, la gestion des ressources, la maintenance, la planification, etc. L'association de ces deux derniers sous-systèmes constitue le système de gestion de production, évoqué dans cette section.

En fait, la gestion de production assure l'organisation du système de production afin de fabriquer les produits en quantités et qualités définies, ainsi dans un temps voulus compte tenu des moyens humains ou technologiques disponibles.

L'objectif de la gestion de production est de gérer les systèmes de production au mieux. Cette gestion s'effectue par un ensemble de décisions qui peuvent être hiérarchisées suivant des gra-

nularités et des horizons temporels différents. Ces décisions sont habituellement classées selon trois catégories introduites par Anthony [4] et reprises dans [135] en gestion de production, à savoir : les décisions stratégiques, tactiques et opérationnelles.

1.1.1 Décisions stratégiques

Les décisions stratégiques définissent la politique de l'entreprise à long terme. Ces décisions portant sur la stratégie de l'entreprise sont prises par la direction générale de l'entreprise et portent essentiellement sur la gestion des ressources durables afin que celles-ci soient toujours suffisantes pour assurer la pérennité de l'entreprise [81]. Les ressources visées peuvent être des ressources humaines (compétences à posséder, plans de formation), équipements (bâtiment, machines), des informations de production (les produits à lancer ou développer) ou des données techniques, etc.

1.1.2 Décisions tactiques

Les décisions tactiques définissent la politique de l'entreprise à moyen terme. Ces décisions sont prises par le personnel d'encadrement de l'entreprise et elles sont destinées à obtenir la meilleure exploitation des moyens mis en œuvre.

En effet, ces décisions s'inscrivent dans un cadre logique dessiné par les décisions stratégiques. Comme exemples de décisions tactiques, on peut citer : la planification de la production qui est une programmation prévisionnelle de la production pour une période qui varie généralement entre 6 et 18 mois (selon l'entreprise), les problèmes d'allocation (des fournisseurs ou des produits), la définition des niveaux de stock ainsi que le choix des modes de transport, etc.

1.1.3 Décisions opérationnelles

Les décisions opérationnelles définissent la politique de l'entreprise à court terme. Ces décisions assurent la flexibilité quotidienne nécessaire pour faire face aux fluctuations prévues de la demande et des disponibilités de ressources et réagir aux aléas, dans le respect des décisions tactiques. Généralement, ces décisions sont prises par des responsables d'activités, des chefs d'équipe ou des agents maîtrise.

Parmi les décisions opérationnelles concernant la gestion de la production, on trouve par exemple la gestion des stocks et l'ordonnancement de la production. La gestion des stocks assure la mise à disposition des produits et des composants alors que l'ordonnancement consiste a une programmation prévisionnelle détaillée des ressources mobilisées (opérateurs, équipements et outillages) afin de fabriquer des produits définis par la planification.

En effet, l'ordonnancement constitue une classe importante de ces décisions et joue un rôle privilégié dans la gestion informatisée des flux de production au sein de l'entreprise. Il s'occupe de la réalisation des décisions venant des niveaux supérieurs et couvre ainsi un ensemble d'actions

qui transforment les décisions de fabrication définies par le programme directeur de production en instructions d'exécution détaillées destinées à piloter et contrôler à court terme l'activité des postes de travail dans l'atelier [64].

Dans la suite de ce travail, on se place au niveau des décisions opérationnelles et nous nous intéressons uniquement à l'ordonnancement de la production.

Dans la section suivante, ce concept va être détaillé et va faire l'objet de notre étude.

1.2 L'ordonnancement de la production

La théorie d'ordonnancement est une branche de la recherche opérationnelle, qui consiste à ordonner un ensemble d'opérations tout en satisfaisant un ensemble de contraintes et en optimisant un ou plusieurs objectives. L'ordonnancement joue un rôle essentiel dans de nombreux secteurs d'activités à savoir, en : informatique (ordonnancement de processus), administration (établissement d'un emploi du temps, gestion des ressources humaines), industrie (gestion des ateliers de production), construction (gestion des chantiers routiers), logistique (gestion des livraisons et des stocks).

Parmi ces nombreux types de problèmes d'ordonnancement, nous nous sommes intéressés dans le cadre de cette thèse aux problèmes d'ordonnancement d'ateliers dans les systèmes de production.

Un atelier de production est un espace physique où la fabrication se déroule. Il est composé de ressources humaines et matérielles, et caractérisé par les types de tâches à exécuté, les type de ressources et la gamme de fabrication, que nous présentons en détail dans les sous-sections suivantes.

Nombreuses sont les définitions proposées au problème d'ordonnancement d'atelier, nous tirons de la littérature les trois définitions ci-dessous :

Definition 1.2.1 *[92]*
Scheduling is the process of organizing, choosing, and timing resource usage to carry out all the activities necessary to produce the desired outputs at the desired times, while satisfying a large number of time and relationship constraints among the activities and the resources.

Definition 1.2.2 *[82]*
Scheduling is concerned with the allocation of scarce resources to activities with the objective of optimizing one or more performance measure.

Definition 1.2.3 *[106]*
Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives.

D'après les définitions ci-dessus, on retrouve l'aspect commun de l'affectation de ressources aux tâches. Donc nous pouvons dire que l'ordonnancement d'ateliers consiste à programmer dans le temps l'exécution des tâches selon la disponibilité des ressources pour répondre à un ou plusieurs d'objectifs, tout en respectant les contraintes techniques de fabrication.

1.2.1 Formulation d'un problème d'ordonnancement

Dans un problème d'ordonnancement, quatre notions fondamentales interviennent : les tâches, les ressources, les contraintes et les objectifs. Dans ce qui suit, on donnera une définition détaillée de chacun de ces notions. Ces définitions sont basées sur celles proposées par [87].

1.2.1.1 Les tâches

Une tâche est une entité élémentaire de travail localisé dans le temps par une date de début t_i (start time) et une date de fin c_i (completion time), dont la réalisation est caractérisée par une durée positive p_i (processing time) telle que $p_i = c_i - t_i$.

Certaines caractéristiques relatives à l'exécution d'une tâche sont définies ainsi :

- Une date de disponibilité r_i (release date) qui correspond à la date de début au plus tôt.
- Une date d'échéance d_i (due date) qui correspond à la date de fin au plus tard.
- Un poids w_i (weight) qui représente le facteur de priorité qui dénote l'importance de la tâche i relativement aux autres.

La figure suivante donne une représentation de la tâche en désignant ses principales caractéristiques.

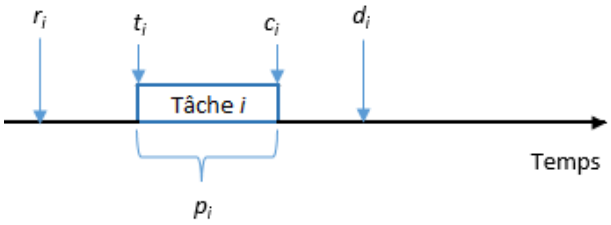


FIGURE 1.2 : Caractéristique d'une tâche

On distingue deux types de tâches :

- Des tâches morcelables (préemptibles) qui peuvent être exécutées par morceaux par une ou plusieurs ressources.
- Des tâches non-morcelables (indivisibles) qui doivent être exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles ne soient complètement achevées.

Généralement, en ordonnancement d'atelier, le terme « tâche » correspond à une « opération » et le terme « travail » ou « job » désigne l'ensemble d'opérations constituant le même job.

1.2.1.2 Les ressources

Une ressource est un moyen technique ou humain utilisé pour la réalisation d'une tâche et disponible en quantité limitée. Dans un atelier, plusieurs types de ressources sont distingués.

1. Selon leurs disponibilités au cours du temps, on trouve :
   — Les ressources renouvelables, comme c'est le cas pour les machines, personnels, équipements, etc. La ressource est dite renouvelable si après avoir été utilisé par une ou plusieurs opérations, elle est à nouveau disponible en même quantité.
   — Les ressources non-renouvelables, souvent appelées ressources consommables ou bien ressources financières. On dit que la ressource est non-renouvelables si ça disponibilité décroît après avoir été allouée à une opération. C'est le cas pour la matière première, budget.
   — Les ressources doublement contraintes, ces ressources combinent les contraintes liées aux deux catégories précédentes. Leur utilisation instantanées et leur consommation globale sont toutes les deux limitées. C'est le cas des ressources d'énergie (pétrole, électricité, etc.).
2. Selon leurs capacités, on trouve :
   — Les ressources disjonctives (ou non-partageables), il s'agit des ressources qui ne peuvent exécuter qu'une seule opération à la fois c'est le cas par exemple de machine-outil ou robot manipulateur.
   — Les ressources cumulatives (ou partageables), il s'agit des ressources qui peuvent être utilisé par plusieurs opérations simultanément (équipes d'ouvriers, poste de travail, etc.).

1.2.1.3 Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre conjointement les variables de décision [1]. En d'autres termes, les contraintes représentent les conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Plus les contraintes sont nombreuses, plus le problème d'ordonnancement devient plus difficile.

Dans les problèmes d'ordonnancement, deux types de contraintes sont distingués : les contraintes temporelles et les contraintes de ressources.

1. Les contraintes temporelles décrivent les interdépendances temporelles entre les opérations, elles intègrent :
   — Les contraintes de dates limites : c'est des contraintes imposées individuellement à chaque opération i. Par exemple, l'opération i ne peut débuter avant une certaine date

1. ensemble des variables qui régissent la situation à modéliser

(livraison de matière première, conditions climatiques, etc.) ou encore l'opération i ne peut commencer avant sa date de disponibilité r_i et doit être terminée avant une date d'échéance d_i.

- Les contraintes d'antériorité : c'est des contraintes qui relient la date de début ou la date de fin de deux opérations par une relation linéaire. De manière générale, c'est des contraintes qui décrivent le positionnement relatif de certaines opérations par rapport à d'autres, c'est le cas par example des gammes opératoires dans les d'ateliers de production.

2. Les contraintes de ressources traduisent l'utilisation et la disponibilité des ressources utilisées par les opérations. Deux types de contraintes liées à la nature cumulative ou disjonctive des ressources peuvent alors être distingués.
   - Les contraintes disjonctives : ces contraintes imposent la non-réalisation simultanée de deux opérations sur la même ressource.
   - Les contraintes cumulatives : ces contraintes expriment le fait qu'à tout instant, le total des ressources utilisées ne dépasse pas une certaine limite fixée.

Il existe une autre classification liée au système de production présentée dans [66], dont les contraintes peuvent être classées en deux types : endogène et exogène.

1. Les contraintes endogènes : elles constituent des contraintes directement liées au système de production et à ses performances dont :
   - les capacités et les dates de disponibilité des machines.
   - Les séquences des opérations à exécuter.
2. Les contraintes exogènes : elles sont indépendantes du système de production, on distingue :
   - les priorités de quelques commandes et de quelques clients.
   - Les retards possibles accordés pour certains produits.

1.2.1.4 Les objectifs

Les objectifs dits aussi les critères d'évaluation sont les indicateurs de performance sur lesquels se base le choix d'un ordonnancement satisfaisant. En ordonnancement, les critères à optimiser consistent à minimiser ou maximiser une fonction objectif. Cette fonction objectif est généralement liée aux temps, aux ressources ou bien aux coûts.

- les objectifs liés au temps : c'est la catégorie des objectifs les plus étudiés en optimisation, parmi les plus classiques, nous pouvons citer :
  - Le temps total d'exécution : connue sous le nom de *makespan* et défini par $C_{max} = \max_{i \in E} C_i$ (E désigne l'ensemble d'opération à ordonnancer) qui représente la date de fin

du job le plus tardif. La minimisation de ce critère est souvent rencontrée puisque ça conduit à une meilleure utilisation de ressources (productivité).

- La somme des dates d'achèvement des jobs : on lui réfère aussi comme *total flow time* ou *total completion time* définie par $\sum_{i \in E} C_i$.
- Le retard algébrique maximal : connue sous le nom de *lateness* et défini par $L_{max} = \max_{i \in E} L_i$, tel que $L_i = C_i - d_i$ est le retard algébrique pour chaque opération. L'objectif de la minimisation du retard algébrique maximal consiste donc à minimiser la quantité L_{max}.

— Les objectifs liés aux ressources : les objectifs de ce type correspondent par exemple à :
- Maximisation de la charge d'une ressource e.g., maximiser l'utilisation de la machine ayant un bon rendement ou la machine la moins gourmand en énergie, etc.
- Minimisation de nombre de ressources nécessaires pour réaliser un ensemble d'opérations.

— Les objectifs liés au coût : ces objectifs consistent généralement à minimiser les coûts de lancement, de production, de stockage, ou de transport.

Généralement, l'objectif souhaité dans un problème d'ordonnancement quelconque définit le critère à optimiser. Lorsque cet objectif renvoie un seul critère, il s'agit bien du cas d'optimisation mono-objectif, le cas le plus fréquent dans la littérature des problèmes d'ordonnancement. Cependant, lorsque l'objectif renvoie simultanément plusieurs critères à la fois, nous parlons dans ce cas d'optimisation multi-objectif. Cette dernière est devenue plus en plus important face à l'évolution et à la concurrence des systèmes de production.

Une caractéristique intéressante pour un critère est sa régularité. Un critère est dit **régulier**, si la valeur à minimiser est une fonction croissante des dates de fin des opérations, nous citons, à titre d'exemple : makespan, flow time, lateness, etc. Ces critères réguliers servent à exprimer de différentes façons deux sources d'insatisfaction : celle de l'entreprise liée aux volumes trop importants d'en-cours, et celle des clients liée au non-respect des délais [24].

Les critère **irréguliers** sont des critères non-réguliers, c'est-à-dire qui ne sont pas des fonctions monotones des dates de fin des opérations. Ces critères peuvent augmenter lorsque l'on termine un travail plus tôt. En particulier, si le fait de terminer un travail avant sa date de fin souhaitée implique un coût, alors tout critère incluant ce coût n'est pas régulier [24, 66]. Soit à titre d'exemple : la minimisation des encours, la minimisation du coût de stockage des matières premières, etc.

1.2.2 Typologie des problèmes d'ordonnancements

Une typologie des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre et la nature des machines ainsi que l'ordre d'enchaînement des opérations (gamme de fabrication).

Deux grandes familles de problèmes d'ordonnancement se présentent. La première famille regroupe les problèmes pour lesquels chaque job nécessite une seule opération. La deuxième re-

groupe ceux dont les jobs nécessitent plusieurs opérations.

1.2.2.1 Problèmes à une opération

En se basant sur la configuration des machines, nous distinguons pour la première catégorie :

- ***Problèmes à une machine*** : les problèmes d'atelier à une machine (*single machine problem*) consistent à ordonnancer, sur une seule machine, des jobs constitué d'une seule opération.
- ***Problèmes à machines parallèles*** : les problèmes d'atelier à machines parallèles (*parallel machine problem*) sont une généralisation des problèmes à une machine. Ce type d'atelier se caractérise par le fait que chaque opération peut être réalisée par n'importe quelle machine, disposée en parallèle, mais n'en nécessite qu'une seul. Le problème d'ordonnancement consiste donc à déterminer l'affectation des opérations aux machines puis le séquencement de ces opérations sur chaque machine.

  Dans le dernier cas, il est possible de distinguer trois classes de machines :

  - Machines parallèles identiques (*identical parallel machines*) : la durée d'exécution des opérations est la même sur toutes les machines.
  - Machines parallèles uniformes (*uniform parallel machines*) : la durée d'exécution des opérations varie uniformément en fonction de la performance de la machine choisie.
  - Machines parallèles indépendantes ou non liées (*unrelated parallel machines*) : les durées opératoires dépendent complètement des machines utilisées.

1.2.2.2 Problèmes à plus d'une opération

Les problèmes de la deuxième catégorie sont dits problèmes d'atelier du fait de la nécessité du passage de chaque job sur deux ou plusieurs machines dédiées. Suivant le mode de passage des opérations sur les différentes machines, trois types d'ateliers sont distingués à savoir :

- ***Problèmes Flow-Shop*** : les ateliers de type *Flow-Shop* appelés également ateliers à cheminement unique, il s'agit d'un ensemble de m machines disposées en séries. Toutes les opérations de tous les jobs passent par les machines dans le même ordre (flot unidirectionnel). Un cas particulier important est celui d'un *Flow-Shop de permutation*, le *Flow-Shop* est dit de *permutation* s'il existe une contrainte selon laquelle la séquence des opérations des différents jobs est la même sur chaque machine [107].
- ***Problèmes Job-Shop*** : dans cette classe d'ateliers, appelés aussi ateliers à cheminements multiples, chaque opération passe sur les machines dans un ordre fixé, mais à la différence du *Flow-Shop*, cet ordre peut être différent pour chaque job (flot multidirectionnel).

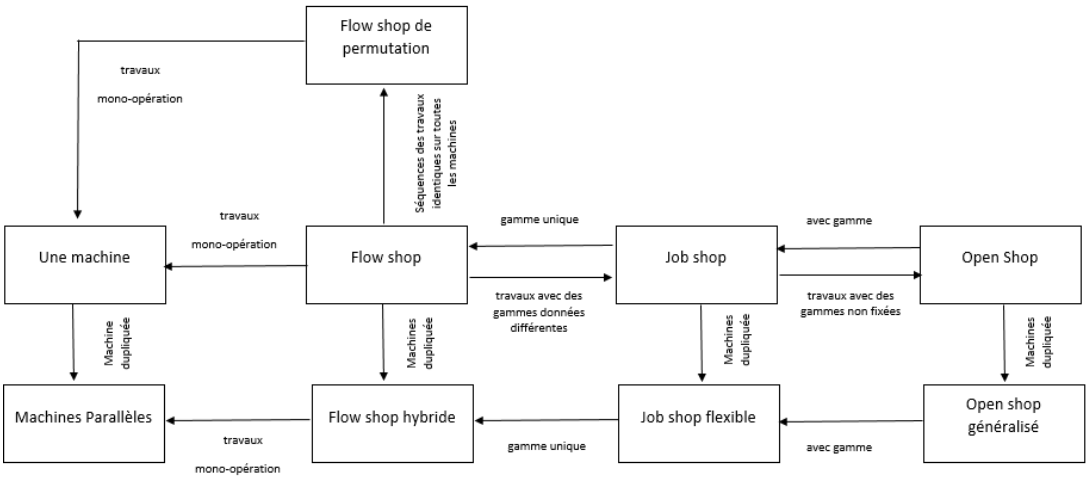


FIGURE 1.3 : Typologie des problèmes d'ordonnancement

— ***Problèmes Open-Shop*** : dans cette classe d'ateliers, appelés aussi ateliers à cheminement libre, les gammes opératoires des différents jobs ne sont pas fixées a priori contrairement au problème d'atelier *Job-Shop*. Les opérations d'un même job peuvent donc être exécutées dans un ordre quelconque. Le problème consiste d'une part à déterminer le cheminement de chaque job et d'autre part à ordonnancer les jobs en tenant compte des gammes trouvées.

À partir de ces ateliers de base, de nombreuses extensions ont été proposés afin de traiter des problèmes industriels spécifiques. L'une des principales extensions consiste à proposer plus d'une machine pour la réalisation d'une opération. Le cas de *flow shop hybride* correspond à une généralisation de l'environnement *Flow-Shop* standard et de l'environnement de *machines parallèles*. Autrement dit, au lieu d'avoir m machines en séries, il existe k étages en séries où dans chaque étage les machines sont en parallèles. De même pour les ateliers de type *Job-Shop* et *Open-Shop* qui donnent respectivement le *Job-Shop flexible* et l'*Open-Shop généralisé*. Ces types d'atelier offrent ainsi plus de flexibilité par rapport aux modèles classiques et nécessitent la détermination des affectations adéquates des opérations aux machines en plus de l'établissement des ordres de passages des différentes opérations sur les machines. La figure 1.3 présente d'une manière exhaustive, une typologie de ces problèmes d'ordonnancement.

1.2.3 Formalisation des problèmes d'ordonnancements

1.2.3.1 Classification et notation

Vu la très grande variété de problèmes d'ordonnancement, Graham et al. [51] ont proposé une méthode de classification permettant une distinction facile entre ces problèmes. Cette méthode se base sur une notation à trois champs distincts notée $\alpha \mid \beta \mid \gamma$.

Le champ α désigne l'environnement machine. Il se compose généralement de deux sous-champs $\alpha_1\alpha_2$. Le premier permet d'indiquer le type de problème étudié (problème à machines parallèles, Flow-Shop, ...) tandis que le second précise le nombre de machines. Le champ β décrit les contraintes liées à l'exécution des tâches. Ce champ peut être vide $\varnothing$ là où aucune contrainte n'est imposée et peut contenir une concaténation de 1 à k sous-champs comme présentés dans le Tableau 1.1. Finalement, le champ γ spécifie le (ou les) critère(s) à optimiser. Il contient dans la plupart des cas un seul champs.

Les différentes notations utilisées pour les valeurs des champs α, β et γ figurent généralement sous forme d'abréviations (cf. tableau1.1) ayant chacune d'elles un sens particulier. La description de ces abréviations est décrite dans les tableaux 1.2, 1.3 et 1.4 respectivement.

TABLE 1.1 : Classification de Graham

Champ		Sous champs	Notation
α	(α_1)	Type de machine	$\{\varnothing, 1, P, Q, R, F, J, O, FH, JF, OG\}$
	(α_2)	Nombre de machines	$\{\varnothing, m\}$
β	(β_1)	Mode d'exécution des jobs	$\{\varnothing, pmtn\}$
	(β_2)	Ressources supplémentaire	$\{\varnothing, res\}$
	(β_3)	Relation de précedence	$\{\varnothing, prec, tree, cahins\}$
	(β_4)	Dates de disponibilité	$\{\varnothing, r_i\}$
	(β_5)	Durées opératoires	$\{\varnothing, p_i = p\}$
	(β_6)	Dates d'échéance	$\{\varnothing, d_i\}$
	(β_7)	Propriété d'attente	$\{\varnothing, nwt\}$
γ		/	$\{C_{max}, \sum w_i C_i, L_{max}, T_{max}, \sum w_i T_i, \sum U_i, \sum w_i U_i\}$

TABLE 1.2 : Interprétation des notations du champ α_1

Notation	Description
1	Problème à une seule machine
P	Problème à machines parallèles identiques
Q	Problème à machines parallèles uniformes
R	Problème à machines parallèles indépendantes
F	Flow-Shop
J	Job-Shop
O	Open-Shop
FH	Flow-Shop hybride
JF	Job-Shop flexible
OG	Open-Shop généralisé

À titre d'exemple :

- $F_2||C_{max}$ dénote un problème d'ordonnancement d'un atelier de type Flow-Shop à deux machines avec minimisation de makespan C_{max}. L'absence de valeurs dans le champ β

TABLE 1.3 : Interprétation des principales notations possibles de sous-champs du champ β

Notation	Description
$pmtn$	La préemption des opérations est autorisée
$prec$	Existence des contraintes de précedence entre les opérations
res	L'opération nécessite l'emploi d'une ou plusieurs ressources supplémentaires
nwt	Les opérations de chaque job doivent se succéder sans attente
$p_i = p$	Les temps d'exécution des tâches sont identiques et égaux à p
r_i	Une date de début au plus tôt est associée à chaque job i
d_i	Une date d'échéance est associée à chaque job i

TABLE 1.4 : Interprétation des principales notations du champ γ

Notation	Expression	Description
C_{max}	$\max_{i\in\{1,\dots,n\}} C_i$	La durée totale de l'ordonnancement
L_{max}	$\max_{i\in\{1,\dots,n\}} C_i - d_i$	Le plus grand retard algébrique
T_{max}	$\max_{i\in\{1,\dots,n\}}\{\max(C_i - d_i, 0)\}$	Le plus grand retard vrai
$\sum[w_i]C_i$	$\sum_{i\in\{1,\dots,n\}}[w_i]C_i$	La somme [pondéré] des dates de fin des tâches
$\sum[w_i]T_i$	$\sum_{i\in\{1,\dots,n\}}[w_i]\{\max(C_i - d_i, 0)\}$	La somme [pondéré] des retards
$\sum[w_i]U_i$	$\sum_{i\in\{1,\dots,n\}}[w_i]\lvert\{J_i/C_i > d_i\}\rvert$	Le nombre [pondéré] des tâches en retard

implique qu'il n'y a pas de contraintes particulières.

- $P_m|pmtn|L_{max}$ désigne le problème de la minimisation du retard maximum L_{max} dans un environnement à m machines parallèles identiques où la préemption est autorisée.
- $1|prec, r_i|\sum w_i C_i$ s'agit d'un problème à une machine dont les tâches présentent une contrainte de précedence et elles ne sont disponibles qu'à la date r_i. Le but est de minimiser la somme pondérée des dates de fin des tâches.

Bien que les valeurs affectées aux différents champs permettent de modéliser une variété de problèmes d'ordonnancement, multitudes extensions ont été proposées en particulier au niveau des champs β et γ dans le but de supporter d'autres catégories de problèmes particulières .

1.2.3.2 Modélisation

La modélisation représente une étape très importante dans la résolution d'un problème. Elle est caractérisée par une écriture simplifiée de toutes les données de problème tout en utilisant un formalisme bien adapté. Principalement, deux types de modélisation existent pour les problèmes d'ordonnancement. La modélisation graphique sous forme de graphe et la modélisation analytique sous forme de programme mathématique.

1.2.3.2.1 Modélisation graphique

Cette modélisation apporte une aide incontestable à la manipulation d'un problème. Elle a connu une très importante évolution surtout avec l'apparition des Réseaux de Pétri, qui permettent de traduire plusieurs notions fondamentales ayant un lien avec les problèmes d'ordonnancement (les durées opératoires, les gammes, etc.). Cette méthode de modélisation présente un caractère visuel facilitant la vérification de la cohérence du problème considéré et l'interprétation des solutions, ce qui lui a permettre d'être la méthode la plus utilisé dans la littérature. Dans cette modélisation, deux types de graphes sont utilisés, le graphe PERT et le graphe des potentiels.

- La méthode PERT (Program Evaluation Research Task) est une méthode Américaine élaborée pour résoudre les problèmes d'ordonnancement. Elle consiste à associer à un problème d'ordonnancement un graphe constitué d'un ensemble de nœuds reliés entre eux par des arcs où :
  - Chaque nœud correspond à un instant de déclenchement d'exécution d'une tâche représentée par l'arc sortant du nœud et également à un instant d'achèvement d'une tâche représentée par l'arc entrant au nœud.
  - Sur les arcs, nous trouvons généralement les durées d'exécution associées aux tâches. Dans le cas où la valeur d'un arc est égale à 0 cela présente une contrainte de précédence.
- La méthode des potentiels a été développée vers la fin des années 50 parallèlement à la méthode PERT. Elle est appelée également la méthode MPM (méthode des potentiels Metra) ou encore méthode des potentiels – tâches. Elle se base aussi sur une modélisation par graphe constitué d'un ensemble de nœuds et d'arcs. Les nœuds du graphe représentent les tâches composant les travaux à réaliser, auxquels s'ajoutent deux sommets fictifs appelés « source » et « puits » correspondant respectivement au début et à la fin des travaux. Ainsi, les arcs peuvent être de deux types :
  - Arcs conjonctifs connectent deux tâches consécutives d'un même travail (contraintes de précedence). Ces arcs sont pondérés par la durée opératoire de la tâche, exceptés les arcs de la source qui sont pondérés par 0.
  - Arcs disjonctifs connectent deux tâches appartenant à des travaux différents qui utilisent la même machine (contraintes de ressources).

1.2.3.2.2 Modélisation analytique

Un problème d'ordonnancement peut également être modélisé sous une forme analytique. Cette modélisation, couramment utilisée, est souvent sous forme de programme mathématique dont les données, les contraintes et la fonction d'évaluation des critères sont écrites sous forme d'équations et d'inéquations mathématiques. Cette modélisation permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également de le résoudre.

1.2.3.3 Représentation des solutions

Afin de visualiser une solution d'un problème d'ordonnancement, nous utilisons couramment une représentation graphique appelée diagramme de Gantt. Ce diagramme constitue un formalisme graphique qui a été mis au point par Henry Gantt en 1910.

Dans le cas d'un problème d'atelier, le diagramme est formé de deux axes orthogonaux et peut avoir deux représentations :

1. La représentation « Jobs »
2. La représentation « Machines »

Pour la représentation « Jobs », l'axe horizontal représente le temps et l'axe vertical représente les jobs. Pour chaque job, nous dessinons des rectangles représentant l'ensemble des machines utilisées dans le temps pour la réalisation du job.

Pour la représentation « Machines », l'axe horizontal représente le temps et l'axe vertical représente les machines. Pour chaque machine, nous représentons l'ensemble des opérations effectuées dans le temps par des barres ayant des longueurs proportionnelles à leurs durées opératoires. Cette représentation permet de visualiser l'occupation des machines, l'enchaînement des opérations sur celles-ci, les dates de début et de fin de chaque opération ainsi que le temps d'inactivité des machines.

À titre d'illustration, une solution réalisable du problème $F3|perm|C_{max}$ avec les données fournies dans le tableau 1.5 est donnée par la figure 1.4. Nous utilisons la représentation (b), pour présenter notre solution dans ce qui suit.

TABLE 1.5 : Temps opératoires pour 3 jobs et 3 machines

J_i	p_{i1}	p_{i2}	p_{i3}
J_1	2	1	3
J_2	3	5	2
J_3	4	2	1

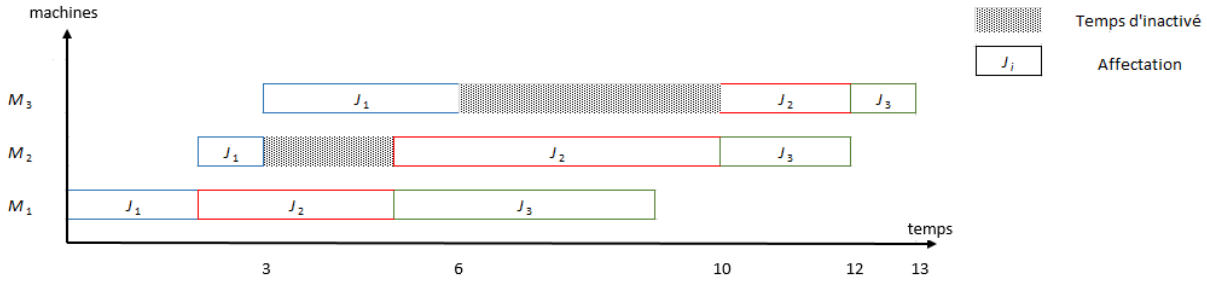


FIGURE 1.4 : Diagramme de Gantt d'une solution du problème $F3|perm|C_{max}$

1.3 La théorie de la complexité

Les problèmes d'optimisation sont des problèmes dont la résolution consiste à trouver parmi un ensemble de solutions celle qui répond le mieux à certains critères décrits sous forme d'une fonction objectif. Lorsque le domaine de solutions est discret, on parle alors de problèmes d'optimisation combinatoire. Les problèmes d'ordonnancement font partie des problèmes d'optimisation combinatoire.

Selon Cook [26], la théorie de la complexité a pour but d'analyser les coûts de résolution, notamment en terme de temps de calcul, des problèmes d'optimisation combinatoire.

La définition de la complexité d'un problème découle de la définition de la complexité des algorithmes [2]. Informellement, la complexité d'un problème peut être définie comme la complexité du meilleur algorithme permettant sa résolution.

1.3.1 Complexité algorithmique

La complexité algorithmique est un concept fondamental qui permet de mesurer les performances d'un algorithme. Ces performances sont évaluées sur la base du temps alloué pour l'exécution de l'algorithme ou encore par rapport à l'espace mémoire requis pour résoudre le problème. Généralement, le temps d'exécution est le facteur dominant pour déterminer l'efficacité d'un algorithme, pour cela, on se concentre principalement sur ce facteur.

La complexité temporelle d'un algorithme représente le nombre maximum d'opérations élémentaires effectuées pour résoudre un problème donné. Cette complexité se base essentiellement sur la mesure d'un ordre de grandeur qui est évalué en fonction de la taille du problème n. La notation O est utilisé pour représenter cet ordre de grandeur. Par exemple, pour un algorithme donné, si la solution est donnée en environ n opérations, on dit que l'algorithme a une complexité en $O(n)$.

Definition 1.3.1 *Un algorithme est dit polynomiale si pour tout n, l'algorithme s'exécute en moins de n^k opérations élémentaires (k étant une constante).*

En d'autres termes, un algorithme polynomial est un algorithme dont la complexité temporelle est bornée par un polynôme p de n, $O(p(n))$: par exemple $O(n^k)$ avec k une constante.

Definition 1.3.2 *Un algorithme est dit non-polynomial si le nombre d'opérations n'est pas borné par un polynôme de n.*

En d'autres termes, si le nombre d'opérations est borné par une forme exponentielle de la taille de problème alors l'algorithme est dit non-polynomial ou exponentiel : par exemple $O(a^n)$ avec $a > 1$ une constante.

2. Le mot "algorithme" vient du nom du mathématicien Al Khuwarizmi (latinisé au moyen-âge en Algoritmi) et désigne un ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé.

1.3.2 Complexité problématique

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

Selon Garey et Johnson [46], la théorie de complexité d' problème d'optimisation se limite à la seule étude de problème de décision. Celui-ci, aussi appelé problème de reconnaissance, est un problème dont la résolution se limite à la réponse par « oui » ou par « non » à la question de savoir s'il existe une solution à un problème donné.

Pour chaque problème d'optimisation, on peut associer un problème de décision défini par la question suivante : « Existe-t-il une solution telle que son évaluation est majorée (Resp. minorée) par une constante fixée ? ». Ceci est pour un problème d'optimisation, dont la fonction objectif est la minimisation (resp. maximisation). Par exemple, dans un problème de type Flow-Shop $F_m||C_{max}$ ayant comme fonction objectif la minimisation du makespan, le problème de décision associé est : « Existe-t-il un ordonnancement pour un Flow-Shop où le makespan est inférieur à une certaine valeur z ».

Alors si la réponse a la question existe et est obtenue grâce à un algorithme polynomial alors il existera un algorithme polynomial résolvant le problème d'optimisation associé. Cette propriété nous permet de classer les problèmes d'optimisation grâce à leurs problèmes de décision. Ces problèmes peuvent être classés en deux classes principales : la classe P et la classe NP.

- **La classe P (Polynomiale)** : les problèmes appartenant à la classe P sont ceux dont le problème de décision correspondant peut être résolu à l'optimum, à l'aide d'un algorithme en temps polynomial. C'est en quelque sorte la classe des problèmes dits « facile ».
- **La classe NP (Non-deterministic Polynomial)** : cette classe a un nom trompeur, NP ne correspond pas à Non Polynomial, mais à Polynomial Non-déterministe ou en anglais « Non-deterministic Polynomial ». Cette classe est une extension de la classe P, elle représente la classe des problèmes de décision pour lesquels un algorithme non-déterministe [3] peut vérifier en temps polynomial la validité d'une solution du problème traité.

La classe NP contient donc des problèmes plus « difficiles » que la classe P. Mais de même dans la classe NP, on peut trouver des problèmes encore plus difficiles. Pour cette raison, la classe des problèmes NP-complet a été créée. Pour définir la classe NP-Complet, il est intéressant de définir la notion de réduction polynomiale.

Le concept de réduction consiste à convertir un problème en un autre dans le but de pouvoir utiliser la solution du second problème pour résoudre le problème initial. La réduction d'un problème P1 vers un problème P2 est dite polynomiale si et seulement s'il existe un algorithme polynomial construisant les données d'une instance de P2 à partir des données d'une instance quelconque de P1, de sorte que la réponse à P1 soit « oui » si et seulement si la réponse à P2 est « oui ».

3. Un algorithme non-déterministe est un algorithme muni d'une instruction qui permette, chaque fois qu'elle ait appliqué, de faire le bon choix.

Ainsi, s'il existe un algorithme polynomial pour résoudre P2, il est facile d'en trouver un pour résoudre P1. La réduction polynomiale de P1 en P2 est notée P1 $\propto$ P2, cette propriété est transitive, c'est-à-dire si P1 $\propto$ P2 et P2 $\propto$ P3 alors P1 $\propto$ P3. De même, s'il n'existe pas un algorithme polynomial pour P1 alors il n'existera pas non plus un algorithme en temps polynomial pour P2, puisque P1 est un cas particulier du problème P2.

— **La classe NP-complet** : cette classe réunit l'ensemble des problèmes de décision qui n'ont pas d'algorithmes polynomiaux connus pour leurs résolutions. En effet, un problème de décision P est dit NP-complet, si :
  — Il appartient à la classe NP
  — Tout problème P′de la classe NP peut se réduire polynomialement à lui.

$$\forall\ P' \in NP,\ P' \propto P$$

  Certains auteurs font une distinction entre les problèmes faiblement NP-complet et les problèmes fortement NP-complet. Un problème est dit NP-complet au sens faible si on peut construire des algorithmes très efficaces appelés les algorithmes « pseudo-polynomiaux [4] » . Dans le cas contraire, il est dit NP-complet au sens fort.

Definition 1.3.3 *Un problème d'optimisation est dit NP-difficile au sens fort (resp. NP-difficile au sens faible) si le problème de décision qui lui correspond est NP-complet au sens fort (resp. NP-complet au sens faible).*

Les différentes classes présentées précédemment ne sont que les principales classes dans la théorie de la complexité. Pour plus amples détails et pour une présentation bien plus formelle, nous conseillons le lecteur de se référer à [46].

1.3.3 Hiérarchie de complexité pour les problèmes d'ordonnancement

La théorie de la complexité représente un outil important dans la théorie d'ordonnancement puisqu'elle permet d'établir une classification des problèmes d'ordonnancement en plusieurs classes de difficulté et donne une orientation sur la méthode de résolution de ces problèmes.

On utilisant la réduction polynomiale, une hiérarchie de complexité (aussi nommé l'arbre de réduction) de certains problèmes d'ordonnancement peut être établie en fonction de l'environnement machine, les contraintes ainsi que les critères. La figure 1.5 présente une hiérarchie de la complexité en fonction de l'environnement machine. Cet arbre s'interprète de la manière suivante : pour des contraintes données et pour un critère donné, si un problème est NP-difficile, tous ses successeurs dans l'arbre le sont également. La même hiérarchie de complexité peut être tracée pour les contraintes et les critères. Figure 1.6 présente la hiérarchie de complexité en fonction des contraintes alors que la figure 1.7 illustre la hiérarchie de complexité en fonction des critères à optimiser. Ces deux arbres ont une interprétation similaire à celle d'arbre d'environnement machine.

4. Un algorithme pseudo-polynomial est un algorithme dont la complexité est bornée par une fonction polynomiale de la représentation unaire des données du problème considéré

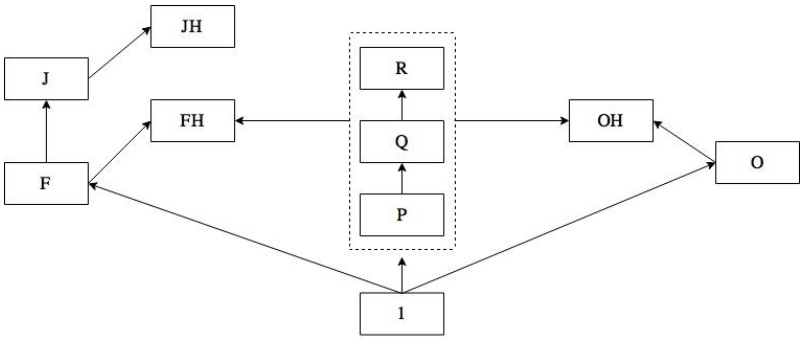


FIGURE 1.5 : Hiérarchie de complexité des problèmes d'ordonnancement en fonction de l'environnement machine [42]

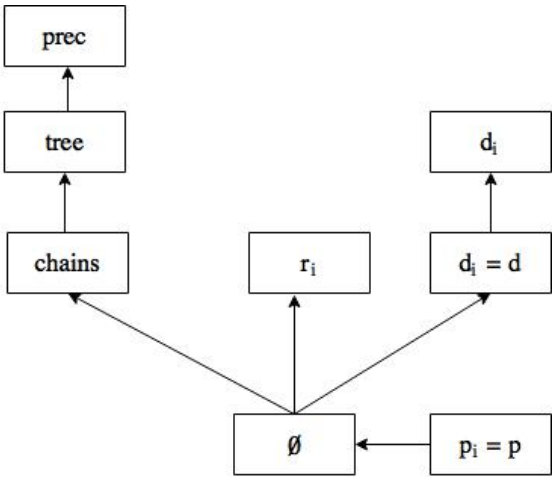


FIGURE 1.6 : Hiérarchie de complexité des problèmes d'ordonnancement en fonction des contraintes [42]

En se basant sur les hiérarchies de complexité présentées par la figure 1.5 et 1.7, ainsi que les résultats de la littérature, la classe de complexité des problèmes d'ordonnancement mono-critère sans contraintes sont représenté dans le Tableau 1.6 où P représente la classe P, NP : classe NP-difficile au sens fort et NP* : classe NP-difficile au sens faible.

Nous référons le lecteur au site web `http://www2.informatik.uni-osnabrueck.de/knust/class/` développé par Brucker et al. [17] pour des résultats de complexité des problèmes d'ordonnancement mono-critère avec considération de contraintes. Pour la complexité des problèmes d'ordonnancement multi-critère, le lecteur intéressé peut se référer à [127].

1.4 Conclusion

Dans ce chapitre, nous avons positionné l'ordonnancement dans le système de production, où l'accent est mis sur son importance cruciale dans le processus de production. Nous avons présenté les différentes notations, définitions, classification des problèmes d'ordonnancement, ainsi que

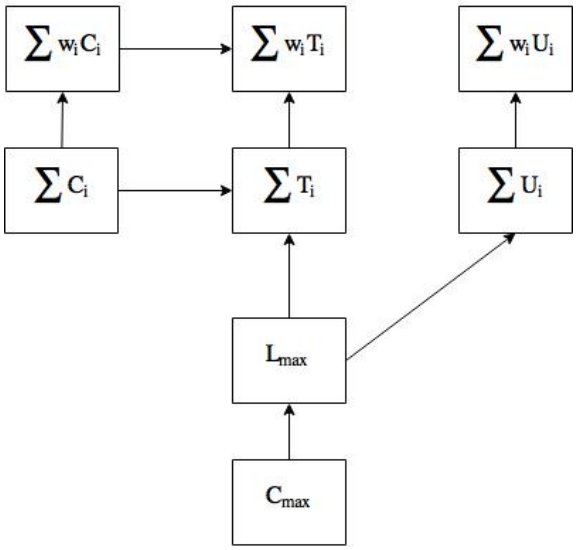


FIGURE 1.7 : Hiérarchie de complexité des problèmes d'ordonnancement en fonction des critères [106]

TABLE 1.6 : Classe de complexité des problèmes d'ordonnancement mono-critère

	Critères		C_{max}	L_{max}	$\sum C_i$	$\sum w_i C_i$	$\sum T_i$	$\sum w_i T_i$	$\sum U_i$	$\sum w_i U_i$
Problèmes	Une seule machine		P	P	P	P	NP*	NP	P	NP*
	Machines parallèles	P	NP	NP	P	NP	NP	NP	NP	NP
		Q	NP	NP	P	NP	NP	NP	NP	NP
		R	NP	NP	P	NP	NP	NP	NP	NP
	Flow-Shop		NP	NP	NP	NP	NP	NP	NP	NP
	Job-Shop		NP	NP	NP	NP	NP	NP	NP	NP
	Open-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 géneralisé		NP	NP	NP	NP	NP	NP	NP	NP

les types et les représentations des ordonnancements. Nous avons exposé à la fin de ce chapitre la complexité des problèmes d'optimisation et nous avons mentionné que l'étude de complexité des problèmes d'ordonnancement conduit au choix de leurs méthodes de résolution. Donc, il est essentiel d'identifier la classe de complexité de ces problèmes. Si le problème d'ordonnancement appartient à la classe P donc il dispose d'un algorithme polynomial spécifique qui permette de le résoudre. Sinon, si le problème appartient à la classe NP-difficile alors pour l'appréhender, différentes méthodes de résolution sont proposées dans la littérature. Le chapitre suivant permettra d'expliciter ces méthodes.

Chapitre 2

Techniques d'optimisation

Les méthodes exploitées dans la résolution des problèmes d'ordonnancement sont généralement celles de la discipline de la recherche opérationnelle. Cette dernière regroupe différentes méthodes et techniques pour la résolution des problèmes d'optimisation combinatoire.

Dans ce chapitre, nous allons passer en revue les méthodes les plus connues en les répertoriant en méthodes exactes et méthodes approchées. Pour chaque méthode, l'idée générale est rapportée. Plus de détails ne sont donnés que pour les méthodes que nous utilisons dans notre étude. La plupart des références données concernent la résolution des problèmes d'ordonnancement Flow-Shop avec ou sans contraintes.

La section 2.1 concerne les méthodes exactes qui tentent de trouver des solutions optimales aux problèmes d'optimisation, en particulier les problèmes d'ordonnancement. La section 2.2 présente les méthodes approchées qui sont utilisés comme une alternative aux méthodes exactes pour trouver de bonnes solutions. La section 2.3 concerne les méthodes hybrides qui représentent une combinaison de plusieurs méthodes (exactes et/ou approchées).

2.1 Méthodes exactes

Les méthodes exactes sont celles qui nous permettent de fournir une solution optimale pour les problèmes d'optimisation combinatoire, grâce à une exploration intelligente de l'espace des solutions. Ces méthodes se caractérisent par un temps de calcul souvent exponentiel, ce qui explique qu'elles ne sont utilisables que pour des problèmes de petites tailles. Les méthodes les plus couramment utilisées sont : la méthode de séparation et évaluation, la programmation dynamique et la programmation linéaire. Ces méthodes sont décrites dans les sous-sections suivantes.

2.1.1 Méthode de séparation et évaluation

La méthode de séparation et évaluation, plus connue sous son appellation anglaise Branch and Bound (B & B) est considérée parmi les méthodes exactes les plus reconnues dans la résolution optimale des problèmes d'optimisation combinatoires. Cette méthode est basée sur une énumération implicite et intelligente de l'ensemble des solutions, mais l'analyse des propriétés du problème évite l'énumération de larges classes ne contenant pas de solution optimale. En d'autres termes, seulement les solutions potentiellement bonnes seront énumérées.

Le principe de cette méthode repose comme son nom l'indique sur deux notions clé : la séparation et l'évaluation. La séparation consiste à décomposer l'ensemble des solutions en plusieurs sous-ensembles qui sont décomposés à leur tour selon une démarche itérative. Ce processus peut se visualiser sous la forme d'un arbre d'énumération ou les nœuds de l'arbre correspondent aux sous-ensembles et les feuilles correspondent à des solutions réalisables. Pour accélérer la recherche de la solution optimale en évitant l'exploration inutile de certains nœuds, on utilise une procédure d'évaluation. Cette dernière réduit le nombre de solutions explorées en permettant de prouver mathématiquement que l'ensemble des solutions réalisables associées à un des nœuds de l'arbre ne contient pas une solution intéressante pour le problème initial. Pour cela, la méthode la plus générale consiste à déterminer une borne inférieure (dans le cas des problèmes de minimisation) ou une propriété de dominance pour confirmer qu'une branche ne contient pas de solution optimale.

Cette méthode a été introduite pour la première fois par Dantzig et al. [28] pour la résolution du problème du voyageur de commerce. Depuis, des centaines, voire des milliers de procédures par séparation et évaluation ont été proposées pour des problèmes d'ordonnancement dont on peut citer [20, 89].

2.1.2 Programmation dynamique

La programmation dynamique est considérée comme une approche permettant de concevoir des méthodologies robustes de résolution exacte des problèmes d'optimisation combinatoire dont la fonction objectif présente une propriété dite de décomposabilité [91]. Cette méthode a été initiée par Richard Bellman dans les années cinquantes [10] dans le cadre d'une recherche du plus court chemin dans un graphe. Elle est conçue principalement pour la résolution de certains problèmes qui présentent certaines caractéristiques permettant leur résolution à l'aide d'une formule de récurrence. Son efficacité repose sur le principe d'optimalité, dit de Bellman « Toute politique optimale est composée de sous politiques optimales ».

Alors, le principe de la programmation dynamique est de transformer la résolution d'un problème P en résolution d'une série de sous-problèmes $(P_0, P_1, ..., P_n)$ liés par une relation de récurrence portant sur la valeur de la fonction objectif. Les informations obtenues lors de la résolution des sous-problèmes P_0 à P_{k-1} sont utilisées pour résoudre de façon optimale le sous-problème P_k.

Ainsi, pour obtenir la solution optimale du problème P, il suffit de revenir en arrière $(P_n, ..., P_0)$ à travers l'ensemble des décisions prises et stockées à chaque résolution de sous-problème, comme le montre la figure 2.1.

Cette méthode peut s'avérer couteuse en temps et en mémoire. Cependant, elle permet de fournir des algorithmes polynomiaux et pseudo-polynomiaux pour des cas particuliers [1].

Comme exemples d'utilisation de la programmation dynamique pour l'ordonnancent d'atelier Flow-Shop, nous citons les travaux de [102, 133]

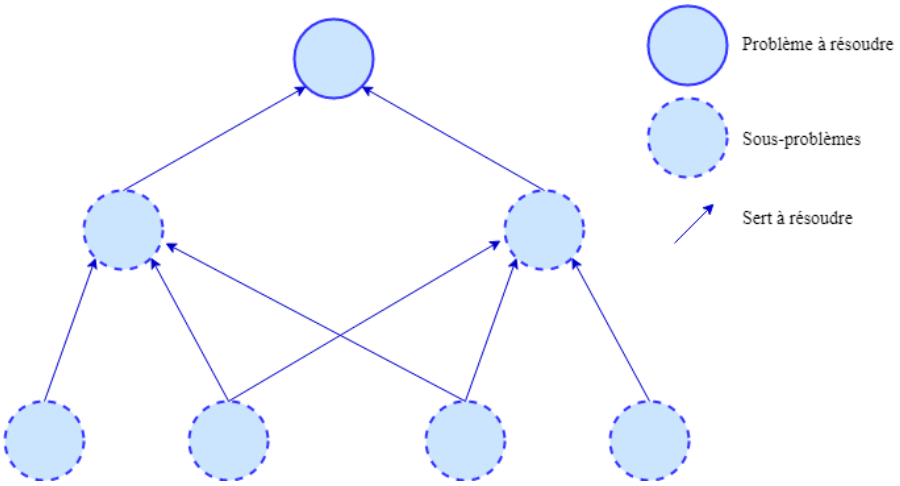


FIGURE 2.1 : Schéma de la programmation dynamique

2.1.3 Programmation linéaire

La programmation linéaire (PL) est l'une des méthodes les plus puissantes en recherche opérationnelle. Elle se base sur la modélisation mathématique du problème. Son utilisation demande que le problème posé puisse se ramener à un programme linéaire, ce dernier se compose d'une fonction linéaire à optimiser (maximiser ou minimiser) désignant l'objectif du problème et d'un ensemble d'équations et/ou d'inéquations linéaires représentant les contraintes imposées par le problème.

Généralement, dans le processus de modélisation d'un problème, nous commençons par la définition des variables de décision. En ordonnancement, ces variables peuvent être des variables indexées par le temps (variables temporelles) indiquant si un job est en train d'être exécuté à un instant t, des variables de précédence indiquant si deux jobs se précèdent dans une séquence et des variables de positions indiquant si un job est en position k dans une séquence [21]. La figure 2.2 montre un exemple de ces trois variables sur une séquence de cinq jobs.

Un modèle linéaire peut s'exprimer sous la forme compacte suivante :

$$\begin{cases} Max \sum_{i=1}^{n} c_i X_i \\ s.c. \\ \sum_{i=1}^{n} a_i X_i \geqslant b_i \\ X_i \geqslant 0 \end{cases}$$

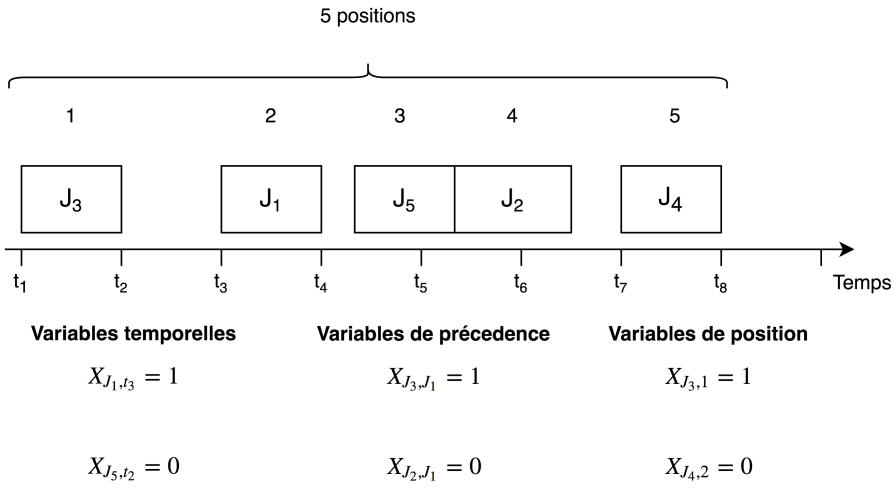


FIGURE 2.2 : Exemple de variables de décision

Suivant l'ensemble de définition des variables, nous distinguons trois types pour cette méthode :

- Programme linéaire continu si les variables sont des nombres réels.
- Programme linéaire en nombres entiers si les variables sont des entiers.
- Programme linéaire en nombres mixtes si les variables sont des entiers et des réels.

La résolution exacte des problèmes linéaires à variables continues utilise des méthodes telles que la méthode de simplexe [29], ou la méthode des points intérieurs. Lorsqu'il y a des variables discrètes, on parle de programmes linéaires en nombre entiers (*PLNE* ou *IP*) ou encore des programmes linéaires à variables mixtes (*PLM* ou *MIP*). Pour résoudre ces problèmes, des approches polyédriques sont utilisées telle que la génération de coupes [50] ou des méthodes issues de l'intelligence artificielle (programmation par contraintes, et des relaxations diverses).

Grâce aux progrès de l'informatique, l'offre de logiciels commerciaux permettant de résoudre des programmes linéaires a considérablement augmenté. Parmi ces logiciels, on peut citer CPLEX, XPRESS, GAMS.

Parmi les modèles linéaires proposés pour la résolution des problèmes d'ordonnancement de type Flow-Shop classique, nous citons le modèle de Wagner [136], Wilson [140], Manne [88] et Liao et al. [86].

Dans Tseng et al. [131] une étude a été menée pour comparer les quatre modèles cités précédemment. Les auteurs utilisent le temps de calcul nécessaire pour chaque modèle pour trouver

la solution optimale comme critère pour comparer les modèles linéaires. Les résultats obtenus sont les suivants : le modèle de Wagner est le meilleur modèle linéaire suivi du modèle de Wilson comme la deuxième meilleure formulation mathématique pour le problème Flow-Shop de permutation et au final le modèle de Manne traîne derrière le modèle de Liao et You.

Nous citons aussi quelques références utilisant la programmation linéaire pour la résolution des problèmes Flow-Shop sous divers contraintes [7, 39, 130]. En particulier, Trabelsi et al. [130] ont proposé un modèle mathématique linéaire en nombres entiers à base de variables de position pour résoudre le problème Flow-Shop sous contraintes de blocage.

Ces méthodes exactes sont souvent utilisées pour pouvoir évaluer l'erreur que commettent les méthodes approchées qui donnent beaucoup plus rapidement une solution aux problèmes considérés.

2.2 Méthodes approchées

Les difficultés rencontrées par les méthodes exactes pour la résolution des problèmes d'optimisation de grande taille en un temps raisonnable ont incité les chercheurs à trouver des alternatives connues sous l'appellation : méthodes approchées. Ces méthodes nous permettent de trouver une solution acceptable en un temps de calcul raisonnable. La performance de telles méthodes est généralement donnée par l'estimation du pourcentage d'erreur entre la valeur de la solution fournie et la valeur de la solution optimale si elle est calculable. Dans les cas où la solution optimale est non calculable, il est également possible d'étudier et d'évaluer expérimentalement le comportement d'une méthode en comparant ses performances soit à celles d'autres méthodes, soit à des bornes inférieures.

Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle heuristiques. Cependant, lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de métaheuristiques.

2.2.1 Heuristiques

Les heuristiques sont des méthodes empiriques qui cherchent de bonnes solutions à un coût raisonnable sans être en mesure de garantir l'optimalité. Ces méthodes se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Leur principe général est d'intégrer des stratégies de décision pour construire une solution proche de l'optimum. Les heuristiques sont souvent utilisées pour des problèmes particulaires, de sort qu'une méthode qui est destinée à résoudre un problème donné ne peut pas être utilisée pour résoudre un autre problème.

Parmi ces heuristiques, on peut citer l'algorithme de Johnson [65] développée dans le cadre de la recherche d'une séquence de durée minimale dans un atelier de type Flow-Shop de permutation

à deux machines et l'algorithme NEH [97] pour $F_m||C_{max}$.

Nous retrouvons également parmi les heuristiques les plus couramment utilisées, les algorithmes de liste dont le principe consiste à trier la liste des tâches selon une stratégie de décision appelée « règles de priorité » telle que SPT (Shortest Processing Time), LPT (Longest Processing Time), EDD (Earliest Due Date), FIFO (First In First Out), LIFO (Last In Last Out), FAM (First Available Machine), etc.

Les règles de priorité sont des règles qui définit les priorités entre les travaux en attente de traitement. Chaque fois qu'une machine devient inactive, une règle de priorité inspecte les travaux en attente et sélectionne celui avec la plus haute priorité.

La recherche sur les règles de priorité est active depuis plusieurs décennies et de nombreuses règles ont été élaborées et étudiées dans la littérature. Aussi, des approches permettant de combiner, par agrégation, différentes règles élémentaires afin de réaliser des compromis pour tenter de satisfaire des objectifs multiples ont également été développées dont nous citons, la concaténation de l'algorithme de Johnson [65] avec la règle FAM pour résoudre le problème Flow-Shop hybride à deux étages, nous retrouvons aussi l'application de la règle LPT avec FAM pour résoudre le problème $P||C_{max}$, cette dernière consiste à ordonner les tâches selon l'ordre décroissant de leurs temps opératoires et à affecter chaque tâche à la première machine libre.

Les heuristiques sont des méthodes de résolution non fondées sur un modèle formel et qui fournissent rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale. Elles se distinguent par leur rapidité et leur grande simplicité. Parfois, ces heuristiques peuvent donner de solutions optimales pour des problèmes particuliers en ordonnancement. A titre d'example, SPT qui consiste à séquencer les tâches dans l'ordre croissant des durées opératoires est optimale pour $1||\sum C_i$ [122], EDD qui consiste à séquencer les tâches dans l'ordre croissant des dates de fin au plus tard est optimale pour $1||L_{max}$ et $1||T_{max}$ [61], l'algorithme de Johnson [65] est optimale pour $F_2||C_{max}$, etc.

2.2.2 Metaheuristiques

Les métaheuristiques sont des techniques puissantes et généralement applicable à un grand nombre de problèmes. Formellement, une métaheuristique fait référence à une stratégie de maître interactif qui guide et modifie les opérations heuristiques subordonnées en combinant intelligemment différents concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou proches de d'optimales [100]. Les performances de ce type de méthodes dépendent de la qualité de la solution obtenue et du temps de calcul nécessaire.

Les métaheuristiques n'offrent généralement pas de garantie d'optimalité, mais leur succès est dû à la capacité de résoudre en pratiques certains problèmes difficiles. Ces méthodes ont en commun un certain nombre d'avantages [14, 35] :

— Elles sont souvent inspirées par des analogies avec la physique (recuit simulé), avec la biologie (algorithmes génétiques, recherche tabou, etc) ou avec l'éthologie (colonies de fourmis, essaims particulaires).
— Certaines de ces méthodes se caractérisent par un effet stochastique. Cet effet possède l'avantage de diversification des différentes solutions explorées au cours du processus de calcul.
— Elles sont souvent d'origine discrète à l'exception de certaines comme les essaims de particules.
— La combinaison de deux méthodes approchées ou plus est généralement permise.

Elles partagent aussi les mêmes inconvénients suivants :

— Difficulté de réglage des paramètres.
— Temps de calcul parfois élevé.

Les métaheuristiques sont habituellement classées en fonction du nombre de solutions qu'elles manipulent : les métaheuristiques à solution unique telles que le recuit simulé et la recherche tabou, et les métaheuristiques à population de solutions telles que les algorithmes génétiques, optimisation par essaims particulaires, les colonies de fourmis, etc.

2.2.2.1 Métaheuristiques à solution unique

Les métaheuristiques à solution unique appelées aussi méthodes de recherche locale ou méthodes de trajectoire, se basent sur la notion de voisinage qui représente un ensemble de solutions obtenues à partir d'une solution donnée en effectuant un certain nombre de transformations. Le but de ces transformations locales est d'explorer le voisinage de la solution courante afin d'améliorer progressivement sa qualité au cours des différentes itérations. Comme le montre la figure 2.3, ces techniques tendent à trouver l'optimum global (G) sans être piégé par les optima locaux (L).

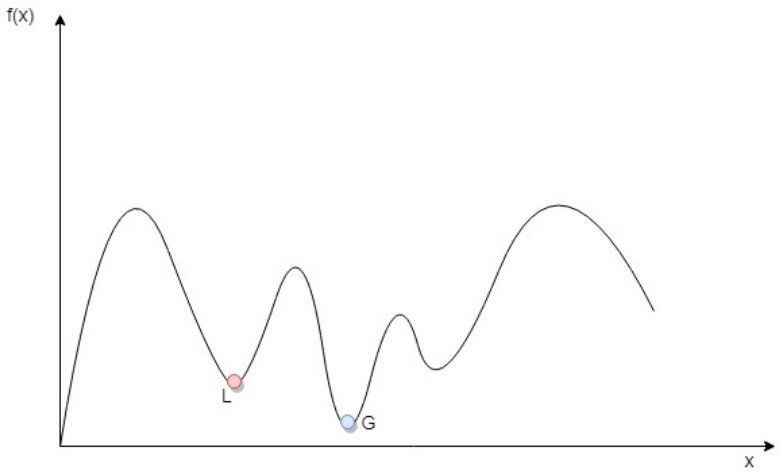


FIGURE 2.3 : Optima local et global

Il existe un grand nombre de méthodes à recherche locale. Les plus répondues sont présentées

par la suite.

2.2.2.1.1 Le Recuit Simulé

Le recuit simulé (SA : Simulated Annealing) est une métaheuristique connue pour être la plus ancienne [15]. Elle a été indroduite par Kirkpatrick et al. [72] pour résoudre des problème d'optimisation combinatoire, mais les origines de la méthode remontent aux expériences réalisées par Metropolis et al. [90].

Cette méthode est inspirée d'un processus utilisé en métallurgie dont le principe consiste à porter un métal à une température très élevée, puis à le refroidir très lentement. Cela permet la solidification du métal dans une structure d'énergie minimale.

La méthode du recuit simulé transpose le processus du recuit à la résolution d'un problème d'optimisation là où une solution est associée à un état du métal, son équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine.

En optimisation, le principe de cette méthode repose sur une procédure itérative qui cherche des solutions de coûts plus faibles tout en acceptant de manière contrôlée des solutions qui dégradent la fonction objectif. À chaque itération, une nouvelle solution $s\prime$ est choisie parmi l'ensemble des voisins $N(s)$ de la solution courante s. Les nouvelles solutions sont toujours acceptées si le coût de la fonction objectif diminue, c'est-à-dire si $f(s\prime) \leq f(s)$ avec $f(s)$ le coût de la fonction objectif associée à la solution s. Dans le cas contraire, la solution $s\prime$ est acceptée ou rejetée selon la probabilité de Métropolis [90]. Soit $\Delta f = f(s\prime) - f(s)$ la différence entre les fonctions objectifs de la nouvelle solution $f(s\prime)$ et la solution courante $f(s)$.

L'acceptation d'une solution est déterminée de la façon suivante : on choisit un nombre aléatoire, noté q, de l'intervalle $[0, 1]$ que l'on compare avec la probabilité

$$prob(\Delta f, T) = \exp(-\Delta f / k.T)$$

où T est un paramètre de contrôle appelé la température et k est la constante dite de Boltzmann. Si $q \leq prob(\Delta f, T)$ alors la nouvelle solution est adoptée comme solution courante. Autrement, la nouvelle solution est rejetée.

La probabilité $prob(\Delta f, T)$ dépend essentiellement de deux facteurs :

- De l'importance de la dégradation Δf, les dégradations plus faibles sont plus facilement acceptées.
- D'un paramètre de contrôle T (la température), une température élevée correspond à une probabilité plus grande d'accepter des dégradations.

L'acceptation des solutions moins bonnes permet à l'algorithme de sortir d'optima locaux. Cette procédure est répétée pendant un nombre spécifié de cycles jusqu'à atteindre un quasi-équilibre.

La température est ensuite diminuée et une nouvelle itération est exécutée. En pratique, l'algorithme s'arrête lorsque soit la température est plus petite qu'une certaine valeur T_f appelée la température finale, soit lorsque aucune configuration voisine n'a été acceptée pendant un certain nombre d'itérations [141]. Ce principe peut se résumer par l'algorithme 1.

Algorithme 1 Pseudo code de la méthode du Recuit Simulé.

Début
$s \leftarrow s_0$ ▷ s_0 est la solution initiale
$T \leftarrow T_0$ ▷ T_0 est la température initiale du système
Tant que $T > T_f$
  Générer une solution $s' \in N(s)$ aléatoirement
  Calculer $\Delta f = f(s') - f(s)$
  if $\Delta f \leq 0$ **then**
    $s \leftarrow s'$
  else
    Calculer $prob(\Delta f, T) \leftarrow \exp(-\Delta f / T)$
    Générer q uniformément dans l'intervalle $[0, 1]$
    if $q < prob(\Delta f, T)$ **then**
      $s \leftarrow s'$
    else
      s' est rejetée
    end if
  end if
  $T \leftarrow T \times \theta$ ▷ $0 < \theta < 1$ coefficient de refroidissement
Fin Tant que
Fin

Parmi les difficultés de cette méthode est la détermination de la valeur initiale de la température T_0 et le coefficient de refroidissement de la température θ. Le réglage de ces paramètres est assez délicat et repose sur des essais. L'avantage est la flexibilité concernant les évolutions du problème et la facilité de mise en œuvre.

Les algorithmes de recuit simulé sont largement utilisés pour les problèmes d'ordonnancement. En 1989, Osman et al. [101] ont proposé un algorithme de recuit simulé afin de minimiser le makespan dans un environnement de type Flow-Shop de permutation, les résultats obtenus montrent l'efficacité de l'algorithme proposé par rapport à des heuristiques constructives bien connue. Blazewicz et al. [12] ont développé un algorithme de recuit simulé pour un problème d'ordonnancement d'atelier de type Flow-Shop avec une contrainte de disponibilité de ressources.

2.2.2.1.2 La recherche tabou

La recherche tabou (TS : Tabu Search) est une métaheuristique d'optimisation proposé par Glover en 1990 [47]. Son idée de base consiste à introduire la notion de mémoire dans la politique d'exploration de voisinage. Cette idée est inspirée de la mémoire humaine.

La recherche tabou est une procédure itérative qui, partant d'une solution initiale (cette solution peut être construite par une heuristique ou générée aléatoirement) tente de converger vers la meilleur solution en exécutant à chaque itération un mouvement dans l'espace de recherche. Chaque itération consiste d'abord à engendrer un ensemble de solutions voisines de la solution courante pour ensuite en choisir la meilleure.

La recherche tabou utilise une mémoire afin de conserver pendant un moment les informations sur les solutions déjà visitées. Ces informations sont déclaré taboues et elles sont stockées dans une liste de longueur donnée, appelée la « liste des tabous » ou la « mémoire tabou » (qui a donné le nom à la méthode). Les informations données par la liste tabou sont utilisées pour établir une restriction appelée « restriction tabou », qui permet de classer certains mouvements comme étant interdits et permet ainsi d'éviter de retourner à des solutions déjà visitées dans un passé récent (phénomène de cyclage).

La procédure s'arrête lorsqu'une condition d'arrêt est satisfaite, généralement après un nombre fixé d'itérations ou encore après un nombre d'itérations sans amélioration de la solution. Pour certains problèmes d'optimisation, la recherche tabou donne de bons résultats. De plus, dans sa forme de base, la méthode contient moins de paramètres que le recuit simulé, ce qui la rend simple à utiliser. Pour plus de détails sur les différents paramètres de la recherche tabou, nous renvoyons le lecteur à l'article de Glover et al. [48].

L'algorithme 2 décrit le schéma général de la recherche tabou.

Algorithme 2 Pseudo code de la recherche tabou.

```
Début
1 :   s ← s_0                                   ▷ s_0 est la solution initiale
2 :   L = {}                                    ▷ la liste des tabous est vide initialement
3 :   Tant que la condition d'arrêt n'est pas vérifiée
4 :       Générer un ensemble N' ⊆ N(s) de solution voisines de s
5 :       Choisir la meilleur solution s' ∈ N' telle que s' ∉ L
6 :       Mettre à jour la liste L des solutions taboues
7 :       s ← s'
8 :   Fin Tant que
Fin
```

Les algorithmes de recherche tabou ont été appliqués pour résoudre les problèmes d'ordonnancement dans les environnements de production Flow-Shop, comme le travail de Taillard [123], celui de Nowicki et al. [99] et celui de Chen et al. [23] en 2008.

2.2.2.2 Métaheuristiques à base de population

Contrairement aux méthodes de recherche locale qui font intervenir une solution unique, les méthodes à base de population de solutions ou méthodes de recherche globale manipulent une panoplie de solutions réalisable à chacune des étapes du processus de recherche. L'idée cen-

trale consiste 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.

Ces méthodes se différencient par leur manière de représenter les problèmes à résoudre et par leur façon de faire évoluer la population d'une génération à l'autre. Une grande variété de métaheuristiques basées sur une population de solutions ont été proposées dans la littérature pour résoudre les problèmes d'optimisation, tel que : les algorithmes génétiques [58], les algorithmes de colonies de fourmis [34], Les algorithmes de recherche Cuckoo [144], les algorithmes d'optimisation par essaim de particules [71], les algorithmes de la colonie d'abeilles [70], etc.

Dans ce qui suit, nous présentons, un peu plus en détails les métaheuristiques utilisées dans le cadre de cette thèse, notamment les algorithmes génétiques et les algorithmes d'optimisation par essaim de particules.

2.2.2.2.1 Les algorithmes génétiques

Les algorithmes génétiques (AG) font partie de la famille des algorithmes évolutionnaires inspirés par l'évolution biologique des espèces. Ils ont été introduits par Holland [58] de l'université du Michigan en 1975 dans l'ouvrage "Adaptation of Natural and Artificial System". L'utilisation de ces algorithmes pour résoudre des problèmes d'optimisation a été développée initialement par Goldberg [57]. Suite à la publication de Goldberg, un intérêt scientifique croissant a été marqué pour cette technique d'optimisation.

Ces algorithmes génétiques se basent sur le concept d'une évolution biologique, dans laquelle la forme physique de l'individu détermine sa capacité de survivre et de reproduire. Leur principe est donc une transposition artificielle des concepts de la génétiques et des lois de survie énoncé par Charles Darwin.

Par analogie à la génétique, les algorithmes génétiques partent d'une population initiale d'individus, de taille bien déterminée, de solutions admissibles. Les solutions sont aussi appelées individus ou chromosomes. Chaque individu est codé par des chaînes de symboles et caractérisé par un fitness (une sorte de mesure d'adaptation au milieu). Chaque individu de la population est donc évalué afin de mesurer ces performances ou son adaptation. Par la suite, la structure de la population change en appliquant des évolutions progressives sur plusieurs générations. Pendant chaque génération, des nouveaux individus sont générés par les opérateurs de sélection, de croisement et de mutation. Dans la phase de la sélection, les parents les plus adaptés sont choisis, ensuite un opérateur de croisement est appliqué aux parents pour produire des enfants. L'opérateur de mutation est appliqué par la suite avec une certaine probabilité sur les nouveaux individus dont le but de modifié légèrement le code génétique de l'individu. Ainsi, seuls les individus les plus performants sont gardés pour former la population de la génération suivante. Le processus est répété jusqu'à ce que les conditions d'arrêts soient satisfaites.

Afin d'expliciter le fonctionnement des algorithmes génétiques, nous présentons dans la suite de cette section les différentes étapes d'un algorithme génétique standard qui sont illustrées par l'algorithme 3.

Algorithme 3 Structure générale d'un algorithme génétique

Début
Initialiser les paramètres de l'algorithme génétique.
Générer la population initiale.
Tant que critère d'arrêt non satisfait
  Évaluer les individus.
  Choisir les parents en se basant sur une stratégie de sélection.
  Appliquer l'opérateur de croisement avec un taux de croisement associé.
  Appliquer l'opérateur de mutation avec un taux de mutation associé.
  Remplacement de la population
Fin Tant que
Retourner le meilleur individu.
Fin

1. **Codage**

   Le codage est une fonction qui permet de passer de la donnée réelle du problème traité à la donnée utilisée par l'algorithme génétique. Le choix du codage est l'élément le plus important dans la conception de l'algorithme puisqu'il permet d'une part de représenter les données, les paramètres et les solutions et d'autre part, il influe sur la mise en œuvre des opérations génétique telles que le croisement et la mutation qui influent directement sur le bon déroulement de l'algorithme génétique et de son convergence vers la bonne solution. Plusieurs types de codage sont utilisés pour coder les individus, on distingue :

   — *Codage binaire* : ce codage a été le premier à être utilisé dans les algorithmes génétiques, où chaque gène peut prendre seulement les valeurs 0 ou 1.
   — *Codage réel* : contrairement au codage binaire, le codage réel associe à chaque gène une valeur réelle.
   — *Codage de permutation* : dans ce codage, les chromosomes contiennent typiquement une séquence de gènes où les gènes sont des nombres entiers ou des lettres, dans laquelle l'ordre est significatif. Ce type de codage est bien adapté aux problèmes d'ordonnancement.

2. **Population initiale**

   Une fois le codage choisi, une population initiale formée de solutions admissibles du problème doit être généré. Cette étape est très importante, car elle affecte la qualité de la solution d'une part et le nombre de générations au bout duquel on obtient de bonnes solutions d'autre part [119]. Plusieurs mécanismes de génération de la population initiale sont

utilisés dans la littérature. La méthode la plus classique consiste à générer aléatoirement les individus constituant la population initiale. Cette méthode répond à la nécessité d'avoir une population variée permettant d'explorer des zones diverses de l'espace de recherche. Cependant, la population peut être aussi générée à l'aide des méthodes heuristiques ou bien un pourcentage avec des méthodes heuristiques et le reste aléatoirement, et cela, dans le but d'accélérer la convergence de l'algorithme génétique, et en même temps de garder une diversité nécessaire de la population initiale.

La génération de la population initiale concerne aussi bien la taille de la population que la manière dont elle est générée. Nous revenons sur ce point dans chapitre 5.

3. **Évaluation**
   Une fois que la population est initialisée ou une nouvelle population est créée, une fonction d'évaluation est introduite afin de mesurer les performances de chaque individu de la population qui correspond à une solution donnée du problème à résoudre. Cette fonction correspond au profit ou à l'utilité de la solution par rapport au problème. Elle permet de quantifier la capacité d'un individu à survivre en lui affectant un poids couramment appelé « fitness ».
4. **Sélection**

   La sélection est un procédé qui permet d'identifier les individus répondant au mieux au problème étudié, c'est-à-dire, que chaque individus est choisi en fonction de sa valeur d'évaluation. Ce procédé ne permet pas de créer de nouveaux individus, mais de privilégier les individus en fonction de leur valeur d'évaluation, et cela, afin de se présenter au croisement et à la mutation (reproduction).

   Dans l'algorithme génétique, il existe de nombreuses techniques de sélection. Nous présentons ci-dessous les techniques les plus utilisées :

   — *Sélection par roulette* : pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème. Donc plus les individus sont adaptés au problème, plus ils ont de la chance d'être sélectionnés. Afin de sélectionner un individu, on utilise le principe de la roue de loterie biaisée. Cette roue est une roue de la fortune classique sur laquelle chaque individu est représenté par une portion proportionnelle à son adaptation, son "fitness". La roue étant lancée, l'individu sélectionné est celui sur lequel la roue s'est arrêtée. Cette méthode favorise les meilleurs individus, mais tous les individus conservent néanmoins des chances d'être sélectionnés.

   — *Sélection par rang* : cette sélection permet de classer la population suivant la fonction d'adaptation, chaque individu de la population se voit accorder un rang. Le plus mauvais individu est dans le rang 1 et le meilleur individu et dans le rang n (pour une population de n individus), c'est-à-dire que plus l'individu est bon, plus son rang est élevé. Le principe de la sélection par rang est similaire à la sélection par roulette, la différence est que la proportion est calculée sur les rangs et non sur la valeur de la fonction d'adaptation.

— *Sélection par tournoi* : cette méthode consiste à choisir aléatoirement deux ou plusieurs individus de la population, puis à sélectionner le meilleur individu dans ce groupe en fonction de son fitness. Ce processus est répété à chaque fois, avec ou sans remise, jusqu'à ce que le nombre d'individus nécessaires pour le croisement soit atteint (en d'autre terme, jusqu'à l'obtention de n individus, avec n la taille de la population).

5. **Croisement**

L'opération de croisement permet de simuler la reproduction d'individus pour créer de nouvelles solutions. Le croisement consiste donc à générer à partir de deux individus sélectionnés (parents), un ou deux nouvelles solutions fils composés chacune d'une partie des caractéristiques de leurs parents. Le croisement se fait généralement avec des opérateurs de croisement et le choix d'un opérateur dépend du codage adapté et les caractéristiques du problème traité. Parmi les opérateurs de croisement les plus connus dans la littérature, nous citons :

— *Croisement à un point* : cette opérateur consiste à diviser chacun des deux parents en deux parties à la même position p ($1 \leq p \leq l$) choisie aléatoirement avec l la longueur du chromosome et à recopier la partie inférieure du parent $[1, ..., p]$ à l'enfant et à compléter les gènes manquants de l'enfant à partir de l'autre parent en maintenant l'ordre des gènes.

— *Croisement à deux points* : ce type de croisement est utilisé en choisissant aléatoirement deux points de coupure pour dissocier chaque parent en 3 fragments. Les deux fragments en extrémités pour le parent 1 (respectivement parent 2) sont copiés à l'enfant 1 (respectivement enfant 2). La partie restante de l'enfant 1 est complété par les éléments du parent 2 et la partie restante de l'enfant 2 est complété par les éléments du parent 1 en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis.

Dans le but de garder quelques individus parents dans la prochaine population, une probabilité de croisement notée P_c est associe à l'algorithme génétique qui permet de décider si les parents seront croisés entre eux ou s'ils seront tout simplement recopiés dans la population suivante.

6. **Mutation**

La mutation permet de maintenir la diversité qui empêche la dégénérescence des solutions entre elles dans la population. L'opérateur de mutation apporte donc aux algorithmes génétiques la propriété d'ergodicité de parcours d'espace de solutions. Cette propriété indique que l'algorithme génétique sera susceptible d'atteindre tout les points de l'espace de recherche, sans pour autant les parcourir tous dans le processus de résolution [36]. L'opérateur de mutation joue le rôle d'un élément perturbateur qui consiste en un changement au niveau des gènes d'un individu choisi avec une certaine probabilité p_m appelée la probabilité ou le taux de mutation. Selon la littérature, le taux de mutation est généralement faible

puisqu'un taux élevé risque de conduire à une marche aléatoire dans l'espace de recherche.

Comme pour le croisement, de nombreuses méthodes de mutation ont été développés dans la littérature, nous citons les plus connues :

- *Mutation par échange* : cet opérateur de mutation permet de sélectionner deux gènes et les inter-changer.
- *Mutation par insertion* : cet opérateur consiste à sélectionner au hasard un gène et une position dans le chromosome à muter, puis à insérer le gène en question dans la position choisie.
- *Mutation par inversion* : cet opérateur consiste à choisir aléatoirement deux points de coupure et inverser les positions des gènes situés au milieu.

7. **Remplacement**

   L'opérateur de remplacement consiste à choisir les individus qui vont constituer la génération suivante. Deux principaux types sont distingués dans la littérature :

   - *Remplacement stationnaire* : ce type de remplacement est le plus simple dans lequel la population des enfants remplace automatiquement la population des parents sans prendre en considération leurs performances respectives. Dans ce type de remplacement, deux façons différentes sont mises en œuvre :
     - la première se contente de remplacer l'intégralité de la population des parents par la population des enfants, cette méthode est connue sous le nom de remplacement générationnel.
     - la deuxième consiste à choisir une certaine proportion d'individus d'enfants qui remplaceront leurs parents (proportion égale à 100 % dans le cas du remplacement générationnel).
   - *Remplacement élitiste* : le facteur performance est pris en compte dans cette stratégie qui consiste a garder au moins l'individu le plus performant lors du passage d'une génération à la suivante.

8. **Critère d'arrêt**

   Le critère d'arrêt détermine quand le processus génétique arrête d'évoluer. Ce critère peut être de nature diverse, par exemple :

   - Un nombre maximum de générations fixées au départ est atteint.
   - Un certain temps de calcul à ne pas dépasser.
   - Un état de stagnation de la population, c'est-à-dire que l'algorithme arrête s'il n'y a pas un changement de la fonction fitness d'une population pour un nombre de générations spécifier.

Cette métaheuristique a été largement utilisé pour résoudre les problèmes d'ordonnancement, et plus particulièrement les problèmes d'atelier Flow-Shop. Nous citons à titre d'exemple les travaux

de [22, 59, 94, 110, 112, 130, 145]. Ruiz et al. [116] ont proposé deux algorithmes génétiques pour un problème d'ordonnancement Flow-Shop de permutation pour minimiser le makespan, ces algorithmes utilisent de nouveaux opérateurs génétiques ainsi qu'une initialisation efficace de la population. Dans [130], les auteurs ont adapté un algorithme génétique afin de minimiser le temps total d'exécution pour un problème d'ordonnancement Flow-Shop avec contrainte de blocage.

2.2.2.2.2 Algorithme d'optimisation par essaim particulaire

L'optimisation par essaim particulaire (OEP) ou Particle Swarm Optimization (PSO) est une métaheuristique à population, mise au point par le sociologue James Kennedy et l'ingénieur électrique Russell C. Eberhart en 1995 [38]. Cette métaheuristique est issue d'une analogie avec les déplacements collectifs de certains animaux évoluant en essaim, tels que les bancs de poissons et les oiseaux migrateurs.

L'optimisation par essaim particulaire présente quelques similarités avec les algorithmes génétiques dans le sens où elle fonctionne sur la base d'une population de solutions qui interagissent entre elles afin de trouver la meilleure solution possible à un problème d'optimisation. Cependant, contrairement aux algorithmes génétiques où l'évolution est basée sur la compétition et l'élimination des individus les moins performants (sélection), l'optimisation par essaim particulaire se base sur la coopération entre les individus afin de faire évoluer chacun [78].

L'essaim de particule correspond à une population d'individus appelés particules où chaque particule représente une solution potentielle du problème à traité. Le concept principal de cette méthode est que les particules évaluent de manière itérative l'aptitude des solutions candidates et se souviennent de l'endroit où elles ont eu leur meilleure valeur de fitness.

Chaque particule dont l'espace de recherche est doté d'une position et une vitesse. La position représente ces cordonnées dans l'ensemble de définitions et la vitesse correspond à un vecteur de déplacement, de modification de ça position actuelle, définissant sa future position probable. En outre, chaque particule dispose d'une mémoire concernant sa meilleure solution visitée ainsi que la capacité de communiquer avec les particules de son entourage.

À chaque itération, les particules se déplacent en tenant compte de leur meilleure position, mais aussi de la meilleure position de ses voisins. L'objectif est de modifier leur trajectoire pour qu'elles se rapprochent le plus possible de l'optimum. Le déplacement de chaque particule lors d'une itération est une combinaison de trois comportements sociaux (cf. figure 2.4) :

- Comportement inertiel, dont lequel la particule tend à suivre le déplacement induit par sa vitesse.
- Comportement cognitif, dont lequel la particule tend à se diriger vers sa meilleure position visitée.
- Comportement social, dont lequel la particule tend à se diriger vers la meilleure position

atteint par ses voisines.

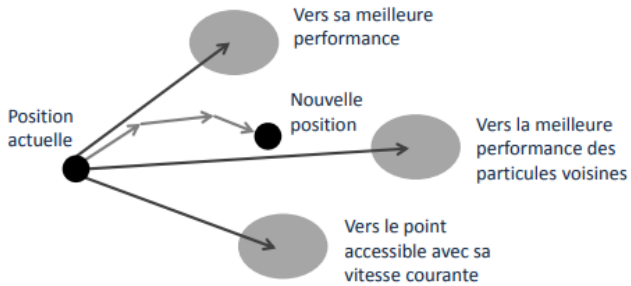


FIGURE 2.4 : Déplacement d'une particule [80]

Plus formellement, dans un espace de recherche de dimension d, une particule i de l'essaim est modélisée par son vecteur position $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ et son vecteur vitesse $v_i = (v_{i1}, v_{i2}, ..., v_{id})$. Désignons par $p_i = (p_{i1}, p_{i2}, ..., p_{id})$ la meilleur position atteinte par la particule i et $p_g = (p_{g1}, p_{g2}, ..., p_{gn})$ la meilleure position atteinte par toutes les particules de l'essaim. Étant donné un état de l'essaim à un temps t, au temps $t+1$ la vitesse et la position de la i^{ieme} particule sont recalculées selon les équations 2.1 et 2.2 respectivement :

$$v_i^{t+1} = w\, v_i^t + c_1 r_1 (P_i^t - x_i^t) + c_2 r_2 (p_g^t - x_i^t) \tag{2.1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.2}$$

où $i = 1, 2, ..., n$ avec n le nombre de particules (taille de l'essaim), w est une constante appelée coefficient d'inertie, $c1$ et $c2$ sont deux constantes appelées coefficients d'accélération, $r1$ et $r2$ sont deux nombres aléatoires tirés uniformément dans [0, 1].

L'algorithme 4 d'écrit le déroulement global de l'algorithme d'optimisation par essaim particulaire où le critère d'arrêt peut être défini comme étant un nombre maximum d'itérations, comme il peut correspond au temps de calcul à ne pas dépasser, etc.

L'algorithme d'optimisation par essaim particulaire a été conçue originellement pour résoudre les problèmes d'optimisation continue, mais qu'il a été transformé par divers auteurs afin de l'adapter à la résolution des problèmes d'optimisation combinatoire, dont l'ordonnancement.

Différentes approches discrètes ont été proposées dans la littérature pour l'OEP. Ces approches diffèrent de l'approche continue par la manière dont une particule est associée à une solution discrète, ainsi que par le modèle de vitesse utilisé. Selon la littérature [3], il existe trois manières d'associer une particule à une solution discrète : représentation binaire, la représentation par valeurs réelles et la représentation par permutation. De même, trois modèles de vitesse ont été utilisés, à savoir le modèle de valeurs réelles, le modèle stochastique et le modèle basé sur une liste de déplacements.

Dans ce contexte, Liao et al. [85] ont développé un algorithme d'OEP discret pour résoudre un

Algorithme 4 Structure générale d'algorithme d'optimisation par essaim particulaire

Début
Initialiser les paramètres de l'algorithme et la taille de l'essaim.
Initialiser les vitesses et les positions des particules.
Pour chaque particule i, $p_i = x_i$.
Évaluer les positions des particules $f(x_i)$.
Calculer p_g. ▷ meilleur de p_i
Tant que critère d'arrêt non satisfait
  for i allant de 1 à n **do**
    Calculer la nouvelle vitesse à l'aide de l'équation 2.1.
    Trouver la nouvelle position à l'aide de l'équation 2.2.
    Calculer $f(x_i)$ de chaque particule.
    if $f(x_i)$ est meilleur que $f(p_g)$ **then**
      $p_i = x_i$.
    end if
    if $f(p_i)$ est meilleur que $f(p_g)$ **then**
      $p_g = p_i$.
    end if
  end for
Fin Tant que
Retourner la meilleure solution trouvée p_g.
Fin

problème d'ordonnancement Flow-Shop, l'algorithme développé est caractérisé par une représentation binaire et un modèle de vitesse stochastique. Les auteurs ont montré que l'algorithme proposé est très compétitif, on comparant avec deux algorithmes génétiques. Pan et al. [104] ont présenté un algorithme d'OEP discret basé sur une représentation par permutation et un algorithme de recherche locale pour résoudre un problème d'ordonnancement Flow-Shop sous contrainte de "sans-attente" ou ("no-wait constraint" en Anglais) afin de minimiser le temps d'exécution maximal (makespan).

De même, Tasgetiren et al. [126] et Jarboui et al. [63] ont exposé un algorithme d'OEP discret pour un problème d'ordonnancement Flow-Shop de permutation avec la minimisation du makespan et la minimisation du flow time. Dans [63], les auteurs ont utilisé le modèle par permutation pour représenter une particule, ainsi une phase d'amélioration basée sur une procédure de recuit simulé a été intégrée.

Lian et al. [84] ont utilisé le modèle par permutation pour résoudre le problème de minimisation du makespan dans un atelier de type Flow-Shop. Les auteurs conçoivent une technique particulière pour définir la vitesse de chaque particule, cette technique consiste à emprunter les concepts de croisement et de mutation aux algorithmes génétiques.

2.3 Méthodes hybrides

Les différentes méthodes vues précédemment possèdent bien entendu leurs propres avantages et inconvénients, en termes de qualité de solutions fournies, de la complexité temporelle et la simplicité d'implémentation. Une tendance actuelle en optimisation combinatoire consiste à co-opérer/hybrider plusieurs méthodes parmi celles que nous venons de citer. La motivation derrière une telle hybridation est d'avoir des méthodes plus sophistiquées et plus efficaces.

L'hybridation est une technique qui essaie de tirer profit des points forts de chaque méthode de résolution utilisée pour améliorer le comportement global de la méthode hybride. Donc, une méthode hybride est une méthode de recherche constituée d'au moins deux procédures de recherche distinctes, y compris les méthodes exactes et/ou approchées (sus-décrites). Elle consiste à exploiter les avantages respectifs de ces méthodes tout en combinant leurs algorithmes suivant une approche synergétique. Nous citons à titre d'exemple le cas des algorithmes mimétiques [93] ou encore connus sous le nom d'algorithmes génétiques hybrides qui combinent une recherche locale qui assure l'intensification de la recherche et un algorithme génétique qui renforce la diversification.

À ce sujet, Duvivier [37] a proposé une classification des techniques d'hybridation en s'appuyant sur les travaux du groupe PERFORM [1] [108]. Les principales formes hybridation proposées sont classifiées en trois catégories, selon leurs architectures :

— Hybridation séquentielle : consiste à exécuté séquentiellement différentes méthodes de recherche de telle manière que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante comme le montre la figure 2.5. Cette technique d'hybridation est la plus simple et la plus populaire.

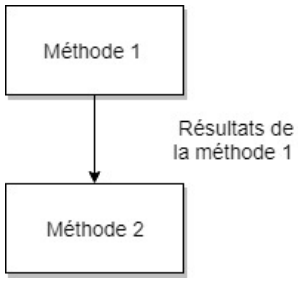


FIGURE 2.5 : Hybridation séquentielle

— Hybridation parallèle synchrone : consiste à incorporer une méthode de recherche particulière dans un opérateur d'une autre. C'est en quelque sort une hybridation plus fine que la précédente étant donné que la méthode est englobé dans une autre comme le montre la figure 2.6 . Cette technique est plus complexe à mettre en œuvre que la précédente vu qu'il faut tenir compte des fortes interactions entre les méthodes incorporées pour ne pas

1. PERFORM(Parallel gEnetic algoRithms FOR optiMization) est un groupe de travail qui rassemble des chercheurs de LIFL (Laboratoire d'Informatique Fondamentale de Lile) et LIL (Laboratoire d'informatique de Littoral)

aboutir à une méthode hybride moins efficace [37]. Un exemple de ce type d'hybridation est de remplacer l'opérateur de mutation d'un algorithme génétique par une recherche taboue.

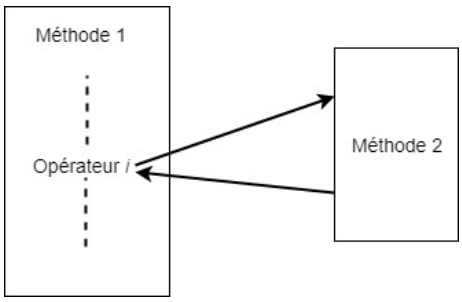


FIGURE 2.6 : Hybridation parallèle synchrone

— Hybridation parallèle asynchrone : consiste à faire évoluer en parallèle différentes méthodes de recherche. Cette coévolution permet une bonne coopération des méthodes de recherche au travers d'un coordinateur qui est chargé d'assurer les échanges d'informations entre les méthodes de recherche constituant l'hybride (cf. Figure 2.7).

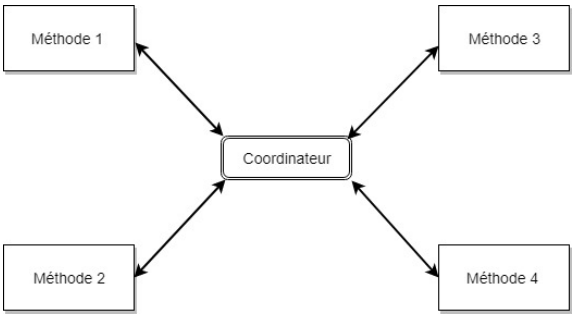


FIGURE 2.7 : Hybridation parallèle asynchrone

Nous nous limitons dans le cadre de cette thèse à cette classification. Cependant, il existe bien d'autre classification des stratégies d'hybridation, nous citons celle de Talbi [125] qui a proposé une taxonomie de métaheuristiques hybrides, cette taxonomie comporte deux aspects principaux : une classification hiérarchique permettant d'identifier la structure générale de l'hybridation et une classification générale spécifiant les détails des approches impliqués dans l'hybridation. L'auteur a également introduit une grammaire qui permet de décrire précisément les différents schémas d'hybridation. Une autre classification axée sur l'hybridation de méthodes exactes et approximatives a été proposée par Puchinger et al. [109]. Cette classification a été divisée en deux principales catégories : une hybridation collaborative et intégrative. Dans l'hybridation collaborative, les algorithmes échangent des informations de façon séquentielle, parallèle ou entrelacée. Cependant, l'hybridation intégrative se fait de manière à ce qu'un des deux algorithmes soit un composant intégré à l'autre algorithme.

Des méthodes hybrides ont été développées dans la littérature pour minimiser le makespan dans

un atelier de type Flow-Shop, à titre d'exemple, Zobolas et al. [145] ont combiné un algorithme génétique avec une recherche à voisinage variable (VNS : Variable Neighborhood Search) pour améliorer la solution. De même, Jarboui et al. [62] ont proposé un algorithme génétique hybride pour la résolution de problème Flow-Shop avec contrainte sans délai (no-wait). Dans [63], les auteurs ont adapté un algorithme d'optimisation par essaim de particules dont laquel ils ont incorporé une procédure d'amélioration basée sur l'approche du recuit simulé.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les techniques utilisées pour résoudre les problèmes d'optimisation, et en particulier, les problèmes d'ordonnancement. Pour les deux classes de méthodes : exactes et approximatives, les techniques les plus représentatives sont décrites. Pour chaque méthode, nous avons présenté l'idée générale. Seules les méthodes que nous avons utilisées dans notre étude sont plus détaillées. Pour la plupart des méthodes décrites, certaines références sont données pour les problèmes d'ordonnancement Flow-Shop avec ou sans contraintes.

Chapitre 3

Revue de la littérature

Le présent chapitre contient une description de la plupart des importants travaux de recherches effectués sur le problème d'ordonnancement étudié dans cette thèse.

3.1 État de l'art sur les problèmes d'ordonnancement sous contraintes de ressources non-renouvelables

Une quantité considérable de recherches sur les problèmes d'ordonnancement de la production a été menée au cours des dernières décennies. Des introductions générales sur l'ordonnancement sont aussi fournies dans les ouvrages de Blazewicz et al. [13] Brucker [16] et Pinedo [106]. Cependant, dans la majorité des travaux, peu d'attention a été portée à la contrainte de ressources non-renouvelables qui fait l'objet de cette thèse. La présence de cette contrainte dans les problèmes d'ordonnancement signifie l'existence de ressources additionnelles dans le système de production. Ces ressources sont de nature consommable ou non-renouvelable, ces derniers sont sollicités pour que les tâches soient exécutées sur les machines. Une tâche est donc considéré comme prêt pour être exécuté quand la quantité de ressource consommable requise est disponible. Chaque ressource consommable dispose d'un stock initial et de quelques fournitures supplémentaires à des moments connus à priori et en quantités connues.

Les problèmes sous contraints de ressources non-renouvelables ont une grande importance pratique. Des problèmes comme celui-ci se posent par exemple dans les environnements de production (automobile, textile), Ordonnancent de projet (les entreprises de construction ne peuvent pas commencer une partie de leur projet si elles ne reçoivent pas l'argent nécessaire pour cette partie.), etc.

Les problèmes d'ordonnancement sous contraintes de ressources non-renouvelables ont attiré une communauté de chercheurs en raison de leur intérêt pratique considérable. Dans ce qui suit, nous allons présenter dans un ordre chronologique un aperçu des travaux traitant la contrainte de ressources non-renouvelables. Nous limitons notre recherche bibliographique, aux problèmes

statiques et déterministes d'ordonnancement d'atelier.

Les problèmes d'ordonnancement sous contraintes de ressources non-renouvelables ont été introduits par Carlier [18] et Slowinski [121] au début des années 1980. En particulier, Carlier [18] a appliqué une modification légère pour introduire cette contrainte dans la classification de Graham [51]. En plus, il a étudié la complexité des problèmes d'ordonnancement à contrainte de ressource consommable pour des différentes fonctions économiques. La notation adaptée par Carlier pour les problèmes d'ordonnancement à une machine et à plusieurs machines identiques sous contrainte de ressource non-renouvelable est la suivante : $A|B|C|D$ avec $A = n$ est le nombre de tâches ; $B = m$ est le nombre de machines disponibles ; C est l'ensemble des paramètres associés aux tâches et D est la fonction économique.

Le champs C peut avoir six composants $C(C1, C2, C3, C4, C5, C6)$ qui sont les suivants :

- $C1$ est associée aux durées des tâches, $C1 \in \{p_i = 1, p_i = P, \epsilon\}$, ϵ si la durée est quelconque.
- $C2$ est associées aux dates de disponibilités des tâches, $C2 = \epsilon$ si elles sont tout égales ; sinon $C2 = r_i$.
- $C3$ est associée aux dates échues des tâches, $C3 = \epsilon$ s'il n'y a pas de dates échues ; $C3 = d_i$ si elles sont quelconques ; $C3 = d$ si elles sont toutes égaux.
- $C4 =<$ si les tâches sont dépendantes, sinon $C4 = \epsilon$.
- $C5 = RC$ qui indique l'existence d'une seule Ressource Consommable (RC).
- $C6$ indique la quantité de ressource a_i consommé par la tâche i, $C6 \in \{a_i = 1, \epsilon\}$, $C6 = \epsilon$ si la quantité consommée est quelconque.

Dans les tableaux 3.1 et 3.2, nous résumons une partie des différents résultats de complexité trouvés par Carlier [18](cf. chapitre VIII de ça thèse) :

TABLE 3.1 : Complexité de minimisation de la durée total

	Taches dépendantes ($P_i = 1$, $r_i = 0$)	Taches indépendantes (P_i quelconques, $r_i = 0$)	Taches indépendantes (P_i = P ou 1, r_i quelconques)
$m = 1$	$n\|1\|p_i = 1, <, RC\|C_{max}$ NP-difficile	$n\|1\|RC\|C_{max}$ NP-difficile	polynomial
	$n\|1\|p_i = 1, <, RC, a_i = 1\|C_{max}$ Polynomial	$n\|1\|RC, a_i = 1\|C_{max}$ Polynomial	
m fixe	$n\|m\|p_i = 1, <, RC\|C_{max}$ NP-difficile	$n\|m\|RC\|C_{max}$ NP-difficile	polynomial
	$n\|m\|p_i = 1, <, RC, a_i = 1\|C_{max}$ Ouvert	$n\|m\|RC, a_i = 1\|C_{max}$ NP-difficile	

Dans sa thèse, Carlier a étudié également les problèmes où les dates échues sont considérées. Il a montré que ces problèmes sont NP-difficile, le seul cas polynomial est lorsque les dates de disponibilité sont tout égales, les durées égales à 1 et les tâches indépendantes.

Slowinski [121] a étudié le problème d'ordonnancement où la préemption des tâches est autorisée sur machines parallèles avec l'utilisation de ressources renouvelables supplémentaires (main-

TABLE 3.2 : Complexité de minimisation de la durée moyenne

	Taches dépendantes ($P_i = 1$, r_i=0)	Taches indépendantes (P_i quelconques, r_i=0)	Taches indépendantes (P_i = 1, r_i quelconques)
m=1	$n\|1\|p_i = 1, <, RC\|\sum C_i$ NP-difficle $n\|1\|p_i = 1, <, RC, a_i = 1\|\sum C_i$ Polynomial	$n\|1\|RC\|\sum C_i$ NP-difficile	$n\|1\|p_i = 1, r_i, RC\|\sum C_i$ NP-difficile
m fixe	$n\|m\|p_i = 1, <, RC\|\sum C_i$ NP-difficile $n\|m\|p_i = 1, <, RC, a_i = 1\|\sum C_i$ NP-difficile	$n\|m\|RC\|\sum C_i$ NP-difficile	$n\|m\|p_i = 1, r_i, RC\|\sum C_i$ NP-difficile

d'œuvre) et la consommation d'une ressource non-renouvelable (argent) qui a un stock initial et qui devient disponible en quantités spécifiques à des dates différentes. L'auteur a supposé que le taux de consommation de la ressource non-renouvelable est constant, cette hypothèse lui a conduit à un algorithme polynomial pour minimiser le makespan. Des résultats supplémentaires ont été trouvés dans Cochand et al. [25] où les auteurs ont considéré que l'approvisionnement de la ressource non-renouvelable varie avec le temps (en escalier ou linéaire par morceaux).

Dans [31], les auteurs ont étudié la version préemptive du problème d'atelier open shop à m machines avec ressources renouvelables et non-renouvelables pour minimiser le makespan, ils ont modélisé le problème par une approche basé sur la coloration de graphe et ils ont également présenté un algorithme polynomial pour le cas où la courbe de la disponibilité de ressource est de type escalier, et cela, afin d'étendre les résultats présentés dans [30]. En particulier, de Werra et al. [30] ont démontré que le problème $O|pmtn|Cmax$ avec une ressource renouvelable ou consommable est NP-difficile au sens fort.

Toker et al. [128] ont étudié la minimisation du makespan dans un problème d'ordonnancement à une machine avec une seule ressource non-renouvelable. Ils ont montré que, si la quantité de ressource non-renouvelable disponible au cours de chaque période est constante (approvisionnement à taux constant) alors le problème devient équivalent à un Flow-Shop à deux machines sans contraintes de ressources, qui peut donc être résolu polynômialement à l'aide de l'algorithme de Johnson [65].

Le schéma de classification utilisé dans [128] est une version modifiée de celle de Rinnooy Kan [69] dont le format est le suivant : $(\alpha|\beta|\gamma, \Gamma|\delta)$, où

- α = nombre de tâches.
- β = nombre de machines.
- γ = environnement machines (Flow-Shop, Job-Shop, Open-Shop)
- Γ = toute contrainte imposée au problème. $\Gamma = NR : \alpha_t$ indique l'existence d'une ressource Non-Renouvelable (NR) qui devient disponible avec une quantité α_t en période t.
- δ = Critère de performance.

Le résultat de Toker et al. [128] est résumé dans le théorème suivant :

Théorème 3.1

Le problème $(n|1|NR : \alpha_t = 1|C_{max})$ est équivalent au problème $(n|2|F|C_{max})$

Toker et al. [129] ont étendu l'hypothèse présentée dans [128] au problème d'ordonnancement d'atelier Job-Shop sous une contrainte de ressource non-renouvelable. L'hypothèse considérée a conduit à un algorithme d'approximation permettant de minimiser le makespan.

Xie [143] a généralisé le résultat présenté dans [128] au problème d'ordonnancement à une machine avec plusieurs ressources non-renouvelables de type financière. Il a démontré que le problème peut être réduit à un problème d'ordonnancement d'atelier Flow-Shop à deux machines si les ressources consommables arrivent de manière uniforme dans le temps. Il a également montré que la règle LPT génère une solution optimale au problème si les ressources non-renouvelables sont consommées de manière uniforme par tous les tâches. Ces résultats sont résumés dans les théorèmes 3.2 et 3.3 où FC_k indique l'existence de k différents types de ressources financières (FC : Financial Constraints) et r_{jh} représente la quantité de ressource h que le job j consomme.

Théorème 3.2

Le problème $1|FC_k : \alpha_{ht} = 1|C_{max}$ peut être réduit au problème $(n|2|F|C_{max})$

Théorème 3.3

La règle LPT génère une solution optimale au problème $1|FC_k : r_{jh} = r_h|C_{max}$.

Grigoriev et al. [52] ont également étudié la complexité des problèmes d'ordonnancement sur une machine en présence d'une ou plusieurs ressources non-renouvelables de nature matières premières. Une attention particulière a été portée aux problèmes de minimisation du retard et de minimisation du makespan avec des temps d'exécution unitaires ou égaux.

Les auteurs ont étendu aussi la notation de Graham et al. [51] avec les restrictions rm (Raw Materials) : $rm = m$ qui signifie qu'il existe m ressources (matières premières).

Les résultats obtenus sont résumés dans les théorèmes suivants :

Théorème 3.4

Le problème $1|rm = 1, p_j = 1|L_{max}$ peut être résolu en temps polynomial.

Théorème 3.5

Le problème $1|rm = 2, p_j = 1|L_{max}$ est NP-difficile au sens fort.

TABLE 3.3 : Un aperçu sur les résultats de complexité [44].

Fonction objective	Cas particuliers	Complexité
C_{max}	/	NP-difficile au sens fort
$\sum U_j$	/	NP-difficile au sens fort
L_{max}	/	NP-difficile au sens fort
$\sum C_j$	/	NP-difficile au sens fort
$\sum T_j$	$d_j = d$	NP-difficile au sens fort
$\sum T_j$	$g_j = g$	NP-difficile au sens fort
$\sum T_j$	$p_j = p$	NP-difficile au sens fort
$\sum T_j$	$d_j = d, g_j = g$	NP-difficile au sens fort
$\sum T_j$	$p_j = p, g_1 \leq g_2 \leq .... \leq g_n, d_1 \leq d_2 \leq .... \leq d_n$	Polynomiale

Théorème 3.6

Le problème $1|rm = 2, p_j = 1|L_{max}$ est NP-difficile au sens fort.

Pour le Théorème 3.4, un algorithme basé sur la règle EDD est utilisé pour résoudre le problème. De plus, Grigoriev et al. [52] ont étudié un cas particulier où chaque tâche nécessite son propre type de matière première. Les auteurs ont montré que ce problème est équivalent à celui où l'on impose des dates de disponibilités aux tâches. Formellement, $1|ddc|$. est équivalent à $1|r_j|$. où le script ddc correspond à différentes matières premières dédiées pour chaque tâche. Par conséquent, il est bien connu que le problème $1|r_j|L_{max}$ est NP-difficile [79] ce qui rend le problème $1|ddc|L_{max}$ NP-difficile aussi. À l'exception des problèmes $1|r_j, prec, p_j = p|L_{max}$ et $1|r_j, prec, pmtn|L_{max}$ où des algorithmes polynomiaux sont connus [6].

De la même manière, $1|ddc|C_{max}$ est équivalent à $1|r_j|C_{max}$ et il est clair que l'ordonnancement des tâches dans l'ordre croissant des r_j fournit une solution optimale pour le problème $1|r_j|C_{max}$ et par conséquent, $1|ddc|C_{max}$ est un problème polynomial.

Des algorithmes d'approximation ont également été proposés par Grigoriev et al. [52].

Gafarov et al. [44] ont complété les travaux de Grigoriev et al. [52] par des résultats de complexité supplémentaires, dans lesquels une attention particulière a été accordée au problème de la minimisation du retard total. Le tableau 3.3 résume les résultats de complexité obtenus dans leur papier.

Dans sa thèse, Carrera [21] aborde un problème d'optimisation de la planification et d'ordonnancement des plateformes logistiques. Les problèmes d'ordonnancement pour la plateforme qui exécute des activités de préparation de commande sont étudiés dans le chapitre trois de la thèse. L'auteur se limite à des problèmes d'ordonnancement sur une machine avec des contraintes de ressources consommables et avec des dates de livraison fixes. Pour résoudre ce problème, l'auteur a proposé un modèle linéaire en nombres entiers et une procédure par séparation et évaluation.

Dans György et Kis [53], un schéma d'approximation polynomial PTAS (*Polynomial-Time Ap-*

proximation Scheme) pour l'ordonnancement d'une seule machine avec une ressource non-renouvelable et un nombre constant de dates d'approvisionnement a été développé. Ainsi qu'un schéma d'approximation entièrement polynomial FPTAS (*Fully Polynomial-Time Approximation Scheme*) a été conçu pour le cas particulier avec deux dates d'approvisionnement et une seul ressource non-renouvelable.

Belkaid et al. [9] ont présenté un modèle mathématique linéaire en nombres entiers, ainsi qu'une métaheuristique à base d'un algorithme génétique pour résoudre un problème de machines parallèles avec contraintes de ressources consommables afin de minimiser le makespan. De plus, certaines heuristiques basées sur des règles de priorité ont été également proposées.

Les résultats présentés dans [53] ont été étendus par György et Kis [54] pour les problèmes d'ordonnancement sur machines parallèles avec ressources consommables. Des PTASs ont été fournies sous diverses hypothèses où l'objectif d'optimisation est soit le makespan, soit le retard maximal.

3.2 Identification de problème et objectif de la thèse

Afin d'identifier notre problématique, nous résumons dans le tableau 3.4 les travaux présentés dans la section précédente par ordre chronologique en se concentrant sur trois catégories principales : l'environnement machine, fonction objectif et la méthodologie. Comme le montre le tableau 3.4 et à notre connaissance, le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables n'a pas été traité dans la littérature auparavant. Cela nous a donné donc la motivation d'étendre et d'accomplir les travaux précédents tout en intégrant les contraintes de ressources non-renouvelables dans les environnements de production Flow-Shop.

Dans cette thèse, nous nous intéressons plus particulièrement aux problèmes de minimisation de makespan dans un atelier de type Fow-Shop avec contraintes de ressources non-renouvelables. Le makespan est l'un des critères les plus importants dans tous les systèmes de production. La minimisation de ce dernier permet d'équilibrer la charge entre les machines et offre généralement un taux élevé de leurs utilisations ce qui revient à maximiser la productivité. Tandis que le Flow-Shop représente près du quart des systèmes de production et qu'il est aussi l'un des problèmes le plus traité de la théorie de l'ordonnancement ; par contre sa complexité fait de lui l'un des problèmes de l'optimisation combinatoire les plus difficiles à résoudre. Nous referons le lecteur aux articles de [43] et [41] pour un état de l'art sur les problèmes Flow-Shop avec minimisation de makespan.

Comme on peut constater aussi du tableau 3.4 que la majorité des auteurs se concentrent sur les résultats de complexité et les algorithmes polynomiaux pour des cas particuliers des problèmes d'ordonnancement avec contraintes de ressources non-renouvelables. Peu de travaux ont utilisé les méthodes de résolutions exactes ou approchées. Le but de cette thèse est donc de développer des méthodes de résolution efficaces (exactes et approchée) pour les problèmes d'or-

donnancement rencontrés dans les systèmes de production, en particulier les ateliers de type Flow-Shop et on intégrant des contraintes pratiques telles que la disponibilité des ressources non-renouvelables. Notons que l'étude de cette dernière n'est que récente malgré sa pertinence au niveau industriel. En effet, cette contrainte est souvent négligée par les chercheurs. Ceci peut être du fait que son intégration rend le problème d'ordonnancement nettement plus difficile à résoudre que sa forme classique.

TABLE 3.4 : Une taxonomie des problèmes examinés

Référence	Environnement					Objectif		Méthode
	SM [a]	PM [b]	JS [c]	FS [d]	OS [e]	Makespan	Autres	
Carlier [18]	×					×		Algorithmes polynomiaux
Slowinski [121]		×				×		Algorithme polynomial
Cochand et al. [25]		×				×		Algorithme polynomial
Toker et al. [128]	×					×		Algorithme polynomial
de Werra et al. [31]					×		×	Algorithmes polynomiaux
Toker et al. [129]			×			×		Algorithme d'approximation
Grigoriev et al. [52]	×					×	×	Algorithmes d'approximation
Carrera [21]	×					×		PLNE + B&B
György et al. [53]	×					×		PTAS, FPTAS
Belkaid et al. [9]		×				×		PLNE + AG
Györgyi et al. [54]		×				×	×	PTAS

[a] Single Machine [b] Parallel Machine [c] Job-Shop [d] Flow-Shop [e] Open-Shop

3.3 Conclusion

Ce chapitre montre l'engouement de la littérature pour l'étude des problèmes d'ordonnancement sous contraintes de ressources non-renouvelables. Dans ce chapitre, nous avons noté un manque d'étude concernant la résolution des problèmes Flow-Shop sous contraintes de ressources non-renouvelables. C'est pourquoi nous proposons de résoudre ce problème dans cette thèse.

Deuxième partie

Le Flow-Shop sous contraintes de resources non-renouvelables : Modélisation mathématique et méthodes de résolution

Chapitre 4

Méthode de résolution exacte

Après avoir identifié le problème dans le chapitre précédent, nous présentons dans ce chapitre une approche de modélisation mathématique pour résoudre le problème d'ordonnancement d'atelier Flow-Shop avec contraintes de ressources non-renouvelables. Cette approche est souvent négligée par les chercheurs en raison de la forte complexité du problème. Dans cette étude, nous avons choisi de développer cette approche, car elle permet de modéliser formellement le problème et de résoudre de façon optimale certaines tailles d'instances, elle permet ainsi d'évaluer la qualité des solutions fournies par les méthodes approchées que nous verrons dans les chapitres suivants.

Le chapitre est organisé comme suit : dans la section 4.1, nous décrivons notre problème et nous donnons les notations utilisées ainsi que la complexité. Ensuite, dans la section 4.2 nous développons un modèle mathématique linéaire en nombres entiers pour représenter et résoudre le problème en optimisant le makespan. Dans la section 4.3, nous présentons le benchmark qui constitue l'échantillon de test pour notre travail. Finalement, dans la section 4.4, la performance de la méthode exacte proposée pour le problème est évaluée à travers des résultats expérimentaux.

4.1 Le Flow-Shop sous contraintes de ressources non-renouvelables

4.1.1 Description du problème

Dans cette section, nous fournissons une définition formelle du problème d'ordonnancement d'atelier Flow-Shop sous contraintes de ressources non-renouvelables. Étant donné un ensemble $J = \{J_1, J_2, ..., J_n\}$ de n jobs et un ensemble $M = \{M_1, M_2, ..., M_m\}$ de m machines. Chaque job J_i, $i \in \{1, ..., n\}$ est constitué de m opérations O_{ij}, où O_{ij} doit être traité sur la machine M_j, $j \in \{1, ..., m\}$ pour p_{ij} unités de temps. Le temps opératoire p_{ij} est un nombre positif fixe.

L'ordonnancement est réalisable si :

(a) Les opérations de chaque job sont traitées dans un ordre déterminé (M_1 à M_m).

(b) Les opérations sont traitées sans interruption (c'est-à-dire que l'opération O_{ij} s'exécute durant p_{ij} unités de temps consécutives sur la machine M_j).

(c) Chaque machine exécute au maximum un job à la fois.

(d) Chaque job ne peut être exécuté par plus d'une machine à un instant donné.

(e) La séquence des jobs est la même sur chaque machine (ordonnancement de permutation).

Outre que les machines, nous avons un ensemble $R = \{R_1, R_2, ..., R_r\}$ de r ressources non-renouvelables. Chaque ressource R_l, $l \in \{1, ..., r\}$ est livrée par des fournisseurs externes à la machine où il y a un besoin avec des quantités limitées à plusieurs dates de livraison fixes.

Différentes situations peuvent être rencontrées dans ce problème, notamment lorsque : $(r > m)$ ou $(r < m)$ ou $(r = m)$ et pour chaque cas, le job sur chaque machine peut consommer une ou plusieurs ressources.

Dans ce travail, nous supposons que R_1 est livré à M_1, R_2 est livré à M_2 et ainsi de suite (c'est-à-dire que, $r = m$ et que le job pour son traitement sur chaque machine ne consomme qu'une seule ressource). Le problème considéré peut être représenté par la figure 4.1.

Soit q le nombre de moments où une quantité positive de ressource non-renouvelable est fourni. Ces moments sont $u_{1j}, u_{2j}, ..., u_{qj}$ où u_{1j} et u_{qj} représentent respectivement le moment à lequel la première et la dernière livraison est arrivée à la machine M_j. L'utilisation de chaque ressource à tout moment est limitée par sa disponibilité notée b_{tj}, $t \in \{u_{1j}, ..., u_{qj}\}$ qui représente la quantité de ressource non-renouvelable arrivé à l'instant t à la machine M_j.

Chaque opération O_{ij} consomme a_{ij} unités de ressource non-renouvelable dédiée à la machine M_j au début de leur traitement sur cette machine. Le job ne peut commencer le traitement que lorsque la quantité de ressource non-renouvelable requis est disponible en quantités suffisantes. Après l'exécution du job, le stock de la ressource est diminué par la quantité demandée.

Étant donné que la disponibilité des ressources consommables peut être suffisante ou non, la machine restera inactive pendant un certain temps uniquement si les ressources requises ne sont pas disponibles en quantité suffisante lors du démarrage de traitement des jobs.

L'ordonnancement est réalisable si :

(f) Les contraintes de ressources sont respectées, c'est-à-dire qu'à tout moment t et pour chaque ressource R_l, l'offre totale correspond au moins à la demande totale de ces jobs.

Sans perte de généralité, nous supposons que pour chaque ressource non-renouvelable, la quantité totale fournie est égal à la quantité totale demandée par les jobs, c'est-à-dire :

$$\sum_{t=u_{1j}}^{u_{qj}} b_{tj} = \sum_{i=1}^{n} a_{ij} \qquad \forall j \in \{1, ..., m\} \tag{1}$$

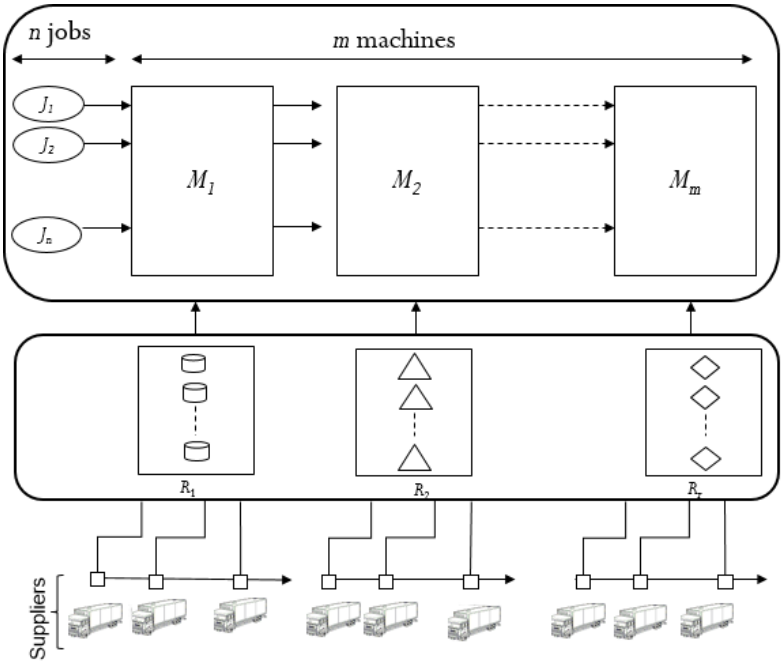


FIGURE 4.1 : Représentation du problème

L'objectif visé par de la résolution de ce problème est de chercher une séquence de jobs en fonction de la disponibilité des ressources renouvelables et non-renouvelables de manière à satisfaire la minimisation du temps d'exécution maximum (C_{max}), aussi appelé le makespan.

$$C_{max} = \sum_{i=1}^{n} p_{im} + \sum_{i=1}^{n} I_{im} + \sum_{i=1}^{n} I'_{im} \tag{2}$$

Où I_{im} et I'_{im} indiquent le temps d'inactivité sur la dernière machine avant le traitement du job J_i, I_{im} est causé par les contraintes du Flow-Shop tandis que I'_{im} est causé par l'indisponibilité des ressources non-renouvelables.

Suivant le schéma de classification utilisé par Toker et al. [128] et Gafarov et al. [44], nous notons le problème d'ordonnancement considéré par $F_m|prmu, NR|C_{max}$, où m est le nombre de machines, $prmu$ indique que seuls les ordonnancements de permutation sont autorisés, NR indique l'existence de ressources non-renouvelables et C_{max} désigne la minimisation de makespan comme critère d'optimisation.

4.1.2 Une instance de problème

Afin d'illustrer l'impact des contraintes de ressources non-renouvelables sur le makespan du problème, nous considérons un exemple d'un Flow-Shop à 3 jobs et 3 machines.

La matrice de temps de traitement suivante p_{ij} contient la liste des jobs J_i en lignes et la liste des machines M_j en colonnes.

$$p_{ij} = \begin{bmatrix} 3 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix} \qquad i, j \in \{1, ..., 3\}$$

Figure 4.2 représente le diagramme de Gantt pour la permutation de jobs $J_1 \longrightarrow J_2 \longrightarrow J_3$ obtenue avec le Flow-Shop de permutation classique « sans contraintes de ressources ». Comme nous pouvons le constater, le résultat est un ordonnancement qui ne contient qu'un seul temps d'inactivité (temps mort) dû aux contraintes de système Flow-Shop. Le makespan correspondant est égal à 10 unités de temps.

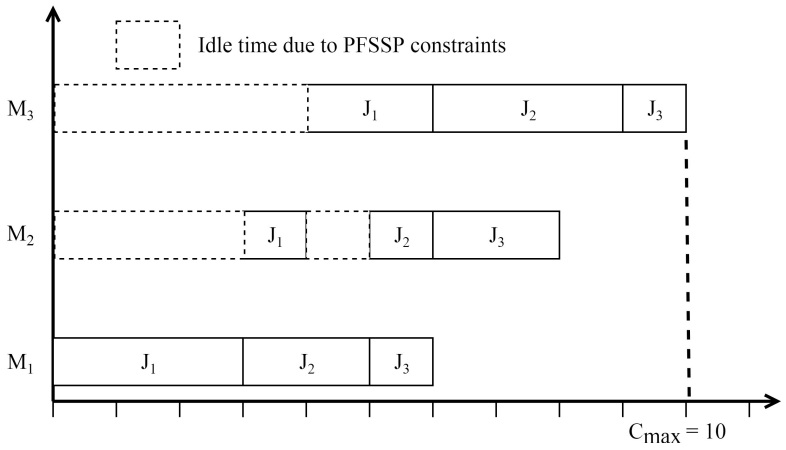


FIGURE 4.2 : Flow-Shop classique

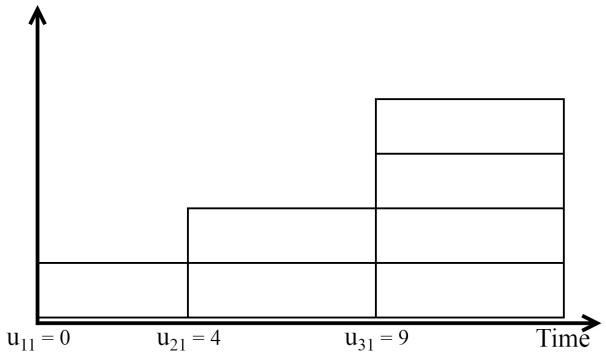
Dans le cas où les contraintes de ressources non-renouvelables sont prises en compte, en plus de la matrice de temps de traitement, nous avons également une matrice de besoin de ressources non-renouvelables.

La matrice de besoins en ressources a_{ij}, a une liste des job J_i en lignes et la liste des machines/-ressources en colonnes (par exemple, J_1 nécessite deux unités de la ressource R_1 sur la première machine, trois unités de la ressource R_2 sur la deuxième machine et quatre unités de la ressource R_3 sur la troisième machine et ainsi de suite).

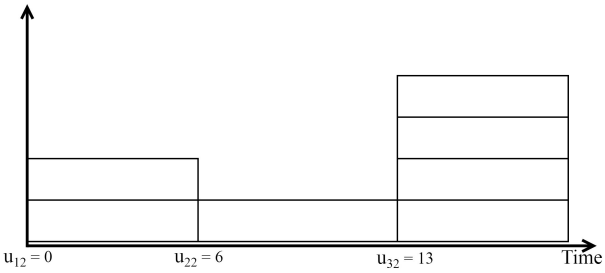
$$a_{ij} = \begin{bmatrix} 2 & 3 & 4 \\ 2 & 2 & 3 \\ 3 & 1 & 1 \end{bmatrix} \qquad i, j \in \{1, ...3\}$$

La disponibilité des ressources est représenté par Figure 4.3 où le nombre de cellules représente les unités de ressource non-renouvelables fournies à un moment donné.

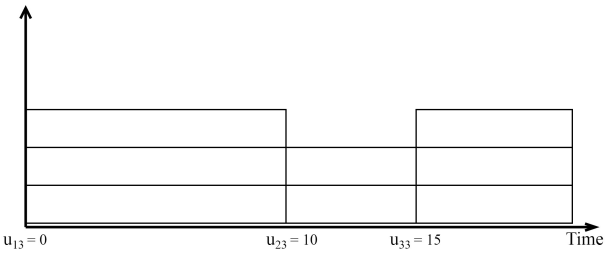
La figure 4.4 illustre le diagramme de Gantt obtenu pour la même permutation de job pour le Flow-Shop de permutation sous contraintes de ressources non-renouvelables. Comme on peut remarquer, le temps total d'exécution augmente de 9 unité temps. Cela est dû à l'indisponibilité



(a) La disponibilité de la ressource R_1



(b) La disponibilité de la ressource R_2



(c) La disponibilité de la ressource R_3

FIGURE 4.3 : La disponibilité des ressources non-renouvelables

de ressources non-renouvelables. Par exemple, le job J_1 n'a pas pu commencé son traitement sur la première machine jusqu'au l'arrivée d'une quantité suffisante de la ressource $R1$.

4.1.3 Complexité de problème

Nous avons présenté dans le chapitre 3 un état de l'art des travaux qui ont étudié la complexité des problèmes d'ordonnancement sous contraintes de ressources non-renouvelables. Puisque le problème d'ordonnancement Flow-Shop sous contrainte de ressources non-renouvelables n'a pas été traité auparavant dans la littérature, nous étudions donc dans cette section la complexité du problème considéré. La preuve de la NP-difficulté des deux corollaires suivants se fera via la méthode de restriction.

Corollaire 1 *$F_m|prmu, NR|C_{max}$ est NP-difficile au sens fort pour $m > 2$.*

preuve 1 *Soit p le problème d'ordonnancement Flow-Shop de permutation sous contraintes de ressources non-renouvelables avec minimisation de makespan. Construisant le problème p' comme étant un problème d'ordonnancement Flow-Shop classique (sans contraintes). Clairement, le problème p' est un cas particulier de problème p (puisque le besoin en ressources peut être limité à 0 dans le problème p). En effet, le problème p' est connu pour être NP-difficile au sens fort pour $m > 2$ [45]. Il en résulte que le problème p est NP-difficile au sens fort pour $m > 2$.*

Corollaire 2 *$F_2|prmu, NR|C_{max}$ est NP-difficile au sens fort.*

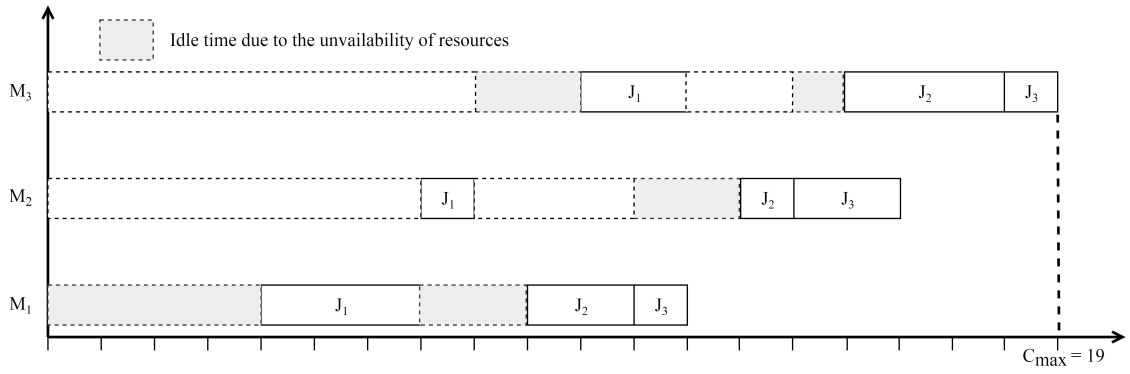


FIGURE 4.4 : Flow-Shop avec la contrainte de ressources non-renouvelables

preuve 2 *Soit p le problème flow-shop à deux machines sous contraintes de ressources non-renouvelables avec minimisation de makespan (les besoins en ressources et leur disponibilité sont arbitraires.). Construisant le problème p' en tant que problème d'ordonnancement d'une seule machine soumis à des contraintes de ressources non-renouvelables avec minimisation de makespan. Nous observons, que le problème p' est un cas particulier du problème p, puisque le temps de traitement et les besoins en ressources sur la deuxième machine peuvent être limités à 0 dans le problème p. Or, le problème p' $(1|NR|C_{max})$ a été prouvé, par Carlier [18] et Gafarov et al. [44], qu'il est NP-difficile au sens fort pour un temps de traitements arbitraire ainsi qu'une consommation et disponibilité de ressources arbitraire. Le résultat du corollaire est donc établi.*

4.2 Modèle mathématique

Après avoir déterminé la complexité du problème considéré dans cette thèse, nous allons procéder dans cette section à sa résolution exacte. Nous avons donc proposé un modèle mathématique linéaire en nombres entiers pour résoudre le problème de minimisation du makespan d'un Flow-Shop de permutation sous contraintes de ressources non-renouvelables. Cette modélisation est inspirée du modèle présenté par Carrera [21] pour minimiser le makespan dans un problème à une machine avec la consommation d'une seule ressource consommable, ainsi que le modèle proposé par Wilson [140] pour le Flow-Shop classique.

Dans ce qui suit, nous allons commencer par présenter les paramètres, les variables et contraintes du modèle.

4.2.1 Paramètres et indices

Les paramètres ainsi que les indices utilisés pour la formulation mathématique du problème d'ordonnancement de type Flow-Shop sous contraintes de ressources non-renouvelables sont les suivants :

- n : nombre de jobs.
- m : nombre de machines.
- i : indice de job, où $i \in \{1, ..., n\}$.
- j : indice de machine, où $j \in \{1, ..., m\}$.
- k : indice de position, où $k \in \{1, ..., n\}$.
- t : Indice d'arrivée de ressource, où $t \in \{u_{1j}, ..., u_{qj}\}$.
- u_{1j} : date de la première livraison arrivée à la machine M_j.
- u_{qj} : date de la dernière livraison arrivée à la machine M_j.
- p_{ij} : temps du traitement du job J_i sur la machine M_j.
- a_{ij} : quantité de ressource que le job J_i consomme sur la machine M_j.
- b_{tj} : nombre total de la ressource dédiée arrivée à la machine M_j à l'instant t.
- q : nombre de dates de livraison.
- M : Constante suffisamment grande.

4.2.2 Variables

- S_{kj} : date de début du job en position k sur la machine M_j.
- W_{kj} : quantité de ressource nécessaire pour traiter le job en position k sur la machine M_j.
- X_{ik} : variable booléenne, égale à 1 si le job J_i se trouve à la position k dans la séquence et 0 sinon.
- Y_{kjt} : variable booléenne, égale à 1 si $S_{kj} \geq t$ et 0 sinon.

4.2.3 Modèle

En utilisant les paramètres et les variables décrits précédemment, le problème $F_m|prmu, NR|C_{max}$ peut être modélisé de la façon suivante :

$$Min \quad (S_{nm} + \sum_{i=1}^{n} X_{in}.p_{im}) \tag{1}$$

(4.1)

Sous les contraintes :

$$\sum_{i=1}^{n} X_{ik} = 1 \qquad \forall k \in \{1,...,n\} \tag{2}$$

$$\sum_{k=1}^{n} X_{ik} = 1 \qquad \forall i \in \{1,...,n\} \tag{3}$$

$$S_{(k+1)1} \geq S_{k1} + \sum_{i=1}^{n} X_{ik}.p_{i1} \qquad \forall k \in \{1,...,n-1\} \tag{4}$$

$$S_{11} \geq 0 \tag{5}$$

$$S_{1(j+1)} \geq S_{1j} + \sum_{i=1}^{n} X_{i1}.p_{ij} \qquad \forall j \in \{1,...,m-1\} \tag{6}$$

$$S_{k(j+1)} \geq S_{kj} + \sum_{i=1}^{n} X_{ik}.p_{ij} \qquad \forall j \in \{1,...,m-1\} \forall k \in \{2,...,n\} \tag{7}$$

$$S_{(k+1)j} \geq S_{kj} + \sum_{i=1}^{n} X_{ik}.p_{ij} \qquad \forall j \in \{2,...,m\} \forall k \in \{1,...,n-1\} \tag{8}$$

$$W_{kj} = \sum_{i=1}^{n} X_{ik}.a_{ij} \qquad \forall k \in \{1,...,n\} \forall j \in \{1,...,m\} \tag{9}$$

$$\sum_{v=1}^{k} W_{vj} \leq \sum_{t=u_{1j}}^{u_{qj}} b_{tj}.Y_{kjt} \qquad \forall k \in \{1,...,n\} \forall j \in \{1,...,m\} \tag{10}$$

$$Z\left(Y_{kjt}-1\right) \leq S_{kj} - t \qquad \forall k \in \{1,...,n\} \forall j \in \{1,...,m\} \forall t \in \left\{u_{1j},...,u_{qj}\right\} \tag{11}$$

$$S_{kj}, W_{kj} \geq 0 \qquad \forall k \in \{1,...,n\} \forall j \in \{1,...,m\} \tag{12}$$

$$X_{ik}, Y_{kjt} \in \{0,1\} \qquad \forall i,k \in \{1,...,n\} \forall j \in \{1,...,m\} \forall t \in \left\{u_{1j},...,u_{qj}\right\} \tag{13}$$

4.2.4 Signification des équations

Nous donnons dans ce paragraphe, la signification de chacune des contraintes de notre modèle :

- Fonction 1 : représente la fonction objectif de notre problème. Qui consiste à minimiser le temps d'exécution maximum, appelé C_{max} ou encore makespan décrit par la date de début du job placé dans la dernière position, dernière machine plus son temps d'exécution.
- Contrainte 2 : cette contrainte impose que chaque position k n'est affecté qu'un seul job J_i dans la séquence.
- Contrainte 3 : cette contrainte force que chaque job j_i n'occupe qu'une seul position j dans la séquence.
- Contrainte 4 : cette contrainte calcule la date de début du job en position k sur la première machine.
- Contrainte 5 : cette contrainte assure que la date de début du job en première position sur la première machine peut être supérieure à zéro si la ressource non-renouvelable n'est pas disponible en quantité suffisante au début d'ordonnancement (Contrairement au flow

shop standard où $S_{11} = 0$).

- Contrainte 6 : cette contrainte calcule la date de début du job en première position sur la machine M_j.
- Contrainte 7 : cette contrainte représente la relation de précédence entre deux opérations successives d'un même job. Un job doit finir son opération sur la machine M_j, avant de pouvoir démarrer son opération sur la machine suivante M_{j+1}.
- Contrainte 8 : cette contrainte assure que le job en position $(k+1)$ ne puisse pas commencer son traitement sur la machine M_j tant que le job en position k n'a pas encore terminé son traitement sur cette même machine.
- Contrainte 9 : cette contrainte permet de calculer la quantité de ressource consommée par le job en position k sur la machine M_j.
- Contrainte 10 : cette contrainte vérifie que la quantité totale de ressource consommée à l'instant de début d'exécution du job en position k sur la machine M_j ne doit pas dépasser la quantité disponible de cette ressource jusqu'à cet instant.
- Contrainte 11 : cette contrainte permet de faire le lien entre la variable Y_{kjt} et la date de début du job en position k.
- Contrainte 12 : la date de début du job et la quantité de ressource requise pour traiter un job ne peuvent pas être négatives.
- Contrainte 13 : cette contrainte établit une restriction sur les variables de décision X_{ik} et Y_{kjt}.

4.3 Benchmarks

Pour tester les performances du modèle mathématique développé dans la section précédente et les méthodes de résolution proposées dans les chapitres suivants, nous avons besoin des jeux-tests ou bien ce qu'on appelle des benchmarks.

Les benchmarks sont des problèmes-types ou un ensemble d'instances construits en explicitant la valeur de chacun des paramètres du problème. Ils sont souvent utiles pour tester les performances des méthodes de résolution.

Sans aucune hésitation, le benchmark le plus utilisé pour les problèmes d'ordonnancement d'atelier de type Flow-Shop est celui de Taillard [124](Voir par exemple [2, 32, 96, 113]). Ces instances ont été générées pour tester des algorithmes pour le problème Flow-Shop de permutation avec minimisation de makespan, bien qu'elles aient également été utilisées sous d'autres critères et conditions [95, 114]. Il existe aussi d'autres benchmarks pour les problèmes Flow-Shop mais ils sont beaucoup moins utilisés, comme ceux de Demirkol et al. [33], Reeves [111] ou récemment celui proposé par Vallada et al. [134] ou des benchmarks plus anciens qui ne sont plus utilisés actuellement, comme ceux de Carlier [19] et Heller [56].

Le benchmark de Taillard pour le problème Flow-Shop est disponible à l'adresse suivante : http:

```
number of jobs, number of machines, initial seed, upper bound and lower bound :
          20           5   873654221        1278        1232
processing times :
 54 83 15 71 77 36 53 38 27 87 76 91 14 29 12 77 32 87 68 94
 79  3 11 99 56 70 99 60  5 56  3 61 73 75 47 14 21 86  5 77
 16 89 49 15 89 45 60 23 57 64  7  1 63 41 63 47 26 75 77 40
 66 58 31 68 78 91 13 59 49 85 85  9 39 41 56 40 54 77 51 31
 58 56 20 85 53 35 53 41 69 13 86 72  8 49 47 87 58 18 68 28
```

FIGURE 4.5 : Exemple d'une instance de Taillard

//mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

Ce benchmark présente les caractéristiques suivantes :

- Il comprend 120 instances avec 12 tailles différentes donnée par $n \times m$ et combinant les valeurs $n \in [20, 50, 100, 200, 500]$ et $m \in [5, 10, 20]$.
- Pour chaque taille, 10 instances sont construites.
- Pour chaque instance, les deux premières lignes contiennent la description de l'instance et le reste représente les temps de traitement.
- Les temps de traitement pour les différentes opérations sont uniformément répartis de 1 à 99.
- A part le temps de traitement, il n'existe pas d'autre paramètres de contraintes temporelles ou contraintes de ressources.

La figure 4.5 nous montre un exemple d'une instance de taille 20×5 (20 jobs et 5 machines) du benchmark de Taillard sur laquelle nous constatons les caractéristiques générales de ce benchmark.

Au cours de la revue de la littérature, nous avons rencontré une difficulté à trouver des instances numériques pour les problèmes d'atelier Flow-Shop de permutation soumises a des contraintes de ressources non-renouvelables. Nous avons trouvé quelques papiers qui investiguent les contraintes de ressources non-renouvelables dans d'autres environnements de production mais qu'ils construisent et génèrent les instances numériques de façon aléatoire ou bien d'autre papiers ou la génération des instances n'est pas bien spécifié [8, 52].

Comme il n'y a pas d'instances dédiées aux problèmes d'ordonnancement Flow-Shop de permutation avec contraintes de ressources non-renouvelables, nous avons donc décidé d'étendre le benchmark de Taillard tout en ajoutant les paramètres de contraintes de ressources non-renouvelables.

Une instance de problème traité est définie par le nombre jobs n, le nombre de machines m, le nombre de ressources non-renouvelables r, les temps opératoires, la quantité de ressource non-renouvelable requiert par chaque job ainsi que la disponibilité de ces ressources.

Pour la génération numérique de ces instances et comme nous avons mentionné auparavant, nos

instances sont une extension des instances proposées par Taillard. Alors, le nombre de jobs et le nombre de machines ainsi que les temps opératoires reste identique comme celui proposé par Taillard.

- Le nombre de jobs est de : 20, 50, 100, 200, 500. Mais seulement les trois premières tailles sont utilisées, car les autres consomment beaucoup de temps pour les algorithmes proposés dans le chapitre 5 et 6.
- Le nombre de machines est de : 5, 10, 20.
- Les temps opératoires sont des entiers uniformément distribués dans [1, 99].

Le reste des données dédié au ressources non-renouvelables sont générées comme expliqué ci-dessous :

- Le nombre de ressources non-renouvelables est déterminé par le nombre de machines (nous supposons une ressource par machine.).
- La quantité de ressource non-renouvelable requiert par chaque job est généré uniformément dans [1, 9].
- La disponibilité de ressources non-renouvelables est donné par une courbe précisant les dates $u_{1j}, u_{2j}, ..., u_{qj}$ où la ressource affectée à la machine j est disponible ainsi que les quantités b_{tj},$(t \in u_{1j}, ..., u_{qj})$ arrivant a ces dates.
- Pour chaque ressource consommable, nous supposons que le nombre de dates d'approvisionnement est égale au nombre de machines i.e., $q = m$.
- Pour chaque ressource l, nous générons les dates d'approvisionnement entre 0 et $\sum p_{i*}$ $(* = l)$ et nous divisons $\sum a_{ij}$ $(j = l)$ unités de ressource l aléatoirement entre les dates d'approvisionnement.
- Pour la disponibilité de ressources, nous pouvons distinguer plusieurs fonctions d'arrivée représentées par la figure 4.6 et présentées comme suite :
  - $F1$: la disponibilité de la ressource non-renouvelable est représentée par une fonction en escalier croissante.
  - $F2$: la disponibilité de la ressource non-renouvelable est représentée par une fonction en escalier décroissante.
  - $F3$: la disponibilité de la ressource non-renouvelable est représentée par une fonction en escalier aléatoire.

Deux configurations d'arrivée de ressources sont testées :

1. Arrivée uniforme (UAF*), dans lequel toutes les ressources non-renouvelables du système suivent la même fonction d'arrivée (F* est soit $F1$ ou $F2$ ou $F3$.).
2. Arrivée non-uniforme (NUA), dans lequel la fonction de disponibilité d'une ressource non-renouvelable est supposée être soit $(F1)$ ou $(F2)$ ou $(F3)$.

En résumé, nous générons des instances pour les (n, m) paires suivantes : (20, 5), (20, 10), (20, 20), (50, 5), (50, 10), (50, 20), (100, 5), (100, 10) et (100, 20). De plus, afin de tester les limites du

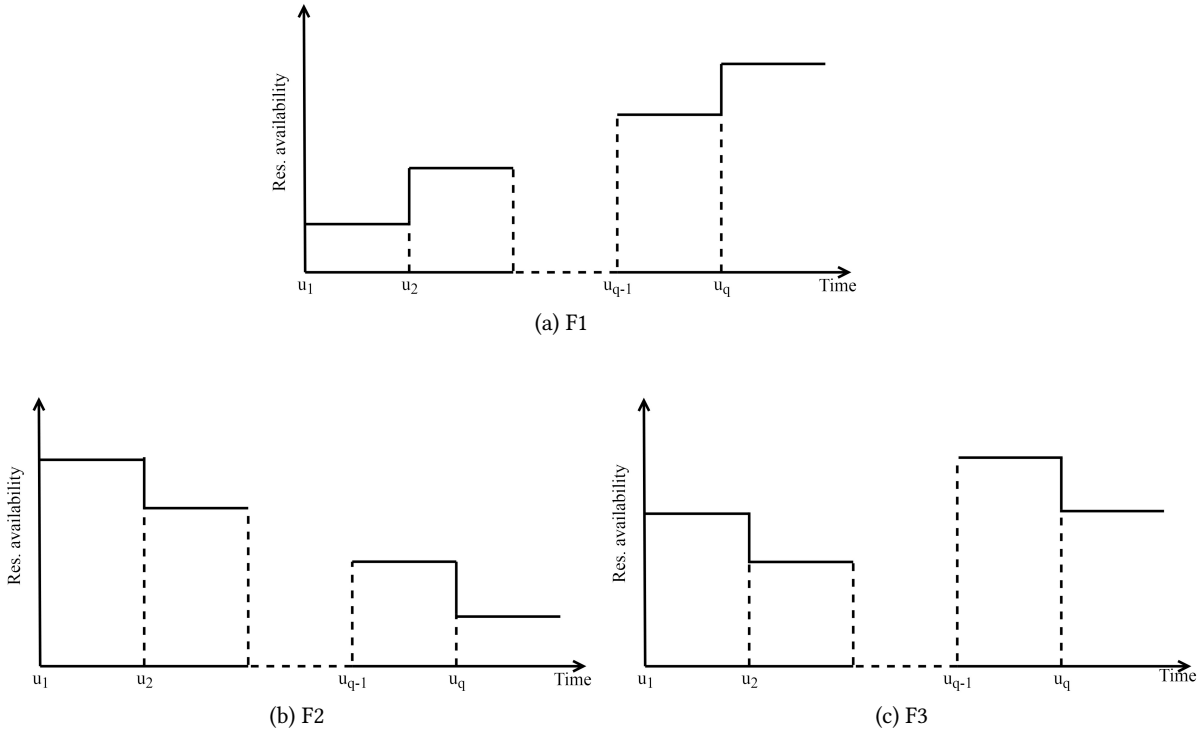


FIGURE 4.6 : Disponibilité de ressources

modèle mathématique, outre l'ensemble de paires de Taillard considérés, nous générons un autre ensemble comme suit : (10,5), (10,10), (10,20). Pour chaque cas, nous examinons ces instances avec les différentes configurations d'arrivée de ressources (UAF1, UAF2, UAF3, NUA) et cela afin de montrer l'influence de la disponibilité de ressources sur les résultats. Cela donne un total de 48 classes d'instances.

La figure 4.7 nous montre un exemple d'une instance de taille (20,5) de notre benchmark ou la disponibilité de ressources est représentée par une arrivée uniforme croissante (UAF1).

4.4 Résultats numériques

Dans cette partie, nous présentons les résultats de l'application numérique du modèle mathématique décrit dans ce chapitre sur les instances du problème d'ordonnancement d'atelier de production Flow-Shop soumise a des contraintes de ressources non-renouvelables.

Afin d'établir les limites du modèle mathématique développé pour ce problème et étudier l'influence de la disponibilité de ressources sur les résultats, nous exécutons le modèle lorsque la disponibilité de ressources est représentée par une arrivée uniforme croissante (UAF1), uniforme décroissante (UAF2), uniforme aléatoire (UAF3) et non-uniforme (NUA).

Les expérimentations sont effectuées en utilisant le solveur commercial CPLEX 12.6 sur le bench-

Number of job, number of machines, number of non-renewable resources, resources availability

20 5 5 UAF1

p

54	83	15	71	77	36	53	38	27	87	76	91	14	29	12	77	32	87	68	94
79	3	11	99	56	70	99	60	5	56	3	61	73	75	47	14	21	86	5	77
16	89	49	15	89	45	60	23	57	64	7	1	63	41	63	47	26	75	77	40
66	58	31	68	78	91	13	59	49	85	85	9	39	41	56	40	54	77	51	31
58	56	20	85	53	35	53	41	69	13	86	72	8	49	47	87	58	18	68	28

a

5	6	7	4	6	7	5	3	5	5	7	6	2	6	2	8	8	4	1	4
7	3	5	4	4	8	8	7	5	6	7	1	2	3	4	7	8	2	7	6
2	8	6	6	8	2	1	7	3	2	6	1	3	2	7	1	4	4	3	8
3	3	2	3	2	1	7	7	6	7	7	6	2	6	5	3	3	2	5	3
8	1	7	4	8	5	8	3	8	6	3	2	5	7	4	6	3	4	6	2

b

6	13	20	26	36
3	17	19	28	37
1	5	19	24	35
7	12	17	21	26
11	13	17	20	39

u

0	32	176	621	786
0	79	106	564	809
0	110	491	621	958
0	322	617	864	1087
0	282	756	921	1158

FIGURE 4.7 : Exemple d'une instance du problème traité

mark généré suivant la procédure proposée dans la section précédente. La résolution de ces instances est bornée à 1800 secondes.

Dans le tableau 4.1, $\overline{C_{max}^{CPLEX}}$ exprime le makespan moyen obtenu par CPLEX pour cinq réplications et $\overline{CPU}$ représente le temps de calcul moyen en secondes.

À partir du tableau 4.1, nous constatons que le modèle proposé peut résoudre de manière optimale les instances jusqu'à 20 jobs et 5 machines dans un délai de 1800 s, quelle que soit la configuration UAF1, UAF2, UAF3, ou NUA. Par conséquent, la complexité du problème augmente avec le nombre de jobs à ordonner et les configurations considérées de la disponibilité des ressources n'affectent pas la complexité du problème.

En outre, nous observons aussi que le temps de calcul moyen obtenu par toutes les instances résolues lorsque les ressources non-renouvelables sont fournies par une arrivée uniforme décroissante (UAF2) est inférieur au temps de calcul moyen obtenu par les autres configurations.

TABLE 4.1 : Résultats obtenus par le solveur CPLEX

Problème	Configuration							
	UAF1		UAF2		UAF3		NUA	
	$\overline{C_{max}^{CPLEX}}$	$\overline{\text{CPU}}$	$\overline{C_{max}^{CPLEX}}$	$\overline{\text{CPU}}$	$\overline{C_{max}^{CPLEX}}$	$\overline{\text{CPU}}$	$\overline{C_{max}^{CPLEX}}$	$\overline{\text{CPU}}$
10 x 5	886.40	8.39	798.40	5.76	965.40	7.28	895.40	4.84
10 x 10	1142.20	48.00	1081.60	34.86	1157.60	36.13	1150.60	37.69
10 x 20	1775.00	768.88	1695.20	353.30	1764.80	859.53	1760.00	769.33
20 x 5	1538.20	415.10	1388.25	217.11	1516.40	440.20	1496.75	96.96
20 x 10	-	> 1800	-	> 1800	-	> 1800	-	> 1800

4.5 Conclusion

Dans ce chapitre, nous avons examiné le problème d'ordonnancement d'atelier Flow-Shop avec contraintes de ressources non-renouvelables noté $F_m|prmu, NR|C_{max}$. L'objectif est de trouver une séquence appropriée de jobs en fonction de la disponibilité des ressources renouvelables et non-renouvelables, de manière à minimiser le temps d'exécution maximum (makespan). Dans un premier temps, nous avons décrit formellement le problème et nous avons étudier sa complexité. Ainsi, nous avons donné un exemple pour illustrer l'impact des contraintes de ressources non-renouvelables sur le makespan du problème. Dans un second temps, nous avons traité ce problème par une méthode exacte basée sur un modèle linéaire en nombres entiers. Cette modélisation a permis de mieux comprendre le problème grâce aux expressions mathématiques.

Le modèle mathématique a été formulé et validé par le solveur CPLEX, les résultats obtenus par ce dernier montrent que le modèle proposé peut résoudre de manière exacte jusqu'à 20 jobs et 5 machines dans la limite du temps. Nous avons constaté également que la configuration de ressources non-renouvelables n'affecte pas la complexité de problème.

Vu la complexité du problème considéré, les méthodes exactes en général ne peuvent pas être utilisées pour résoudre les instances de moyenne à grande taille. C'est pour cela que nous abordons dans les chapitres suivants les méthodes approchées pour résoudre en temps raisonnable les instances de taille importante.

Chapitre 5

Algorithme génétique pour le problème Flow-Shop sous contraintes de ressources non-renouvelables

5.1 Introduction

Dans le chapitre précédent, nous avons présenté le problème traité dans ce manuscrit qui est un problème d'atelier Flow-Shop soumis à des contraintes de ressources non-renouvelables. Nous avons proposé un modèle mathématique pour décrire le problème et nous avons exposé les résultats obtenus pour différentes configurations de disponibilité de ressources. Nous avons constaté que le temps d'exécution augmente avec le nombre de jobs et le nombre de machines et que pour les problèmes de taille importante, il n'est pas possible d'obtenir une solution optimale à ces problèmes en un temps raisonnable. C'est pour cette raison que dans ce chapitre nous abordons les méthodes approchées comme une alternative pour trouver des solutions de bonnes qualités pour résoudre les problèmes de taille importante dans un temps raisonnable.

Le Flow-shop de permutation avec contraintes de ressources non-renouvelables est un problème d'ordonnancement dont l'objectif consiste à déterminer la meilleure séquence de jobs tout en respectant la disponibilité des ressources consommables. Si nous prenons ce problème séparément, c'est-à-dire, les problèmes à ressources consommables d'une part et les problèmes d'ordonnancement d'ateliers Flow-Shop d'autre part. Ces deux problèmes ont attiré l'attention d'une communauté de chercheurs, en raison de leur importance dans les contextes théoriques et appliqués. Pour cela, plusieurs approches ont été proposées pour résoudre ces problèmes. Une grande partie des chercheurs ont employé les approches métaheuristiques pour faire face à la complexité des problèmes Flow-Shop sous divers contraintes e.g., [11, 62, 120, 130, 137, 138, 145]. Cependant, l'application des métaheuristiques sur les problèmes d'ordonnancement sous contraintes de ressources consommables est très limité.

Comme nous avons vu au chapitre 2, il existe de nombreuses métaheuristiques pour les problèmes d'optimisation combinatoire, et en particulier, pour les problèmes d'ordonnancement Flow-Shop. Ces métaheuristiques allant de la simple recherche locale à des algorithmes complexes de recherche globale. Une question qui se pose ici est : « Comment pouvons-nous choisir l'algorithme le plus performant pour le problème d'optimisation considéré dans cette thèse ? ».

Comme le stipule le *No Free Lunch Theorem* énoncé en 1997 par D. H. Wolpert et W. G. Macready dans [142], aucune métaheuristique n'est en moyenne meilleure qu'une autre pour résoudre l'ensemble des problèmes d'optimisation. En effet, une métaheuristique peut s'avérer très performant sur un problème particulier donné, et avoir de moins bonnes performances sur un autre type de problème. En d'autres termes, la meilleure performance d'un algorithme A par rapport à un algorithme B pour un problème donné signifie simplement que l'algorithme A (c'est-à-dire l'algorithme lui-même, mais aussi un choix de paramètres et une implémentation donnée) est mieux adapté à ce problème et qu'il peut être alors moins performant que B sur l'ensemble des autres problèmes d'optimisations.

La chose que nous devons garder à l'esprit au sujet de l'évaluation des performances des algorithmes est que la performance dépend de la structure du problème à optimiser et au paramétrage de l'algorithme. Par conséquent, le choix d'un algorithme qui convient le mieux pour un problème donné représente une tâche complexe qui nécessite une bonne connaissance des algorithmes et du problème à optimiser.

Dans ce chapitre, la méthode approchée suggérée pour résoudre le problème d'ordonnancement d'atelier Flow-shop avec contraintes de ressources non-renouvelables est une métaheuristique à base d'algorithme génétique. La suite de ce chapitre est organisée comme suit : dans la section 5.2, nous présentons les éléments essentiels qui constituent notre algorithme génétique. Dans la section 5.3, nous proposons une procédure de recherche locale pour améliorer le fonctionnement de la métaheuristique proposée. Dans la section 5.4, nous étudions l'influence de différents paramètres et opérateurs de l'algorithme, et nous déterminons le paramétrage approprié. Dans la section 5.6, nous évaluons la performance de l'algorithme développé sur différentes configurations d'arrivée de ressources consommables et nous comparons aussi cette performance à celle de la méthode exacte proposée dans le chapitre 4.

5.2 Algorithme Génétique (AG)

Les algorithmes génétiques constituent la famille des métaheuristiques les plus utilisées en pratique pour surmonter les limites des méthodes exactes et trouver des solutions satisfaisantes dans un temps de calcul raisonnable pour les problèmes complexes. Ils ont prouvé leur efficacité sur différents types de problèmes d'optimisation, et en particulier, les problèmes d'altier Flow-Shop. Les résultats obtenus par l'algorithme génétique pour le problème d'ordonnancement sur machines parallèles avec ressources consommables exposés dans les travaux de Belkaid et al.

[9] nous ont encouragé à utiliser cette méthode sur les problèmes d'ordonnancement d'atelier Flow-Shop sous contraintes de ressources non-renouvelables.

Les algorithmes génétiques, décrits en détail dans le deuxième chapitre de cette thèse, font partie de la famille des algorithmes évolutionnaires inspirés par l'évolution biologique des espèces et s'appuient sur des concepts de la génétique. Dans ces algorithmes, on part d'une population initiale de taille bien déterminée, constitué d'un ensemble de solutions réalisables générées aléatoirement ou avec des méthodes heuristiques ou encore avec un mélange de solutions aléatoires et heuristiques.

Les solutions sont aussi appelées individus ou chromosomes, représentés par une séquence finie de gènes pouvant prendre des valeurs significatives. Chaque chromosome est caractérisé par une fonction fitness (qui mesure son adaptation à l'environnement). L'idée de base d'un algorithme génétique consiste à sélectionner les individus les plus adaptés (appelés parents) pour la reproduction. Ensuite, un opérateur de croisement est appliqué aux parents pour produire des enfants. Les individus peuvent également subir une transformation individuelle appelée opérateur de mutation, qui consiste à appliquer certaines modifications sur le code génétique de l'individu. Ainsi, seules les solutions les plus performantes sont gardées et sélectionnées pour former la population de la génération suivante. Le processus est répété jusqu'à ce qu'une condition d'arrêt est satisfaite.

Les étapes de l'algorithme génétique adaptées à notre problème sont les suivantes :

5.2.1 Codage utilisé

L'étape cruciale dans la conception d'un algorithme génétique consiste à définir un codage, c'est-à-dire un moyen de représenter une solution. Pour résoudre le problème d'ordonnancement Flow-Shop de permutation sous contraintes de ressources non-renouvelables en utilisant l'algorithme génétique, la première attaque consiste à coder une solution du problème sous forme de chromosome.

Puisque nous sommes dans le cadre de la résolution d'un problème d'ordonnancement de permutation, c'est-à-dire que nous cherchons la permutation des jobs qui minimise le makespan en présence des contraintes de ressources non-renouvelables. Le codage que nous avons retenu est basé sur la représentation par permutation. Ce codage a été largement utilisé dans de nombreux papiers pour résoudre des problèmes d'ordonnancement Flow-Shop de permutation classiques ou avec des contraintes additionnelles tel-que la contrainte de blocage, la contrainte sans temps d'arrêt (no-idle) et la contrainte sans délai (no-wait), etc [60, 62, 130].

La permutation des jobs détermine l'ordre relatif d'exécution des jobs sur les machines. La figure 5.1 représente le codage d'une solution quelconque du problème considéré où le job numéro 3 sera lancé en production le premier sur toutes les machines, suivi du job numéro 5 et ainsi de suite.

J_3	J_5	J_1	J_4	J_6	J_2	J_7

FIGURE 5.1 : Codage d'un chromosome

Il est important de noter que la permutation doit être réalisable, c'est-à-dire que la longueur du chromosome est égale au nombre de jobs dans le système et qu'il ne pourrait y avoir aucun job manquant ou répétitive.

5.2.2 Génération de la population initiale

Dans cette étape, nous générons un ensemble de chromosomes, où chaque chromosome représente une solution réalisable du problème abordé. La population initiale joue un rôle essentiel dans la détermination de la qualité de la solution finale et également le nombre de générations au bout duquel on obtient de bonnes solutions [119].

Traditionnellement, dans les AGs, la population initiale est générée de manière aléatoire. La plus part des auteurs utilisant l'initialisation aléatoire déclarent que cette procedure est le meilleur choix, car elle permet l'hétérogénéité de la population. Cependant, après une investigation sur les documents traitant les problèmes Flow-Shop et les AGs, Ruiz et al. [116] proclament que cette procédure ne donne pas de bons résultats et que la procédure d'initialisation dans un tel problème combinatoire doit être faite avec beaucoup de soin, pour assurer une meilleurs convergence, dans un temps raisonnable.

Donc, trouver une bonne population initiale avec une meilleure taille est un problème difficile et une règle générale ne peut pas être appliquée à tout type de problème. Afin de confirmer si la population initial à un impact sur les résultats de notre problème, nous avons testé deux procédures :

(i) Génération aléatoire notée *Rand* qui génère aléatoirement P_s solutions où P_s indique la taille de la population.

(ii) Génération combinée notée *RandHeur* qui est une combinaison entre des solutions générés à l'aide des heuristiques constructives et des solutions générés aléatoirement. L'objectif de cette procédure est d'acquérir une population de solutions diversifiées et de qualité. Dans notre implémentation, trois heuristiques constructives bien connues pour le Flow-Shop de permutation : NEH [97], NEHKK1[67] et Palmer [103] sont initialement utilisées pour générer des solutions tout en respectant les contraintes de ressources non-renouvelables et cela afin d'assurer une bonne qualité de solutions. En outre, pour maintenir la diversité de la population, le reste $(P_s - 3)$ des solutions sont générés aléatoirement.

5.2.2.1 Les heuristiques constructives utilisées

L'utilisation de ces heuristiques est justifiée par la bonne qualité des résultats donnés au sein de différents travaux, notamment ceux de Reeves [112], Ruiz et al. [116] et Zobolas et al. [145].

5.2.2.1.1 Heuristique NEH

L'heuristique NEH (Nawaz-Enscore-Ham) proposée par Nawaz et al. [97] est considéré comme l'une des meilleures heuristiques constructives pour les problèmes Flow-Shop de permutation dans le cas de minimisation du makespan. Selon l'analyse et la comparaison effectuées par Ruiz et al. [117] et kalczynski et al. [68], cette heuristique est compétitive en considérant à la fois la qualité de la solution obtenue et le temps du processus de recherche.

L'idée de NEH consiste à construire progressivement une séquence correspondant à la solution en insérant à chaque étape le job ayant une durée totale d'exécution élevée, parmi les jobs non encore ordonnancé, à la meilleure position dans une séquence partielle, de manière à minimiser le makesapan. L'adaptation de cette heuristique pour initialiser une solution du problème considéré est illustrée par l'algorithme 5.

Algorithme 5 Algorithme NEH

Entrées : Ensemble J de n jobs
Sorties : Séquence π

Début
1 : **for** $i = 1$ to n **do**
2 : **Calculer** la somme de temps opératoire S_i pour chaque job J_i, avec $S_i = \sum_{j=1}^{m} P_{ij}$
3 : **end for**
4 : **Classer** les jobs dans l'ordre décroissant de S_i
5 : **Sélectionner** les deux premiers jobs et choisir la séquence π de ces deux jobs qui minimise le C_{max} tout en respectant la disponibilité des ressources non-renouvelables
6 : **for** $k = 3$ to n **do**
7 : **Tester** l'insertion du job suivant à n'importe quelle position possible de π tout en respectant la disponibilité des ressources non-renouvelables
8 : **gardez** l'insertion qui donne la plus faible valeur de C_{max}.
9 : **Mise à jour** de π
10 : **end for**
Fin

5.2.2.1.2 Heuristique NEHKK1

Kalczynski et Kamburowski [67] ont observé que l'heuristique NEH peut être améliorée en insérant les jobs dans le même ordre que celui proposé par Johnson [65] pour le problème Flow-Shop de permutation à deux machines. Si nous définissons :

$$\hat{a}_i = \sum_{j=1}^{m} [(m-1)(m-2)/2 + m - j] * P_{ij} \tag{5.1}$$

$$\hat{b}_i = \sum_{j=1}^{m} [(m-1)(m-2)/2 + j - 1] * P_{ij} \tag{5.2}$$

Alors NEHKK1 insère les jobs dans l'ordre décroissant de $\hat{c}_i = \min\{\hat{a}_i, \hat{b}_i\}$ comme le montre l'algorithme 6. L'efficacité de cette méthode a été confirmée par Fernandez-Viagas et Framinan [40].

L'heuristique NEHKK1 a la propriété supplémentaire de résoudre de manière optimale le problème d'ordonnancement Flow-Shop de permutation à deux machines. Nous avons utilisé le résultat de cette heuristique comme point de départ pour l'amélioration de la qualité de la population initiale de notre algorithme génétique tout en respectant les contraintes de ressources.

Algorithme 6 Algorithme NEHKK1

Entrées : Ensemble J de n jobs
Sorties : Séquence π

Début
1 : **for** $i = 1$ to n **do**
2 : **Calculer**
3 : $\hat{a}_i = \sum_{j=1}^{m} [(m-1)(m-2)/2 + m - j] * P_{ij}$;
4 : $\hat{b}_i = \sum_{j=1}^{m} [(m-1)(m-2)/2 + j - 1] * P_{ij}$;
5 : **if** $\hat{a}_i \leq \hat{b}_i$ **then**
6 : $c_i = \hat{a}_i$
7 : **else**
8 : $c_i = \hat{b}_i$
9 : **end if**
10 : **end for**
11 : **Classer** les jobs dans l'ordre décroissant de c_i
12 : **Sélectionner** les deux premiers jobs et choisir la séquence π de ces deux jobs qui minimise le C_{max} tout en respectant la disponibilité des ressources non-renouvelables
13 : **for** $k = 3$ to n **do**
14 : **Tester** l'insertion du job suivant à n'importe quelle position possible de π tout en respectant la disponibilité des ressources non-renouvelables
15 : **gardez** l'insertion qui donne la plus faible valeur de C_{max}.
16 : **Mise à jour** de π
17 : **end for**
Fin

5.2.2.1.3 Heuristique de Palmer

Palmer [103] a introduit le concept d'indice pour déterminer la meilleur séquence de jobs pour les problèmes Flow-Shop. L'heuristique de Palmer consiste en deux phases, dans la phase initiale, l'indice est calculée pour chaque job en utilisant l'équation suivante :

$$S_i = \sum_{j=1}^{m} (m - 2 * j + 1) * P_{ij} \tag{5.3}$$

Cet indice permet de favoriser les travaux ayant des petits temps opératoires sur les premières machines et des grands temps opératoires sur les dernières machines en les plaçant en premier, et dans la configuration inverse en dernier. Ensuite, la meilleure solution est obtenue en agençant les jobs dans l'ordre décroissant par rapport à leur indice S_i, comme le montre l'algorithme 7.

Cette heuristique se caractérise par un minimum d'effort de calcul et une méthodologie efficace pour atteindre un meilleur makespan.

Algorithme 7 Algorithme de Palmer

Entrées : Ensemble J de n jobs
Sorties : Séquence π

Début
1 : | **for** $i = 1$ to n **do**
2 : | | **Calculer** $S_i = \sum_{j=1}^{m} (m - 2 * j + 1) * P_{ij}$;
3 : | **end for**
4 : | **Classer** les jobs dans l'ordre décroissant de S_i
5 : | **Retourner** la sequence obtenue π
Fin

5.2.3 Évaluation de la solution

Dans cette étape, on associe à chaque individu une fonction d'évaluation (fitness) pour calculer sa force d'adaptation. La fonction fitness est basé sur l'objectif à optimiser et attribue à chaque individu de la population une valeur reflétant leur infériorité ou supériorité (déterminer si l'individu peut survivre et produire des enfants, ou mourir). Pour un problème de maximisation, la mesure de la performance constitue généralement la fonction objectif. Cependant, pour les problèmes de minimisation, la méthode pour déterminer la fonction fitness diffère des problèmes de maximisation.

Dans notre cas, le problème a pour but de minimiser le makespan, alors la force de l'individu i peut être exprimé tout simplement par l'équation 5.4.

$$F(i) = 1/f(i) \tag{5.4}$$

Où i est le $i$ème individu dans la population, $f(i)$ est la valeur de makespan pour l'individu i et $F(i)$ est sa fonction fitness.

5.2.4 Opérateur de sélection

Après avoir généré la population initiale, la sélection vise à identifier les meilleurs individus de la population pour la reproduction, en fonction de leur fitness. Cependant, ce n'est pas parce qu'un individu est bon qu'il survive nécessairement et de même, ce n'est pas parce qu'il est mauvais qu'il doit disparaître. En effet, bien, souvent, une bonne génération peut descendre d'un individu décrété mauvais. Pour la mise au point de notre algorithme génétique, nous avons recours à la procédure de sélection par roulette définie dans le chapitre 2, qui est sans doute la procédure la plus connue et utilisée. Cette procédure, basée sur le principe de la roue de loterie, favorise les individus forts. Contrairement à d'autres procédures, la sélection choisie maintient la diversité de la population en laissant une chance aux individus faibles pour être sélectionnés.

Le principe de cette procédure consiste à associer à chaque individu un secteur de la roue dont l'angle est proportionnel à la valeur de sa fonction fitness $F(i)$. Si la population des individus est de la taille égale à P_{size}, alors la probabilité de sélection $P(i)$ d'un individu i est :

$$P(i) = F(i)/\sum_{i=1}^{P_{size}} F(i) \tag{5.5}$$

Les différentes étapes de cette procédure sont données par l'algorithme 15.

5.2.5 Opérateur de croisement

Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Un croisement fusionne l'information génétique de deux individus appelés « parents » et génère deux nouveaux individus appelés « enfants ». Les individus survivants à la phase de sélection vont subir le croisement avec une probabilité P_c, le choix de la méthode de croisement dépend de la technique de codage utilisée (caractéristiques du problème).

Dans notre algorithme génétique, deux méthodes de croisement ont été testées : croisement à un point (one-point crossover (OPX)) et croisement à deux points (two-point crossover (TPX)). Ces deux méthodes ont été choisies de manière à garantir la faisabilité et la légalité de toutes les solutions codées avec la représentation par permutation.

Les différentes étapes de ces deux procédures sont données respectivement par les algorithmes 9 et 10.

Algorithme 8 Sélection par roulette

Entrées : Population ; P_{size} taille de la population ; $f(i)$ makespan de l'individu i
Sorties : Individus sélectionnés pour la reproduction.

Début
for $i = 1$ jusqu'à P_{size} **do**
  Calculer la fonction fitness $F(i)$ en utilisant l'équation 5.5
end for
for $i = 1$ jusqu'à P_{size} **do**
  Calculer la probabilité de sélection $P(i)$ en utilisant l'équation 5.5
end for
for $i = 1$ jusqu'à P_{size} **do**
  if $i == 1$ **then**
    $CumProb(i) = P(i)$ ▷ $CumProb(i)$ probabilité cumulée du $i$ème individu
  else
    $CumProb(i) = CumProb(i-1) + P(i)$
  end if
end for
for $i = 1$ jusqu'à P_{size} **do**
  Générer aléatoirement une probabilité x où $x \in [0, \sum_{i=1}^{P_{size}} CumProb(i)]$
  if $CumProb(i) < x < CumProb(i+1)$ **then**
    Choisir l'individu $(i+1)$
  end if
end for
Retourner l'ensemble des individus sélectionnés pour la reproduction
Fin

Example :
Considérons un problème à sept jobs. Un exemple de croisement à un point est représenté par la figure 5.2.

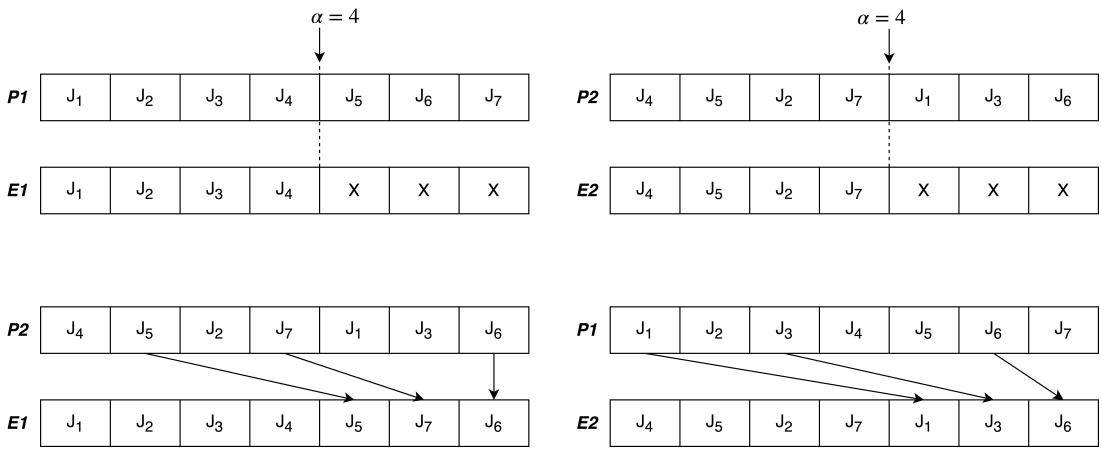


FIGURE 5.2 : Opérateur de croisement à un point

Algorithme 9 Algorithme de croisement à un point (OPX)

Entrées : $P1$, $P2$ individus parents ; n longueur d'individu ; α entiers $\in [1, n]$; P_c probabilité de croisement
Sorties : $E1$, $E2$ individus enfants.

Début
1 : **Choisir** deux parents $P1$ et $P2$ pour générer deux enfants $E1$ et $E2$
2 : **Générer** un nombre x aléatoirement $x \in [0, 1]$
3 : **if** $x \leq P_c$ **then**
4 : **Générer** un entier aléatoire α en tant que « point de croisement » (ce point de croisement ou coupure est placé aux mêmes endroits pour les deux parents)
5 : **Copier** tous les jobs qui se situe d'un côté du point de coupure α des parents choisis sur les deux enfants respectivement, ici, nous copions ceux du parent $P1$ à l'enfant $E1$, ceux du parent $P2$ à l'enfant $E2$.
6 : **Compléter** la partie restante de l'enfant $E1$ par les éléments du parent $P2$ et la partie restante de l'enfant $E2$ par les éléments du parent $P1$ en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis
7 : **end if**
8 : **Mettre à jour** les individus $E1$ et $E2$
Fin

Example :
Considérons le même exemple que précédemment, mais dans ce cas il y a deux points de croisement α et β. Cet opérateur est illustré par la figure 5.3

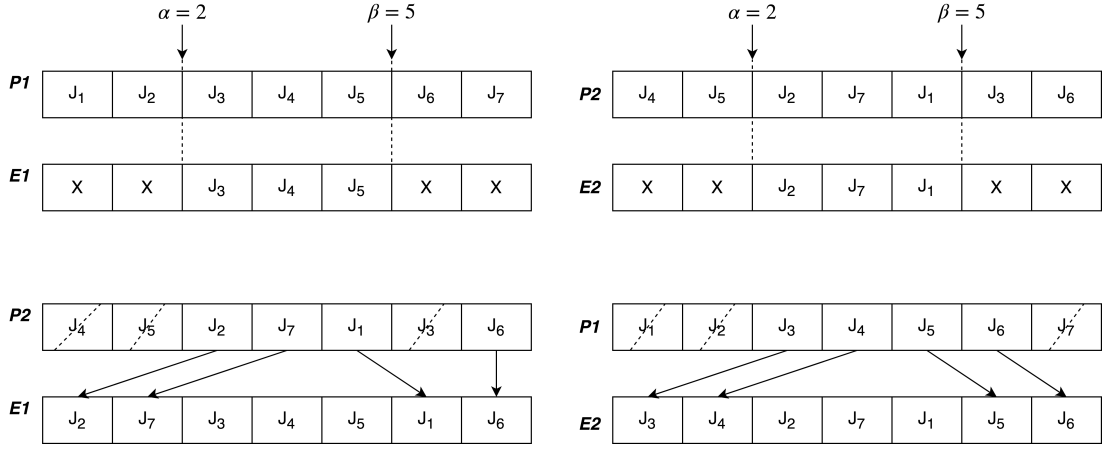


FIGURE 5.3 : Opérateur de croisement à deux point

5.2.6 Opérateur de mutation

Après application de l'opérateur de croisement, une fraction d'individus est sélectionnée avec une probabilité P_m pour subir la mutation. La mutation est le processus de changement aléatoire des valeurs des gènes dans les chromosomes, l'objectif visé par l'application de cet opérateur

Algorithme 10 Algorithme de croisement à deux point (TPX)

Entrées : $P1$, $P2$ individus parents ; n longueur d'individu ; α, β entiers $\in [1, n]$; P_c probabilité de croisement
Sorties : $E1$, $E2$ individus enfants.

Début
1 : **Choisir** deux parents $P1$ et $P2$ pour générer deux enfants $E1$ et $E2$
2 : **Générer** un nombre x aléatoirement $x \in [0, 1]$
3 : **if** $x \leq P_c$ **then**
4 : **Générer** aléatoirement deux points de croisements α, $\beta \in [2,..., n-1]$ avec $\alpha < \beta$
5 : **Copiez** le segment entre les deux points sélectionnés α et β de parent $P1$ à l'enfant $E1$ et de parent $P2$ à l'enfant $E2$
6 : **Supprimez** les jobs de Parent $P2$ inclus dans le segment hérité du Parent $P1$. Copiez les jobs restants dans la position disponible de l'enfant $E1$ (de la même manière on construit l'enfant $E2$)
7 : **end if**
8 : **Mettre à jour** les individus $E1$ et $E2$
Fin

est de maintenir la diversité des chromosomes et introduire une variabilité supplémentaire dans la population, et cela, afin de garantir que l'algorithme génétique sera susceptible d'atteindre la plupart des points du domaine réalisable et éviter une convergence prématurée vers un minimum local.

Nous avons vu au chapitre 2, qu'il existe plusieurs méthodes de mutation, mais le choix de cet opérateur dépend du problème étudier, autrement dit du codage utilisé. Les méthodes les plus courantes pour la représentation par permutation sont la mutation par échange et la mutation par insertion. Dans cette partie, ces deux méthodes de mutation ont été testées.

(i) La mutation par échange (*swap mutation*) consiste à interchanger deux gènes du même chromosome, la procédure de cette mutation est définit par l'algorithme 11 et représentée par la figure 5.4.

(ii) La mutation par insertion (*shift mutation*) consiste à choisir aléatoirement un gène et à l'insérer dans une position choisie au hasard dans le chromosome à muter, comme décrit dans l'algorithme 12 et la figure 5.5.

Example :
La figure 5.4 démontre un exemple de cette procédure de mutation, où les jobs placés aux deuxième et cinquième positions sont échangés (J_7 et J_5).

Algorithme 11 Algorithme de mutation par échange

Entrées : i individu ; n longueur d'individu ; α, β entiers $\in [1, n]$; P_m probabilité de mutation
Sorties : $i\prime$ individu muté

Début
1 : **Sélectionner** aléatoirement un individu i
2 : **Générer** un nombre γ aléatoirement $\gamma \in [0, 1]$
3 : **if** $\gamma \leq P_m$ **then**
4 : **Choisir** aléatoirement deux positions α et β ($\alpha \neq \beta$), correspondant respectivement aux jobs J_k et $J_k\prime$ de l'individu i
5 : **Permuter** les jobs J_i et J_k
6 : **end if**
7 : **Mettre à jour** l'individu muté $i\prime$
Fin

Algorithme 12 Algorithme de mutation par insertion

Entrées : i individu ; n longueur d'individu ; α entiers $\in [1, n]$; P_m probabilité de mutation
Sorties : $i\prime$ individu muté

Début
1 : **Sélectionner** aléatoirement un individu i
2 : **Générer** un nombre γ aléatoirement $\gamma \in [0, 1]$
3 : **if** $\gamma \leq P_m$ **then**
4 : **Choisir** aléatoirement un job J_k et une position α de l'individu i
5 : **Insérer** le job J_k à la position α
6 : **Décaler** à droite les autres jobs.
7 : **end if**
8 : **Mettre à jour** l'individu muté $i\prime$
Fin

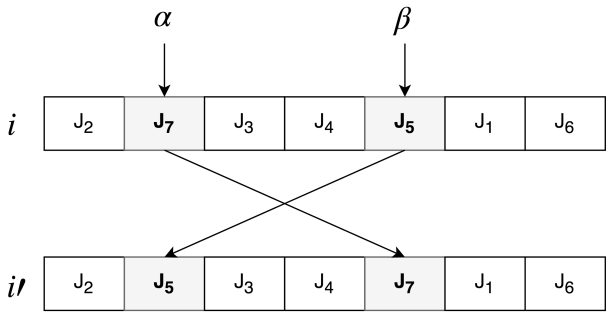


FIGURE 5.4 : Mutation par échange

Example :
L'opérateur de mutation par insertion est illustré à la figure 5.5, où le gène sélectionné est cerclé et la nouvelle position est indiquée par une flèche.

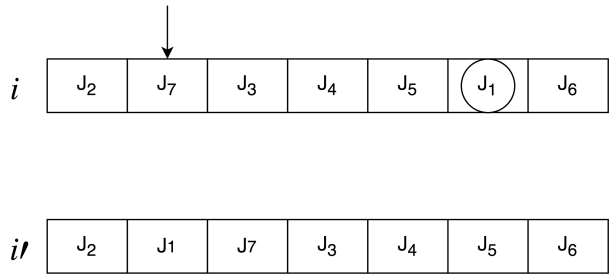


FIGURE 5.5 : Mutation par insertion

5.2.7 La stratégie de remplacement

Après la reproduction, nous devons décider quels individus seront survécu et transférés à la génération suivante. Le processus de formation de la prochaine génération d'individus en remplaçant ou en retirant des enfants ou des parents est effectué par l'opérateur de remplacement. Le remplacement aide à trouver les individus qui remplaceraient la génération actuelle pour former la génération suivante.

Dans la stratégie de remplacement utilisée, l'ensemble des parents et des enfants sont regroupés en une unité (taille de cette unité égale à $2 \times P_{size}$). Les parents et les enfants rivalisent pour survivre. Ils sont triés selon leur fonction fitness. Ensuite, les meilleurs chromosomes équivalents à la taille de la population sont choisis pour former la génération suivante.

Algorithme 13 Algorithme de Remplacement

Entrées : P_{size} taille de population ; P populations parentes ; E population enfants ; makespan pour chaque individu.
Sorties : *NewPop* nouvelle population

Début
1 : $Pool \leftarrow$ P + E
2 : **for** i dans *Pool* **do**
3 : **Calculer** la fonction fitness $F(i)$ en utilisant l'équation 5.5
4 : **end for**
5 : **Trier** *Pool* dans un ordre croissant selon la fonction fitness
6 : **Stocker** les P_{size} meilleurs individus dans *NewPop*
7 : P $\leftarrow$ *NewPop*
8 : **Mise à jour** de P
Fin

5.2.8 Mécanisme d'arrêt

Pour mettre fin à l'exécution de l'algorithme génétique, un critère d'arrêt est nécessaire. Comme il n'y a pas de règle praticable pour définir une condition d'arrêt appropriée, la méthode habituelle consiste à définir un nombre maximal de générations.

Dans cette étude, le critère d'arrêt que nous avons utilisé est conditionné par un nombre maximal de générations, fixé en fonction de la taille du problème.

5.3 Recherche locale

Plusieurs travaux sur les problèmes d'ordonnancements ont montré que l'incorporation de la recherche locale améliore les performances des métaheuristiques. Motivés par cette observation, nous proposons de renforcer notre approche décrite dans la section précédente en l'hybridant avec une procédure de recherche locale.

Il est bien connu que l'algorithme génétique est une méthode d'optimisation globale, mais qu'elle ne dispose pas d'une bonne performance de la recherche locale. L'idée d'améliorer les performances de l'algorithme génétique en l'hybridant avec une méthode heuristique a été fréquemment étudiée pour résoudre les problèmes Flow-Shop (cf. [23, 27, 132, 139**?**]).

Dans cette thèse, pour renforcer les performances de notre l'algorithme génétique, nous proposons de l'hybrider avec une procédure de recherche locale. La recherche locale peut être résumée comme étant une procédure de recherche itérative qui, à partir d'une solution réalisable, l'améliore progressivement en appliquant une série mouvements.

La recherche locale que nous proposons correspond à la technique d'insertion proposée par Nawaz et al. [97]. Plus précisément, cette recherche locale est considéré comme une fonction qui prend comme entré une séquence aléatoire de jobs (au lieu d'une séquence de jobs triée par ordre décroissant de la somme de temps de traitement). Ensuite, les étapes de l'algorithme NEH sont simplement suivies. Si la nouvelle séquence résultante conduit à une meilleure fonction objectif, la solution actuelle est remplacée par la nouvelle séquence.

Le pseudo-code de cette procédure est décrit par l'algorithme 14, où :

- ρ est une séquence partielle.
- $\sigma(i)$ est le i^{me} élément dans la séquence σ.
- $card(\rho)$ indique le nombre de jobs dans la séquence ρ
- $\rho(r,k)$ est la séquence partielle résultante dans laquelle le job r est inséré dans la k^{me} position de la séquence ρ.
- k^* est la position pour laquelle le minimum makespan est atteint.
- $E(\sigma)$ est la fonction objectif de la séquence σ

Cette recherche locale avec le voisinage insertion (*insertion neighborhood*), a été considéré comme

Algorithme 14 Recherche locale

Entrées : Séquence σ de n jobs

Début

1 : **set** $\rho = (\sigma(1))$
2 : **set** $\kappa = card(\rho) = 1$
3 : **let** $r = \sigma(\kappa + 1)$
4 : **for** $k = 1$ to $\kappa + 1$ **do**
5 : **Calculer** $E(\rho(r,k))$ pour trouver la séquence $\rho(r,k^*)$ qui minimise le makespan
6 : **set** $\rho = \rho(r,k^*)$
7 : **set** $\kappa = \kappa + 1$
8 : **if** $\kappa < n$ **then**
9 : **Retourner** à l'étape 3
10 : **else**
11 : ρ est la séquence finale
12 : **end if**
13 : **end for**
14 : **if** $E(\rho) < E(\sigma)$ **then**
15 : $\sigma \longleftarrow \rho$
16 : **end if**

Fin

adéquate pour les problèmes Flow-Shop de permutation [116].

Dans ce travail, nous étudions l'influence de l'hybridation de l'AG avec la recherche locale proposée. Notre algorithme utilise l'hybridation qui se produit avec deux procédures différentes :

(1) PsyLS (Partiel systematic local search) : cette procédure se produit pour différents pourcentages de générations et pour chaque pourcentage, la recherche locale peut être appliqué de trois manières différentes : aux générations initiales, au milieu et à celles de la fin. Puisque les métaheuristique arrêtent leur diversification dans les dernières itérations, nous appliquons la recherche locale systématiquement au meilleur individu pour 30, 50, 80 et 100 pourcents des dernières itérations de l'algorithme proposé.

(2) RanLS (*Random local search*) : pour chaque meilleur individu de chaque génération, nous effectuons la recherche locale mentionnée ci-dessus avec une probabilité prédéfinie P_{ls}. Contrairement à PsyLS, RanLS génère un nombre aléatoire α entre $[0,1]$, si $\alpha < P_{ls}$ alors la recherche locale est appliquée au meilleur individu de chaque génération, mais pas à tous les individus de la population.

Dans cette procédure, trois niveaux de P_{ls} sont testés : faible, moyen et élevé, avec une probabilité de 0.1, 0.5 et 0.8 respectivement.

5.4 Paramétrage de l'algorithme génétique proposé

Dans les deux sections précédentes, nous avons défini les différentes phases de l'algorithme génétique proposé. Cependant, la difficulté majeure de ce dernier ne réside pas uniquement dans la mise en œuvre de l'algorithme lui-même, mais plutôt, dans le choix des valeurs adéquates des différents paramètres de cet algorithme à savoir : la taille de la population, la probabilité de croisement et de mutation et les techniques à utiliser pour les réaliser.

5.4.1 Taille de population

Taille de population P_s diffère d'un algorithme génétique à un autre et d'un problème à un autre, i.e., qu'il n'y a pas de standardisation quant au choix de la taille de population. En générale, une grande population peut abaisser exagérément le taux de convergence et altérer les performances de l'algorithme (l'augmentation du temps de résolution). En revanche, une population de petite taille ne permet pas une bonne exploration de l'espace de recherche et par conséquent, elle peut conduire à une convergence prématurée de la population vers un optimum local. Donc, la taille de la population doit donc être choisie de façon à trouver un bon compromis entre le temps de calcul et la qualité de solution.

5.4.2 Probabilité de croisement

L'opérateur de croisement a pour but d'enrichir la diversité de la population. Cet opérateur est appliqué avec une certaine probabilité appelée taux ou probabilité de croisement notée P_c (typiquement proche de l'unité). Plus cette probabilité est élevée plus il y aura de nouvelles structures dans la population. En revanche, une faible probabilité de croisement appauvrit la population et ralentit l'évolution. C'est pourquoi, la probabilité de croisement est généralement supérieure à 0.6.

5.4.3 Probabilité de mutation

L'opérateur de mutation permet d'étendre l'espace de recherche et réduire la probabilité de converger vers un minimum local. Cet opérateur est appliqué avec une certaine probabilité appelée taux ou probabilité de mutation notée P_m. Cette probabilité est généralement faible puisqu'une probabilité élevée risque de disperser les solutions par génération et dégrader le taux de convergence.

Donc, une conception appropriée de ces paramètres (P_s, P_c, P_m) a un impact significatif sur l'efficacité de l'algorithme génétique. Mais à noter que le choix de ces paramètres et les techniques utilisées pour les réaliser dépend fortement du problème à résoudre. Cependant, dans la plupart des travaux portant sur les problèmes d'ordonnancement Flow-Shop, les auteurs ont ajusté les paramètres des métaheuristiques proposées, et en particulier les algorithmes génétiques, manuel-

lement sur la base des valeurs de référence de la littérature similaire ou bien, ils ont opté pour un paramétrage standard et aucune étude n'a été effectuée afin d'adapter le meilleur paramétrage des algorithmes proposés. Compte tenu de l'importance d'un bon paramétrage, ce dernier fait l'objet de cette section.

Le problème à résoudre consiste à régler a priori les paramètres de l'algorithme génétique proposé pour la résolution du problème Flow-Shop sous contraintes de ressources non-renouvelables. L'objectif à atteindre est de trouver la combinaison de ces paramètres qui donne le meilleur comportement pour notre l'algorithme génétique. Pour ce faire, nous adoptons une approche basée sur les méthodes de plans d'expériences *(en anglais, design of experiments ou DOE)* pour régler au mieux les paramètres de l'algorithme proposé. Avant d'utiliser cette approche expérimentale, les concepts suivants doivent être définis :

— Réponses : c'est la variable qui mesure le phénomène auquel on s'intéresse. Dans notre problème, c'est le fitness de la solution.
— Facteurs : appelée aussi variables d'entrées, ce sont les variables que l'on désire étudier. Dans notre problème, les facteurs sont les paramètres de l'algorithme génétique à régler.
— Niveau d'un facteur : le niveau d'un facteur indique les valeurs que peux prend ce facteur dans le plan d'expériences. Ces valeurs peuvent être quantitatives ou qualitatives.

Donc, un plan d'expérience peut être défini comme une technique qui permet de :

— Quantifier les effets des variables d'entrée (facteurs) sur la variable de sortie (réponse) en un nombre minimal d'expériences pour sélectionner les facteurs qui ont un effet significatif et éventuellement les classer en fonction de leurs effets.
— Déterminer la meilleure combinaison des facteurs d'entrée qui permet d'optimiser la variable de sortie toujours en un nombre minimal d'expériences.

Différentes méthodes de plans expérimentaux existent, chacun permettant de répondre à des problématiques expérimentales différentes. Citons entre autres, les plans complets, les plans factoriels fractionnaires, les plans de Koshal, les plans de Plackett-Burman et les plans de Taguchi.

La méthode utilisée pour réaliser ce travail est celle des plans de Taguchi. Ces plans sont issus des travaux de M. Genichi Taguchi au début des années 1960, pour lequel l'aspect industriel a été pris en compte [105].

La méthode de Taguchi vient pour enrichir les méthodes de plans d'expériences en apportant une amélioration considérable aux plans factoriels complets et fractionnaires. Elle a pour but de simplifier le protocole expérimental afin de mettre en évidence les effets de facteurs sur la réponse. Cette méthode se distingue par une réduction importante du nombre d'essais, tout en gardant une bonne précision. L'expérimentateur dans la méthode de Taguchi choisit librement les facteurs et les interactions à étudier. La spécificité et la puissance de cette méthode résident dans l'utilisation d'un outil de mesure de performance, dénommé rapport Signal sur Bruit (symbolisé par : S/N).

La méthode de Taguchi a connu, dans un premier temps, un succès dans les secteurs industriels et, en particulier, dans le domaine agroalimentaire, puis elle a suscité l'intérêt de la communauté de statisticiens pour un développement et une étude assez larges [118].

Pour de plus amples détails sur cette méthode, nous référons le lecteur au livre de Ranjit Roy [115].

5.4.4 Application de la méthode de Taguchi

La mise en œuvre de la méthode de Taguchi nécessite le suivi d'une démarche basée sur les septs étapes suivantes :

— Étape 1 : identifier le problème et les objectifs à atteindre.
— Étape 2 : identifier les facteurs qui affectent la réponse.
— Étape 3 : déterminer les niveaux de chaque facteur.
— Étape 4 : sélectionner une table orthogonale (table de Taguchi) appropriée.
— Étape 5 : construire le plan en fonction de la table de Taguchi.
— Étape 6 : réaliser les expériences.
— Étape 7 : analyser les résultats.

Comme nous avons cité précédemment, le problème à résoudre consiste à régler a priori les paramètres de l'algorithme génétique proposé pour la résolution du problème Flow-Shop sous contraintes de ressources non-renouvelables. L'objectif à atteindre est donc de trouver la combinaison de ces paramètres qui donne le meilleur comportement pour notre algorithme génétique.

Dans cette étude, nous avons choisi parmi l'ensemble des facteurs d'algorithme génétique à ajuster, six principaux facteurs couramment étudiés qui sont :

(A) La méthode dont la population initiale est générée (population initiale).

(B) La technique de croisement.

(C) La technique de mutation.

(D) La taille de la population.

(E) La probabilité de croisement.

(F) La probabilité de mutation.

Après une analyse préliminaire approfondie de l'algorithme, les facteurs sélectionnés et leurs niveaux sont présentés dans le tableau 5.1. Ces derniers ont été choisis à partir de données issues d'articles publiés dans des revues spécialisées portant sur les problèmes Flow-Shop et d'ouvrages généraux sur les algorithmes évolutionnistes.

En fonction du nombre de paramètres pris en compte et le nombre de niveaux de paramètres, un plan factoriel complet nécessite 216 ($2^3 \times 3^3$) expériences. En outre, le nombre d'expériences sera répété cinq fois afin de tenir compte du comportement stochastique de l'algorithme génétique. Par conséquent, le nombre d'expériences requis pour un problème sera 1080 (216×5) expériences.

TABLE 5.1 : Facteurs et leurs niveaux

	Facteurs					
	Population initiale	Croisement	Mutation	P_s	P_c	P_m
Codage	A	B	C	D	E	F
Niveau 1	Rand	OPX	Swap	100	0.8	0.1
Niveau 2	Randheur	TPX	Shift	150	0.9	0.15
Niveau 3				200	1	0.2

La réalisation de toutes ces expériences nécessite un temps de calcul considérable. Pour cette raison, nous avons choisi la technique de Taguchi afin de réduire le nombre d'expériences.

Dans notre problème, la taille de la population, la probabilité de croisement et la probabilité de mutation ont trois niveaux, mais les autres facteurs tels que la méthode dont la population initiale est générée, la technique de croisement et de mutation ont deux niveaux. Donc, à partir des tables de Taguchi, la table L36 est sélectionnée comme la table ou bien la matrice d'expériences la plus adaptée à nos exigences.

Après avoir exécuté l'algorithme génétique sur des instances de tailles différentes en fonction de la matrice d'expériences sélectionnée, les réponses sont individuellement transformées en rapport S/B. Comme notre fonction objectif vise à minimiser le makespan, alors le rapport S/N est calculé à l'aide de la formule d'optimum est un minimum « *Lower-is-Better* », décrite ci-dessous :

$$\mathrm{S/N} = -10.\log\left[1/n.\sum_{i=1}^{n} y_i^2\right] \tag{5.6}$$

Où n représente le nombre de répétitions de l'expérience et y représente la valeur de la réponse obtenue à la i^{me} répétition.

Ces expériences sont testées en utilisant le logiciel statistique Minitab 17, la table de réponse et le graphique produits sont illustrés dans le tableau 5.2 et la figure 5.6.

Comme l'illustre le tableau de réponse 5.2, le facteur A (population initiale) joue un rôle important dans le processus d'exécution de l'algorithme génétique proposé. En outre, l'influence des six facteurs sur la minimisation du makespan pour le problème Flow-Shop sous contraintes de ressources non-renouvelables est, de l'ordre de : population initiale, technique de mutation, technique de croisement, taille de la population, probabilité de croisement et probabilité de mutation.

De plus, selon la figure 5.6, le meilleur niveau de chaque facteur est celui avec le rapport S/N élevé. Par conséquent, la meilleure combinaison de facteurs est : A2 B2 C2 D3 E2 F3 (A2 : population initiale = RandHeur, B2 : technique de croisement = PTX, C2 : technique de mutation = shift mutation, D3 : taille de la population = 200, E2 : probabilité de mutation = 0.9 et F3 : probabilité de mutation = 0.15).

TABLE 5.2 : Tableau de réponses pour les rapports S/N

Niveau	Facteur					
	A	B	C	D	E	F
1	-67,29	-67,09	-67,1	-67,09	-67,08	-67,08
2	-66,93	-67,06	-67,05	-67,07	-67,07	-67,07
3				-67,07	-67,08	-67,07
Delta	0,36	0,02	0,05	0,02	0,01	0,01
Rang	1	3	2	4	5	6

La figure 5.6 montre aussi que le facteur A a l'effet le plus important, c'est-à-dire que ce facteur a un impact significatif sur le rapport S/N. Nous pouvons donc affirmer que le fait de démarrer notre algorithme génétique avec des solutions satisfaisantes conduit à des meilleures résultats.

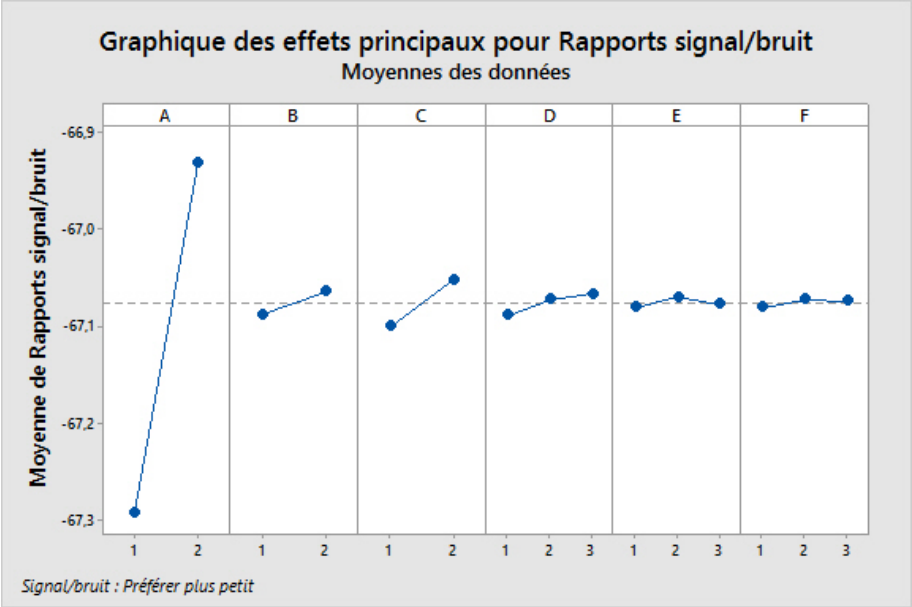


FIGURE 5.6 : Graphe de réponses pour les rapports S/N

5.5 Résultats expérimentaux

Afin de tester les performances de l'algorithme génétique hybride développé pour le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables, nous effectuons des simulation lorsque la disponibilité de ressources non-renouvelables est représentée par une arrivée uniforme croissante (UAF1), uniforme décroissante (UAF2), uniforme aléatoire (UAF3) et non-uniforme (NUA).

Pour chaque taille de problème sous différentes configurations, cinq essais indépendants sont effectués et les résultats sont présentés séparément pour l'AG avec recherche locale aléatoire

TABLE 5.3 : Paramètres de l'AG utilisés dans la simulation

Paramètre	Valeur
Taille de la population	200
Population initiale	RandHeur
Méthode de croisement	Croisement à deux points
Méthode de mutation	Mutation par insertion
Taux de croisement	0.9
Taux de mutation	0.15

(AG_RanLS) et l'AG avec recherche locale partielle systématique (AG_PsyLS). Pour les deux recherches locales, les résultats sont résumés respectivement dans les tableaux 5.4 et 5.5 pour les problèmes de petite taille et les tableaux 5.6 et 5.7 pour les problèmes de moyenne à grande taille.

L'algorithme génétique proposé a été tourné en utilisant les paramètres résultant de la méthode de Taguchi (cf. section 5.4), ces paramètres sont résumés dans le tableau 5.3. Tous les algorithmes proposés dans cette thèse sont codés en Langage MATLAB et exécutés sur un PC avec CPU CORE i5 1.9 GHz.

5.5.1 Résultats de calcul pour les problèmes de petites tailles

Les résultats des tableaux 5.4 et 5.5 montrent le comportement des algorithmes au moment de la résolution des problèmes de petites tailles sur les diffiérents configurations proposés. Dans ces tableaux, les $\overline{ARE}$ représentent l'erreur relative moyenne par rapport à la solution optimale trouvée par CPLEX, calculées comme :

$$\overline{ARE}\% = \frac{C_{max}^{mthode} - C_{max}^{CPLEX}}{C_{max}^{CPLEX}} \times 100 \tag{17}$$

où C_{max}^{mthode} représente la solution générée par la méthode considérée .

Premièrement, nous démontrons l'efficacité de la recherche locale en comparant l'AG hybride (AG_PsyLS et AG_RanLS) avec l'AG sans recherche locale (AG_NoLS).

Le tableau 5.4 illustre l'effet de P_{ls} sur la qualité de la solution et le temps de calcul. P_{ls} détermine la probabilité d'utilisation de la recherche locale proposée pour le meilleur individu de chaque génération.

À partir du tableau 5.4, nous constatons que les performances obtenues par AG lorsque P_{ls} (qui signifie qu'aucune recherche locale n'est appliquée) sont pires que ceux obtenus lorsque $P_{ls} > 0$.

Les résultats montrent que lorsque P_{ls} augmente, le temps de calcul moyen (c'est-à-dire, $\overline{CPU}$) augmente aussi, tandis que la qualité de la solution varie avec une petite magnitude en particulier lorsque nous comparons AG_NoLS avec AG_RanLS (P_{ls}=0.1). C'est-à-dire que AG_RanLS avec

une probabilité de 0.1 n'affecte pas trop la qualité de solution d'AG.

En même temps, nous pouvons observer du tableau 5.4 que les valeurs $\overline{ARE}$ résultant de l'AG avec la recherche locale aléatoire (AG_RanLS) pour les problèmes (10×5) et (10×10) sont proches de zéro lorsque la disponibilité de ressources est une fonction uniforme. Cependant, avec l'augmentation du nombre de jobs et le nombre de machines, on peut constater que l'erreur relative obtenue par l'AG hybride dans le cas ou la disponibilité de ressources non-renouvelables est une fonction uniforme décroissante est meilleure que les autres configurations considérées.

TABLE 5.4 : Comparaison de résultats - problèmes de petites tailles : AG_RanLS

Problème	Méthode	Configuration							
		UAF1		UAF2		UAF3		NUA	
		$\overline{ARE}$	$\overline{CPU}$	$\overline{ARE}$	$\overline{CPU}$	$\overline{ARE}$	$\overline{CPU}$	$\overline{ARE}$	$\overline{CPU}$
10 x 5	AG_NoLS	1.26	1.91	1.35	1.90	**0.00**	2.04	0.51	1.91
	AG_RanLS ($P_{ls} = 0.1$)	0.45	2.03	1.35	1.92	**0.00**	2.05	0.51	1.97
	AG_RanLS ($P_{ls} = 0.5$)	0.23	2.09	1.20	1.99	**0.00**	2.07	0.20	1.98
	AG_RanLS ($P_{ls} = 0.8$)	0.23	2.14	0.43	2.02	**0.00**	2.11	0.04	2.01
10 x 10	AG_NoLS	0.96	2.79	0.87	2.80	0.98	2.78	0.78	2.73
	AG_RanLS ($P_{ls} = 0.1$)	0.93	2.82	0.85	2.83	0.09	2.84	0.75	2.77
	AG_RanLS ($P_{ls} = 0.5$)	0.79	2.90	0.59	3.02	0.09	2.95	0.19	2.82
	AG_RanLS ($P_{ls} = 0.8$)	0.79	2.99	0.54	3.05	**0.00**	3.01	**0.00**	2.95
10 x 20	AG_NoLS	0.54	4.84	0.17	4.72	0.76	4.88	0.75	4.95
	AG_RanLS ($P_{ls} = 0.1$)	0.16	4.90	0.14	4.77	0.62	4.94	0.75	4.98
	AG_RanLS ($P_{ls} = 0.5$)	0.16	5.18	0.02	5.02	0.46	5.20	0.52	5.03
	AG_RanLS ($P_{ls} = 0.8$)	0.15	5.28	**0.00**	5.11	0.39	5.37	0.35	5.18
20 x 5	AG_NoLS	2.44	5.58	2.19	5.41	3.75	5.28	2.45	5.42
	AG_RanLS ($P_{ls} = 0.1$)	1.76	5.92	1.45	5.71	3.19	5.46	1.53	5.76
	AG_RanLS ($P_{ls} = 0.5$)	1.44	6.39	1.10	6.37	2.59	5.98	0.91	6.28
	AG_RanLS ($P_{ls} = 0.8$)	1.25	6.71	0.75	6.46	2.20	6.38	0.79	6.68

Le tableau 5.5 montre que les valeurs $\overline{ARE}$ obtenues par l'AG avec différents pourcentages de recherche locale (AG_PsyLS) sont bien meilleures que celles obtenues par l'AG sans recherche locale (AG_NoLS), ce qui démontre l'efficacité de l'incorporation de la recherche locale dans AG proposé. De plus, nous observons que la qualité de la solution s'améliore considérablement à mesure que nous augmentons le pourcentage d'hybridation, par exemple, dans le cas ou la disponibilité de ressources non-renouvelables est une fonction uniforme aléatoire (UAF3) et pour le problème de taille (10 × 10), la qualité de la solution s'est améliorée de 0.98 % sans hybridation (AG_NoLS) à 0.47% avec 10% d'hybridation et à la valeur optimale avec 100% d'hybridation.

Puisque la recherche locale est appliquée, le temps de calcul de AG_PsyLS est supérieur à celui de AG_NoLS.

Un comportement similaire à AG_RanLS peut être observé pour l'impact de la disponibilité des ressources sur les performances de AG_PsyLS. On peut dire que les meilleurs résultats obtenus par AG_PsyLS sont en moyenne très proches de l'optimum lorsque la disponibilité des ressources est une fonction uniforme. Cependant, pour plus de 10 machines et 10 jobs, AG_PsyLS a une

performance moyenne supérieure dans le cas ou la disponibilité de ressources non-renouvelables est une fonction uniforme décroissante.

À partir des tableaux 5.4 et 5.5, nous observons une régularité dans les performances de AG_RanLS et AG_PsyLS en particulier lorsque $P_{ls} \geq 0.5$ (pourcentage ≥ 50).

TABLE 5.5 : Comparaison de résultats - problèmes de petites tailles : AG_PsyLS

Problème	Méthode	Configuration							
		UAF1		UAF2		UAF3		NUA	
		ARE	CPU	ARE	CPU	ARE	CPU	ARE	CPU
10 x 5	AG_NoLS	1.26	1.91	1.35	1.90	**0.00**	2.07	0.51	1.91
	AG_PsyLS (10%)	0.45	1.92	0.43	1.91	**0.00**	2.13	0.25	1.93
	AG_PsyLS (50%)	0.36	1.99	0.33	1.96	**0.00**	2.13	0.16	1.99
	AG_PsyLS (80%)	0.23	2.07	0.33	2.02	**0.00**	2.19	0.00	2.00
	AG_PsyLS (100%)	0.23	2.23	0.33	2.12	**0.00**	2.23	**0.00**	2.13
10 x 10	AG_NoLS	0.96	2.82	0.87	2.71	0.98	2.74	0.78	2.67
	AG_PsyLS (10%)	0.96	2.83	0.63	2.74	0.47	2.84	0.76	2.73
	AG_PsyLS (50%)	0.79	2.87	0.63	2.80	0.43	2.85	0.75	2.83
	AG_PsyLS (80%)	0.65	2.94	0.57	2.89	0.35	2.95	0.59	2.85
	AG_PsyLS (100%)	0.19	3.01	0.44	2.99	**0.00**	3.08	**0.00**	3.04
10 x 20	AG_NoLS	0.54	4.84	0.17	4.72	0.76	4.88	0.75	4.95
	AG_PsyLS (10%)	0.47	4.94	0.17	4.76	0.49	4.92	0.72	5.04
	AG_PsyLS (50%)	0.16	5.06	0.12	4.76	0.43	4.97	0.58	5.21
	AG_PsyLS (80%)	0.16	5.12	0.12	4.86	0.25	4.99	0.47	5.25
	AG_PsyLS (100%)	0.15	5.34	**0.00**	5.00	0.17	5.19	0.18	5.66
20 x 5	AG_NoLS	2.44	5.58	2.19	5.41	3.75	5.28	2.45	5.42
	AG_PsyLS (10%)	1.94	5.74	1.87	5.89	2.65	5.38	2.01	5.52
	AG_PsyLS (50%)	1.70	5.86	1.28	6.04	2.35	5.66	1.61	5.86
	AG_PsyLS (80%)	1.51	6.34	0.75	6.46	1.99	6.27	1.25	6.29
	AG_PsyLS (100%)	0.87	7.03	0.75	6.61	1.32	6.75	0.71	7.07

5.5.2 Résultats de calcul pour les problèmes de moyennes à grandes tailles

Après avoir étudié l'impact de la disponibilité de ressources non-renouvelables sur les performances de l'AG hybride (AG_RanLS et AG_PsyLS) pour les instances de petites tailles, nous avons l'intention dans cette section de comparer l'AG proposé et son hybridation avec un ensemble d'instances de moyennes à grandes tailles.

Puisque que le problème que nous considérons dans cette thèse n'a pas été abordée dans la littérature, nous avons rencontré une difficulté pour trouver des instances numériques ou bien même des résultats de méthodes visant à résoudre le problème d'atelier Flow-Shop sous contraintes de ressources non-renouvelables avec minimisation du makespan.

Ainsi, pour montrer l'efficacité de l'algorithme proposé et l'impact de la disponibilité de ressources sur les performances de l'AG hybride, nous avons décidé de comparer les résultats obtenus à l'heuristique NEH tout en respectant les contraintes de ressources non-renouvelables.

Dans les tableaux 5.6 et 5.7, le $\overline{ARI}$ dénote l'amélioration moyenne de chaque méthode par rapport aux résultats de l'heuristique NEH, calculée comme suit :

$$\overline{ARI}\% = \frac{C_{max}^{NEH} - C_{max}^{mthode}}{C_{max}^{NEH}} \times 100 \tag{18}$$

Où C_{max}^{NEH} représente la solution générée par NEH.

Le tableau 5.6 représente les résultats obtenus par l'AG avec recherche locale aléatoire (AG_RanLS) sur différentes configurations de disponibilité de ressources. À partir de ce tableau, les améliorations par rapport à l'heuristique NEH (c'est-à-dire, les valeurs d'ARI) montrent qu'une augmentation de la probabilité de recherche locale conduit, comme prévu, à une amélioration des résultats. Cependant, dans la plupart des cas, ces améliorations ne sont pas très importantes et coûtent beaucoup en temps de calcul. Par exemple, pour le problème (50 × 20) et lorsque la disponibilité de ressources est une fonction non-uniforme (NUA), l'amélioration de AG_RanLS avec $P_{ls} = 0.1$ à AG_RanLS avec $P_{ls} = 0.5$ est d'environ , cependant, le temps moyen de calcul augmente de 68. 94 à 111.56 secondes.

D'autre part, les résultats de ce tableau indiquent que la qualité de solutions obtenues par AG_RanLS est au moyen supérieure dans la majorité des cas où la disponibilité de ressources non-renouvelables est une fonction non-uniforme.

Le tableau 5.7 contient les résultats obtenus par l'AG ayant appliqué une recherche locale systématique partielle (AG_PsyLS) sur différentes configurations de disponibilité de ressources. Dans ce tableau, nous constatons que AG_PsyLS peut considérablement améliorer le résultat de NEH en augmentant le pourcentage de la recherche locale. Ceci démontre l'efficacité d'utilisation de la recherche locale partielle systématique.

Le meilleur résultat est obtenu lorsque le pourcentage de la recherche locale est égal à 100 % avec un taux d'amélioration allant jusqu'à 6.89 % par rapport à l'heuristique NEH. En outre, nous observons également que AG_PsyLS nécessite moins de temps de calcul en comparant avec AG_RanLS. Cela peut résulter du fait que AG_RanLS applique la recherche locale de manière aléatoire, contrairement à AG_PsyLS.

Selon les résultats présentés dans le tableau 5.7 et dans la majorité des instances, les meilleurs résultats donnés par AG_PsyLS c'est lorsque les ressources non-renouvelables sont fournies de manière non-uniforme (NUA).

5.6 Conclusion

Comme les méthodes approchées constituent une alternative appropriée aux méthodes exactes pour résoudre les problèmes d'optimisation combinatoire NP-difficile, nous avons donc proposé dans ce chapitre un algorithme génétique permettant de résoudre le problème d'ordonnancement

Flow-Shop sous contraintes de ressources non-renouvelables.

Pour déterminer la meilleure combinaison des paramètres de l'AG, la méthode de Taguchi a été utilisée. De plus, pour enrichir le comportement de recherche et améliorer les solutions, une recherche locale basée sur l'heuristique NEH a été utilisée. Deux stratégies d'application de la recherche locale ont été étudiées. La première, appelée recherche locale aléatoire (RanLS), consiste à appliquer la recherche locale de manière probabiliste au meilleur individu de chaque génération et une recherche locale partielle systématique (PsyLS), qui est appliqué systématiquement à un pourcentage prédéfini du meilleur individu de chaque génération.

Les expérimentations effectuées montrent que l'algorithme génétique hybride proposé que se soit avec la recherche locale RanLS (AG_RanLS) ou PsyLS (AG_PsyLS) donne de bons résultats en particulier lorsque la probabilité de recherche locale est supérieur à 0.5 (respectivement 50%)). Cependant, l'AG_RanLS est plus gourmand en temps de calcul en comparant avec l'AG_PsyLS.

Dans le chapitre suivant, nous adaptons une autre métaheuristqiue qui a récemment donné de bons résultats sur le problème Flow-Shop classique et avec contraintes.

TABLE 5.6 : Comparaison de résultats - problèmes de moyennes à grandes tailles : AG_RanLS

Problème	Méthode	Configuration							
		UAF1		UAF2		UAF3		NUA	
		$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$
20 x 10	AG_NoLS	3.50	8.13	2.22	7.23	2.75	7.33	**4.35**	7.67
	AG_RanLS (P_{ls}=0.1)	4.39	8.27	2.61	7.59	3.57	7.62	**5.43**	7.88
	AG_RanLS (P_{ls}=0.5)	4.74	9.21	3.48	8.30	4.39	8.45	**6.40**	8.58
	AG_RanLS (P_{ls}=0.8)	4.94	9.86	3.50	8.92	5.10	9.15	**6.50**	9.23
20 x 20	AG_NoLS	**1.60**	13.52	1.40	13.81	1.35	13.53	0.68	12.26
	AG_RanLS (P_{ls}=0.1)	1.75	13.94	1.59	14.26	**1.80**	13.91	1.49	12.72
	AG_RanLS (P_{ls}=0.5)	2.04	15.70	1.98	15.91	**2.09**	15.52	1.81	14.21
	AG_RanLS (P_{ls}=0.8)	**2.39**	17.09	2.38	17.13	2.21	17.01	1.98	15.53
50 x 5	AG_NoLS	4.84	14.59	3.04	16.48	4.72	23.84	**5.45**	22.70
	AG_RanLS (P_{ls}=0.1)	5.34	16.60	4.13	17.86	5.33	28.18	**5.96**	25.96
	AG_RanLS (P_{ls}=0.5)	6.03	25.14	4.62	26.46	5.59	41.66	6.42	37.05
	AG_RanLS (P_{ls}=0.8)	**6.58**	31.53	4.78	33.44	5.78	51.35	6.48	46.65
50 x 10	AG_NoLS	1.79	34.49	2.98	35.16	2.28	32.74	**3.07**	32.76
	AG_RanLS (P_{ls}=0.1)	3.20	40.64	3.35	40.62	3.41	37.24	**3.98**	38.27
	AG_RanLS (P_{ls}=0.5)	3.47	62.14	3.47	62.59	3.98	59.69	**4.42**	59.30
	AG_RanLS (P_{ls}=0.8)	3.80	77.77	3.59	78.86	**5.49**	75.62	5.06	73.55
50 x 20	AG_NoLS	1.90	61.06	1.38	61.50	1.34	58.02	**2.12**	57.50
	AG_RanLS (P_{ls}=0.1)	**2.39**	70.41	2.27	70.90	2.09	67.23	2.33	68.94
	AG_RanLS (P_{ls}=0.5)	2.56	116.30	**2.76**	114.13	1.91	110.48	2.56	111.56
	AG_RanLS (P_{ls}=0.8)	2.92	152.13	**3.11**	145.58	2.36	141.42	2.94	142.48
100 x 5	AG_NoLS	2.75	54.71	1.45	55.73	3.55	53.19	**4.17**	54.53
	AG_RanLS (P_{ls}=0.1)	4.18	80.94	1.84	79.98	4.33	80.47	**5.12**	80.24
	AG_RanLS (P_{ls}=0.5)	4.80	185.02	2.57	182.96	4.48	183.69	**5.69**	181.94
	AG_RanLS (P_{ls}=0.8)	5.21	272.36	2.89	262.40	5.05	267.39	**5.92**	263.99
100 x 10	AG_NoLS	**3.10**	84.43	1.57	83.81	3.03	87.49	2.52	91.24
	AG_RanLS (P_{ls}=0.1)	4.01	133.22	3.28	130.39	**4.20**	138.17	3.60	140.69
	AG_RanLS (P_{ls}=0.5)	4.07	330.32	3.86	339.79	**4.71**	339.24	4.61	353.47
	AG_RanLS (P_{ls}=0.8)	5.15	487.45	4.85	470.72	**5.78**	491.67	4.86	508.54
100 x 20	AG_NoLS	1.46	132.66	0.52	120.92	**1.62**	125.32	1.54	131.27
	AG_RanLS (P_{ls}=0.1)	**4.18**	227.76	0.56	190.58	2.64	213.67	2.05	204.37
	AG_RanLS (P_{ls}=0.5)	**4.24**	605.76	0.98	519.08	2.94	560.40	2.50	568.29
	AG_RanLS (P_{ls}=0.8)	**5.04**	884.52	1.37	748.69	3.05	818.68	3.40	819.62
Average of all		3.70	133.68	2.64	124.43	3.53	130.61	3.93	131.34

TABLE 5.7 : Comparaison de résultats - problèmes de moyennes à grandes tailles : AG_PsyLS

Problème	Méthode	Configuration							
		UCF1		UCF2		UCF3		NUA	
		$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$	$\overline{ARI}$	$\overline{CPU}$
20 x 10	AG_NoLS	3.50	8.13	2.22	7.23	2.75	7.29	**4.35**	7.67
	AG_PsyLS (10%)	4.74	8.20	2.68	7.28	3.17	7.33	**5.53**	7.76
	AG_PsyLS (50%)	4.82	8.71	2.84	7.98	4.35	7.78	**5.82**	8.17
	AG_PsyLS (80%)	5.20	9.03	3.20	8.41	4.61	8.54	**6.29**	8.81
	AG_PsyLS (100%)	5.30	10.32	3.64	9.37	5.10	9.54	**6.62**	9.61
20 x 20	AG_NoLS	**1.60**	13.52	1.40	13.81	1.35	13.53	0.68	12.26
	AG_PsyLS (10%)	1.89	13.56	**1.93**	13.83	1.57	13.54	1.12	12.37
	AG_PsyLS (50%)	**2.24**	14.94	2.06	14.75	1.87	14.90	1.31	13.55
	AG_PsyLS (80%)	**2.36**	15.41	2.22	15.85	2.03	15.76	1.76	14.97
	AG_PsyLS (100%)	**3.16**	18.10	2.67	17.95	2.42	17.88	2.30	16.33
50 x 5	AG_NoLS	4.84	14.59	3.04	16.48	4.72	23.84	**5.45**	22.70
	AG_PsyLS (10%)	5.41	14.85	4.15	16.54	5.09	24.32	**5.88**	22.82
	AG_PsyLS (50%)	6.06	21.46	4.53	22.84	5.51	38.28	**6.16**	33.29
	AG_PsyLS (80%)	**6.31**	26.41	4.74	29.75	5.69	47.57	6.28	43.18
	AG_PsyLS (100%)	**6.89**	35.08	4.92	38.26	6.30	58.17	6.74	54.39
50 x 10	AG_NoLS	1.79	34.45	2.98	35.16	2.28	32.74	**3.07**	32.76
	AG_PsyLS (10%)	2.69	34.49	3.19	35.32	3.40	33.40	**4.00**	33.03
	AG_PsyLS (50%)	3.68	44.77	3.40	50.21	3.78	40.63	**3.90**	45.66
	AG_PsyLS (80%)	4.12	56.00	3.74	63.97	5.09	61.18	**5.11**	59.94
	AG_PsyLS (100%)	4.53	89.84	4.19	90.59	**6.33**	86.48	5.62	86.49
50 x 20	AG_NoLS	1.90	60.71	1.38	61.50	1.34	58.02	**2.12**	57.50
	AG_PsyLS (10%)	2.25	61.06	1.59	61.73	1.77	59.97	**2.38**	57.80
	AG_PsyLS (50%)	2.72	79.78	2.10	73.83	2.39	72.68	**2.73**	83.77
	AG_PsyLS (80%)	2.95	108.95	2.58	115.70	2.88	101.29	**3.02**	100.54
	AG_PsyLS (100%)	**3.68**	172.04	2.86	160.31	3.21	163.39	3.42	162.67
100 x 5	AG_NoLS	2.75	54.22	1.45	55.73	3.55	53.19	**4.17**	54.53
	AG_PsyLS (10%)	3.73	54.71	1.69	66.73	4.04	58.64	**5.00**	55.60
	AG_PsyLS (50%)	4.03	126.03	2.26	156.49	4.37	130.96	**5.28**	134.06
	AG_PsyLS (80%)	4.23	179.16	2.71	196.94	4.63	225.50	**5.58**	194.23
	AG_PsyLS (100%)	5.00	323.22	3.14	310.49	5.26	321.16	**6.11**	315.67
100 x 10	AG_NoLS	3.10	83.62	1.57	83.37	**3.03**	85.64	2.52	91.24
	AG_PsyLS (10%)	**4.14**	84.43	3.15	83.81	3.82	87.49	3.61	134.02
	AG_PsyLS (50%)	**4.48**	167.96	3.91	137.91	4.15	250.63	4.19	387.92
	AG_PsyLS (80%)	**5.04**	256.19	4.82	249.58	4.73	288.01	4.73	616.63
	AG_PsyLS (100%)	**5.77**	587.05	5.43	576.73	5.98	593.81	5.50	682.64
100 x 20	AG_NoLS	1.46	132.66	0.52	120.92	**1.62**	124.69	1.54	131.27
	AG_PsyLS (10%)	**3.32**	132.75	0.85	153.24	2.66	125.32	1.75	135.06
	AG_PsyLS (50%)	**4.06**	165.47	1.01	297.13	3.03	302.99	2.26	319.42
	AG_PsyLS (80%)	**4.85**	451.25	1.04	636.86	3.37	309.55	2.74	378.85
	AG_PsyLS (100%)	**5.13**	1087.18	1.73	998.94	3.75	994.41	3.45	615.25
	Average of all	3.89	121.51	2.74	127.84	3.67	124.25	4.00	131.36

Chapitre 6

Algorithme d'optimisation par essaims particulaires pour le Flow-Shop sous contraintes de ressources non-renouvelables

6.1 Introduction

L'application des métaheuristiques pour la résolution des problèmes d'ordonnancement sous contraintes de ressources non-renouvelables est assez limité (voir un à deux articles). C'est ce qui nous a motivés à considérer ces approches pour résoudre le problème considéré, et cela, afin d'étendre les travaux rapportés dans la littérature.

Dans le chapitre précédent, un algorithme génétique a été proposé pour résoudre le problème d'ordonnancement Flow-Shop sous contraintes de ressources consommables. Rappelons que le choix de cet algorithme a été basé sur les travaux de Belkaid et al. [8].

Dans ce chapitre, nous proposons une autre métaheuristique a la base d'un algorithme d'optimisation par essaims particulaires. Cette métaheuristique à été initialement conçue pour l'optimisation continue, mais qui récemment fait l'objet de quelques adaptations pour résoudre les problèmes d'optimisation discrets, en particulier les problèmes Flow-Shop et a donné de bons résultats. L'objectif de ce chapitre consiste à valider le potentiel de cette métaheuristique pour résoudre le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

6.2 Conception d'un algorithme d'OEP discrèt pour le Flow-Shop sous contraintes de ressources

L'algorithme d'OEP repose généralement sur un ensemble de solutions (essaim) disposées aléatoirement au début de l'algorithme. Chaque solution est appelée particule et se situe à une position donnée dans l'espace de recherche. Chaque particule dispose d'une mémoire dans laquelle est conservée sa meilleure position visitée ainsi que la meilleure position atteinte par les particules de l'essaim. Cet algorithme fonctionne aussi de manière itérative où à chaque itération, les particules se déplacent dans l'espace de recherche selon une certaine vitesse. Les étapes de l'algorithme d'OEP adaptées à notre problème sont les suivantes :

6.2.1 Représentation de particule

L'un des principaux problèmes lors de la conception d'un algorithme d'OEP réside dans la représentation de la solution (particule). Puisque nous sommes dans le cadre de la résolution d'un problème d'ordonnancement de permutation, c'est-à-dire que nous cherchons la permutation des jobs qui minimise le makespan en présence des contraintes de ressources non-renouvelables. Alors, nous gardons le schéma de codage basé sur la permutation de jobs.

Chaque particule i est représentée par une séquence $\sigma_i^t = \{\pi_{i1}^t, \pi_{i2}^t, ..., \pi_{in}^t\}$ où n représente le nombre de jobs dans le système et π_{ik}^t représente le job affecté à la position k du particule i à l'itération t.

Soit $\sigma_i^t = \{J_2, J_4, J_1, J_5, J_3\}$ une solution potentielle du i^{ime} particule dans la t^{ime} itération, $\pi_{i4}^t = J_5$ signifie que le job J_5 est ordonnancé à la position 4 dans la particule.

Dans l'algorithme proposé, nous considérons un ensemble de s particules (s représente la taille de l'essaim) où pour chaque particule i nous associons une séquence σ_i^t à l'itération t et une fonction objectif représentée par le makespan donné par la valeur du coût $E(\sigma_i^t)$.

6.2.2 Initialisation de l'essaim

Dans l'algorithme d'OEP de base, l'essaim initial est souvent généré de manière aléatoire. Dans ce travail, pour garantir un essaim initial $\Phi^0 = \{\sigma_1^0, \sigma_2^0, \sigma_3^0, ..., \sigma_s^0\}$ avec une certaine qualité et diversité, les permutations de trois heuristiques constructives bien connues pour le Flow-Shop (a) NEH [97], (b) NEHKK1 [67] and (c) Palmer [103] sont inclus tout en respectant la disponibilité des ressources non-renouvelables. Outre que ces trois solutions initiales, afin de maintenir la diversité de la population, le reste de l'essaim (s - 3) est construit de manière à ce qu'une permutation soit produite de manière aléatoire.

6.2.3 Mise à jour de la vitesse et de la position

Dans cette partie, nous empruntons l'opérateur de l'AG pour mettre à jour la position de la particule dans l'algorithme proposé. L'état de particule dans l'espace de recherche étudié est caractérisé par deux facteurs : sa position σ_i^t et sa vitesse v_i^t qui est aussi une permutation dans l'espace de recherche. On note, $Pbest_i^t$ la meilleure position personnelle qui représente la séquence avec la plus petite valeur de makespan trouvée par la particule jusqu'à l'itération t et $Gbest^t$ la meilleure position globale qui représente la meilleure position atteinte par toutes les particules de l'essaim. .

S'inspirant du modèle présenté par Lian et al. [83], le processus de génération d'une nouvelle position dans l'essaim est représentée par les équations suivantes :

$$v_i^{t+1} = Pbest_i^t \otimes Gbest^t \tag{6.1}$$

$$\sigma_i^{t+1} = \sigma_i^t \otimes v_i^{t+1} \tag{6.2}$$

Où $\otimes$ est la dénotation de l'opérateur de croisement qui sera détaillée dans la section 6.2.5, et t représente le nombre de générations.

6.2.4 Stratégie de diversification

Contrairement à l'OEP de base, l'avantage du modèle de vitesse précédent est qu'il n'y a pas de paramètres à ajuster, à part la taille de l'essaim. Cependant, ce modèle peut facilement être piégé dans les optima locaux, lorsque :

(a) La meilleure position personnelle a la même séquence que la meilleure position globale.

(b) La nouvelle vitesse obtenue a la même séquence que la position actuelle.

Pour le cas (a), la particule perde la capacité de trouver la nouvelle vitesse. Alors que dans le cas (b) aucun mouvement ne peut être effectué pour une particule.

Afin d'éviter ces états de stagnations indésirables, une perturbation basée sur la recherche du voisinage de la meilleure position globale est appliquée comme mentionner dans l'équation 6.3 et pour le deuxième cas, une perturbation de la position actuelle est opérée comme le montre l'équation 6.4.

$$v_i^{t+1} = mutation(Gbest^t) \tag{6.3}$$

$$\sigma_i^{t+1} = mutation(\sigma_i^t) \tag{6.4}$$

Le principal avantage de la mise en œuvre de la mutation est d'accroître la diversité de l'essaim. La mutation pourrait réduire le piégeage de l'algorithme OEP proposé et produire de meilleur

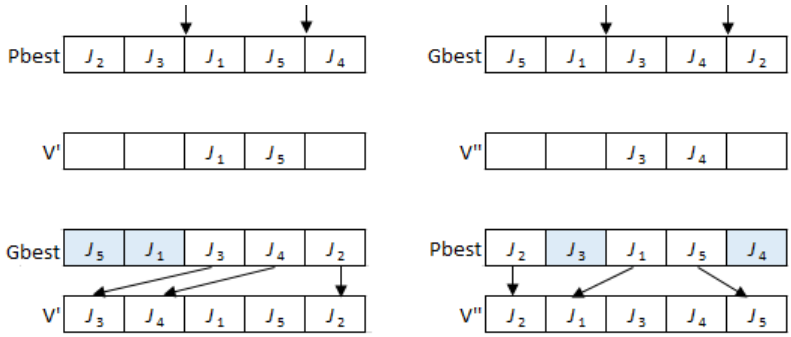


FIGURE 6.1 : Illustration de la manière dont l'opérateur ⊗ fonctionne

résultat. La mise à jour de la meilleure position personnelle de chaque particule et de la meilleure position globale est obtenue respectivement comme suit :

$$Pbest_i^{t+1} = \begin{cases} \sigma_i^{t+1} & \text{if } E(\sigma_i^{t+1}) < E(Pbest_i^t) \\ Pbest_i^t & \text{otherwise} \end{cases} \tag{6.5}$$

$$Gbest^{t+1} = \begin{cases} \arg\min_{\forall i}\{E(Pbest_i^{t+1})\} \text{ if} \\ E(Gbest^t) > \min_{\forall i}\{E(Pbest_i^{t+1})\} \\ Gbest^t \qquad \text{otherwise} \end{cases} \tag{6.6}$$

6.2.5 Opérateurs génétiques empruntés

6.2.5.1 Opérateur de croisement

Le croisement est un opérateur génétique utilisé après la sélection dans l'algorithme génétique [49]. Cet opérateur est appliqué dans notre algorithme d'OEP pour mettre à jour la position des particules dans le modèle de vitesse défini. Dans notre mise en œuvre, l'opérateur de croisement à deux points est appliqué.

Dans cet opérateur, deux points de coupure sont choisis au hasard, les jobs à l'intérieur des deux points sélectionnés sont directement hérités des parents aux enfants, tandis que les jobs restants sont remplis en balayant les parents de gauche à droite et en ne reprenant que les jobs non encore transmis comme illustré par la figure 6.1.

Comme le montre la figure 6.1, cette procédure génère deux particules distinctes ; nous les comparons et laissons la particule avec la valeur de fitness la plus petite, soit la particule finale de cet opérateur.

6.2.5.2 Opérateur de mutation

L'opérateur de mutation est introduit pour empêcher les particules de rester bloqué dans les optima locaux (cf. section 6.2.4). Étant donné que l'opérateur de mutation par insertion est le plus approprié pour les problèmes Flow-Shop [98], ce dernier est utilisé dans notre algorithme d'OEP. Cet opérateur est déjà décrit dans le chapitre 5.

6.2.6 Stratégie d'intensification

Dans cette section, nous introduisons une procédure de recherche locale pour améliorer l'intensification au sein de l'algorithme l'OEP proposé. Nous appliquons la même recherche locale utilisé dans le chapitre précédent. Le pseudo-code de cette recherche locale est décrit par l'algorithme 14 du chapitre 5.

La stratégie d'incorporation de la recherche locale susmentionnée dans l'algorithme d'OEP proposé est la suivante : à chaque itération et pour tous $Pbest_i$, on effectue la recherche locale avec une probabilité prédéfinie P_{ls}.

Des valeurs de probabilité allant de [0.1, 0.2,..., 1] ont été testées, un bon compromis entre le temps de calcul et la qualité de solution est obtenu pour $P_{ls} = 0.1$.

Le pseudo-code d'algorithme d'OEP proposé est présenté dans l'algorithme 15, cet algorithme est noté dans la suite de ce chapitre par IDPSO (Improved Discrete Particle Swarm Optimization).

6.3 Paramétrage de l'algorithme

6.3.1 Taille de l'essaim

La performance de l'algorithme d'optimisation par essaims particulaires dépend fortement du choix des paramètres appropriés. Dans l'algorithme proposé, le seul paramètre qui doit être réglé est la taille de l'essaim (c'est l'avantage de notre implémentation.). Par conséquent, nous avons testé différentes valeurs de taille d'essaim ($s = 50, 100, 150, 200$) sur certains problèmes afin de déterminer la meilleure taille pour le Flow-Shop sous contraintes de ressources non-renouvelables.

Pour ce faire, nous avons sélectionné cinq types de problèmes qui représentent la diversité entre le nombre jobs, nombre de machines ainsi que la disponibilité de ressources non-renouvelables ($n \times m \times$ configuration) comme le montre le tableau 6.1.

Les résultats sont illustrés dans la figure 6.2, qui représente la valeur moyenne de la fonction d'objectif et le temps de calcul requis pour chaque valeur de la taille de l'essaim. Selon cette figure, nous constatons que plus la taille d'essaim augmente, plus le temps de calcul (en secondes) augmente, tandis que la qualité de la recherche s'améliore. Ainsi, dans ce travail, $s = 200$ est recommandé.

TABLE 6.1 : Problèmes testés

Problèmes testés	
1	10 × 10 × UAF2
2	20 × 20 × NUA
3	50 × 5 × UAF3
4	50 × 20 × UAF2
5	100 × 10 × UAF1

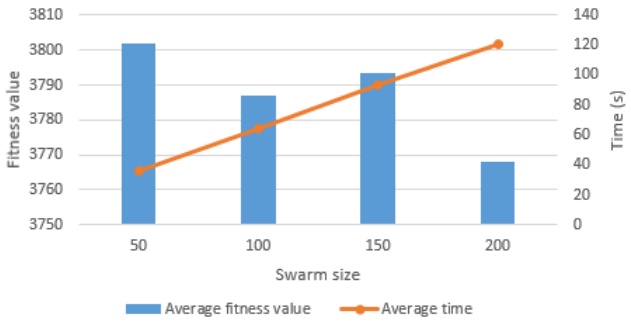


FIGURE 6.2 : Effets de la taille de l'essaim

6.3.2 Critère d'arrêt

Il n'existe pas de règle pratique permettant de définir un critère d'arrêt approprié, car le choix de ce dernier dépend fortement de la taille du problème. Par conséquent, le critère d'arrêt pour l'algorithme proposé est conditionné par un nombre de générations maximum fixé en fonction de la taille du problème (200 itérations pour les petites instances, 500 et 700 itérations pour les moyennes et grandes instances, respectivement.).

6.4 Résultats expérimentaux

Afin de tester les performances de l'algorithme d'OEP proposé pour le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables, nous effectuons des simulation lorsque la disponibilité de ressources non-renouvelables est représentée par une arrivée uniforme croissante (UAF1), uniforme décroissante (UAF2), uniforme aléatoire (UAF3) et non-uniforme (NUA).

6.4.1 Résultats de calcul pour les problèmes de petites tailles

Le tableau 6.3 démontre à la fois l'efficacité de la recherche locale proposée en comparant IDPSO avec IDPSO sans recherche locale, notée IDPSO_NOLS, et la performance de IDPSO sous différentes configurations de ressources non-renouvelables pour les instances de petites tailles.

Dans le tableau 6.3, les mesures de performance BRE (Best Relative Error), ARE (Average Relative Error) et WRE (Worst Relative Error) désignent respectivement la meilleure erreur relative, l'erreur relative moyenne et la pire erreur relative par rapport à la meilleure solution trouvée par CPLEX.

Ces mesures sont calculées comme suit :

$$BRE(\%) = \min_{i=1...R} \left(\frac{\left(C_{max_i}^{Alg} - C_{max_i}^{Cplex} \right) \times 100}{C_{max_i}^{Cplex}} \right) \tag{6.7}$$

$$ARE(\%) = \sum_{i=1}^{R} \left(\frac{\left(C_{max_i}^{Alg} - C_{max_i}^{Cplex} \right) \times 100}{C_{max_i}^{Cplex}} \right) \Big/ R \tag{6.8}$$

$$WRE(\%) = \max_{i=1...R} \left(\frac{\left(C_{max_i}^{Alg} - C_{max_i}^{Cplex} \right) \times 100}{C_{max_i}^{Cplex}} \right) \tag{6.9}$$

Où $C_{max_i}^{Alg}$ représente le makespan trouvé par l'algorithme IDPSO alors que $C_{max_i}^{Cplex}$ représente la valeur optimale trouvée par CPLEX et R est le nombre de réplications.

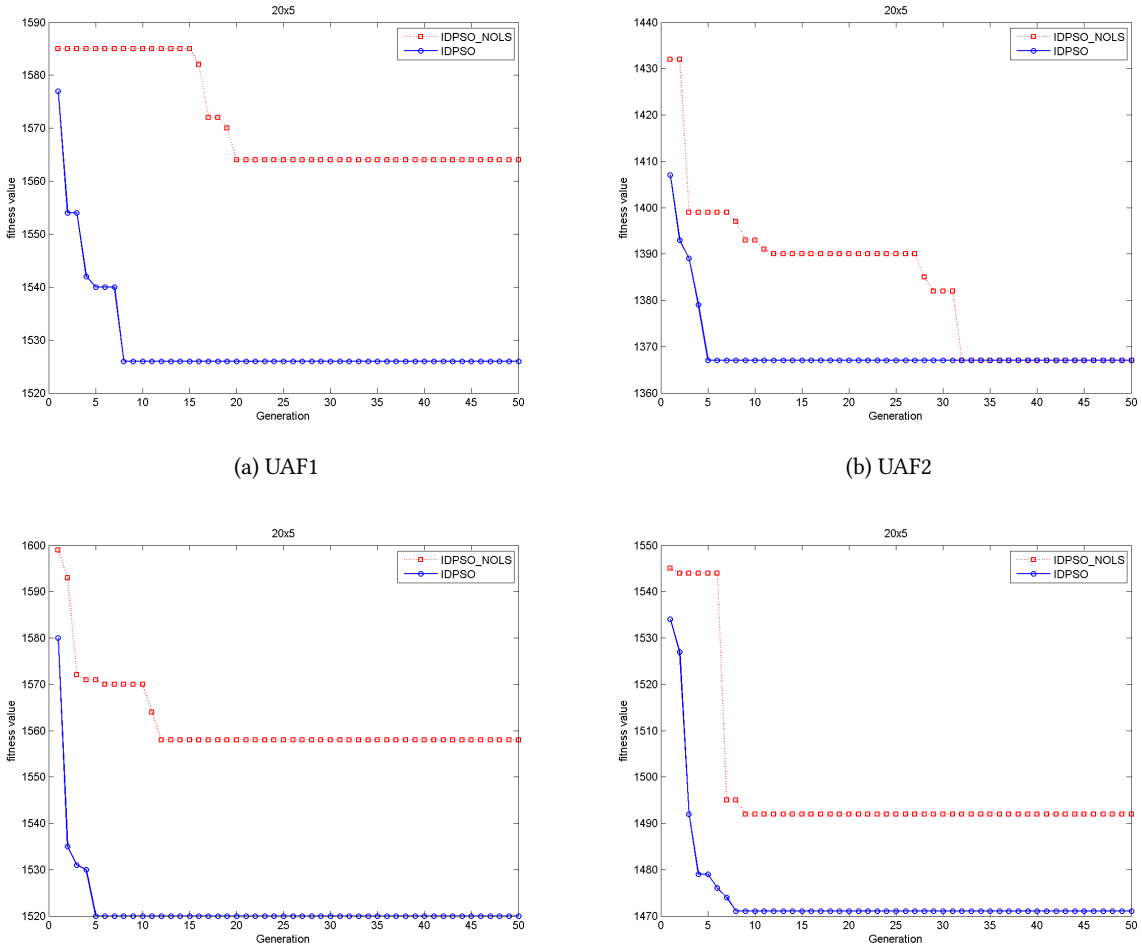
À partir du tableau 6.3, nous pouvons observer que l'algorithme IDPSO produit presque les mêmes valeurs BRE que IDPSO_NOLS, à l'exception de problème avec 20 jobs et 5 machines où la disponibilité de ressources non-renouvelables est une fonction uniforme croissante (UAF1) et uniforme aléatoire (UAF3). De plus, les valeurs ARE et WRE obtenues par IDPSO sont bien meilleures que celles obtenues par IDPSO_NOLS.

Ces résultats montrent l'efficacité de l'incorporation de la recherche locale dans l'algorithme IDPSO et ceci pour toutes les configurations de ressources considérés. Cela signifie que la supériorité en termes de qualité de recherche et de performances de l'algorithme IDPSO provient de la combinaison de la recherche globale et locale, plus explicitement, de l'équilibre entre le processus d'exploration et d'exploitation.

La convergence des valeurs de fitness en fonction de générations pour IDPSO_NOLS et IDPSO sous différentes configurations est illustrée par la figure 6.3 où, pour des raisons de clarté, seules les 50 premières générations sont présentées pour une instance arbitraire du problème 20 jobs et 5 machines (20 x 5).

Comme le montre la figure 6.3, la valeur de fitness (makespan) d'IDPSO proposé décroît rapidement. Par exemple, dans la figure 6.3 .b, IDPSO converge vers environ 5 générations, alors que IDPSO sans recherche locale prend environ 33 générations. C'est-à-dire que le taux de convergence d'IDPSO est plus rapide que celui d'IDPSO_NOLS, et cela, pour toutes les configurations de ressources non-renouvelables considérées.

Le temps de calcul requis pour IDPSO et IDPSO_NOLS est présenté dans tableau 6.2. Puisque la recherche locale est appliquée alors l'IDPSO proposé mettant plus de temps que l'IDPSO sans re-



(a) UAF1

(b) UAF2

(c) UAF3

(d) NUA

FIGURE 6.3 : Convergence d'une instance du problème sous différentes configurations

cherche locale. Cela peut être interprété comme le coût pour éviter une convergence prématurée afin de parvenir à des meilleures performances.

Pour la performance de l'algorithme IDPSO sous différentes configurations, le tableau 6.3 montre que les valeurs BRE résultant d'IDPSO sont égales aux solutions optimales fournies par CPLEX et cela pour toutes les configurations de ressources non-renouvelables considérées. Au même temps, on peut constater que la différence entre les valeurs BRE et ARE est très faible pour toutes les configurations, ce qui signifie que IDPSO présente des performances supérieures pour les problèmes de petites tailles.

De plus, on peut remarquer que les valeurs ARE et WRE résultent d'IDPSO lorsque la disponibilité des ressources est une fonction non-uniforme (NUA) sont légèrement meilleure que les valeurs des autres configurations.

TABLE 6.2 : Temps de résolution pour IDPSO et IDPSO_NOLS

Problem	UAF1		UAF2		UAF3		NUA	
	IDPSO_NOLS	IDPSO	IDPSO_NOLS	IDPSO	IDPSO_NOLS	IDPSO	IDPSO_NOLS	IDPSO
10 x 5	13.20	4.94	13.53	4.68	14.05	5.65	13.60	5.74
10 x 10	17.57	22.11	15.52	20.02	17.82	22.28	17.60	21.82
10 x 20	26.51	39.79	25.56	36.76	27.16	39.83	27.77	38.98
20 x 5	24.53	57.92	25.80	58.49	23.93	58.28	24.97	57.52
Average	20.45	33.69	20.10	32.49	20.74	34.01	20.99	33.51

6.4.2 Résultats de calcul pour les problèmes de moyennes à grandes tailles

Après avoir étudié les performances de l'algorithme IDPSO sur les problèmes de petites tailles et afin de continuer avec la résolution du problème Flow-Shop qui prend en compte les contraintes de ressources non-renouvelables, nous présentons les tests d'IDPSO sur les instances de moyennes à grandes tailles.

Dans cette partie, nous comparons toujours les performances d'IDPSO et d'IDPSO_NOLS afin de vérifier si la recherche locale proposée améliore toujours la qualité de solution d'IDPSO pour les instances de moyennes à grandes tailles. De plus, pour tester l'efficacité d'IDPSO, une comparaison est effectuée aussi avec les résultats obtenus par l'algorithme génétique (AG_PsyLS (100%)) proposé dans le chapitre **??**.

Afin d'effectuer une comparaison équitable, IDPSO et IDPSO_NOLS utilisent le même temps de calcul que AG_PsyLS (100%), ce temps de calcul est fixé en fonction de la taille du problème. Sa valeur est à 200 secondes pour les problèmes $(20\times10),(20\times20)$; 300 secondes pour les problèmes $(50\times5),(50\times10),(50\times20)$ et 500 secondes pour les problèmes restants.

Dans le tableau 6.4, MRI et ARI indiquent respectivement l'amélioration maximale et moyenne de chaque algorithme par rapport aux résultats de NEH tout en respectant les contraintes de ressources non-renouvelables. MRI (Maximum Relative Improvements) et ARI (Average Relative Improvements) sont calculées comme suit :

$$MRI(\%) = \max_{i=1\ldots R}\left(\frac{\left(C^{NEH}_{max_i} - C^{Alg}_{max_i}\right)\times 100}{C^{NEH}_{max_i}}\right) \tag{6.10}$$

$$ARI(\%) = \sum_{i=1}^{R}\left(\frac{\left(C^{NEH}_{max_i} - C^{Alg}_{max_i}\right)\times 100}{C^{NEH}_{max_i}}\right)\Big/R \tag{6.11}$$

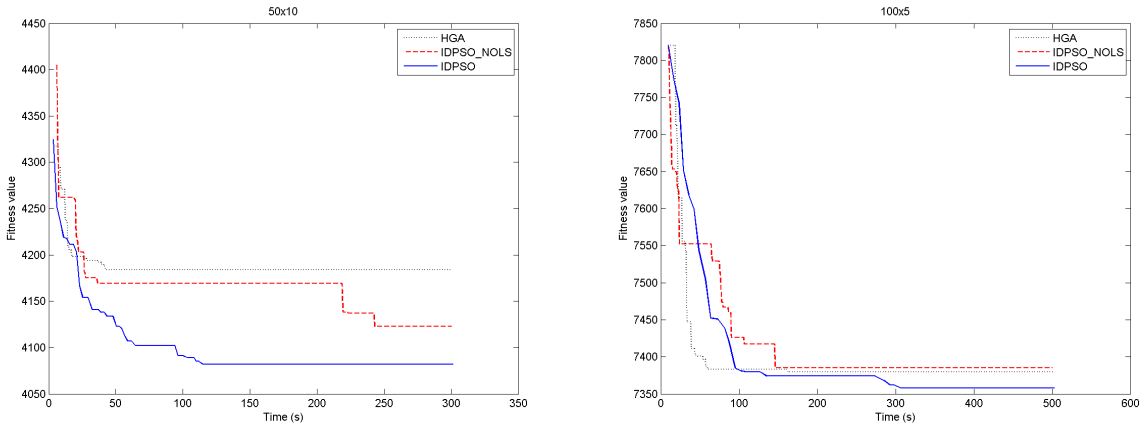
Où $C^{NEH}_{max_i}$ est la valeur retournée par NEH tout en respectant les contraintes de ressources non-renouvelables.

À partir du tableau 6.4, nous observons que les résultats de calcul générés par l'algorithme IDPSO peuvent améliorer considérablement le résultat de NEH, en particulier pour une arrivée non-

uniforme de ressources non-renouvelables (NUA). Deuxièmement, les valeurs MRI et ARI obtenues par IDPSO_NOLS sont pires que celles obtenues par IDPSO pour presque toutes les instances. Nous concluons donc que la recherche locale proposée reste efficace pour les instances de moyennes à grandes tailles. Troisièmement, selon les valeurs ARI, la qualité de recherche d'IDPSO est très supérieure à celle d'AG_PsyLS (100%) et cela pour toutes les instances de chaque configuration.

Cependant, en ce qui concerne les valeurs de MRI lorsque la disponibilité des ressources non-renouvelables est non-uniforme, AG_PsyLS (100%) est compétitif à IDPSO pour les problèmes $(50 \times 5), (50 \times 20)$ et (100×20). Par ailleurs, nous observons du tableau 6.4 que IDPSO _NOLS domine AG_PsyLS (100%) pour la majorité des instances. Donc, nous confirmons que l'algorithme d'optimisation par essaim particulaire proposé est plus efficace que l'algorithme génétique pour le problème Flow-Shop sous contraintes de ressources non-renouvelables.

Pour étudier la convergence des algorithmes précédents, nous traçons les valeurs de fitness en fonction du temps (en secondes) pour deux instances arbitraires de problème (50×10) et (100×5), à partir de la figure 6.4 il est clair que IDPSO est meilleur que AG_PsyLS (100%) et IDPSO_NOLS.



(a) 50×10 (b) 100×5

FIGURE 6.4 : Convergence d'AG, IDPSO_NOLS et IDPSO

6.5 Conclusion

Comme les méthodes approchées constituent une alternative appropriée aux méthodes exactes pour la résolution des problèmes d'optimisation combinatoire NP-difficile, nous avons présenté dans ce chapitre une autre méthode approchée visant à résoudre le problème de minimisation de makespan dans un atelier Flow-Shop sous contrainte de ressources non-renouvelables.

La méthode approchée développée dans ce chapitre est un algorithme d'optimisation par essaim particulaire. Ce dernier à été initialement conçu pour l'optimisation continue, mais qui récem-

ment fait l'objet de quelques adaptations pour résoudre les problèmes d'optimisation discrets. Nous avons décidé d'implémenter cet algorithme grâce aux résultats obtenus dans les différents problèmes d'ordonnancement, plus particulièrement les problèmes de type Flow-Shop qui ont été mentionnés dans le deuxième chapitre de cette thèse. Contrairement à l'algorithme génétique proposé dans le chapitre précédent, l'algorithme d'optimisation par essaim particulaire se base sur la coopération des individus pour faire évoluer.

Dans ce chapitre, afin de rendre l'algorithme d'optimisation par essaim particulaire adapté au problème Flow-Shop sous contrainte de ressources non-renouvelables qui est un problème combinatoire, nous avons modifié la représentation des particules ainsi que le modèle de vitesse. Tout d'abord, pour la représentation des particules, une représentation par permutation a été adoptée. Deuxièmement, les vitesses sont implémentées a l'aide d'un opérateur de croisement, emprunté de l'algorithme génétique. De plus, afin de garantir un essaim initial avec une certaine qualité et diversité, trois solutions issues des heuristique bien connue ont été intégrées à l'essaim initial. Nous avons également introduit une stratégie de diversification en utilisant un opérateur de mutation emprunté aussi de l'algorithme génétique pour empêcher l'algorithme de rester bloqué dans les optima locaux. En outre, une hybridation de l'algorithme avec une recherche locale basée sur l'heuristique NEH a été utilisée pour intensifier la qualité de la recherche et améliorer la convergence de l'algorithme.

La performance de l'algorithme proposé a été testée sur les instances générées dans le quatrième chapitre de cette thèse, et comparée à différentes configurations de disponibilité de ressources non-renouvelables aux solutions optimales rapportées par le modèle mathématique du chapitre 4 (PLNE - lancé sur CPLEX) pour les instances de petites tailles et aux résultats d'algorithme génétique proposé dans le chapitre 5 pour les instances de moyennes à grandes tailles. Les résultats obtenus montrent l'efficacité de l'algorithme proposé pour la résolution de problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

Algorithme 15 Pseudo-code d'IDPSO

Début

Set $t = 0$
for $i = 1$ jusqu'à s **do**
  Générer σ_i^t.
  Évaluer $E(\sigma_i^t)$.
  Set $Pbest_i^t \longleftarrow \sigma_i^t$.
end for
$Gbest^t \longleftarrow \{\sigma_m^t | m = \arg\min_{\forall i}\{E(\sigma_i^t)\}\}$.
Tant que $(t < t_{max})$
  for $i = 1$ jusqu'à s **do**
    if $Pbest_i^t == Gbest^t$ **then**
      $v_i^{t+1} \longleftarrow mutation(Gbest^t)$.
    else
      $v_i^{t+1} \longleftarrow Pbest_i^t \otimes Gbest^t$.
    end if
    if $v_i^{t+1} == \sigma_i^t$ **then**
      $\sigma_i^{t+1} \longleftarrow mutation(\sigma_i^t)$.
    else
      $\sigma_i^{t+1} \longleftarrow \sigma_i^t \otimes v_i^{t+1}$
    end if
    if $E(\sigma_i^{t+1}) < E(Pbest_i^t)$ **then**
      $Pbest_i^{t+1} \longleftarrow \sigma_i^{t+1}$.
    else
      $Pbest_i^{t+1} \longleftarrow Pbest_i^t$.
    end if
  end for
  for i in $Pbest$ **do**
    Générer une probabilité α où $\alpha \in [0, 1]$.
    if $\alpha < p_{ls}$ **then**
      Effectuer la recherche locale pour le $Pbest$ sélectionné.
      Mise à jour $Pbest_i^{t+1}$.
    end if
  end for
  if $E(Gbest^t) > \min_{\forall i}\{E(Pbest_i^{t+1})\}$ **then**
    $Gbest^{t+1} \longleftarrow \{Pbest_m^{t+1} | m = \arg\min_{\forall i}\{E(Pbest_i^{t+1})\}\}$
  end if
  $t \longleftarrow t + 1$.
Fin Tant que

Fin

Table 6.3 : Comparaison de résultats pour les petites instances : IDPSO_NOLS / IDPSO

Problème	UAF1						UAF2						UAF3						NUA					
	IDPSO_NOLS			IDPSO			IDPSO_NOLS			IDPSO			IDPSO_NOLS			IDPSO			IDPSO_NOLS			IDPSO		
	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	WRE
10 x 5	0.00	0.25	1.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.64	1.93	0.00	0.00	0.00
10 x 10	0.00	0.79	2.99	0.00	0.17	0.63	0.00	0.31	1.55	0.00	0.00	0.00	0.00	0.61	2.52	0.00	0.00	0.00	0.00	0.60	2.99	0.00	0.00	0.00
10 x 20	0.00	0.26	0.65	0.00	0.01	0.06	0.00	0.14	0.59	0.00	0.02	0.12	0.00	0.06	0.32	0.00	0.06	0.32	0.00	0.06	0.32	0.00	0.06	0.32
20 x 5	0.69	1.84	4.37	0.00	0.11	0.54	0.00	0.71	1.74	0.00	0.28	1.20	0.76	0.71	5.77	0.00	0.28	1.14	0.00	1.20	3.29	0.00	0.17	0.63
Average	0.17	0.79	2.26	0.00	0.07	0.31	0.00	0.29	0.97	0.00	0.08	0.33	0.19	0.35	2.15	0.00	0.09	0.36	0.00	0.63	2.14	0.00	0.06	0.24

Table 6.4 : Comparaison de résultats - instances de moyennes à grandes tailles : IDPSO_NOLS / IDPSO et AG

Problème	UAF1						UAF2						UAF3						NUA					
	GA		IDPSO_NOLS		IDPSO		GA		IDPSO_NOLS		IDPSO		GA		IDPSO_NOLS		IDPSO		GA		IDPSO_NOLS		IDPSO	
	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI	MRI	ARI
20 x 10	5.54	3.93	6.37	4.14	6.37	4.86	5.25	2.56	5.54	3.49	5.66	3.80	7.82	4.37	7.14	4.26	8.29	5.08	9.42	5.20	9.00	6.33	9.48	6.70
20 x 20	2.51	1.93	3.65	2.72	5.36	3.24	3.22	1.72	3.38	2.35	3.71	2.82	3.28	2.17	2.51	1.94	3.30	2.84	3.56	1.97	3.37	2.17	4.18	2.71
50 x 5	7.45	6.72	6.99	6.19	7.58	6.89	7.16	4.41	6.81	4.45	7.29	4.87	7.38	5.13	8.50	5.72	8.50	6.09	10.25	5.37	9.12	6.04	10.15	7.28
50 x 10	5.78	4.00	6.67	4.29	6.67	4.65	5.15	3.56	5.99	3.35	6.80	4.38	7.69	3.83	7.26	4.10	9.18	6.23	7.00	4.54	6.95	5.07	7.71	5.91
50 x 20	4.77	2.25	4.03	2.48	5.34	4.07	4.07	2.75	4.00	2.76	4.96	3.36	5.08	2.65	4.14	2.31	5.63	3.34	6.22	2.43	4.06	2.36	5.82	3.21
100 x 5	6.51	4.67	7.20	4.63	7.87	5.69	6.61	2.43	7.57	3.13	7.75	3.33	7.81	5.12	8.01	5.78	8.01	6.18	7.07	5.72	7.88	6.42	7.88	6.80
100 x 10	7.08	4.04	8.66	5.27	8.66	5.63	4.54	3.51	7.02	4.25	8.09	4.74	7.87	5.27	8.11	4.79	8.11	5.46	7.55	4.28	7.76	4.88	7.76	5.10
100 x 20	7.48	4.20	6.83	4.63	6.86	4.66	2.75	1.06	1.65	0.85	3.22	1.54	4.97	2.24	7.87	6.75	5.79	2.99	5.38	2.83	3.71	2.70	5.13	3.21
Average	5.89	3.97	6.30	4.29	6.84	4.96	4.84	2.75	5.25	3.08	5.94	3.61	6.49	3.85	6.69	4.46	7.10	4.78	7.06	4.04	6.48	4.50	7.26	5.11

Conclusions et perspectives

Les enjeux de la productivité et de l'efficacité imposent de focaliser la réflexion sur les contraintes fréquemment rencontrées dans la pratique industrielle et surtout celles qui sont susceptibles de ralentir le processus de production si elles ne sont pas prises en compte. En particulier, les contraintes de ressources non-renouvelables qui ont une importance irrévocable puisque leur indisponibilité permet de perturber et affecté l'efficacité du processus de fabrication. Ce cas se produit fréquemment dans la pratique et constitue la principale motivation de cette thèse.

Les travaux effectués dans cette thèse contribuent aux études menées sur les problèmes d'ordonnancement sous contraintes de ressources non-renouvelables. La recherche bibliographique réalisée a montré une littérature limitée dans la prise en compte de cette contrainte dans les problèmes d'ordonnancement et en particulier dans les problèmes d'ateliers (Flow-Shop, Job-Shop, Open-Shop).

L'environnement de production Flow-Shop à une grande pertinence en ingénierie, représentant près du quart des systèmes de production, ceci nous a motivés à considérer les contraintes de ressources non-renouvelables dans cet environnement. Dans cette optique, nos travaux sont articulés autour des problèmes d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables pour la minimisation du makespan. Au meilleur de notre connaissance, ce problème n'a pas été traité dans la littérature auparavant.

Selon notre recherche bibliographique, nous avons également remarqué que la majorité des travaux proposent soit des résultats de complexité [44, 128], soit des algorithmes d'approximation [52, 54]. Cependant, il n'y a que des résultats de calcul sporadiques sur ce problème. Ceci nous a encore motivé à implémenter des méthodes exactes et des méthodes approchés afin de fournir des résultats de calcul.

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

Le dernier chapitre de cette partie est consacré à un état de l'art où une description de la plu-

part des importants travaux de recherche effectués sur le problème d'ordonnancement sous contraintes de ressources non-renouvelables a été fournie.

Selon l'état de l'art, nous avons noté un manque d'études concernant le problème Flow-Shop sous contraintes de ressources non-renouvelables. C'est pourquoi nous avons proposé de résoudre ce problème dans cette thèse.

La résolution du problème d'ordonnancement dans un atelier Flow-Shop avec contraintes de ressources non-renouvelables a fait l'objet de la deuxième partie de cette thèse.

Comme première attaque à ce problème, nous avons construit un modèle mathématique basé sur la programmation linéaire en nombres entiers. Ce modèle a été formulé et validé par le solveur CPLEX , les résultats obtenus par ce dernier montrent que le modèle proposé peut résoudre de manière exacte jusqu'à 20 jobs et 5 machines dans la limite du temps (1800s) quel que soit la configuration de ressources de non-renouvelables considérée. Cette modélisation nous a permet d'une part de résoudre de manière optimale les instances de petites tailles et d'autre part d'évaluer la qualité des méthodes approchées développées.

Comme les méthodes approchées constituent une alternative appropriée aux méthodes exactes pour la résolution des problèmes d'optimisation combinatoire NP-difficile, nous avons choisi de développer dans cette thèse des méthodes d'optimisation efficaces, basées sur l'algorithme génétique, algorithme d'optimisation par essaims particulaires et l'hybridation de ces deux derniers avec une procédure de recherche locale basée sur l'heuristique NEH pour résoudre le problème d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

Nous avons décidé d'implémenter un algorithme génétique grâce aux résultats obtenus par les différents problèmes d'ordonnancement et plus particulièrement par le problème de machines parallèles sous contraintes de ressources consommables proposé par Belkaid et al. [8]. Concernant l'algorithme d'optimisation par essaims particulaires, nous avons aussi trouvé une augmentation sur son application aux problèmes d'ordonnancement et spécialement dans le problème de type Flow-Shop.

L'algorithme d'optimisation par essaims particulaires présente quelques similarités avec l'algorithme génétique dans le sens qu'il fonctionne sur la base d'une population de solutions qui interagissent entre elles afin de trouver la meilleure solution possible à un problème d'optimisation. Cependant, contrairement à l'algorithme génétique où l'évolution est basée sur la compétition et l'élimination des individus les moins performants, l'algorithme d'optimisation par essaims particulaires se base sur la coopération entre les individus afin de faire évoluer. Chaque individu dans cet algorithme utilise les connaissances que ses voisines possèdent du milieu afin de se déplacer efficacement et de s'améliorer (aucune solution n'est éliminée.).

De cette façon, avec ces méthodes proposées, nous avons étudié différentes façons pour explorer l'espace de solution de notre problème d'ordonnancement de type Flow-Shop avec contraintes de ressources non-renouvelables.

Les méthodes de résolution proposées ont été comparées entre-elles. Cette comparaison a été faite sur différentes configurations de disponibilité de ressources non-renouvelables et sur des instances générées à partir d'une procédure proposée dans le chapitre 4 de la thèse. Les résultats de calcul ont démontré la supériorité de l'algorithme d'optimisation par essaims particulaires pour résoudre les problèmes d'ordonnancement Flow-Shop sous contraintes de ressources non-renouvelables.

Les résultats que nous avons obtenus sont prometteurs et encourageants. Il serait donc intéressant de développer certains aspects dans les travaux futurs que nous envisageons.

— Prise en compte de nouveaux critères à optimiser ou de nouvelles contraintes pouvant rendre le problème étudié plus réaliste.
— Généralisation des approches développées à d'autres problèmes d'ordonnancement (e.g., Flow-Shop hybride, Job-Shop, etc.).
— Étude du cas où le nombre de ressources non-renouvelables est supérieurs (respectivement inférieur) au nombre de machines, ainsi le cas où les tâches consomment plus d'une ressource non-renouvelable par machine.
— Les travaux futurs pourraient également se concentrer sur l'exploration des caractéristiques propres du problème, telles que l'étude de complexité des cas particuliers (e.g., que se passe-t-il si les tâches ont des temps de traitement unitaires ou égaux ?, etc)

Les travaux présentés dans cette thèse ont donné lieu à plusieurs publications : dans des revues internationales ([76, 77]), et à des communications dans des conférences internationales ([73–75].

Bibliographie

[1] Riad Aggoune. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. PhD thesis, Metz, 2002.

[2] Ekaterina Alekseeva, Mohand Mezmaz, Daniel Tuyttens, and Nouredine Melab. Parallel multi-core hyper-heuristic grasp to solve permutation flow-shop problem. *Concurrency and Computation : Practice and Experience*, 29(9), 2017.

[3] Davide Anghinolfi and Massimo Paolucci. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193(1) :73–85, 2009.

[4] 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.

[5] Younes BAHMANI. *Optimisation multicritère de l'ordonnancement des activités de la production et de la maintenance intégrées dans un atelier Job Shop*. PhD thesis, Université Mustapha Ben Boulaid Batna 2, 2017.

[6] Kenneth R Baker, Eugene L Lawler, Jan Karel Lenstra, and Alexander HG Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 31(2) :381–386, 1983.

[7] Joaquín Bautista-Valhondo and Rocío Alfaro-Pozo. Mixed integer linear programming models for flow shop scheduling with a demand plan of job types. *Central European Journal of Operations Research*, pages 1–19, 2018.

[8] F. Belkaid, F. Yalaoui, and Z. Sari. An efficient approach for the reentrant parallel machines scheduling problem under consumable resources constraints. *International Journal of Information Systems and Supply Chain Management*, 9 :1–25, 2016.

[9] F. Belkaid, F. Yalaoui, and Z. Sari. *Investigations on Performance Evaluation of Scheduling Heuristics and Metaheuristics in a Parallel Machine Environment*, pages 191–222. Springer International Publishing, 2016.

[10] Richard Bellman. Dynamic programming. 1957.

[11] Imène Benkalai, Djamal Rebaine, Caroline Gagné, and Pierre Baptiste. Improving the migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent set-up times. *International Journal of Production Research*, 55(20) :6145–6157, 2017.

[12] Jacek Błażewicz, Joachim Breit, Piotr Formanowicz, Wiesław Kubiak, and Günter Schmidt. Heuristic algorithms for the two-machine flowshop with limited machine availability. *Omega*, 29(6) :599–608, 2001.

[13] Jacek Blazewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Handbook on scheduling : from theory to applications.* Springer Science & Business Media, 2007.

[14] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3) :268–308, 2003.

[15] Christian Blum, Andrea Roli, and Enrique Alba. An introduction to metaheuristic techniques. *Parallel Metaheuristics : A New Class of Algorithms*, 47 :1, 2005.

[16] Peter Brucker and P Brucker. *Scheduling algorithms*, volume 3. Springer, 2007.

[17] Peter Brucker and Sigrid Knust. Complexity results for scheduling problems., 2009.

[18] J. Carlier. Problemes d'ordonnancements a contraintes de ressources : algorithmes et complexité. *These d'etat en science.*, Universitté Pierre et Marie Curie (Paris VI), 1984.

[19] Jacques Carlier. Ordonnancements a contraintes disjonctives. *RAIRO-Operations Research*, 12(4) :333–350, 1978.

[20] Jacques Carlier and Ismaïl Rebaï. Two branch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90(2) :238–251, 1996.

[21] S. Carrera. Planification et ordonnancement de plateformes logistiques. *These doctorat.*, Ecole doctorale IAEM Lorraine, 2010.

[22] Chuen-Lung Chen, Venkateswara S Vempati, and Nasser Aljaber. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80(2) :389–396, 1995.

[23] Jen-Shiang Chen, Jason Chao-Hsien Pan, and Chien-Min Lin. A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. *Expert systems with applications*, 34(1) :570–577, 2008.

[24] Chengbin Chu. *Nouvelles approches analytiques et concept de mémoire artificielle pour divers problèmes d'ordonnancement.* PhD thesis, Université de Metz, 1990.

[25] M Cochand, D de Werra, and R Slowinski. Preemptive scheduling with staircase and piecewise linear resource availability. *Zeitschrift für Operations Research*, 33(5) :297–313, 1989.

[26] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[27] Antonio Costa, Fulvio Antonio Cappadonna, and Sergio Fichera. A hybrid genetic algorithm for minimizing makespan in a flow-shop sequence-dependent group scheduling problem. *Journal of Intelligent Manufacturing*, 28(6) :1269–1283, 2017.

[28] George Dantzig, Ray Fulkerson, and Selmer . Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4) :393–410, 1954.

[29] George Bernard Dantzig. Linear programming and extensions. 1963.

[30] D De Werra, J Błażewicz, and W Kubiak. A preemptive open shop scheduling problem with one resource. *Operations Research Letters*, 10(1) :9–15, 1991.

[31] Dominique de Werra and Jacek Blazewicz. Some preemptive open shop scheduling problems with a renewable or a nonrenewable resource. *Discrete applied mathematics*, 35(3) :205–219, 1992.

[32] Suash Deb, Zhonghuan Tian, Simon Fong, Rui Tang, Raymond Wong, and Nilanjan Dey. Solving permutation flow-shop scheduling problem by rhinoceros search algorithm. *Soft Computing*, pages 1–10, 2018.

[33] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1) :137 – 141, 1998.

[34] Marco Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[35] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[36] Nicolas Durand. *Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien*. PhD thesis, INPT, 2004.

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

[38] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43, 1995.

[39] Kan Fang, Nelson A Uhan, Fu Zhao, and John W Sutherland. Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research*, 206(1) :115–145, 2013.

[40] Victor Fernandez-Viagas and Jose M Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45 :60–67, 2014.

[41] Victor Fernandez-Viagas, Rubén Ruiz, and Jose M Framinan. A new vision of approximate methods for the permutation flowshop to minimise makespan : State-of-the-art and computational evaluation. *European Journal of Operational Research*, 257(3) :707–721, 2017.

[42] J.M. Framinan, R. Leisten, and R.R. García. *Manufacturing Scheduling Systems : An Integrated View on Models, Methods and Tools*. SpringerLink : Bücher. Springer London, 2014.

[43] Jose M Framinan, Jatinder ND Gupta, and Rainer Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12) :1243–1255, 2004.

[44] E. R. Gafarov, A. Lazarev, and F. Werner. Single machine scheduling problems with financial resource constraints : Some complexity results and properties. *Mathematical Social Sciences*, 62 :7–13, 2011.

[45] M . R. Garey and D. S. Johnson. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 2 :117–129, 1976.

[46] M. R. Garey and D. S. Johnson. Computers and intractability : A guide to the theory of np-completeness. *W. H. Freeman and Co. New York, NY, USA*, 1979.

[47] Fred Glover. Tabu search : A tutorial. *Interfaces*, 20(4) :74–94, 1990.

[48] Fred Glover and Manuel Laguna. Tabu search. 1997.

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

[50] Ralph E Gomory et al. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5) :275–278, 1958.

[51] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete mathematics*, 5 :287–326, 1979.

[52] A. Grigoriev, M. Holthuijsen, and J. Klundert. Basic scheduling problems with raw material constraints. *Naval Research Logistics*, 52 :527–535, 2005.

[53] P. György and T. Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17 :135–144, 2014.

[54] Péter Györgyi and Tamás Kis. Approximation schemes for parallel machine scheduling with non-renewable resources. *European Journal of Operational Research*, 258(1) :113 – 123, 2017.

[55] JM Harrington. Shift work and health–a critical review of the literature on working hours. *Annals of the Academy of Medicine, Singapore*, 23(5) :699–705, 1994.

[56] Jack Heller. Some numerical experiments for an m× j flow shop and its decision-theoretical aspects. *Operations Research*, 8(2) :178–184, 1960.

[57] JH Holland and D Goldberg. Genetic algorithms in search, optimization and machine learning. *Massachusetts : Addison-Wesley*, 1989.

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

[59] Srikanth K Iyer and Barkha Saxena. Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers & Operations Research*, 31(4) :593–606, 2004.

[60] Srikanth K. Iyer and Barkha Saxena. Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers & Operations Research*, 31(4) :593 – 606, 2004.

[61] James R Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, CALIFORNIA UNIV LOS ANGELES NUMERICAL ANALYSIS RESEARCH, 1955.

[62] Bassem Jarboui, Mansour Eddaly, and Patrick Siarry. A hybrid genetic algorithm for solving no-wait flowshop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 54(9) :1129–1143, 2011.

[63] Bassem Jarboui, Saber Ibrahim, Patrick Siarry, and Abdelwaheb Rebai. A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers & Industrial Engineering*, 54(3) :526–538, 2008.

[64] Georges Javel. *Organisation et gestion de la production-4e édition : Cours, exercices et etudes de cas.* Dunod, 2010.

[65] S. M. Johnson. Optimal two and three stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1 :61–68, 1954.

[66] Imad Kacem. *Ordonnancement multicritère des job-Shop flexibles : formulation, bornes inférieures et approche évolutionniste coopérative.* PhD thesis, 2003. Thèse de Doctorat, Université des Sciences et Techniques de Lille1.

[67] P. J. Kalczynski and J. Kamburowski. An improved neh heuristic to minimize makespan in permutation flow shops. *Computers and Operations Research*, 35 :3001–3008, 2008.

[68] Pawel Jan Kalczynski and Jerzy Kamburowski. On the neh heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1) :53–60, 2007.

[69] AHG Rinnooy Kan. *Machine scheduling problems : classification, complexity and computations*. Martinus Nijhoff, The Hague, 1976.

[70] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[71] J Kennedy and R Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

[72] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[73] I. Laribi, F. Yalaoui, F. Belkaid, and Z. Sari. Heuristics for solving flow shop problem under resources constraints. *IFAC-PapersOnLine*, 12 :1478–1483, 2016.

[74] I. Laribi, F. Yalaoui, F. Belkaid, and Z. Sari. Metaheuristics for non-renewable resources constraints flow shop scheduling problems. In *12th Metaheuristics International Conference (MIC 2017)*, 2017.

[75] Imane Laribi, Fayçal Belkaid, Zaki Sari, and Farouk Yalaoui. Investigation pour la résolution des problèmes flow shop de permutation sous contrainte de ressource. In *Xème Conférence Internationale : Conception et Production Intégrées*, 2015.

[76] Imane Laribi, Farouk Yalaoui, and Zaki Sari. An improved discrete particle swarm optimization algorithm for flow shop scheduling problem under non-renewable resources constraints. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 2018. Soumis.

[77] Imane Laribi, Farouk Yalaoui, and Zaki Sari. Permutation flow shop scheduling problem under non-renewable resources constraints. *Int. J. Mathematical Modelling and Numerical Optimisation*, 2018. Forthcoming.

[78] Eunice Adjarath Lemamou. *Ordonnancement de projet sous contraintes de ressources à l'aide d'un algorithme génétique à croisement hybride de type OEP*. Université du Québec à Chicoutimi, 2009.

[79] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[80] Julien Lepagnot. *Conception de métaheuristiques pour l'optimisation dynamique : application à l'analyse de séquences d'images IRM.* PhD thesis, Université Paris-Est, 2011.

[81] 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.

[82] Joseph YT Leung. *Handbook of scheduling : algorithms, models, and performance analysis.* Chapman and Hall/CRC, Boca Raton, Florida, 2004.

[83] Zhigang Lian, Xingsheng Gu, and Bin Jiao. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied mathematics and computation*, 175(1) :773–785, 2006.

[84] Zhigang Lian, Xingsheng Gu, and Bin Jiao. A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals*, 35(5) :851 – 861, 2008.

[85] Ching-Jong Liao, Chao-Tang Tseng, and Pin Luarn. A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, 34(10) :3099–3111, 2007.

[86] Ching-Jong Liao and Chii-Tsuen You. An improved formulation for the job-shop scheduling problem. *Journal of the Operational Research Society*, 43(11) :1047–1054, 1992.

[87] Pierre Lopez and François Roubellat. *Ordonnancement de la production.* Hermès science publications, 2001.

[88] Alan S Manne. On the job-shop scheduling problem. *Operations Research*, 8(2) :219–223, 1960.

[89] Mohammad Mahdavi Mazdeh and Mohammad Rostami. A branch-and-bound algorithm for two-machine flow-shop scheduling problems with batch delivery costs. *International Journal of Systems Science : Operations & Logistics*, 1(2) :94–104, 2014.

[90] 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) :1087–1092, 1953.

[91] Michel Minoux. Programmation mathématique, théorie et algorithmes, volume 2 of collection technique et scientifique des télécommunications, 1983.

[92] Thomas Morton and David W Pentico. *Heuristic scheduling systems : with applications to production systems and project management*, volume 3. John Wiley & Sons, 1993.

[93] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826 :1989, 1989.

[94] Tadahiko Murata, Hisao Ishibuchi, and Hideo Tanaka. Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4) :1061 – 1071, 1996.

[95] Bahman Naderi and Rubén Ruiz. The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4) :754–768, 2010.

[96] Marcelo Seido Nagano, Adriano Seiko Komesu, and Hugo Hissashi Miyata. An evolutionary clustering search for the total tardiness blocking flow shop problem. *Journal of Intelligent Manufacturing*, pages 1–15, 2017.

[97] M. Nawaz, E. Enscore, and I. Ham. A heuristic algorithm for the m-machine , n-job flowshop sequencing problem. *The International Journal of Management Science*, 11 :91–95, 1983.

[98] Andreas C Nearchou. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88(2) :191–203, 2004.

[99] Eugeniusz Nowicki and Czesław Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1) :160–175, 1996.

[100] Ibrahim H Osman and Gilbert Laporte. Metaheuristics : A bibliography, 1996.

[101] Ibrahim H Osman and CN Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6) :551–557, 1989.

[102] Ansis Ozolins. Improved bounded dynamic programming algorithm for solving the blocking flow shop problem. *Central European Journal of Operations Research*, pages 1–24, 2017.

[103] D. S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining a near optimum. *Operational Research Quarterly*, 16 :101–107, 1965.

[104] Quan-Ke Pan, Ling Wang, M Fatih Tasgetiren, and Bao-Hua Zhao. A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, 38(3-4) :337–347, 2008.

[105] Maurice Pillet. *Introduction aux plans d'expériences par la méthode Taguchi*. Editions d'Organisation, 1994.

[106] Michael Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Springer, 2016.

[107] Chris N Potts, David B Shmoys, and David P Williamson. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10(5) :281–284, 1991.

[108] Ph Preux and E-G Talbi. Towards hybrid evolutionary algorithms. *International transactions in operational research*, 6(6) :557–570, 1999.

[109] Jakob Puchinger and 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.

[110] H. F. Rahman, R. Sarker, and D. Essam. A genetic algorithm for permutation flow shop scheduling under make to stock production system. *Computers & Industrial Engineering*, 90 :12 – 24, 2015.

[111] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1) :5 – 13, 1995.

[112] Colin R Reeves. A genetic algorithm for flowshop sequencing. *Computers & operations research*, 22(1) :5–13, 1995.

[113] Imma Ribas, Ramon Companys, and Xavier Tort-Martorell. Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, 74 :41–54, 2017.

[114] Débora P Ronconi and Luís RS Henriques. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. *Omega*, 37(2) :272–281, 2009.

[115] Ranjit K Roy. *Design of experiments using the Taguchi approach : 16 steps to product and process improvement*. John Wiley & Sons, 2001.

[116] R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34 :461–476, 2006.

[117] Rubén Ruiz and Concepción Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2) :479–494, 2005.

[118] Rachid Sabre. Plans d'expériences-méthode de taguchi. 2007.

[119] Sadiq M Sait and Habib Youssef. *Iterative computer algorithms with applications in engineering : solving combinatorial optimization problems*. IEEE Computer Society Press, 1999.

[120] H. Samarghandi. A particle swarm optimisation for the no-wait flow shop problem with due date constraints. *International Journal of Production Research*, 53(9) :2853–2870, 2015.

[121] R. Slowinski. Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15 :366–373, 1984.

[122] Wayne E Smith. Various optimizers for single-stage production. *Naval Research Logistics (NRL)*, 3(1-2) :59–66, 1956.

[123] Eric Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1) :65–74, 1990.

[124] Eric Taillard. Benchmarks for basic scheduling problems. *European journal of operational research*, 64(2) :278–285, 1993.

[125] E-G Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5) :541–564, 2002.

[126] M Fatih Tasgetiren, Yun-Chia Liang, Mehmet Sevkli, and Gunes Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European journal of operational research*, 177(3) :1930–1947, 2007.

[127] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria scheduling : theory, models and algorithms*. Springer Science & Business Media, 2006.

[128] A. Toker, S. Kondakci, and N. Erkip. Scheduling under a non-renewable resource constraint. *Operational Research Society*, 42 :811–814, 1991.

[129] A. Toker, S. Kondakci, and N. Erkip. Job shop scheduling under a non-renewable resource constraint. *Operational Research Society*, 45 :942–947, 1994.

[130] W. Trabelsi, C. Sauvey, and N. Sauer. Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. *Computers and Operations Research*, 39(11) :2520–2527, 2012.

[131] Fan T Tseng, Edward F Stafford Jr, and Jatinder ND Gupta. An empirical analysis of integer programming formulations for the permutation flowshop. *Omega*, 32(4) :285–293, 2004.

[132] Lin-Yu Tseng and Ya-Tai Lin. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1) :84–92, 2009.

[133] Ji Ung Sun. Dynamic programming approach to a two machine flow shop sequencing with two-step-prior-job dependent setup times. 5 :126–131, 01 2007.

[134] Eva Vallada, Rubén Ruiz, and Jose M Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3) :666–677, 2015.

[135] GIARD Vincent. Gestion de la production. *Economica, Paris*, 1988.

[136] Harvey M Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2) :131–140, 1959.

[137] Hongfeng Wang, Min Huang, and Junwei Wang. An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs. *Journal of Intelligent Manufacturing*, pages 1–10, 2018.

[138] Ling Wang, Quan-Ke Pan, and M Fatih Tasgetiren. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering*, 61(1) :76–83, 2011.

[139] Ling Wang, Liang Zhang, and Da-Zhong Zheng. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research*, 33(10) :2960–2971, 2006.

[140] JM Wilson. Alternative formulations of a flow-shop scheduling problem. *Journal of the Operational Research Society*, 40(4) :395–399, 1989.

[141] Cathy Wolosewicz. *Approche intégrée en planification et ordonnancement de la production.* PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2008.

[142] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1) :67–82, 1997.

[143] J. Xie. Polynomial algorithms for single machine scheduling problems with financial constraints. *Operations Research Letters*, 21 :39–42, 1997.

[144] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE, 2009.

[145] GI Zobolas, Christos D Tarantilis, and George Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4) :1249–1267, 2009.