Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tlemcen
Faculté de Technologie
Département de Génie Electrique et Electronique

Thèse de Doctorat en Productique

Intitulée :

Investigation sur l'ordonnancement des systèmes à machines parallèles

Présentée le : **Mardi 14 Janvier 2014**

Par :

Fayçal BELKAID

Devant le Jury :

Président :		
Mohammed Amine CHIKH	Professeur	Université de Tlemcen, Algérie
Examinateurs :		
Farouk YALAOUI	Professeur	Université de Technologie de Troyes, France
Hassane ALLA	Professeur	Université de Technologie de Grenoble, France
Yasmina KERBOUA-ZIARI	Professeur	Université des Sciences et de la Technologie Houari-Boumediene, Algérie
Directeur de Thèse :		
Zaki SARI	Professeur	Université de Tlemcen, Algérie

Table des matières

Remerciements

Je tiens tout d'abord à adresser mes remerciements les plus sincères et mon profond respect à mon Directeur de thèse, Monsieur ***Zaki Sari***, Professeur à l'université de Tlemcen, d'avoir participé à ma formation universitaire et à mon apprentissage de la recherche, pour son précieux encadrement, d'avoir su me guider avec attention et de m'avoir fait profiter de ses compétences et sa grande expérience. Je tiens aussi à le remercier pour sa patience, ses encouragements, ses judicieux conseils et de la confiance qu'il m'a accordée en souhaitant que ce travail soit à la hauteur de ses espérances.

Je tiens à remercier vivement Monsieur ***Mohammed Amine Chikh,*** Professeur à l'université de Tlemcen d'avoir pris le temps d'examiner cette thèse et de me faire le privilège d'en assurer la présidence. Qu'il trouve à travers ces lignes mon profond respect.

J'adresse également mes vifs remerciements à Monsieur ***Farouk Yalaoui,*** Professeur à l'université de technologie de Troyes, de me faire l'honneur d'examiner cette thèse. Je lui témoigne ma profonde gratitude pour sa collaboration dévouée, son entière disponibilité, ses chaleureux accueils dont il a fait preuve envers moi lors des deux séjours que j'ai effectués dans son laboratoire et de l'intérêt qu'il a porté à mes travaux depuis le début. Je le suis reconnaissant pour tout le savoir qu'il m'a prodigué en espérant profiter davantage de ses larges et inestimables connaissances.

Je voudrais remercier Monsieur ***Hassane Alla,*** Professeur à l'université de technologie de Grenoble, pour les louables efforts qu'il a fourni à mon égard, ses précieux conseils et ses orientations ainsi que d'avoir eu l'amabilité de juger et d'évaluer ce manuscrit, malgré ses nombreuses charges.

J'adresse également mes remerciements à Madame ***Yasmina Kerboua-Ziari***, Professeur à l'université des sciences et de la technologie Houari-Boumediene, pour ses conseils constructifs ainsi que de m'avoir honoré en acceptant de participer à mon jury de thèse et d'examiner mon travail.

Je voudrais également remercier Monsieur *Noureddine Ghouali* et Monsieur *Abdellatif Megnounif* pour leurs chaleureux accueils, leurs entières disponibilités et leurs aides tout au long de mon parcours universitaire. Ils étaient derrière la réalisation et l'avancement de cette thèse en me permettant de bénéficier de deux stages scientifiques à l'université de Technologie de Troyes (France). Une dédicace spéciale à tous les chercheurs que j'ai eu le plaisir de côtoyer durant les séjours que j'ai effectué au laboratoire d'optimisation de systèmes industriels (Troyes).

J’aimerais adresser un remerciement particulier à Mademoiselle Ghomri Latéfa et à Monsieur Hassam Ahmed, Maîtres de Conférences à l’université de Tlemcen, pour leurs sympathies et leurs soutiens tout au long de ma formation universitaire.

Je tiens particulièrement à remercier profondément les membres du Laboratoire de Productique de Tlemcen, pour les discussions que j'ai eu la chance d'avoir avec eux ainsi que pour leurs aides et encouragements qui m’ont été très bénéfiques. Cela fut un plaisir de travailler avec eux durant ces trois années et restera pour moi une expérience mémorable.

Mes remerciements les plus vifs s’adressent à toute ma famille, particulièrement à mes parents, qui m’ont toujours fait confiance, soutenu et conseillé. C’est grâce à leurs encouragements, leurs présences et leurs dévouements exemplaires que j’ai pu arriver jusque-là. Je leurs serais infiniment reconnaissant. Mes remerciements s’adressent aussi à ma sœur Chahrazed pour son aide et sa patience, et à mon frère Rachid pour ses suggestions et sa disponibilité.

Je désire aussi remercier tous les enseignants qui ont participé à ma formation universitaire et scientifique ainsi toutes les personnes ayant contribué de loin ou de prés à la réalisation de ce travail.

Enfin, je remercie Dieu tout puissant de la patience ainsi que la volonté qu’il m’a donné qui étaient indispensables durant tout mon parcours d’étude.

Introduction générale

Durant ces dernières années, la pression sur les entreprises s'est accrue en raison de l'évolution des exigences du marché. L'augmentation de la productivité va de pair avec la réduction des délais de fabrication, la diminution des prix et ceux, sans pour autant affecter le niveau de satisfaction des clients qui sont devenus très exigeants. C'est pourquoi, une entreprise doit constamment chercher à améliorer son efficacité et sa productivité, ce sont les conditions de sa survie et de son développement ; pour cela elle doit être dotée d'un système de production performant. Ce dernier représente l'ensemble des pratiques, des règles, des outils et des méthodes qui forment la culture industrielle de l'entreprise.

En outre, une bonne gestion d'un système de production est aujourd'hui une nécessité de plus en plus préoccupante pour les entreprises, l'amélioration constante de ses performances est tributaire de la gestion opérationnelle, des méthodes d'organisation et d'exploitation des ressources dont elle dispose. Un ordonnancement efficace peut être un outil essentiel pour les entreprises afin d'atteindre des performances élevées. En effet, l'ordonnancement constitue un aspect important dans la gestion de production, il consiste à définir la planification de l'utilisation des ressources disponibles (consommables et renouvelables) de façon à déterminer à quel moment traiter quelle tâche avec quelle ressource tout en réglant un ensemble de conflits et en optimisant un ou plusieurs critères.

Les problèmes d'ordonnancement avec des ressources consommables sont devenus fréquents dans la gestion des opérations. Toutefois, une partie importante des études des problèmes d'ordonnancement est placée dans un contexte où les ressources sont toujours disponibles, or ces ressources peuvent être consommés et peuvent être indisponible pour des raisons diverses, de ce fait cette hypothèse risque de ne pas être satisfaite dans de nombreuses situations pratiques et par conséquent, le manque de ressources perturbe et affecte l'efficacité de l'ordonnancement. D'autre part, il est possible que les tâches nécessitent d'être traitées plusieurs fois sur la même machine dans ce cas, ce genre de problème est dit problème d'ordonnancement avec processus réentrant.

De ce fait, il existe une multitude de configurations dans les systèmes de production actuels, ces derniers peuvent être caractérisés par des processus réentrants ou non ce qui rend les problèmes d'ordonnancement caractérisés par un grand nombre de contraintes qui sont relatives soit aux ressources (consommables ou renouvelables), soit aux tâches (préemptives ou non, indépendantes ou non). Pour faire face à toutes ces contraintes, ces systèmes sont

dans le besoin de traduire les objectifs de l'entreprise sous forme de mesures de performance (critères) qui sont souvent contradictoires (minimisation du makespan et des retards, maximisation du taux de production, etc.). Pour cela, les problèmes d'ordonnancement ne cessent de croitre en complexité et sont considérés par conséquent comme des problèmes NP-difficiles.

Dans ce contexte, il est très peu probable que nous puissions les résoudre et obtenir une solution optimale dans un temps polynomial en utilisant les méthodes exactes. Il faut alors développer des méthodes dites approchées qui permettent d'obtenir des solutions de bonnes qualités en des temps raisonnables. Parmi ces méthodes approchées, les métaheuristiques ont été approuvées pour résoudre ce type de problème. Elles ont eu un grand succès qui s'explique par leur capacité à fournir des solutions proches de l'optimum dans un délai raisonnable pour la résolution de plusieurs problèmes NP-difficiles ainsi que pour la minimisation du makespan.

Tout ceci, nous a motivé à développer, dans notre thèse une étude de recherche qui a pour objectif l'investigation sur l'ordonnancement des systèmes à machines parallèles. Elle s'articule autour de quatre chapitres :

Le premier présente un aperçu sur les problèmes d'ordonnancement dans les systèmes de production. Nous commençons par présenter les systèmes de production, ensuite nous décrivons la terminologie des problèmes d'ordonnancement et leurs importances. Nous présentons quelques études, à travers lesquelles, nous montrons la nature de ces problèmes, les différentes méthodes de résolution ainsi que la complexité de leurs processus de résolution. Nous finalisons ce chapitre, par un aperçu sur les travaux s'intéressant à l'ordonnancement des systèmes à machines parallèles avec ressources consommables et par une recherche bibliographique détaillée portant sur les investigations des problèmes d'ordonnancement réentrant.

Le deuxième est consacré à un état de l'art sur les métaheuristiques. Nous décrivons, leurs caractéristiques et leurs classifications. Ensuite nous présentons les métaheuristiques les plus répandues à savoir les approches à base d'une seule solution ou d'une population de solutions, tout en précisant leurs origines et leurs algorithmes de bases. La fin de ce chapitre apporte une importance particulière à la méthode d'optimisation basée sur les algorithmes génétiques, qui représente la principale approche de résolution de cette thèse.

Le troisième s'intéresse à l'ordonnancement sur machines parallèles avec ressources consommables. Nous nous intéressons au développement d'un programme linéaire en nombres entiers pour la résolution de ce problème. Cependant, la stratégie de résolution s'avère très complexe car les décideurs doivent faire face à des situations difficiles que même l'utilisation d'une méthode exacte reste inefficace. Pour cela, et pour y remédier nous développons aussi, dans notre étude une méthode approchée. Cette dernière repose sur l'utilisation d'une métaheuristique à base d'algorithme génétique dont le but est d'affecter dans un premier temps les tâches aux machines et dans un second temps de déterminer la

séquence des tâches assignées à la même machine afin de minimiser le makespan tout en respectant les différentes contraintes de ressources.

Le dernier volet de ce manuscrit est axé essentiellement sur l'adaptation d'un algorithme génétique dans un environnement de machines parallèles avec des ressources consommables et processus réentrant afin de minimiser le makespan et d'analyser l'impact des différentes approches étudiées sur les performances du système.

Cette thèse est clôturée par une conclusion et des perspectives par lesquelles nous présentons quelques extensions et orientations possibles pour les futures recherches.

Chapitre 1

L'ordonnancement dans les systèmes de production

1.1. Introduction

Actuellement, les environnements de production ont évolué et sont devenus de plus en plus complexes. Cette complexité dépend non seulement de l'évolution des systèmes de production mais aussi du processus de production, ce qui rend les problèmes d'ordonnancement caractérisés par un grand nombre de contraintes relatives aux ressources (consommables ou renouvelables), aux processus (réentrant ou non). Les problèmes d'ordonnancement de production sont confrontés par plusieurs entreprises manufacturières à travers le monde et de nombreux industriels se sont intéressés à leurs résolutions mais (cependant) le développement des ordonnancements de production efficaces reste une tâche difficile en raison de l'importante variante de problèmes rencontrés ce qui conduit pratiquement à un nombre incommensurable d'environnements de production. Ce contexte a suscité l'intérêt de plusieurs chercheurs et praticiens du domaine de la gestion de production et les a poussé à proposer des approches pour faire face à ces problèmes. Cependant, le développement de méthodes a porté principalement sur un nombre limité de problèmes classiques qui, la plupart du temps, ne peuvent pas être appliquées directement à des structures complexes et par conséquent, ils se sont retrouvés contraints à développer des méthodologies de résolution efficaces et robustes, qui peuvent être modifiées et appliquées à différents environnements de production.

Dans ce chapitre, nous présentons un aperçu sur les problèmes d'ordonnancement dans les systèmes de production. La section suivante vise à présenter les systèmes de production. Pour cela, nous exposons leurs définitions puis leurs typologies et finalement les différents types de décisions. La troisième section est consacrée à des généralités sur l'ordonnancement dans les systèmes de production. La quatrième section présente les problèmes d'ordonnancement dans les systèmes de production et leurs méthodologies de résolution. Finalement, les sections cinq et six donnent un aperçu sur les travaux d'ordonnancement dans un environnement de machines parallèles avec ressources consommables dans un premier temps et processus réentrant dans un second temps.

1.2. Les systèmes de production

1.2.1 Définition

Le concept de production a toujours été au centre de l'analyse économique, et sa définition a donné lieu à de grandes controverses entre les courants théoriques. Giard [Giard 88] décrit la production comme étant une transformation de ressources appartenant à un système productif et conduisant à la création de biens et de services. La production d'un bien s'effectue par une succession d'opérations consommant des ressources et transformant les caractéristiques morphologiques ou spatiales de matières conduisant à la création de produits. Un système de production est généralement vu comme l'association d'un ensemble de ressources en interaction pour réaliser une activité de production [Tamani 92]. Il se divise généralement en trois systèmes coopérants [Le Moigne, 1990].

- ✓ Le système physique de fabrication,
- ✓ Le système d'information,
- ✓ Le système de décision.

1.2.2 Les typologies des systèmes de production

La typologie des systèmes de production dépend de l'objectif à atteindre et des moyens disponibles, en effet l'organisation choisie doit permettre un équilibre adéquat entre limitations de temps, disponibilité des données et autres ressources. De plus, il existe de nombreux systèmes de production qui ont été mis en place, utilisant des critères divers et pouvant se trouver dans plusieurs classes différentes, de ce fait, il est maintenant très difficile d'établir une classification exhaustive de toutes ces caractéristiques, nous allons donc proposer un rapide survol sur des classifications qui dressent un bilan synthétique sur les aspects qui nous semble les plus intéressants.

1.2.2.1 Classification selon le mode de production

Selon Woodward [Woodward 65], Baranger [Baranger 87] et Mulkens [Mulkens 93] quatre modes de productions peuvent être définis :

- **Production unitaire** : Il s'agit de la fabrication d'un produit unique. Ce type de production fait toujours l'objet d'un grand projet. Quant à ses caractéristiques, nous citons :
  - ✓ le produit est généralement complexe, nécessite la coordination de plusieurs ressources devant intervenir simultanément ou séquentiellement afin de livrer au moment convenu.
  - ✓ la taille du produit ou la demande impose une production de très faible quantité.
  - ✓ la tâche principale de la production consiste à réunir les moyens nécessaires au bon moment et au bon endroit.
- **Production en petite et moyenne série** : Qualifiée aussi par production de lots, ce genre de systèmes est utilisé pour la fabrication des produits dits "personnalisables" c'est-à-dire en fonction des besoins du client, il s'agit ici d'ateliers dans lesquels la

diversité des produits ne permet pas une spécialisation des moyens de production. Les caractéristiques de cette classe sont les suivantes :

- ✓ Les différents produits suivent leur propre chemin sur des ressources communes de flexibilité élevée souvent regroupées par fonctionnalités équivalentes.
- ✓ le produit est plus ou moins de taille modeste pour que ce produit se déplace de machine en machine dans un même atelier ou un groupe d'ateliers. Le procédé de fabrication d'un produit est, dans ce cas, exprimé par la gamme de fabrication ou par le routage.
- ✓ Les ressources de production sont très polyvalentes, flexibles, capables de passer rapidement d'une production à une autre, quant au niveau de l'automatisation, il est généralement faible ou nul.

- **Production en grande série** : Dite aussi production de masse, ces types de systèmes sont utilisés lorsque les produits réalisés par fabrication ou bien par assemblage sont en très grande quantité mais avec peu de variantes, ils sont caractérisés par les points suivants :
  - ✓ Une chaîne de production représentée par un ensemble de machines différentes pour minimiser les temps de transfert et d'attente des pièces selon le procédé de fabrication.
  - ✓ L'ordre de passage des produits sur les ressources étant toujours le même, celles-ci peuvent être placées dans un ordre figé dépendant du produit à fabriquer.
  - ✓ Les machines doivent avoir le même rendement et le même débit pour que les problèmes puissent être résolus rapidement avant d'entraîner l'arrêt complet de la chaîne.
  - ✓ Les ressources de production (hommes et machines) sont fortement spécialisées et dédiées à des tâches précises et le niveau d'automatisation est en général élevé.
- **Production en continu** : C'est un cas particulier de la production en grande série. Elle est définie par les caractéristiques suivantes :
  - ✓ La matière circule en flux continu, c'est-à-dire que l'attente entre deux ressources est exclue ou très limitée.
  - ✓ Il n'y a pas de stocks intermédiaires et toutes les machines qui constituent cette chaîne ont le même débit.
  - ✓ Les équipements de production sont dédiés avec un niveau d'automatisation très élevé.
  - ✓ Tous les produits sont fabriqués suivant la même séquence d'opérations, à travers une succession plus ou moins longue de postes. Ce type de production est utilisé généralement dans la sidérurgie, les aciéries, les cimenteries, les stations d'épuration, les raffineries et les activités pétrolières.

1.2.2.2 Classification selon la structure du produit

Concernant cette classification, deux types de structures ont été distingués [Courtois 95] :

- **Les structures convergentes :** Les produits finis en variété limitée sont assemblés au départ d'un nombre important de composants eux-mêmes usinés ou formés à partir de matières premières très variées. Cette structure est caractérisée par une arborescence présentant plusieurs niveaux qui correspondent à des sous-ensembles du produit final (ex : industrie automobile).
- **Les Structures divergentes :** Cette structure est celle des produits réalisés à partir de la transformation d'une matière première unique avec une abondance de produits finis (ex : raffinerie de pétrole).

1.2.2.3 Classification selon la circulation des produits dans l'atelier

Dans cette classe, les modèles les plus connus sont ceux d'un atelier dont les produits circulent suivant un cheminement unique c'est à dire l'ordre de passage des différents jobs sur l'ensemble des ressources disponibles dans l'atelier est le même (Flow shop) où suivant un cheminement multiple c'est à dire que chaque job possède une gamme spécifique (Job shop) [Widmer 91].

1.2.2.4 Classification selon le mode de déclenchement de la production

Cette classification, proposée par Giard [Giard 88] se base sur la façon dont la fabrication est lancée, pour cela trois systèmes de gestion de production ont été définis :

- **Production sur stock :** Elle est déclenchée par l'anticipation d'une demande solvable s'exerçant sur un produit dont les caractéristiques sont définies par le fabricant. Ce type de production s'applique dans les cas ou l'éventail des produits finis est restreint ou lorsque la demande de chaque produit doit être importante et prévisible.
- **Production sur commande :** Une production s'effectue à la commande lorsque tout ou partie de la fabrication est déclenché par la commande ferme d'un client, on parle de fabrication à la commande quand, en réponse à une commande précise, il faut effectuer un travail de conception pouvant ou non nécessiter la création de nouveaux composants.
- **Production par programme :** Elle se base sur des commandes assurées et des prévisions de vente pour déterminer un programme de production.

1.2.2.5 Classification selon le mode de pilotage :

En se situant au niveau opérationnel et sur la base du mode de déclenchement de la production, cette classification proposée par Giard [Giard 03] sépare les systèmes fonctionnant à flux tirés de ceux fonctionnant à flux poussés :

- **Les systèmes à flux tirés :** Appelé aussi gestion sur commande ou pilotage par l'aval puisque les quantités à produire de chaque référence sont définies par la demande des clients qui vont servir à planifier le dimensionnement des stocks, du plan de production et des ressources nécessaires à la fabrication des produits. Le but étant de

satisfaire le client au plus vite et donc minimiser les coûts dus aux en-cours de stocks. Ce type de fonctionnement vise à maintenir un stock de produits finis nul.

- **Les systèmes à flux poussés :** appelé aussi gestion sur stock ou pilotage par l'amont, ici, contrairement aux flux tirés, c'est la direction qui décide des quantités à produire. Le déclenchement de l'ordre de fabrication va entraîner d'autres qui vont se succéder au fur et à mesure de l'élaboration du produit jusqu'à son stockage puis sa vente. Ce sont donc des prévisions de la demande des clients et non pas les demandes réelles des clients qui vont servir à planifier le dimensionnement des stocks, du plan de production et des ressources nécessaires à la fabrication des produits. Cette méthode de production implique souvent le stockage des produits finis avant leur livraison.

1.2.2.6 Classification selon la nature et le volume des flux physiques :

Cette classe est fondée sur le volume des produits fabriqués et sur la nature du système physique**.** Elle peut être divisée en trois autres classes [Giard 03], comme suite :

- **Système à flux continu :** Ce type de système concerne surtout les industries dont la production nécessite la manipulation des matières liquides ou gazeuses.
- **Système à flux discret :** Il regroupe les trois modes de production cités précédemment : production unitaire, production en moyenne série et production en grande série.
- **Système à flux hybride ou discontinu :** Ce système fait l'hybridation entre le système à flux continu et le système à flux discret.

Dans cette section, nous avons regroupé les différentes classifications des systèmes de production qui existent depuis ces dernières années, chacune d'entre elles se basent sur un critère qui permet de différencier un système de production d'un autre. Nous pouvons rajouter deux autres critères qui peuvent être aussi pris en considération lors d'une classification c'est la complexité et l'incertitude [Gousty 88].

1.2.3 Les différents types de décisions

Les décideurs sont confrontés à des situations difficiles qui surgissent du secteur industriel et environnement de production où plusieurs décisions doivent être prises chaque jour et aucune d'entre elles n'est identique à une autre, chacune a une incidence différente sur le fonctionnement, la rentabilité, la performance, l'activité, l'évolution et parfois même la survie de l'entreprise. De ce fait, plusieurs types de décisions se situant à différents niveaux hiérarchiques ont été mis en place, dans ce qui suit, nous allons décrire les trois plus grandes architectures décisionnelles : stratégique, tactique et opérationnel, correspondant respectivement à des horizons à long, moyen et court terme.

1.2.3.1 Les décisions stratégiques

Ce niveau représente les orientations à long terme, il concerne la partie conception de l'entreprise. Ces décisions partent de la nature et la répartition judicieuse des ressources en vue de commercialiser des produits et de conquérir des marchés selon la combinaison qui donnera le meilleur résultat au regard des critères ou objectifs retenus. Elles ont un impact

global dans la mesure où elles remettent en question l'existant au niveau de toutes les fonctions de l'entreprise. Les enjeux et les dimensions sont tellement importants qu'ils constituent en soit un véritable phénomène d'inertie. Elles sont prises dans le cadre par exemple d'ouverture ou de fermeture de certains sites de production ou leur délocalisation, d'affectation d'une nouvelle zone de marché à un centre de distribution, le développement d'un nouveau produit, la configuration d'usine, de mode de fonctionnement, etc.

1.2.3.2 Les décisions tactiques

Ces décisions représentées par les orientations à moyen terme s'inscrivent dans le cadre logique dessiné par les décisions stratégiques, elles regroupent tout ce qui est ressource physique et informationnelle nécessaires à la production. Les décisions tactiques s'intéressent à l'aspect planification de la production, l'affectation des clients aux centres de distribution, le choix de la politique de gestion des stocks, etc.

1.2.3.3 Les décisions opérationnelles

Ce sont généralement des décisions journalières, car elles ont une portée plus limitée dans le temps. Ce niveau représente les orientations à court terme. Cette catégorie vise à obtenir et à analyser sur son aspect technique, le maximum de rentabilité. L'impact opérationnel est limité sur le plan spatial au niveau de l'entreprise, il concerne bien souvent une fonction interne de l'entreprise. Les décisions opérationnelles assurent la flexibilité quotidienne pour faire face aux fluctuations prévues de la demande et la disponibilité des ressources et réagir aux aléas. Elles sont prises, dans le cas par exemple, de la gestion de stocks, de l'ordonnancement de la production, etc.

En effet, l'ordonnancement constitue une classe importante de ces décisions. Ce concept va être détaillé et va faire l'objet de notre étude.

1.3. Généralités sur l'ordonnancement dans les systèmes de production

Les problèmes d'ordonnancement apparaissent dans tous les domaines, en particulier dans les industries (activités des ateliers en gestion de production et problèmes de logistique). Nous sommes confrontés à un problème d'ordonnancement lorsqu'il s'agit d'accomplir de multiples tâches, en tenant compte des contraintes de temps et de capacité portant sur l'utilisation et la disponibilité des ressources requises pour ces tâches.

1.3.1 Définition

Il décrit l'exécution des tâches et l'allocation des ressources au fil du temps dans le but de répondre à un ou plusieurs objectifs en respectant des contraintes. Un problème d'ordonnancement est l'organisation dans le temps de l'exécution d'un ensemble de tâches, en tenant compte des contraintes de temps (délais, contraintes de précédence, etc.) et de la capacité des contraintes sur les ressources requises pour ces tâches. Dans ce qui suit, nous allons définir les éléments de base qui caractérisent un problème d'ordonnancement à savoir les tâches, les ressources, les contraintes, les objectifs à réaliser, quels sont les critères d'évaluation, et de quelle manière les problèmes d'ordonnancement peuvent être classés.

1.3.2 Concepts de base

Dans un problème d'ordonnancement interviennent deux notions fondamentales : les tâches et les ressources. La présentation de ces concepts est issue des livres d'Esquirol et Lopez [Esquirol 99] et Blazewicz [Blazewicz 07].

1.3.2.1 Les tâches

Une tâche est un travail élémentaire dont la réalisation nécessite un certain nombre d'unités de temps (sa durée) et d'unités de chaque ressource. Chaque tâche est caractérisée par une durée d'exécution et un certain nombre de ressources. Les tâches sont souvent liées entre elles par des relations d'antériorité ; dans ce cas, ces tâches sont interdépendantes, sinon elles sont indépendantes. Une tâche peut être aussi, dans certains cas, morcelable ce qui signifie qu'elle peut être exécutée par morceaux par une ou plusieurs ressources ; ou bien dans le cas contraire, la tâche ne peut pas être interrompue une fois qu'elle ait commencé. Nous parlons alors respectivement de problèmes préemptif et non préemptif.

1.3.2.2 Les ressources

Une ressource est un moyen, technique ou humain, dont la disponibilité limitée ou non est connue à priori. Deux types de ressources peuvent être distingués : les ressources renouvelables et les ressources consommables.

- **Ressource renouvelable** : Une ressource est dite renouvelable si, après avoir été affectée à une tâche, elle redevient disponible pour les autres (machine, fichier, processeur, homme, …). Parmi les ressources renouvelables, nous distinguons les ressources disjonctives, ces dernières, ne peuvent exécuter qu'une opération à la fois ; et les ressources cumulatives, qui, quant à elles peuvent exécuter, simultanément, un nombre limité d'opérations.
- **Ressource consommable** : Une ressource est dite consommable où non-renouvelable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches restantes, comme par exemple l'argent, la matière première, etc. De plus, lorsqu'une tâche n'a besoin que d'une seule ressource pour pouvoir être exécutée, on dira alors que c'est un problème d'ordonnancement mono-ressource, où, au contraire c'est un problème d'ordonnancement multi-ressources, dans le cas où l'exécution d'une tâche nécessite plusieurs ressources.

1.3.2.3 Les contraintes

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre une ou plusieurs variables de décision. Ces variables peuvent être liées à des décisions qui sont en rapport soit avec les ressources, soit avec le temps. Nous parlons respectivement de variables d'affectation et de variables temporelles. Dans ce contexte, Ben Hmida [Ben Hmida 09**]** a distingué deux types de contraintes généralement rencontrées en ordonnancement, les contraintes temporelles et les contraintes de ressources.

- **Les contraintes de ressources** : composées, à leur tour de deux types de contraintes qui sont respectivement les contraintes de capacité et les contraintes d'affectation. Pour les premières, nous distinguons :

- ✓ Les contraintes de ressource disjonctive : La contrainte exige, dans ce cas-là, la réalisation disjointe de deux tâches, en d'autres termes il n'est possible pour qu'une tâche d'utiliser une seule ressource à la fois.
- ✓ Les contraintes de ressource cumulative : La contrainte interdit pour une ressource donnée d'exécuter simultanément un certain nombre de tâches, lorsque ces dernières utilisent une quantité de la ressource excédant sa capacité.

En ce qui concerne les contraintes d'affectation, elles sont divisées en deux types de contraintes :

- ✓ Contrainte de domaine : L'ensemble des ressources candidates pour l'exécution d'une tâche doit être défini par cette contrainte.
- ✓ Contrainte de différence : Cette contrainte impose, quant à elle, l'utilisation de ressources différentes pour la réalisation d'un certain nombre de tâches.

- ➢ **Les contraintes temporelles** : Elles peuvent être différenciées en deux catégories : contraintes de temps absolu et contraintes de temps relatif.
  - ✓ Contrainte de temps absolu : Permet de définir et de gérer la cohérence temporelle des tâches. Par exemple, déterminer quand est-ce qu'une tâche doit débuter au plus tôt et quand est-ce qu'elle doit être achevée au plus tard.
  - ✓ Contrainte de temps relatif : Cette contrainte est relative aux contraintes de cohérence technologique, comme les contraintes de gammes où il faut respecter le positionnement relatif entre les tâches.

Il peut y avoir d'autres types de classification des contraintes, en plus de celles qui viennent d'être citées. Ces contraintes peuvent être liées par exemple avec au système de production, dans ce cas, elles sont soit endogènes soit exogènes [Tangour 07]. Les contraintes endogènes sont directement en rapport avec le système de production et ses performances telles que : les dates de disponibilité des machines, les capacités des machines et des moyens de transport, les séquences des actions à effectuer. Pour les contraintes exogènes, elles sont imposées extérieurement, en dehors du système de production ; comme les priorités de quelques commandes et de quelques clients, les retards possibles accordés pour certains produits. Selon Letouzey [Letouzey 01], les contraintes peuvent être strictes ou "relâchables" : les contraintes strictes sont les exigences à respecter et les contraintes dites "relâchables" peuvent éventuellement ne pas être satisfaites.

1.3.2.4 Les critères d'évaluation

La définition des ressources et des tâches permet de fixer les contraintes, or ces dernières ne suffisent pas pour résoudre les problèmes d'optimisation, il est nécessaire aussi de définir un critère d'évaluation pour juger la pertinence d'un ordonnancement satisfaisant ses contraintes du point de vue exploitation en se basant sur une fonction objectif définie par un critère que nous allons chercher à optimiser. Dans certains cas, où les objectifs de l'entreprise sont multiples, nous cherchons à optimiser plusieurs fonctions objectifs à la fois. Dans ce cas, le problème d'ordonnancement n'est plus un problème simple, mais devient un problème multicritère.

1.3.2.5 Les objectifs

Lors de la résolution d'un problème d'ordonnancement, nous pouvons choisir entre deux principaux types de stratégie, en se concentrant respectivement sur l'optimalité des solutions (par rapport à un ou plusieurs critères), ou sur leur faisabilité (en fonction des contraintes). L'approche fondée sur l'optimisation suppose que les solutions candidates à un problème peuvent être vérifiées, selon un ou plusieurs critères d'évaluation qui expriment leurs qualités. Dans ce sens, Esquirol et Lopez [Esquirol 99] subdivisent **c**es objectifs en plusieurs classes :

- ✓ Liés au temps : Lorsqu'il s'agit de la minimisation du temps total d'exécution, du temps de cycle, des durées totales de réglage ou des retards par rapport aux dates de livraison…
- ✓ Liés aux ressources : Comme par exemple pour la maximisation du taux d'utilisation d'une ressource ou la minimisation du nombre de ressources à employer…
- ✓ Liés au coût : Ces critères permettent de minimiser les coûts de lancement, de production, de stockage, de transport, du retard, de non occupation des machines…

La littérature sur les problèmes d'ordonnancement montre que de nombreux ouvrages de référence ont été publiés dans ce domaine pour la diversité de leurs champs d'application, particulièrement dans le secteur industriel. Parmi ces ouvrages, nous trouvons, [Baker 74], [Rinnooy Kan 76], [Tanaev 94], [Brucker 98], [Esquirol 99], [Lopez 01], [Leung 04], [Blazewicz 07], [Pinedo 08], **[**Pinedo 12].

1.4. Présentation des problèmes d'ordonnancement dans les systèmes de production et leurs méthodologies de résolution

1.4.1 Problèmes d'ordonnancement d'ateliers :

Un problème d'ordonnancement d'atelier consiste à déterminer la séquence de passage d'un certain nombre d'opérations à exécuter sur différentes machines. Il s'agit de prévoir le travail à réaliser, de façon à coordonner l'utilisation des ressources renouvelables et non renouvelables afin de satisfaire les différentes contraintes et d'optimiser un ou plusieurs critères de performances.

- **Classification des problèmes d'ordonnancement**

Bien que les problèmes d'ordonnancement de production possèdent des caractéristiques communes, ils peuvent être classés selon plusieurs concepts, les plus couramment utilisés : le processus d'arrivée des tâches ainsi que leurs caractéristiques, la politique de l'inventaire et le type d'atelier. Lorsqu'il s'agit de processus d'arrivée des tâches, les problèmes d'ordonnancement de production peuvent être soit statiques si tous les travaux arrivent en même temps ou dynamiques si les jobs arrivent par intermittence. Ces problèmes peuvent être également classés comme déterministes si les temps de traitement des tâches et la disponibilité de la machine sont connus a priori ou, au contraire, stochastiques. En ce qui concerne la politique de l'inventaire, un problème pourrait être, dans ce cas, ouvert si tous les produits sont fabriqués à la commande ou fermé si tous les produits sont fabriqués pour être stockés.

En plus de ce qu'on vient de citer, il existe une autre classification indispensable, qui se base sur le type d'atelier. Pour cela, dans la section suivante, nous allons nous intéresser aux problèmes appartenant à cette classe et à leurs méthodologies de résolution. Nous allons discuter des problèmes d'ordonnancement à une seule machine et à machines parallèles. Dans le premier cas, chaque travail est réduit à une seule opération, il faut alors trouver une séquence de travail sur la machine unique. Dans le second cas, nous présentons les problèmes de machines parallèles qui utilisent la même hypothèse de travail d'une seule opération et dont la solution est un ensemble de séquences. Nous allons ensuite étudier les problèmes d'ordonnancement multi-machines où les travaux contiennent plusieurs opérations. Ces problèmes peuvent être répartis en trois catégories qui sont flow shop, job shop et enfin open shop.

1.4.1.1 Problème d'ordonnancement à une machine (single machine problem)

Dans ce genre de problème, chacune des tâches doit être réalisée par une seule machine. Les tâches sont alors composées d'une seule opération qui nécessite la même machine pour être exécutée.

1.4.1.2 Problème d'ordonnancement à machine parallèle (parallel machines scheduling problem)

Il représente une généralisation du problème d'ordonnancement à une machine, les travaux se composent d'une seule opération et chaque opération dispose d'un ensemble de machines parallèles, mais n'en nécessite qu'une pour pouvoir être réalisée. L'ordonnancement s'effectue en deux phases : la première consiste à affecter les opérations aux machines et la deuxième détermine la séquence de réalisation sur chaque machine.

1.4.1.3 Problème d'ordonnancement à cheminement unique (Flow shop)

Dans ce cas, une ligne de fabrication est composée de plusieurs machines disposées en série ; une tâche est alors constituée de plusieurs opérations qui passent par différentes machines, de manière linéaire, c'est-à-dire dans le même ordre.

1.4.1.4 Problème d'ordonnancement à cheminement multiple (Job shop)

Contrairement aux problèmes d'ordonnancement Flow shop, les tâches ne s'exécutent pas sur toutes les machines selon un ordre bien déterminé, chaque tâche peut emprunter son propre chemin.

1.4.1.5 Problèmes d'atelier multi-machines à cheminements libres (Open shop)

Dans ce type de problème, les opérations nécessaires à la réalisation de chaque tâche peuvent être exécutées dans un ordre quelconque. Autrement dit, ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre.

1.4.2 Problèmes d'ordonnancement réentrant

L'atelier de production réentrant représente des configurations dont les tâches peuvent être traitées plusieurs fois sur la même machine. Le problème d'ordonnancement réentrant est l'un des problèmes opérationnels les plus fréquemment rencontrés dans l'industrie. Selon

Kumar (1993), ce problème ne peut pas être considéré ni comme un flow shop avec son routage acyclique fixe ni comme un job shop avec son routage aléatoire. Mais seulement comme un problème qui a comme caractéristique unique de traiter les tâches plus d'une fois. Les tâches peuvent retourner au traitement pour réaliser la même opération ou pour des problèmes de qualité. Selon Lin et Carman [Lin 11] l'ordonnancement dans les systèmes réentrants se caractérise par :

- ✓ L'évitement du blocage
- ✓ La répartition des capacités
- ✓ Le contrôle d'inventaire
- ✓ La conception du système.

- **Classification**

Nous pouvons distinguer plusieurs classifications des problèmes d'ordonnancement. Selon Bellman et Ernest's [Bellman 82], ces problèmes peuvent être classés selon le nombre des machines, le processus des commandes et l'ordre des travaux. Lin et Carman [Lin 11] pensent qu'à travers les caractéristiques des problèmes d'ordonnancement réentrant « *Reentrant Scheduling Problem* » (RSP), leur classification peut être faite comme suit :

- ➢ Problème général d'ordonnancement réentrant « *General Reentrant Scheduling Problem* » (GRSP) : Cette classe regroupe les problèmes qui sont généralement applicables, pas dans un domaine particulier, tels que l'ordonnancement des ressources, la localisation des entrepôts…
- ➢ Problème d'ordonnancement d'ateliers réentrants « *Reentrant Shop Scheduling Problem* » : Il concerne les problèmes qui se produisent dans la manutention du matériel dans les ateliers de production. Selon Lin et Carman [Lin 11], ils existent plusieurs autres classifications différentes de celle qui vient d'être cité. La littérature sur les problèmes d'ordonnancement réentrant montre que cette dernière classe peut être classés selon le type d'atelier utilisé classe peut être encore divisée en deux types d'ateliers qui sont flow shop réentrant « *Reentrant Flow-shop Scheduling Problem* » et job-shop réentrant « *Reentrant Job-shop Scheduling Problem* ». La figure 1.1 illustre la classification proposée par Lin et Carman [Lin 11].

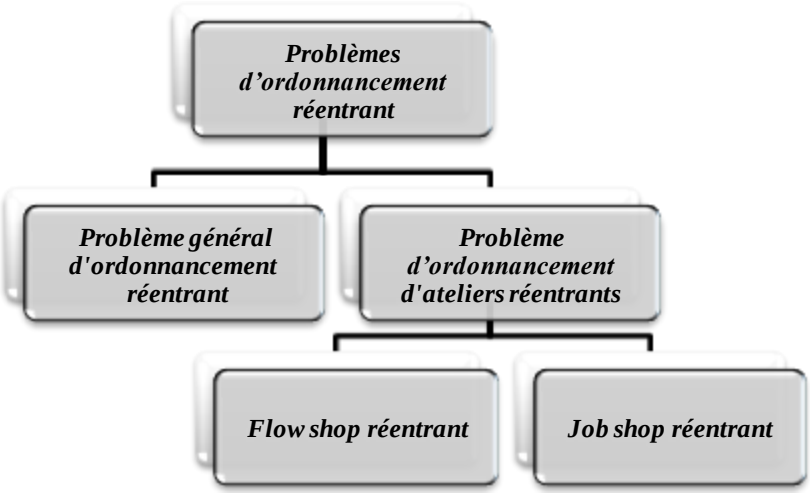

Figure 1.1 : Classification proposée par Lin et Carman [Lin 11]

1.4.3 Complexité des problèmes d'ordonnancement

Les systèmes de productions sont généralement complexes, ils sont caractérisés par différents objectifs à atteindre et plusieurs contraintes à satisfaire ce qui complique leurs problèmes d'ordonnancement. D'autre part, les systèmes définis par un processus réentrant réintègrent les machines de traitement plus d'une fois, ce qui augmente leurs complexités. La théorie de la complexité porte sur l'analyse formelle de la difficulté théorique des problèmes d'optimisations. La complexité d'un problème est équivalente à déterminer le meilleur algorithme pour le résoudre. Concrètement, nous cherchons à savoir si pour un problème étudié, il est possible de trouver un algorithme capable de le résoudre en un temps polynomial. Donc, l'analyse de la complexité d'un algorithme peut être définie comme un indicateur de performance permettant de comparer les différents algorithmes réalisant les mêmes fonctionnalités pour trouver une solution à un problème donné [Karp 72], [Garey 79], elle dépend de deux facteurs essentiels : le temps de calcul des opérations effectuées par l'algorithme et l'espace nécessaire pour stocker les différentes données.

Un aspect important de la théorie computationnelle est de catégoriser les problèmes dans les classes de complexité. Une classe de complexité représente l'ensemble de tous les problèmes qui peuvent être résolus en utilisant des programmes et des algorithmes. Deux classes importantes peuvent être définies pour résoudre les problèmes d'optimisation : P et NP [Talbi 09] [Azem 10]. La classe de complexité P représente l'ensemble des problèmes de décision qui peuvent être résolus par des algorithmes déterministes en un temps polynomial, c'est à dire la complexité dans le pire des cas est polynomiale. Les problèmes appartenant à cette classe sont relativement simples à résoudre. La classe de complexité NP regroupe les problèmes de décision qui peuvent être résolus par un algorithme non-déterministe en un temps polynomial. Ces problèmes peuvent être résolus soit en proposant un algorithme approché qui calcule en temps polynomial une solution approximative, soit en développant un algorithme qui calcule la solution optimale du problème avec une complexité exponentielle. Un problème de décision est NP-complet si tous les autres problèmes de la classe NP peuvent être réduits d'une manière polynomiale à ce problème. En revanche, s'il existe un algorithme déterministe polynomial (la complexité dans le pire des cas est limitée par une fonction polynomiale) pour résoudre un problème NP-complet, alors tous les problèmes de la classe NP peuvent être résolus en temps polynomial. Un problème d'optimisation est NP-difficile si le problème de décision associé est NP-complet.

Pour ces raisons, les entreprises manufacturières doivent réagir efficacement en adoptant une démarche qui vise à analyser et modéliser une situation en vue d'en dégager les éléments pouvant conduire à la prise de décision. Cela peut être réalisé en développant des algorithmes adéquats tout en proposant un ordonnancement efficace de manière à obtenir un fonctionnement optimal de l'utilisation des capacités de production et une amélioration de la production.

1.4.4 Méthodes de résolution

Les problèmes d'ordonnancement, sont généralement similaires à des problèmes d'optimisation combinatoire, ils peuvent être résolus en utilisant deux approches différentes.

La première approche consiste à utiliser des algorithmes exacts, elle est appliquée à des problèmes de petites tailles et conduit à des solutions optimales. La deuxième approche est caractérisée par l'utilisation de méthodes approximatives, elle est appliquée à des problèmes plus complexes et mène à des solutions quasi-optimales.

1.4.4.1 Les méthodes exactes

Les méthodes exactes sont des techniques de résolution qui fournissent des solutions optimales pour des problèmes combinatoires, grâce à une exploration intelligente de l'espace des solutions, mais pas systématiquement dans un temps polynomial. Cependant, il n'existe pas d'algorithmes généraux permettant de résoudre en temps polynomial les problèmes typiques d'optimisation, comme les problèmes d'ordonnancement qui sont généralement NP-difficiles. Par conséquent, ces méthodes sont peu utilisées dans les contextes industriels. Toutefois, les résultats obtenus par les méthodes exactes pour les petites instances peuvent être exploités pour des problèmes de tailles industrielles afin d'essayer de dégager des tendances dans les méthodes de résolution. Les méthodes exactes sont multiples et variées telles que la programmation linéaire, la programmation dynamique, la procédure par séparation et évaluation,…

- **La programmation linéaire et en nombres entiers**

La programmation linéaire est une approche générique basée sur la modélisation mathématique des problèmes d'optimisation combinatoires, où la fonction objectif, les contraintes et les variables de décisions sont linéaires. Ainsi, les problèmes de programmation linéaire déterminent la manière d'obtenir le meilleur résultat. De nombreux problèmes pratiques dans la recherche opérationnelle peuvent être exprimés comme des problèmes de programmation linéaire. Pour traiter un programme linéaire en variables continues, les deux algorithmes les plus utilisés sont la méthode du simplex et la méthode des points intérieurs.

D'autre part, la modélisation des problèmes d'ordonnancement implique souvent l'intégration des variables de décisions entières dans le modèle mathématique, il s'agit donc d'un modèle de programmation linéaire en nombres entiers dont l'inconvénient majeur peut être le grand nombre de contraintes et de variables requises. Par conséquent, la majorité des problèmes d'ordonnancement, qui sont NP-difficiles, ne peuvent être résolus d'une façon optimale par des programmes mathématiques en nombres entiers. En revanche, leur relaxation dans les modèles linéaires continus permet d'obtenir des bornes inférieures pour les problèmes de minimisation. Les meilleures approches utilisées pour résoudre ces modèles sont la procédure par séparation et évaluation, la méthode de simplex ou des solveurs dédiés comme Cplex ou Lingo.

- **La programmation dynamique**

La programmation dynamique est une méthode fondée sur le principe de Bellman qui assure que la sous-stratégie d'une politique optimale est elle même optimale [Bellman 57]. Cette procédure est basée sur la division récursive d'un problème en sous-problèmes plus simples. Ainsi, le système est constitué de plusieurs étapes à résoudre de façon séquentielle. Chaque étape correspond à un sous-problème que nous résolvons de façon optimale en tenant compte des informations obtenues au cours des étapes précédentes. Cela nécessite une

formulation du critère sous la forme d'une relation de récurrence reliant deux niveaux consécutifs. Cependant, le passage d'une étape à une autre peut être basé sur les lois d'évolution du système et d'une décision. La procédure évite une énumération totale de l'espace de recherche par des séquences de décision partielle d'élagage qui ne peuvent pas conduire à la solution optimale.

- **La procédure par séparation et évaluation**

La procédure par séparation et évaluation, introduite par Dantzig et al. [Dantzig 54] pour la résolution du problème du voyageur de commerce, est l'une des méthodes les plus populaires pour résoudre des problèmes d'optimisation de manière exacte. L'algorithme est basé sur une énumération implicite de toutes les solutions du problème d'optimisation considéré en évitant les branches inutiles. L'espace de recherche est exploré dynamiquement par la construction d'un arbre dont le nœud principal représente le problème à résoudre et l'ensemble de son espace de recherche associé. Les nœuds internes représentent les sous-problèmes. Enfin, les nœuds feuilles symbolisent les alternatives possibles. L'exploration d'un tel arbre est effectuée en définissant une stratégie de branchement qui détermine l'ordre dans lequel les branches sont parcourues et en utilisant deux principaux opérateurs : la séparation et l'évaluation.

La procédure de séparation consiste à énumérer toutes les solutions possibles en divisant le problème considéré en plusieurs sous-problèmes. Cependant, la procédure d'évaluation consiste à éliminer les solutions partielles qui ne conduisent pas à des solutions optimales en se basant sur des propriétés mathématiques et donc les sous-arbres qui ne contiennent aucune solution optimale seront élagués. Cela se fait en calculant la borne inférieure associée à une solution partielle afin d'analyser si cette branche peut mener à une solution optimale ou non.

L'inconvénient majeur de cette technique apparaît lorsque nous avons des problèmes de grande taille, nous pouvons tomber sur des situations où nous devons gérer des structures arborescentes importantes. Pour limiter la taille de la recherche de l'arbre, nous pouvons tenter d'appliquer des bornes supérieures et de perfectionner le calcul de bornes inférieures. Nous pouvons également développer des règles de priorités et appliquer des résultats de dominance [Carlier 88] [Esquirol 99] afin d'arrêter l'exploration des nœuds.

1.4.4.2 Les méthodes approximatives

Les méthodes approchées représentent une alternative intéressante pour résoudre des problèmes d'optimisation combinatoires NP-difficiles. Contrairement aux méthodes exactes qui cherchent les solutions optimales, les méthodes approximatives fournissent des solutions acceptables en un temps raisonnable. Les méthodes les plus connues pour leurs aptitudes à résoudre ce type de problèmes sont essentiellement les heuristiques de construction, les heuristiques de décomposition et les métaheuristiques [Azem 10].

- **Les heuristiques de construction**

Ces heuristiques ont pour objectif de construire itérativement une solution à un problème d'ordonnancement. Elles consistent à définir des priorités entre les tâches qui sont en attente de traitement sur une machine. Les méthodes les plus couramment utilisées sont des

algorithmes dont le principe est de trier les opérations selon une stratégie de décision telle que LPT (Longest Processing Time first), SPT (Shortest Processing Time first), etc.

- **Les heuristiques de décomposition**

Cette classe permet de fournir des solutions approximatives à des problèmes complexes en décomposant le problème en plusieurs sous-problèmes. Selon Azem [Azem 10] nous pouvons définir trois familles qui sont :

- ✓ La décomposition hiérarchique proposée par Erscher et al. [Erscher 85] consiste à décomposer le problème en plusieurs niveaux, dont les décisions deviennent des contraintes pour les niveaux inférieurs.
- ✓ La décomposition temporelle proposée par Portmann [Portmann 88] consiste à ordonnancer les sous-ensembles d'opérations disponibles avant une date et inclure les autres dans la séquence partielle. Cette technique est utilisée pour les problèmes d'ordonnancement dynamique.
- ✓ La décomposition spatiale proposée par Portmann [Portmann 88] qui a pour objectif de décomposer l'atelier en plusieurs sous-ateliers avec un minimum de mouvements entre eux, ordonnancer les opérations dans chaque atelier et enfin coordonner l'ensemble.

- **Les métaheuristiques**

Les métaheuristiques sont largement utilisées pour résoudre des problèmes d'optimisation combinatoire. Leur succès est dû au fait qu'elles permettent d'intégrer les différentes contraintes pratiques des problèmes. Elles sont faciles à mettre en œuvre et offrent des solutions acceptables. Cette classe de méthodes de résolution va faire l'objet du chapitre suivant.

1.5. Aperçu sur les travaux d'ordonnancement sur machines parallèles avec ressources consommables

1.5.1 L'importance du Makespan dans l'ordonnancement des problèmes d'ordonnancement sur machines parallèles

Dans la majorité des cas, un problème d'ordonnancement peut être représenté comme un problème d'optimisation en considérant des objectifs à atteindre afin de pouvoir juger la pertinence d'un ordonnancement et mesurer les performances du système tout en satisfaisant ses contraintes du point de vue exploitation. Dans ce contexte, plusieurs chercheurs se sont intéressés à la minimisation du Makespan pour les problèmes d'ordonnancement sur machines parallèles et ceci pour les raisons suivantes [Edis 13] :

- L'objectif du makespan est plus facile à manipuler par rapport à d'autres critères tels que les objectifs à base des pondérations et date dues.
- Plusieurs situations rencontrées dans l'industrie conduisent à la minimisation du makespan pour satisfaire leurs objectifs.
- La minimisation du makespan permet d'équilibrer la charge entre les machines et offre généralement un taux élevé de leurs utilisations [Pinedo 08].

- Plusieurs techniques existent qui fournissent des résultats efficaces comme l'heuristique *longest processing time first*, afin de minimiser le makespan pour les problèmes classiques de machines parallèles.
- Ils existent plusieurs algorithmes qui sont utilisés dans la conception des approches d'approximation pour les problèmes de machines parallèles avec des contraintes de ressources sous le critère du Makespan.
- Des bornes inférieures peuvent être développées simplement par rapport à d'autres critères de performance.

En outre, nous précisons que d'autres objectifs peuvent être considérés comme la minimisation des retards ou du temps de cycle. Si plusieurs fonctions objectifs sont définies à la fois. Dans ce cas, le problème devient un problème multicritère.

1.5.2 Aperçu sur les travaux portant sur les algorithmes génétiques pour l'ordonnancement sur machines parallèles

Dans un contexte industriel, l'ordonnancement sur machines parallèles est un problème qui reste assez fréquent dans plusieurs industries (textiles, en particulier). Il consiste à affecter les tâches aux machines et à déterminer la séquence des travaux assignés à la même machine. Il représente un domaine de recherche important. Cette importance est encore amplifiée par la variété de configurations complexes qui peuvent être réduites à ce type de problème. Les problèmes d'ordonnancement sur machines parallèles reçoivent beaucoup d'attention par la communauté scientifique [Min 03] et de nombreuses approches ont été proposées, une grande partie d'entre elles se basent sur les algorithmes génétiques [Cheng 95], [Yip-Hoi 96], [Min 99], [Sivrikaya 99], [Chiu 99], [Gupta 01], [Luu 02], [Fowler 03], [Wilson 04], [Jou 05], [Chang 05], [Damodaran 06], [Özalp 06], [Min 06], [Malvea 07], [Mok 07], [Çakar 08], [Kashan 08], [Tavakkoli 09], [Balin 11], [Vallada 11], [Li 12], [Amorim 13], [Berrichi 13].

1.5.3 Aperçu sur les travaux d'ordonnancement avec ressources consommables

L'objectif majeur des industriels est d'exploiter efficacement leurs systèmes de productions afin d'améliorer leur réactivité et leur productivité en utilisant toutes les ressources disponibles d'une façon optimale dans des délais minimums. Cependant, une partie importante des problèmes d'ordonnancement suppose la disponibilité totale des ressources, mais ce n'est pas toujours le cas, car les ressources peuvent être consommées et peuvent être indisponibles pour diverses raisons (comme dans les systèmes de production, la matière première est consommée à la fin de chaque tâche) et par conséquent, le manque de ressources, perturbe et affecte l'efficacité de l'ordonnancement. À cet égard, plusieurs travaux se sont intéressés à l'analyse des effets des ressources consommables sur les performances des systèmes de productions afin d'aider les décideurs à prendre les meilleures décisions. Blazewicz *et al.* [Blazewicz 83] et Blazewicz *et al.* [Blazewicz 86] ont divisé les problèmes d'ordonnancement avec contraintes en deux catégories qui sont : l'ordonnancement de projets avec des contraintes associées aux ressources et l'ordonnancement de machines. Depuis ce temps, ces problèmes ont reçu beaucoup d'attention par le milieu académique et plusieurs recherches ont été réalisées, la plupart d'entre elles sont pour l'ordonnancement de projets [Moumene 08] [Briand 09] [Carlier 09] [Quilliot 12] [Zamani 12] ou pour les problèmes à une seule machine.

1.5.3.1 Aperçu sur les travaux sur une seule machine avec ressources consommables

Cette section est réservée à la présentation de quelques travaux qui ont mis l'accent sur les problèmes à une seule machine avec des ressources consommables. Un intérêt particulier est porté au travail réalisé par Carrera [Carrera 10].

Toker et al. [Toker 91] étudient un problème d'ordonnancement à une seule machine avec une seule ressource financière afin de montrer que ce problème est équivalent à un flow shop composé de deux machines sans ressources financières, les auteurs ont appliqué l'algorithme de Johnson dans le but de minimiser le makespan. Xie [Xie 97] quant à lui traite un problème d'ordonnancement à une seule machine avec de multiples contraintes financières, l'auteur réduit ce problème à un flow shop à deux machines. Il démontre que la règle LPT donne une solution optimale au problème si les ressources financières sont consommées de façon uniforme par toutes les tâches. Kaspi et Shabtay [Kaspi 04] traitent un problème d'ordonnancement à une seule machine où le temps de traitement est défini par une fonction convexe décroissante de consommation des ressources. Les temps de traitement dépendent d'une ressource commune limitée. L'objectif est de minimiser le makespan et de déterminer le meilleur moyen pour la permutation et l'allocation des ressources. Lazarev et Werner [Lazarev 09] posent le problème d'ordonnancement sur une seule machine, ils supposent que les temps de traitement et les dates due sont ordonnés d'une manière opposée. Les auteurs fournissent quelques cas polynomiaux pour minimiser le retard total.

Carrera [Carrera 10] aborde un problème de lissage de charge dans une plateforme logistiques pour minimiser les coûts de production. Ensuite, l'auteur étudie un problème d'ordonnancement sur une seule machine avec des ressources consommables et des dates de livraison fixe. Le problème est caractérisé par l'arrivée de plusieurs composants suivant des courbes sous forme d'escaliers. Pour cela, une méthode exacte basée sur la procédure par séparation et évaluation est proposée pour minimiser indépendamment, trois fonctions objectifs à base de makespan. En outre un modèle linéaire en nombres entiers est proposé pour minimiser le makespan. Finalement, une étude de complexité est effectuée affirmant que ce type de problème est NP-difficile. Une partie importante des travaux cités ci-dessus sont résumés dans l'état de l'art réalisé par Janiak et al. [Janiak 07]. Aussi, une analyse sur la complexité des problèmes d'ordonnancement avec une seule machine est faite par Gafarov et Lazarev [Gafarov 10], les auteurs ont accordé une attention particulière au problème de minimisation de la somme des retards.

1.5.3.2 Complexité des problèmes d'ordonnancement sur machines parallèles avec ressources consommables

Dans ce contexte, quelques auteurs se sont intéressés à étudier la nature de ces problèmes et à l'étude de complexité : Lenstra [Lenstra 77] et Sethi [Sethi 77] ont étudié la complexité d'autres problèmes d'ordonnancement sur machines parallèles pour la minimisation du makespan. Ils montrent que ces problèmes sont NP-difficiles, même pour deux-machines. En outre, Carlier et Rinnooy Kan [Carlier 82] montrent que pour les problèmes à une seule machine avec ressources consommables, les problèmes deviennent NP-difficiles au sens fort quand les durées des jobs ne sont pas identiques. De plus, Blazewicz et

al. [Blazewicz 83] confirment que la quasi-totalité des problèmes sur machines parallèles avec contraintes de ressources non-renouvelables sont NP-difficiles sauf pour quelques cas particuliers qui apparaissent lorsque le nombre de machines est égal à deux où ils peuvent être résolus en un temps polynomial. Nous pouvons conclure que l'ordonnancement sur ce type de problème est relativement complexe et généralement, NP-difficiles

1.5.3.3 Aperçu sur les travaux sur machines parallèles avec ressources consommables

Les travaux cités précédemment se focalisent sur les problèmes à une seule machine. Cependant, les problèmes d'ordonnancement sur machines parallèles avec ressources consommables attirent la communauté scientifique, car ils ont un large fond d'application, en particulier dans l'industrie textile, l'industrie des chaussures, l'industrie électronique, etc. Étant donné que la majorité des problèmes d'ordonnancement sur machines parallèles avec des ressources supplémentaires sont NP-difficiles, il devient nécessaire de trouver les approches adéquates pour résoudre ces problèmes. L'implication croissante des machines parallèles avec ressources consommables dans plusieurs domaines explique l'intérêt grandissant des chercheurs pour ce problème. Cela peut être prouvé à travers l'état de l'art effectué récemment par Edis [Edis 13] sur les machines parallèles avec ressources additionnelles, dont lequel il fournit quelques éléments de référence de la littérature que nous enrichissons dans cette section.

Edis et al. [Edis 13] pensent que les approches de résolutions des problèmes d'ordonnancement sur machines parallèles avec contraintes de ressources, peuvent être classées en trois paradigmes regroupant les algorithmes en temps polynomial, les méthodes exactes et les méthodes approximatives. Cette dernière catégorie peut aussi être divisée en heuristiques, métaheuristiques et programmation par contraintes. Nous nous contentons de présenter les travaux issus des méthodes analytiques et des méthodes approchées. De plus, nous avons classé les travaux présentant des études dans ce sens, selon les différentes mesures de performances considérées en deux classes. La première classe (notée, Classe 1) concerne les travaux incluant la minimisation du makespan, la minimisation du temps de cycle, etc. La deuxième classe (notée, Classe 2) contient les études incluant la minimisation des retards, la minimisation de la longueur de la file d'attente, etc. La synthèse de notre revue sur ces modèles est présentée dans le tableau 1.1.

Comme pour tous les problèmes NP-difficiles, les méthodes exactes qui s'intéressent à la résolution des problèmes d'ordonnancement sur machines parallèles avec ressources consommables sont relativement peu utilisées. Slowinski [Slowinski 84] traite un problème d'ordonnancement sur des machines parallèles avec des contraintes de ressources (contraintes financières) et la préemption des tâches est autorisée. L'auteur propose une procédure composée de deux phases en utilisant la programmation linéaire pour minimiser le makespan. Blazewicz et al. [Blazewicz 93] proposent une procédure de séparation et évaluation pour faire face à ce problème afin de minimiser le retard maximum. Ils proposent également deux autres heuristiques.

Daniels et al. [Daniels 96] appliquent une méthode exacte basée sur la procédure par séparation et évaluation pour résoudre le problème d'ordonnancement dynamique sur

machines parallèles. Un problème dans lequel, le temps de traitement est déterminé par la quantité des ressources supplémentaires allouée à une tâche. Les auteurs proposent également une heuristique pour minimiser le makespan. Kellerer et Strusevisch [Kellerer 08] proposent un algorithme de programmation dynamique pour minimiser le makespan, ils montrent que le problème est équivalent à résoudre deux sous-problèmes de sac à dos pour chaque machine.

Concernant les travaux qui se sont basés sur des méthodes approchées, nous pouvons les classer en trois grandes classes : heuristiques, métaheuristiques et programmation par contraintes.

Les méthodes heuristiques proposées dans la littérature pour l'ordonnancement des problèmes avec ressources consommables sont nombreuses. Certaines d'entre elles s'adressent aux problèmes sur machines parallèles avec contraintes de ressources. Olafsson et Shi [Olafsson 00] proposent une heuristique pour résoudre le problème avec ressources consommables avec plusieurs machines parallèles pour minimiser le makespan. Ruiz-Torres et Centeno [Ruiz-Torres 07] proposent de nouvelles heuristiques pour le même problème considéré précédemment par Daniels et al. [Daniels 99], ensuite ils développent une borne inférieure par un recensement complet des alternatives d'affectation des ressources pour minimiser les tâches en retard. Cochand *et al.* [Cochand 89] proposent un algorithme à deux-phases pour résoudre un problème d'ordonnancement sur machines parallèles avec des ressources consommables dont l'approvisionnement varie avec le temps (en escalier ou linéaire par morceaux) pour minimiser le makespan. Shabtay et Kaspi [Shabtay 06] considèrent certains problèmes d'ordonnancement quand les temps de traitement des tâches sont contrôlés par l'affectation des ressources limitées non-renouvelables sur des machines parallèles identiques. La fonction de consommation des ressources est supposée convexe. Les auteurs proposent une méthode pour l'affectation des ressources afin de minimiser le makespan et un algorithme pour minimiser la somme des dates de fin. Quant à Ling et al. [Ling 09], ils proposent une heuristique pour résoudre un problème d'ordonnancement sur des machines parallèles afin de minimiser le makespan lorsque le temps de traitement dépend de la quantité de ressources consommées. Sue et Lien [Sue 09] développent deux heuristiques pour le même problème avec comme objectif la minimisation du makespan. Ils affectent les tâches aux machines puis les ressources aux tâches.

Même si les métaheuristiques constituent une alternative importante de résolution et sont largement utilisées pour les problèmes d'ordonnancement NP-difficiles, il existe seulement quelques études utilisant des métaheuristiques pour résoudre les problèmes d'ordonnancement sur machines parallèles avec des ressources renouvelables. Daniels et al. [Daniels 97] proposent deux métaheuristiques basées sur la recherche tabou. Les deux approches utilisent une stratégie de recherche hiérarchique basée sur la décomposition de trois sous-problèmes pour minimiser le makespan sur machines parallèles avec des ressources non-renouvelable. Daniels et al. [Daniels 99] proposent également une procédure de recherche tabou pour le problème de statique avec des machines identiques. La procédure détermine d'abord l'attribution initiale des tâches vers les machines en utilisant la règle LPT. Compte tenu de cette affectation initiale, ils affectent les ressources disponibles aux machines pour minimiser le makespan. Li et al. [Li 03] proposent une approche basée sur un algorithme génétique pour minimiser le makespan. L'approche proposée intègre un schéma de codage,

qui donne les priorités aux tâches. Ils proposent également trois bornes inférieures pour évaluer la performance de l'approche de l'algorithme génétique proposé. Kai Li et al. [Kai 11] proposent un algorithme de recuit simulé afin de minimiser le makespan pour un problème d'ordonnancement sur machines parallèles identiques avec un temps de traitement contrôlable, ils supposent que les temps de traitement de chaque tâche sont des fonctions linéaires décroissantes de la ressource consommée.

Belkaid et al. [Belkaid 12] mènent une étude préliminaire sur le problème d'ordonnancement sur machines parallèles avec ressources consommables. Les auteurs étendent la problématique considérée par Carrera et al. [Carrera 10]. Ils supposent que le système est composé de plusieurs machines parallèles identiques avec des arrivées aléatoires de composants. Pour résoudre ce problème une méthode d'optimisation est proposée basée sur les algorithmes génétiques pour minimiser le makespan. Quant à Belkaid et al. [Belkaid 13a] présentent une version étendue de celle du travail mené dans Belkaid et al. [Belkaid 12], en effet, ils proposent une heuristique qui consiste à combiner entre les temps de traitement et la consommation des ressources non-renouvelables, puis ils comparent l'algorithme génétique, dans un premier temps, avec une méthode exacte à base d'énumération complète pour les petites instances et, dans un second temps, avec une heuristique pour les grandes instances. Les résultats obtenus démontrent l'efficacité de la métaheuristique proposée. En se focalisant sur le même travail, Belkaid et al. [Belkaid 13b] développent cette fois-ci un modèle mathématique linéaire en nombres entiers, modélisé par des variables de position pour minimiser le makespan. Ensuite ils proposent une procédure de recherche locale afin d'améliorer davantage les résultats obtenus dans [Belkaid 13a]. Les résultats démontrent l'efficacité de l'algorithme génétique hybride.

Finalement, la programmation par contraintes, qui est un domaine de l'intelligence artificielle reste peu utilisée, mais commence à prendre de l'ampleur ces dernières années sur les problèmes de machines parallèles avec ressources limitées. Edis et Ozkarahan [Edis 11] développent trois modèles d'optimisation, un modèle de programmation en nombres entiers, une programmation par contraintes et un troisième qui combine entre les deux premiers pour l'ordonnancement sur machines parallèles avec ressources limitées afin de minimiser le makespan. Edis et Ozkarahan [Edis 12a] considèrent le même modèle décrit précédemment dont l'objectif est de minimiser le makespan, ils développent des heuristiques séquentielles basées sur un modèle en nombres entiers et un modèle de programmation par contrainte. L'attribution des tâches est faite par le modèle en nombres entiers et l'ordonnancement des tâches sur les machines dédiées est obtenu par le modèle d'ordonnancement en nombres entiers (et alternativement avec la programmation par contrainte).

Edis et Oguz [Edis 12b] proposent des approches heuristiques séquentielles basées sur un modèle en nombres entiers et la programmation par contraintes pour un problème d'ordonnancement sur machines parallèles identiques et uniformes. Les auteurs traitent le problème d'allocation des ressources en relaxant le modèle proposé par Grigoriev et al. [Grigoriev 05], en nombres entiers dont le but été de minimiser le makespan. La synthèse des travaux portant sur les problèmes d'ordonnancement dans un environnement de machines parallèles avec ressources consommables est résumée dans le tableau 1.1.

Tableau 1.1 : Synthèse des travaux sur les problèmes d'ordonnancement sur machines parallèles avec des contraintes de ressources

Articles publiés	Les approches de résolution				Les mesures de performances	
	Méthodes analytiques	Métaheuristiques	Programmation par contrainte	Heuristiques	Classe 1	Classe 2
[Slowinski 84] [Kellerer 08]	X				X	
[Cochand 89] [Shabtay 06]				X	X	
[Blazewicz 93]	X			X		X
[Daniels 96]	X			X	X	
[Daniels 97] [Li 03] [Kai 11]		X			X	
[Daniels 99]		X		X	X	
[Olafsson 00] [Ling 09] [Sue 09]				X	X	
[Ruiz-Torres 07]				X		X
[Edis 11] [Edis 12a] [Edis 12b]			X		X	
[Belkaid 12]		X				
[Belkaid 13a]		X	X			
[Belkaid 13b]	X	X				

1.6. Aperçu sur les travaux d'ordonnancement avec processus réentrant

Au cours des dernières années, les problèmes d'ordonnancement réentrants représentent un domaine de recherche particulièrement étudié et qui attire l'attention de plusieurs chercheurs, comme le montre les travaux effectués par Gupta et Sivakumar [Gupta 06] et Lin et Carman [Lin 11] que nous compléterons par la suite. L'intérêt pour les problèmes d'ordonnancement réentrant a été étudié pour la première fois par Graves et al. [Graves 83] en 1983, les auteurs traitent un problème de production de circuits électriques à cheminement unique, ils développent une méthode d'ordonnancement cyclique et une heuristique pour proposer un ordonnancement efficace afin de minimiser le temps de cycle moyen.

Les ateliers de production réentrant visent à améliorer leurs performances en satisfaisant au mieux les demandes et les besoins de leurs clients tout en minimisant les coûts et les délais de production, en augmentant la productivité et en améliorant la qualité. Pour y arriver, ils sont contraints à traduire les objectifs de l'entreprise sous forme de mesures de performance appropriées afin de pouvoir analyser le comportement du système et d'être réactives aux différents changements qui surgissent pendant la production. Cependant, pour faire face à ces systèmes caractérisés par une complexité fonctionnelle et structurelle, les universitaires et les chercheurs développent plusieurs mesures de performances qui se traduisent par un ou plusieurs critères dont certains sont contradictoires entre eux. La classification qui nous semble la plus intéressante est celle proposée par Lin et Carman [Lin 11] (présentée dans la section 1.4.2). Les mêmes auteurs pensent que les fonctions objectifs peuvent être réparties en trois classes : la première classe (notée, Classe 1) contient les mesures liées à l'aspect

synchronisation telles que la minimisation du makespan, la minimisation du temps de cycle, etc. En outre la deuxième classe (notée, Classe 2) a pour objectif d'optimiser le rendement de la production, les travaux en cours, l'inventaire des travaux en cours, le taux d'inactivité des machines, etc. Finalement, la dernière catégorie (notée, Classe 3) est liée au niveau de service tel que la minimisation des retards, la minimisation de la longueur de la file d'attente, etc. [Lin 11]. D'autre part, la complexité des problèmes d'ordonnancement réentrant dépend non seulement de la définition de la fonction objectif et du processus mais aussi du choix de la méthode de résolution appropriée vue que les problèmes qui surgissent dans l'industrie sont d'une difficulté inhérente et se caractérisent par plusieurs contraintes relatives aux ressources, aux machines et aux composants. Plusieurs travaux ont été proposés pour les résoudre afin de pouvoir bénéficier des avantages des systèmes de production. Ces travaux peuvent être divisés, généralement en trois grandes classes incluant les méthodes analytiques, les heuristiques et les métaheuristiques. Dans cette section, nous présentons une synthèse regroupant les différents problèmes avec processus réentrant sous divers critères, ainsi que les méthodes de résolutions proposées. Cette synthèse est résumée dans le tableau 1.2.

1.6.1 Méthodes analytiques pour la résolution des problèmes d'ordonnancement réentrant

Dans la littérature associée aux problèmes d'ordonnancement des systèmes de production réentrant, plusieurs méthodes analytiques fondées sur un raisonnement mathématique sont développées, et chacune présente des avantages spécifiques et servants à décrire l'ensemble des phénomènes rencontrés dans les lignes de production. Ces méthodes sont basées soit sur des modèles mathématiques, sur des réseaux de pétri ou bien sur des procédures par séparation et évaluation.

1.6.1.1 Modèles mathématiques

Dans la littérature, les modèles qui traitent les problèmes d'ordonnancement réentrant sont souvent représentés par des modèles linéaires en nombres entiers où en nombres mixtes et dans certains cas par des modèles non-linéaires. Hwang et Sun [Hwang 99] traitent un problème d'ordonnancement dans une entreprise de fabrication de camions où un ensemble de tâches doit être ordonnancé sur deux machines. Le problème est formulé en un problème général de type flow shop composé de deux machines. Les auteurs proposent, dans un premier temps de résoudre ce problème avec la programmation dynamique afin de minimiser le makespan et dans un second temps, ils développent un algorithme génétique. La performance de ce dernier est démontrée pour les problèmes de grande taille. Quant à Bard et Devanath [Bard 99], ils considèrent un problème de conception des installations d'une usine de semi-conducteurs. L'objectif est de trouver une configuration qui minimise le temps de cycle moyen. En raison de la présence d'un flux réentrant, les auteurs démontrent que cette version du problème peut être modélisée comme un programme non-linéaire en nombres entiers. Les auteurs proposent aussi une nouvelle série d'algorithmes et une technique basée sur le recuit simulé. Moon et Hrymak [Moon 99] abordent un problème d'ordonnancement pour le traitement thermique des bobines d'acier afin de déterminer le plan de mouvement optimal pour satisfaire l'utilisation des équipements partagés. Ce processus peut contenir les flux réentrants, notamment lorsque c'est en rapport avec le temps de réglage et le temps de

transfert. Un modèle de programmation linéaire en nombres entiers est proposé pour l'ordonnancement de ce processus. Pour les problèmes de grande taille, un nouvel algorithme est développé pour fournir des solutions approximatives. Vargas et Rivera [Vargas 00] utilisent, quant à eux, un programme linéaire mixte en nombres entiers pour résoudre un problème d'ordonnancement dans une ligne réentrante de fabrication de semi-conducteurs. Les auteurs proposent aussi un algorithme basé sur une formulation de contrôle prédictif afin d'optimiser le taux de production et de contrôler l'inventaire en cours de fabrication.

Pan et Chen [Pan 03] présentent des modèles mathématiques en nombres entiers mais avec des formulations étendues ainsi que six heuristiques efficaces pour résoudre un problème d'ordonnancement de type flow shop réentrant afin de minimiser le makespan. De plus, une méthode de recherche tabou est proposée de la part des auteurs pour résoudre le problème flow-shop classique. En ce qui concerne, Cochran et Shunk [Cochran 03], ils proposent une technique de modélisation pratique pour minimiser les investissements nécessaires dans une usine de fabrication discontinue. Cette dernière est fondée sur un programme stochastique en nombres entiers constitué de deux étapes : la première caractérise la réponse optimale du système en cas d'incertitude et la seconde sélectionne un ensemble d'outils en fonction de la caractérisation de la première étape, mais en faisant des restrictions budgétaires. Le modèle utilisé est évolutif, permettant de fabriquer de multiples produits en passant par plusieurs opérations avec des chemins de flux multiples, y compris le flux réentrant, et en utilisant plusieurs types d'outils. Pan et Chen [Pan 04] proposent un modèle d'optimisation basé sur la technique de programmation en nombres entiers ainsi que des heuristiques fondées sur l'ordonnancement actif et sans délais afin de résoudre un problème d'ordonnancement dans un environnement flow shop réentrant pour minimiser le makespan et le temps de cycle moyen. Pour ce qui est de Low et al. [Low 05], ils étudient les problèmes d'ordonnancement de type job shop avec des opérations réentrantes où les temps de réglage, varient en fonction de l'opération précédente qui est traitée sur la même machine. Trois mesures de performance sont considérées, dans cette étude, afin de minimiser le temps de cycle, le retard total et le temps d'inactivité. Pour optimiser chaque objectif, un modèle de programmation en nombres entiers est conçu. Chen et Pan [Chen 06b] développent, eux aussi une approche analytique basée non seulement sur la programmation en nombres entiers mais aussi sur les nombres binaires et mixtes (notée, *mixed binary integer programming*) pour résoudre les problèmes d'ordonnancement de type flow shop et job shop réentrants. Aussi, les auteurs proposent huit modèles pour résoudre les problèmes d'ordonnancement d'atelier réentrants pour minimiser le makespan. Ignisio [Ignisio 08] montre, qu'il est possible d'utiliser le même modèle mathématique développé à l'origine pour équilibrer la charge des postes de travail, dans le cadre des chaînes d'approvisionnement réentrantes afin de minimiser le temps de cycle dans les lignes de semi-conducteurs. Quant à Nazarathy et Weiss [Nazarathy 09], ils proposent une méthode pour le contrôle des réseaux de files d'attente. Ils formulent le problème posé comme un problème d'optimisation et proposent pour le résoudre, un programme linéaire continu séparé pour minimiser les coûts et maximiser la productivité. Les auteurs illustrent cette méthode par un exemple simple d'une ligne réentrante en trois étapes. Chung et al. [Chung 09] examinent un problème d'ordonnancement de machine de traitement de lot en parallèle, en considérant plusieurs contraintes. L'objectif visé par les auteurs est de trouver un

ordonnancement optimal afin de minimiser le makespan. Pour cela, un modèle de programmation linéaire en nombres entiers mixtes est proposé. En outre, pour résoudre le problème plus efficacement, un algorithme est proposé pour déterminer le nombre de lots. Enfin, trois autres heuristiques sont également fournies. Dans l'étude menée par Yan et Wang [Yan 12], les auteurs traitent un problème d'ordonnancement dynamique d'une ligne de production réentrante dans laquelle toutes les opérations sont supposées avoir les mêmes routages de traitement et doivent être exécutées sur toutes les machines. Une méthode de programmation dynamique à deux couches est proposée pour résoudre ce problème afin de minimiser l'avance et le retard total. Nous clôturons les travaux qui utilisent les modèles mathématiques par ceux de Gomes et al. [Gomes 13] qui présentent un modèle de programmation linéaire en nombres entiers pour l'ordonnancement d'un job shop flexible caractérisé par un processus réentrant. La formulation utilise une représentation en temps continu afin d'optimiser les inventaires en cours et la somme pondérée des avances et des retards. Aussi, un algorithme de programmation prédictive réactive est dérivé du modèle proposé pour faire face à l'arrivée de nouvelles commandes.

1.6.1.2 Réseaux de pétri

Certains travaux récents portent sur des mesures qui intègrent des outils mathématiques permettant de modéliser le comportement dynamique des systèmes manufacturiers afin d'analyser les processus de production. Ces outils sont connus sous le nom des réseaux de pétri. Nous allons citer quelques travaux les utilisant : Odrey et al. [Odrey 01] présentent une approche de modélisation basée sur l'utilisation des réseaux de pétri généralisés pour une ligne de fabrication de flux réentrant. Plus précisément, trois modèles de réseaux de pétri intégrant trois centres de travail et six machines. Ces modèles peuvent être utilisés pour représenter une variété de règles de files d'attente. Kumar et al. [Kumar 04], quant à eux adaptent dans leurs études une méthode de rééchelonnement de ligne de production d'une industrie de semi-conducteur en utilisant un modèle de réseaux de pétri et en prenant en compte des dates dues afin de minimiser le critère du makespan. Lee et Lee [Lee 06] examinent un problème d'ordonnancement pour un outil de cluster unique avec différents flux de plaquettes réentrants. Les auteurs développent une méthode pratique de modélisation en utilisant les réseaux de pétri ainsi qu'en fournissant une condition nécessaire et suffisante pour éviter une impasse. Ils proposent aussi un modèle de programmation mixte en nombres entiers afin de déterminer la séquence de fonctionnement optimal et d'optimiser la durée du cycle. Quant à Lee et al. [Lee 07], ces derniers traitent simultanément plusieurs types de plaquettes pour effectuer une opération de nettoyage dans l'industrie des semi-conducteurs. Ils proposent comme moyen de modélisation un réseau de pétri. En développant ce modèle, ils instaurent des conditions pour prévenir les blocages et les collisions. Ensuite, ils utilisent un modèle de programmation mixte en nombres entiers pour non seulement la détermination des séquences de tâches robotiques, mais aussi pour établir un ordonnancement de synchronisation afin de satisfaire les contraintes liées aux temps et de minimiser le temps de cycle.

1.6.1.3 Procédure par séparation et évaluation

Quelques chercheurs se sont intéressés au développement des méthodes par séparation et évaluation pour traiter les problèmes d'ordonnancement. Ces méthodes consistent à

énumérer implicitement toutes les solutions du problème considéré en évitant les branches inutiles. Dans ce cadre, Wang et al. [Wang 97] examinent un problème d'ordonnancement de lignes réentrantes dans laquelle le passage de tâches est défini. Ils déterminent une classe particulière des ordonnancements optimaux basés sur certaines propriétés dans le but est de construire deux algorithmes, l'un utilisant la procédure par séparation et évaluation et l'autre d'approximation pour minimiser le makespan. Dupont et Dhaenens [Dupont 02] examinent un problème d'ordonnancement des tâches sur une seule machine de traitement, afin de minimiser le makespan. Les auteurs présentent quelques propriétés de dominance pour un schéma d'énumération général et fournissent une procédure par séparation et évaluation. Pour les problèmes de grande taille, ils utilisent ce schéma d'énumération comme une méthode heuristique. Pour ce qui est de Chen [Chen 06a], il développe un algorithme de séparation et évaluation afin de résoudre un problème d'ordonnancement dans un environnement flow shop réentrant avec permutation pour minimiser le makespan. Yang et al. [Yang 08a] examinent un problème d'ordonnancement avec des flux de production réentrants. Le problème étudié se compose de deux machines, l'objectif est de minimiser le temps d'exécution maximum. Un algorithme de procédure par séparation et évaluation est proposé pour résoudre ce problème. Dans l'étude menée par Choi et Kim [Choi 09a], les auteurs présentent eux aussi un algorithme de séparation et évaluation pour un problème d'ordonnancement dans un environnement flow shop à deux machines avec comme objectif la minimisation du retard total. Dans ce système, toutes les tâches doivent être traitées deux fois sur chaque machine. Ils développent aussi, une borne inférieure, une heuristique et quelques propriétés de dominance pour les intégrer dans l'algorithme de résolution. Choi et al. [Choi 09b] étudient un problème d'ordonnancement dans un flow shop hybride, composé de deux étages en série sous la contrainte des dates dues maximales afin de minimiser le makespan. Pour résoudre ce problème, les auteurs proposent deux types d'algorithmes, le premier est un algorithme de séparation et évaluation et le second est une heuristique.

1.6.2 Heuristiques pour la résolution des problèmes d'ordonnancement réentrant

Un grand nombre de problèmes d'ordonnancement des systèmes de production avec processus réentrant sont complexes. En outre, les méthodes exactes ne sont pas en mesure de les résoudre exactement dans un délai raisonnable. À cet effet, plusieurs chercheurs proposent les heuristiques comme un moyen intéressant pour résoudre cette classe de problèmes. Comme Arzi et Raviv [Arzi 98] qui traitent un problème de répartition des jobs à une station de travail faisant partie d'une ligne de production rentrante. Quatre heuristiques sont examinées et les résultats montrent une bonne performance non seulement en termes de rendement moyen, mais aussi au niveau du temps total de réglage et le nombre de travaux en cours. Cigolini [Cigolini 99] considère une unité de production de circuits intégrés d'une entreprise manufacturière. La nature réentrante du système complique le problème, pour cela les auteurs regroupent un ensemble de tâches en fonction de leurs natures à partir d'un ordonnancement d'atelier basé sur des règles de priorité simples, ensuite ils les classent en fonction de leurs crédibilités en termes de capacité de production et de flexibilité. Enfin, pour chaque classe, une nouvelle règle de priorité est proposée et testée. Quant à Park et al. [Park 02] ils proposent dans cet article une approche approximative pour estimer les mesures de performance d'une ligne réentrante avec des machines qui ne traitent qu'une seule tâche et

des machines de traitement par lots basées sur la méthode d'analyse de la valeur moyenne. Les mesures de performances utilisées sont respectivement l'état d'équilibre de la durée du cycle de chaque classe de tâches, la longueur de la file d'attente et le rendement du système. Mason et Oey [Mason 03] traitent un problème d'ordonnancement de fabrication de plaquettes dans une usine de production de semi-conducteurs afin de minimiser le retard total pondéré ce qui implique la maximisation de la performance de livraison. Le système considéré est de type job shop qui contient des flux de produits réentrants dans des machines fonctionnant en parallèle. Certaines de ces machines encourent des séquences de configurations dépendantes, tandis que d'autres sont capables de traiter plusieurs tâches par lots. Les auteurs démontrent que ce problème peut être modélisé comme un graphe disjonctif, puis ordonnancé à l'aide d'une heuristique appelée *Shifting Bottleneck* modifié. Pour ce qui est de Chen et al. [Chen 04], ces derniers présentent une heuristique dépendante de l'état dynamique d'une usine de fabrication de plaquettes. Cette heuristique utilise dynamiquement différentes règles de priorités en fonction de l'état de système de production. Comme critères de performance, les auteurs utilisent la moyenne et l'écart type du temps de cycle et le nombre des travaux en cours. Barua et al. [Barua 05] proposent une heuristique d'optimisation dans une usine de fabrication de semi-conducteurs où les tâches sont réparties sur l'atelier en fonction de leurs dates de début. La performance de la procédure proposée par rapport à un certain nombre de règles de priorités bien connues est évaluée en utilisant un modèle de simulation d'une usine de fabrication de plaquettes en termes de plusieurs mesures de performance : le retard maximum, la moyenne des retards ainsi que le retard moyen de toutes les tâches. En ce qui concerne Miragliotta et Perona [Miragliotta 05], ils présentent une nouvelle approche d'ordonnancement d'ateliers réentrants pour la minimisation du retard moyen. Les tâches à traiter sont sélectionnées sur la base d'une évaluation équilibrée. Cette approche repose sur une heuristique s'appuyant sur une architecture décentralisée, dans laquelle chaque ressource de production peut agir comme un décideur indépendant et sélectionne les tâches à traiter en fonction de l'évolution dynamique des critères. Adan et Weiss [Adan 06] considèrent deux machines avec trois étapes de traitement dans des lignes réentrantes. La procédure utilisée est une technique qui consiste à faire servir en premier, le dernier buffer. Les délais de traitement sont indépendants. Une analyse du système est faite afin d'obtenir un état de stabilité. Liu et al. [Liu 07] traitent l'infrastructure d'un système d'ordonnancement temps réel avec quatre objectifs de performance spécifiques (maximisation du profit de l'unité ; respect des délais et minimisation de l'inventaire des travaux en cours) dans une usine de semi-conducteurs. Ces critères sont intégrés dans un algorithme d'ordonnancement, ensuite tous les lots de plaquettes seront expédiés dynamiquement par le système en temps réel. Pour ce qui est de Bahaji et Kuhl [Bahaji 08], ils évaluent des règles de priorités et d'autres politiques d'ordonnancement dans deux usines de fabrication de plaquettes. Après la modélisation rigoureuse des deux usines et une analyse statistique, les auteurs trouvent de nouvelles règles de priorités robustes pour optimiser la moyenne et la variance du temps de cycle ainsi que les mesures d'adhérence des dates dues dans les deux modes de production.

Stockton et al. [Stockton 08a] examinent l'utilisation de la politique de maîtrise des points pour améliorer le niveau de service, en répondant plus rapidement aux commandes et en réduisant les niveaux de stocks d'encours dans le système de fabrication. Ensuite, ils

comparent l'efficacité de cette politique à la règle d'ordonnancement RC (Rapport Critique). Pour ce qui est d'Upasani et Uzsoy [Upasani 08], ils améliorent les performances de calcul de l'approche d'ordonnancement proposé par Barua et al. [Barua 05] et utilisent un modèle de simulation d'une usine de fabrication de plaquettes à échelle réduite pour évaluer la performance des procédures proposées dans un environnement multi-produits avec des pannes de machines stochastiques, les auteurs considèrent le rendement moyen de l'atelier comme critère de performance général. Les résultats montrent que la procédure intégrée surpasse les règles de priorités de référence, tout en réduisant considérablement les temps de calcul. Caggiano et Jackson [Caggiano 08] présentent un algorithme général de construction d'ordonnancement pour la mise en œuvre d'une version cyclique des règles de priorités. L'objectif est de minimiser la somme pondérée des temps de cycle. Le problème abordé est caractérisé par des jobs multiples réentrants et non-identiques. En ce qui concerne, Omar et al. [Omar 09], ils examinent une ligne réentrante probabiliste dans une industrie d'assemblage de semi conducteurs, les auteurs introduisent une méthode de résolution afin de dépasser une telle complexité et de permettre ainsi aux gestionnaires opérationnels de calculer les mesures de performance telles que le temps de cycle total et le rendement moyen du système.

Zhang et al. [Zhang 09a] décrivent une approche d'évaluation et d'optimisation des règles de priorités en intégrant la simulation dans une industrie de semi-conducteurs pour optimiser le nombre de travaux en cours en temps réel ce qui conduit à la minimisation du temps de cycle du système. Dans l'étude menée par Choi et al. [Choi 11], les auteurs traitent un problème de flow-shop hybride réentrant dont l'objectif est de déterminer la répartition des tâches aux machines et ceci à chaque étape ainsi que la séquence des tâches assignées à chaque machine. Ils proposent un mécanisme d'ordonnancement en temps réel basé sur les règles de priorités afin de maximiser le rendement du système et de minimiser le temps de cycle, le retard moyen et le nombre de travaux en retard. Jing et al. [Jing 11] étudient un problème d'ordonnancement d'un flow shop réentrant afin de minimiser le temps d'exécution total. Un algorithme basé sur une heuristique est présenté pour résoudre le problème, dans lequel une technique effective est présentée comme une stratégie d'amélioration des itérations. Yang et al. [Yang 11] proposent, quant à eux une stratégie appelée *lean-pull*. La procédure est appliquée à une usine qui fabrique des écrans. L'étude montre que la stratégie proposée permet de réduire le temps de cycle, de diminuer le nombre de travaux en cours de traitement et d'améliorer le rendement. Xie et al. [Xie 11] traitent un problème d'ordonnancement qui émerge d'une ligne de production d'emballage réelle dans l'industrie du fer et de l'acier. Certaines propriétés sont établies pour l'identification d'un ordonnancement optimal. En outre, un algorithme à base d'heuristique est proposé pour minimiser le makespan.

1.6.3 Métaheuristiques pour la résolution des problèmes d'ordonnancement réentrant

Les métaheuristiques sont une famille d'algorithmes de types stochastiques destinées à résoudre des problèmes complexes qui sont généralement NP-difficiles. Elles permettent de trouver des solutions quasi-optimales. Par conséquent, et devant le succès rencontré par ces techniques, plusieurs auteurs les proposent comme des approches adéquates pour la résolution des problèmes possédant un caractère réentrant. Parmi ces auteurs, nous citons : Jain et al.

[Jain 03] qui considèrent un problème de fabrication de plaquettes dans une usine de semi-conducteurs caractérisée par un flux réentrant et un environnement de production dynamique et incertain. Dans ce cadre, les auteurs développent une stratégie d'ordonnancement basée sur le recuit simulé en considérant le temps de cycle moyen et le retard comme des mesures de performance pour obtenir un ordonnancement robuste. Aussi, ils développent un modèle de réseau de pétri stochastique généralisé qui capture les comportements dynamiques lors de la fabrication de plaquettes tels que le traitement réentrant, les défaillances de la machine, le chargement et le déchargement, etc. L'analyse de variance est aussi utilisée pour examiner les effets de l'interaction de diverses règles d'ordonnancement. En ce qui concerne, Lee et Lee [Lee 03], ils étudient un problème d'ordonnancement dans une usine de fabrication de produit continu qui se caractérise par un processus réentrant, dans lequel une séquence de mesures similaires est répétée plusieurs fois. Les auteurs présentent dans cet article, trois types de politiques de contrôle qui sont respectivement : *push*, *push-pull* et *pull*. La politique de type *push* surpasse les deux autres types dans le rendement et le temps de cycle de fabrication. En outre, une formulation exacte basée sur la programmation linéaire est proposée.

Lin et al. [Lin 04] traitent un problème d'ordonnancement avec la présence de contraintes de capacité en utilisant le concept de la théorie des contraintes pour une opération de test de circuits intégrés afin de maximiser le rendement du volume engagé. Le système se caractérise par la présence de nombreuses machines parallèles et différentes contraintes. Dans ce sens, les auteurs développent un modèle de simulation à événements discrets pour mettre en œuvre l'algorithme d'ordonnancement. D'autre part, une comparaison est faite avec d'autres règles d'ordonnancement classiques. Pour ce qui est de Liu et Wu [Liu 04], les auteurs considèrent un problème de lignes de production réentrantes caractérisé par un acheminement de produit qui se compose de plusieurs visites au cours du processus de fabrication. Ils obtiennent quelques résultats concernant l'ordonnancement de la chaîne de production micro-électronique réentrante fondé sur des règles de séquencement, mais cette dernière donne de mauvais résultats en raison de leurs sensibilités à la variation des types de ligne de production. Pour cela, les auteurs développent un algorithme génétique. Les résultats de simulation statiques et dynamiques montrent que l'algorithme améliore les performances de la production micro-électronique non seulement en termes de temps de cycle moyen, mais aussi par rapport aux nombres de produits en cours de traitement et le taux moyen de production. Balasubramanian et al. [Balasubramanian 04] traitent un problème d'ordonnancement trouvé dans les zones de diffusion et d'oxydation de fabrication de plaquettes dans une usine de semi-conducteurs, où les machines peuvent être modélisées comme des processeurs de traitement par lots parallèles afin de minimiser le retard total pondéré. Les auteurs proposent deux versions différentes d'un algorithme génétique, chacune composée de trois phases différentes. Aussi, d'autres heuristiques sont utilisées pour la résolution de ce problème. Galante et Passannanti [Galante 07] traitent, dans cet article, un problème de détection des points d'inspection avec l'hypothèse que l'action à entreprendre est connue quand une pièce défectueuse est détectée dans un environnement job shop et examinent aussi trois politiques de contrôle différentes en termes de nombre d'opérations contrôlées. En ce qui concerne le type d'intervention effectuée sur la détection d'un défaut, les auteurs montrent qu'un processus de décision séquentiel basé sur des mesures d'optimisation distinctes peut conduire à des

résultats non-satisfaisants. Pour cette raison, une approche intégrée est proposée dans une tentative d'identifier une solution optimale afin de minimiser les coûts de production en utilisant un algorithme génétique. Quant à Manikas et Chang [Manikas 08], ils étudient un problème d'ordonnancement d'un job shop réentrant en proposant un algorithme génétique qui peut être facilement appliqué et modifié pour minimiser la somme pondérée d'une variété de critères d'optimisation (les avances, les retards, l'ordre des tâches et des demandes) ensuite une autre mesure est ajoutée qui est la minimisation du makespan dans un environnement de production caractérisé par des temps de réglage dépendant de la séquence à ordonnancer. Chen et al. [Chen 08a] considèrent un environnement de production de type flow shop réentrant où toutes les tâches ont le même cheminement sur les machines de l'atelier. Le but de cette étude est de minimiser le makespan en utilisant l'algorithme génétique pour passer de la solution locale à la solution quasi-optimale pour ce problème d'ordonnancement. En outre, un algorithme génétique hybride est proposé pour améliorer la performance de l'algorithme génétique ensuite, il est comparé avec les solutions optimales obtenues par la technique de programmation en nombres entiers. Chen et al. [Chen 08b] considèrent eux aussi le même type de problème mais ils proposent une méthode de recherche tabou pour minimiser le makespan. Cette méthode est ensuite comparée aux solutions optimales générées par un programme mathématique en nombres entiers. Stockton et al. [Stockton 08b] décrivent l'application d'un algorithme génétique intégré à un modèle de simulation à événements discrets pour sélectionner les valeurs optimales pour la politique de point critique et la taille des buffers. Les caractéristiques utilisées pour mesurer les performances du système sont respectivement la moyenne totale des travaux en cours, l'espace total des buffers et le niveau de service. Les auteurs présentent une méthode pour réduire le nombre de variables afin d'améliorer l'efficacité de l'algorithme génétique.

Sun [Sun 09] traite, dans cette étude un problème d'ordonnancement dans un environnement job shop réentrant avec un réglage de temps qui dépend de la séquence. La minimisation du makespan est utilisée pour détecter les interactions entre les machines. Sur la base de cette représentation, deux procédures heuristiques et un algorithme génétique sont proposés pour obtenir des solutions quasi-optimales pour ce problème. En outre, une méthode expérimentale de conception est aussi présentée pour déterminer les différents paramètres génétiques basés sur l'approche Taguchi. Alfieri [Alfieri 09] étudie un problème d'ordonnancement flow shop afin d'ordonnancer un ensemble de commandes dans les délais et de minimiser les temps de traitement dans une entreprise de carton qui se caractérise par un flux réentrant, des stations multi-machines et des temps de réglage dépendant de la séquence de travail. Ils présentent une méthodologie de résolution basée sur une simulation dans laquelle la séquence de production peut être choisie par l'utilisateur ou trouvée par un algorithme basé sur la recherche tabou. En ce qui concerne, Liu [Liu 09], il étudie un problème d'ordonnancement flow shop de plusieurs commandes effectuées dans un atelier composé de trois machines pour réaliser des opérations de fabrication dans une usine de semi conducteurs. Un algorithme génétique est présenté pour résoudre ce problème afin de minimiser le retard total pondéré. Rau et Cho [Rau 09] proposent une approche basée sur les algorithmes génétiques pour résoudre le problème d'ordonnancement dans un système de production. L'auteur effectue une étude sur les paramètres de l'algorithme afin de suggérer les

plus appropriés à utiliser pour une obtenir une performance dans la production. Choi et Ko [Choi 09c] développent un algorithme génétique, pour le problème d'ordonnancement d'une ligne réentrante d'une usine de production, la métaheuristique suggérée est basée sur la simulation afin d'optimiser le rendement du système de production.

Dugardin et al. [Dugardin 09a] étudient un problème d'ordonnancement d'une ligne de maintenance réentrante constituée de plusieurs étages de machines parallèles. Ils proposent un algorithme de colonie de fourmis multi-objectif avec une procédure de recherche locale afin de minimiser le temps de cycle moyen des produits et de maximiser le taux d'utilisation de la machine goulet. Les mêmes auteurs [Dugardin 09b] proposent, quant à eux deux algorithmes L-NSGA (Lorenz Non dominated Sorting Genetic Algorithm) et SPEA2 (Strength Pareto Evolutionary Algorithm version 2) pour minimiser le makespan et la somme des retards dans un environnement flow shop hybride réentrant. Dans [Dugardin 10] les auteurs proposent un nouvel algorithme génétique multi-objectif L-NSGA pour résoudre un problème d'ordonnancement dans un environnement flow shop hybride, d'autre part, ils développent un modèle stochastique qui est utilisé avec un module de simulation à événements discrets pour maximiser le taux d'utilisation et minimiser le temps d'exécution maximal sur le même système. Ces résultats sont améliorés davantage dans [Dugardin 12] en ajoutant différentes recherches locales à l'algorithme L-NSGA afin de minimiser le makespan et le retard total. Lee et Lin [Lee 10] présentent un algorithme génétique pour l'ordonnancement d'un problème de type flow shop avec des tâches réentrantes. Les objectifs de cette recherche sont de minimiser le retard pondéré et le makespan. Le modèle proposé considère que les tâches possèdent des temps de traitement non-identiques. Elmi et al. [Elmi 11] abordent un problème d'ordonnancement des pièces dans un job shop d'une entreprise manufacturière où les tâches doivent visiter les machines à plusieurs reprises de manière non-consécutive. Une approche de résolution à base de recuit simulé est développée. Ensuite un modèle de programmation linéaire en nombres entiers est présenté pour minimiser le makespan. Quant à Lin et al. [Lin 12], ils considèrent un problème d'ordonnancement flow shop hybride réentrant dans une entreprise de réparation. Ces auteurs proposent une combinaison entre un algorithme génétique et la méthode *analytical hierarchy process* pour remplir les différents critères dont le but est d'obtenir des ordonnancements quasi-optimaux. Ensuite, cette méthode est adoptée dans le processus de sélection de l'algorithme génétique pour minimiser le retard total. Moghaddam et al. [Moghaddam 12a] étudient un problème de type flow shop réentrant composé de deux machines dont les tâches ont des délais strictes pour minimiser le makespan pour les travaux en interne et pour minimiser les coûts en externe pour d'autres travaux. Les auteurs proposent un algorithme génétique efficace basé sur la notion de dominance modifiée. Ce dernier montre son efficacité par rapport à l'algorithme NSGA II (Non dominated Sorting Genetic Algorithm version 2) classique. En ce qui concerne, Li et al. [Li 13b], ces derniers développent un algorithme génétique multi-niveaux pour minimiser le makespan d'un flow shop réentrant, sans prendre en considération l'interdépendance entre les contraintes de ressources. Belkaid et al. [Belkaid 13c] proposent un algorithme génétique pour la résolution d'un problème d'ordonnancement sur machines parallèles avec ressources consommables et processus réentrant. Plusieurs heuristiques sont adaptées dont le but est la minimisation du makespan. Les résultats démontrent l'efficacité de l'algorithme développé.

Tableau 1.2 : Récapitulatif des travaux sur les problèmes d'ordonnancement réentrant

Articles publiés	Type de problème	Les approches de résolution			Les mesures de performances		
		Méthodes analytiques	Métaheuristiques	Heuristiques	Classe 1	Classe 2	Classe 3
[Graves 83] [Arzi 98] [Chen 04] [Caggiano 08] [Xie 11]	GRSP			X	X		
[Cochran 03] [Nazarathy 09]	GRSP	X				X	
[Omar 09] [Zhang 09a] [Yang 11]	GRSP			X	X	X	
[Wang 97] [Moon 99]	GRSP	X		X	X		
[Kumar 04] [Lee 06] [Lee 07] Yang 08a] [Ignisio 08] [Chun 09]	GRSP	X			X		
[Lin 04] [Barua 05] [Upasani 08]	GRSP			X		X	
[Balasubramanian 04] [Choi 09c]	GRSP		X			X	
[Miragliotta 05] [Liu 07]	GRSP			X	X	X	X
[Park 02] [Bahaji 08]	GRSP			X	X		X
[Adan 06] [Stockton 08a]	GRSP			X			X
[Bard 99]	GRSP	X	X		X		
[Vargas 00]	GRSP	X		X		X	
[Odrey 01]	GRSP	X				X	
[Dupont 02]	GRSP	X			X		
[Jain 03]	GRSP	X	X		X		X
[Lee 03]	GRSP	X		X		X	X
[Liu 04]	GRSP		X		X		
[Dugardin 09a]	GRSP		X		X	X	
[Stockton 08b]	GRSP		X			X	X
[Upasani 08]	GRSP			X		X	
[Yan 12]	GRSP	X					X
[Belkaid 13c]	GRSP		X	X	X		
[Liu 09] [Lin 12]	RFSP		X				X
[Alfieri 09] [Rau 09] [Chen 08a] [Chen 08b]	RFSP		X		X		
[Pan 03] [Pan 04] [Choi 09b]	RFSP	X		X	X		
[Dugardin 10] [Moghaddam 12]	RFSP		X		X	X	
[Lee 10] [Dugardin 09b] [Dugardin 12]	RFSP		X		X		X
[Hwang 99]	RFSP	X	X		X		
[Cigolini 99]	RFSP			X		X	
[Chen 06a] [Chen 06b]	RFSP	X			X		
[Choi 09a]	RFSP	X					X
[Choi 11]	RFSP			X	X	X	X
[Jing 11]	RFSP			X	X		
[Li 13b]	RFSP		X		X		
[Mason 03]	RJSP			X	X		
[Low 05]	RJSP	X			X	X	X
[Galante 07]	RJSP		X			X	
[Manikas 08]	RJSP		X		X		X
[Sun 09]	RJSP		X	X	X		
[Elmi 11]	RJSP	X	X		X		
[Gomes 13]	RJSP	X				X	X

1.6.4 Autres investigations sur les problèmes d'ordonnancement réentrants

Dans le cadre de résolution des problèmes d'ordonnancement, quelques chercheurs ont proposé des solutions basées sur la méthode de l'analyse de la valeur moyenne qui est une technique itérative pour analyser un système dans un état stable. Nous citons le travail de Kumar et Omar [Kumar 06] dans lequel les auteurs présentent un modèle analytique modifié efficace basé sur l'analyse de la valeur moyenne approximative avec des lignes réentrantes afin de minimiser le temps moyen d'attente et ensuite le taux moyen de rendement sous divers routages stochastiques. Certaines recherches sur l'ordonnancement avec processus réentrant nécessitent la prise de décision d'une manière décentralisée afin de proposer une solution globale à travers la résolution des sous-problèmes localement par les agents et leurs interactions. Cela ce traduit par l'utilisation des systèmes multi-agents. L'analyse de la revue de la littérature nous permet d'identifier le travail de Zhang et al. [Zhang 07] dans lequel, les auteurs proposent une technologie de modélisation multi-agents pour un système de production réentrant avec la présence de perturbations. L'objectif est de procurer au système plus de flexibilité et d'atteindre une productivité élevée. Un modèle de réseaux de pétri temporisé coloré est proposé, il consiste en un modèle de réseaux de flux de matières et d'un modèle de protocole interactif. Une plateforme de simulation à événements discrets distribués est construite pour évaluer la performance de ce système. En outre, d'autres investigations abordent un problème d'ordonnancement d'atelier simple avec processus réentrant dont le but est de dégager des propriétés théoriques et de fournir un ordonnancement optimal, comme Chu et al. [Chu 10] dans lequel les auteurs proposent diverses propriétés de solutions optimales pour minimiser le makespan et le temps de cycle et à partir de ces propriétés ils arrivent à définir trois cas de solutions optimales sur quatre possibles. Finalement, quelques travaux utilisent des algorithmes auto-adaptatifs comme celui présenté dans l'article de Li et al. [Li 13a] où ils proposent un algorithme auto-adaptatif hybride pour minimiser le makespan d'un flow-shop réentrant avec permutation. En outre, pour améliorer la capacité d'exploitation locale de l'algorithme, une recherche locale basée sur le chemin critique est présentée pour exploiter les différentes régions de l'espace de recherche.

1.6.5 Synthèse des travaux de recherches sur les problèmes d'ordonnancement avec ressources consommables et processus réentrant

Dans la revue de la littérature présentée dans ce chapitre, nous avons mis l'accent sur la résolution de certains problèmes d'ordonnancement dans les systèmes de production. Nous avons synthétisé un état de l'art sur les deux thématiques de recherches qui nous intéresse. Nous avons exposé dans un premier temps la problématique d'ordonnancement dans les systèmes de production, ceci nous a permis de souligner les points essentiels relatifs non seulement aux problèmes d'ordonnancement sur machines parallèles mais aussi sur les problèmes d'ordonnancement avec ressources consommables. Dans un second temps, nous avons abordé, les problèmes d'ordonnancement réentrant en présentant leurs définitions et les différentes particularités extraites de la littérature ensuite, nous avons montré l'importance de ce type de problème qui ne cesse d'attirer sensiblement l'intérêt des chercheurs en raison de la forte augmentation du nombre de publications ces dernières années. Les problèmes d'ordonnancement avec ressources consommables sont devenus fréquents dans la gestion des

opérations des entreprises manufacturières. Toutefois, une partie importante des études des problèmes d'ordonnancement sont basées sur des hypothèses simples, loin de la réalité : le temps de traitement des tâches est indépendant de la consommation des ressources, les composants arrivent au début du traitement ou elles sont toujours disponibles.... En fait, la considération de cette contrainte permet aux décideurs de déterminer le niveau approprié des ressources pour le contrôle du système en fonction de leurs besoins ce qui lui permet de réagir à tous les changements externes ou internes. Par conséquent, la contrainte des ressources consommables mérite une réflexion plus approfondie et l'analyse de ses effets sur d'autres configurations s'avère nécessaire. La revue de la littérature relative aux problèmes d'ordonnancement réentrant montre qu'il existe un arsenal important de travaux s'intéressant à ce domaine vu ses implications sur la gestion opérationnelle et l'utilisation optimale des capacités de production. Ceci a motivé les chercheurs à considérer la caractéristique réentrante du système lors du processus décisionnel pour l'ordonnancement. Dans ce sens, plusieurs approches de résolution ont été proposées, la majorité d'entre elle se place dans un environnement mono critères (makespan, temps de cycle, retards,...), d'autres incluent différents autres facteurs et contraintes liés aux systèmes tout en négligeant d'autres comme la contrainte de ressources. Malheureusement, un système de production est soumis à plusieurs perturbations ; son efficacité dépend étroitement de plusieurs critères de performance dont certains sont contradictoires entre eux (maximiser la rentabilité, minimiser les retards,…) et nécessite la prise en compte de différentes conditions de production qui sont susceptibles de changer au cours du temps, etc. Nous pouvons conclure que les problèmes d'ordonnancement ne cessent de croitre en complexité, par conséquent, le développement d'un système pratique d'ordonnancement reste une phase contraignante et un défi à soulever par les chercheurs…

1.7 Conclusion

Dans ce chapitre, le concept des problèmes d'ordonnancement dans les systèmes de production a été arboré. Les définitions de base et les terminologies de ce domaine ont été discutées. Les diverses classifications et types de décisions ont été illustrés. Ensuite, les différentes structures des problèmes d'ordonnancement et les méthodologies de résolutions déployées ont été brièvement présentées. En outre, afin de recueillir et d'extraire les connaissances existantes et d'étudier l'évolution du domaine de l'ordonnancement, une analyse bibliographique a été effectuée. Mais en raison de l'immensité des problèmes d'ordonnancement dans les entreprises manufacturières, la recherche bibliographique a été focalisée principalement sur les travaux d'ordonnancement dans un environnement de machines parallèles avec ressources consommables dans un premier temps et processus réentrant dans un second temps. Dans la première partie de notre état de l'art, nous avons montré l'importance du makespan et l'intérêt des algorithmes génétique dans l'environnement de machines parallèles ensuite nous avons apporté des détails sur la complexité des problèmes d'ordonnancement sur machines parallèles avec ressources consommables et enfin nous avons présenté l'ensemble des travaux qui traitent ce problème. La seconde partie a été consacrée au regroupement des travaux les plus importants qui s'intéressent à la modélisation et l'adaptation des méthodes analytiques, heuristiques et métaheuristiques pour la résolution des problèmes d'ordonnancement réentrant.

Chapitre 2

État de l'art sur les métaheuristiques pour l'ordonnancement

2.1. Introduction

Les problèmes d'ordonnancement ont été intensivement étudiés ces dernières années et continus encore a attirer l'attention des chercheurs spécialisés dans la gestion, la recherche opérationnelle et la productique. La fréquence des problèmes d'ordonnancement s'accroit dans de nombreuses industries, comme la fabrication de semi-conducteurs, les industries aéronautiques, le transport, la manutention, l'emballage et les systèmes de stockage. Ces problèmes sont exprimés comme des problèmes d'optimisation, qui sont souvent complétés par des informations de contraintes relatives aux ressources (consommables ou renouvelables), aux tâches (préemptives, indépendantes…), aux processus (réentrant ou non)...

En effet, tous les paramètres des solutions adoptées doivent satisfaire ces contraintes et doivent être très réactifs pour faire face à des circonstances imprévues. Autrement, ces solutions ne seraient pas satisfaites. En outre, il a été démontré que l'ordonnancement dans un tel environnement est une tâche extrêmement difficile [Garey 79], puisque ces systèmes sont caractérisés par leurs complexités fonctionnelles et l'évolution rapide de leurs états.

Suite à cela, de nombreux efforts ont été menés depuis longtemps pour résoudre ce type de problèmes et un important arsenal de méthodes d'optimisation traditionnelles a été proposé [Horst 95]. Cependant, les approches d'optimisation traditionnelles telles que la programmation linéaire en nombres entiers ou la programmation dynamique sont souvent inefficaces si la fonction objectif ne possède pas une propriété structurelle particulière, et sont d'un intérêt limité en raison de leur temps de calcul excessif.

En conséquence, des heuristiques ont été proposées. Toutefois, ces approches sont dédiées à un problème spécifique, il est donc difficile de les adapter à d'autres problèmes. En plus, elles sont généralement conçues pour fournir une solution unique alors que la majorité des problèmes de décision ont un grand nombre de solutions possibles. Pour pallier à ces

inconvénients, les métaheuristiques, qui font partie des algorithmes approximatifs, constituent un moyen efficace pour la résolution de cette classe de problèmes.

L'objectif de ce chapitre est de donner un aperçu des dernières tendances des métaheuristiques pour résoudre des problèmes pour lesquels les algorithmes d'optimisation classiques sont incapables de fournir des résultats satisfaisants. La prochaine section de ce chapitre vise la présentation des notions de base liées aux problèmes d'optimisation en exposant leurs définitions et classifications. La troisième section est consacrée aux principes des métaheuristiques les plus connus en présentant leurs définitions, caractéristiques, principaux concepts et classifications. La dernière section a pour objectif de présenter les métaheuristiques les plus répandues qui sont à base de solution unique (le recuit simulé et la recherche tabou) et à base de populations (les colonies de fourmis, les essaims particulaires, les algorithmes génétiques) en rappelant leurs origines, leurs mode de fonctionnement et leurs algorithmes de base.

2.2. Généralités sur les problèmes d'optimisation

2.2.1 Définition

L'optimisation est une branche des mathématiques qui consiste à trouver les conditions qui donnent la valeur optimale d'une ou de plusieurs fonctions dans des circonstances données. En d'autres termes, elle consiste à s'engager dans une démarche visant à trouver la meilleure solution à un problème d'optimisation.

Un problème d'optimisation peut être modélisé au moyen d'un ensemble de variables de décisions avec leurs domaines et contraintes relatives aux variables. Selon Blum et al. [Blum 08], il peut être représenté comme un triplé (S, Ω, f) où : S est le domaine de recherche défini sur un ensemble de variables de décisions, Ω est l'ensemble des contraintes à satisfaire et $f : S \rightarrow R^{+}$est la fonction objectif à optimiser.

Dans le cas de la minimisation, le but est de trouver une solution $s^{*} \in S$ tel que $f(s^{*}) \leq f(s), \forall s \in S$ et dans le cas de la maximisation, l'objectif est de trouver une solution $s^{*} \in S$ tel que$f(s^{*}) \geq f(s), \forall s \in S$.

2.2.2 Classification

L'optimisation constitue un paradigme important avec un large éventail d'applications ce qui nécessite la prise en compte de plusieurs critères, qui peuvent être contradictoires entre eux (minimiser la consommation d'énergie, maximiser les profits,…). En outre, les ressources peuvent être limitées et d'autres facteurs peuvent être considérés, etc., on constate alors que plusieurs contraintes doivent être prises en compte ce qui implique l'intégration d'un nombre important de variables de conception, de contraintes et de fonctions objectifs.

Selon le choix spécifique de ces variables, différents types de problèmes d'optimisation peuvent être trouvés. Le tableau 2.1 présente une classification des différents problèmes d'optimisation recueillis [Foulds 81], [Vanderplaats 99], [Rao 09], [Sahab 13].

Tableau 2.1 Classification des problèmes d'optimisation

Base de classification	**Catégorie**	**Classification**
Nombre de variables de décisions	Une seule variable	Le vecteur de variables de décisions comprend une seule variable
	Multi variable	Le vecteur de variables de décisions comprend plus d'une variable
Nombre de fonctions objectifs	Un seul objectif	Il y a un critère exprimé en une seule fonction objective
	Multi objectif	Il y a plusieurs critères qui sont considérés pour l'optimisation du problème
Présence de contraintes	Sans contraintes	Le problème à traiter ne contient aucune contrainte
	Avec contraintes	Le problème considéré contient un ou plusieurs contraintes
Caractéristiques des contraintes et des fonctions objectifs	Linéaire	La fonction objectif et les contraintes sont linéaires
	Non linéaire	La fonction objectif ou les contraintes peuvent être non linéaires
Nature des variables de décisions	Statique	Les variables de décisions sont indépendantes des autres paramètres
	Dynamique	Les variables de décisions dépendent d'autres paramètres comme le temps
Type des variables de décisions	Discrète	Les variables de décisions prennent des valeurs entières ou discrètes
	Continue	Les variables de décisions prennent des valeurs réelles
	Mixte	Certaines variables de décisions prennent des valeurs discrètes et d'autres prennent des valeurs réelles
Nature des variables d'entrée et des variables de décisions	Déterministe	Toutes les variables du problème à optimiser sont supposées être déterministes
	Probabiliste	Les variables du problème sont supposées être aléatoires ou probabilistes

Cette analyse montre que les décideurs sont confrontés à des problèmes d'une complexité croissante comprenant une variété de fonctions objectifs et plusieurs types de contraintes sur différents domaines. Cette situation est encore plus compliquée par le fait que l'incertitude est presque toujours présente dans les cas à traiter dans l'industrie. Toutes ces raisons impliquent que ces problèmes sont NP-difficiles nécessitant un temps de calcul à croissance exponentielle pour être résolus. Les métaheuristiques qui font partie des algorithmes approximatifs constituent un moyen efficace pour résoudre ce genre de problèmes. Par conséquent, l'objectif est de chercher une conception optimale et robuste des métaheuristiques afin de mener à bien la recherche souhaitée qui peut être intégrée et associée à d'autres composants de modélisation.

2.3 Généralités sur les métaheuristiques

Les métaheuristiques sont une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation pour lesquels il n'existe pas d'approches conventionnelles plus efficaces. Elles sont décrites par leurs natures stochastiques et sont généralement développées sur la base de certains phénomènes naturels [Yang 10]. Les métaheuristiques sont devenues parmi les techniques d'optimisation les plus étudiées ces dernières années en raison de leurs capacités à trouver des solutions prometteuses pour différents problèmes d'optimisation complexes ainsi que leurs aptitudes à gérer à la fois des variables discrètes et continues. Alors que, la complexité du problème, rend impossible de chercher toutes les solutions possibles, les métaheuristiques représentent un excellent moyen pour fournir des solutions efficaces et réalisables dans des délais acceptables avec peu d'informations sur l'espace de recherche [Yang 08b]. Toutefois, la capacité de ces techniques pour optimiser un problème à partir d'un minimum d'informations est contrebalancée par le fait qu'elles ne garantissent pas l'optimalité de la solution trouvée. En général, l'idée est d'avoir un algorithme efficace et capable de produire des solutions de bonne qualité ou presque optimale, mais il n'y a aucune garantie de cette optimalité.

2.3.1 Définition

Les métaheuristiques sont une nouvelle génération de méthodes approchées puissantes et générales, elles sont adaptables et applicables à une large classe de problèmes. Dans la littérature, plusieurs définitions sont proposées pour expliquer ce qu'est une métaheuristique. Nous citons d'après Osman et Laporte [Osman 96]: qu'une métaheuristique est principalement définie comme un processus itératif de génération qui guide une heuristique subordonnée en combinant intelligemment divers concepts pour explorer et exploiter l'espace de recherche en élaborant des stratégies d'apprentissage pour structurer l'information dans le but de trouver au mieux des solutions quasi optimales. Tandis que Voss et al. [Voss 99] voient les métaheuristiques comme un processus itératif qui renforce et modifie les opérations heuristiques subordonnées pour produire des solutions adéquates. Les heuristiques utilisées peuvent être soit des procédures de recherche locale simple, ou bien des méthodes de construction. Le processus conçu peut manipuler une solution unique ou un ensemble de solutions à chaque itération. Ce type d'algorithme combine essentiellement des méthodes heuristiques dont l'objectif est d'explorer efficacement l'espace de recherche conduit par des

mouvements logiques et des connaissances de l'effet d'un mouvement **[**Blum 03]. Blum et al. [Blum 08] définissent les métaheuristiques comme des algorithmes qui permettent d'effectuer des recherches afin de proposer au hasard différentes solutions à un problème donné, ces solutions peuvent être optimales ou quasi optimales, le processus de recherche ne peut être accompli que lorsqu'une condition d'arrêt particulière soit atteinte ou qu'après un nombre prédéfini d'itérations. Quant à Tfaili [Tfaili 08], il considère que les métaheuristiques représentent une famille d'algorithmes inspirés de la nature, utilisés pour proposer des solutions aux problèmes que les algorithmes d'optimisation classiques n'ont pas pu résoudre. Ben Othman [Ben Othman 09] a décrit les métaheuristiques comme des méthodes de recherche générales, présentées sous forme de concepts et qui sont destinées à résoudre des problèmes d'optimisation.

La principale différence entre les métaheuristiques et les approches classiques est la façon dont elles proposent le prochain mouvement dans l'espace des solutions. Ici, les stratégies et les mécanismes utilisés pour proposer des mouvements plus fiables deviennent cruciaux. Cela motive les développeurs à proposer une synergie d'algorithmes d'optimisation. Nous présentons dans le tableau 2.2, les métaheuristiques les plus évoquées dans la littérature avec leurs développeurs, références originales et phénomènes d'inspiration.

Tableau 2.2 : Références des métaheuristiques et leurs inspirations

Métaheuristiques	**Références originales**	**Phénomènes d'inspiration**
Algorithmes génétiques	Holland [Holland 75]	Principe de la sélection naturelle et de l'évolution des espèces
Recuit simulé	Kirkpatrick et al. [Kirkpatrick 83]	Phénomène physique du recuit
Recherche tabou	Glover [Glover 86]	Mémoire humaine
Système immunitaire artificiel	Farmer et al. [Farmer 86]	Système immunitaire naturel
Recherche adaptative gloutonne et randomisée	Feo et Resende [Feo 89]	Mécanisme de construction par étape stochastique
Colonies de fourmis	Dorigo [Dorigo 92]	Comportement de fourmis réelles
Algorithme de kangourou	Fleury [Fleury 93]	Phénomène physique du recuit et le comportement des kangourous
Essaims particulaires	[Eberhart 95]	Comportements sociaux des animaux
Recherche d'harmonie	Geem et Loganathan [Geem 01]	Processus d'improvisation des musiciens
L'électromagnétisme	Birbil et Fang [Birbil 03]	Théorie d'électromagnétisme
Colonies d'abeilles	Karaboga [Karaboga 05]	Comportement d'un essaim d'abeilles
Algorithme luciole	Yang [Yang 08b]	Le comportement clignotant des lucioles (famille d'insecte)
Recherche "Cuckoo"	Yang et Deb [Yang 09]	Parasitisme des oiseaux
Algorithme de chauve-souris	Yang [Yang 10]	Comportement d'écholocation des microchiroptères
Recherche du système chargé	Kaveh et Talatahari [Kaveh 10]	Théorie de la physique et de la mécanique

2.3.2 Caractéristiques des métaheuristiques

Une métaheuristique est un processus de recherche qui s'articule autour de deux concepts très importants appelés diversification et intensification ou exploration et exploitation. Le terme diversification désigne l'exploration de l'espace de recherche pour générer des solutions différentes sur le problème à optimiser en utilisant une stratégie qui consiste à redémarrer périodiquement le processus de recherche à partir d'une solution générée aléatoirement ou bien choisie judicieusement dans une région non encore visitée de l'ensemble des solutions admissibles, ce qui augmente la diversité des solutions ; alors que l'intensification désigne l'exploitation de l'expérience de recherche accumulée, ce qui conduit le processus de recherche à se concentrer sur les zones jugées déjà prometteuses.

Chaque métaheuristique doit se caractériser par un équilibre entre la diversification et l'intensification. Ceci est important, d'une part pour explorer et identifier rapidement les régions de l'espace de recherche pour ne pas assister à une convergence vers des optima locaux et ainsi obtenir des solutions de haute qualité, et d'autre part éviter une exploitation trop longue des régions qui ont été déjà visitées et qui n'ont pas fourni de solutions prometteuses. L'intensification et la diversification sont deux caractéristiques principales des algorithmes métaheuristiques [Blum 03]. Par conséquent, la bonne combinaison de ces deux mécanismes majeurs peut conduire les solutions à converger vers des optimums globaux. Blum et Roli ont élaboré une étude sur l'importance de ces deux concepts dans les métaheuristiques [Blum 03].

Les propriétés fondamentales des métaheuristiques résident dans le fait qu'elles représentent un ensemble de stratégies de haut niveau qui permettent d'explorer des espaces de recherche en utilisant différentes méthodes. Elles s'appliquent à toutes sortes de problèmes discrets comme elles peuvent également s'adapter aux problèmes continus. En outre, ces méthodes partagent les caractéristiques suivantes [Dréo 03], [Clerc 04], [Blum 05] :

- Les métaheuristiques sont des méthodes d'optimisation inspirées par des analogies : avec la physique (recuit simulé...), la biologie (algorithmes évolutionnaires...) ou avec l'éthologie (colonies de fourmis, essaims de particules...);
- Les concepts de base des métaheuristiques peuvent être décrits à un niveau abstrait, sans être pour autant liés à un problème particulier.
- Elles ont pour but d'explorer l'espace de recherche d'une manière efficace afin de trouver des solutions optimales à des problèmes complexes.
- Les métaheuristiques sont des algorithmes qui partent de procédures de recherche locale assez simples à des processus d'apprentissage plus complexes.
- Elles peuvent adopter des mécanismes qui leurs permettent de ne pas être bloquées dans des régions de l'espace de recherche.
- Elles peuvent intégrer des méthodes, plus précisément des heuristiques qui permettent de prendre en compte la spécificité du problème.
- La plupart des ces algorithmes sont généralement non déterministes, approximatifs et ne donnent aucune garantie d'optimalité.
- Les métaheuristiques peuvent être utilisées pour divers problèmes mais leurs efficacités dépendent des paramètres utilisés qui sont généralement difficiles à ajuster.

2.3.3 Principaux concepts des métaheuristiques

Les métaheuristiques constituent une famille de divers algorithmes inspirés de la nature. Actuellement, ce domaine suscite l'intérêt de plusieurs auteurs comme Talbi [Talbi 09] qui consacrent leurs ouvrages uniquement pour ce type d'algorithmes. Les métaheuristiques partagent plusieurs points communs, surtout pendant la phase de conception. Selon Talbi [Talbi 09] l'étape de conception des métaheuristiques itératives dépend de deux concepts essentiels qui peuvent être résumés comme suit :

- ✓ La représentation de la solution
- ✓ La définition de la fonction objectif

2.3.3.1 La représentation de la solution

La représentation de la solution constitue une étape essentielle dans la conception et le développement de toute métaheuristique. Elle joue un rôle majeur quant à l'efficacité de ses algorithmes. Lors de la définition d'une représentation, le choix du codage est très important, il doit être adapté au problème étudié. En effet, il faut savoir comment appliquer les opérateurs de recherche (voisinage, reproduction…) sur cette représentation et de quelle manière la solution va être évaluée. De plus, toute représentation doit avoir les caractéristiques suivantes [Talbi 09] :

- ✓ **La complétude** : Toute solution réalisable doit pouvoir être représentée.
- ✓ **La connexité** : Un lien de recherche doit exister entre deux solutions de l'espace de recherche. Toute solution de l'espace de recherche, en particulier la solution optimale, peut être atteinte.
- ✓ **L'efficacité** : La représentation doit être facile à manipuler par les opérateurs de recherche (la complexité en temps et en espace doit être réduite).

2.3.3.2 La fonction objectif

La fonction objectif représente un élément important dans la conception d'une métaheuristique. Elle permet de guider la recherche vers de bonnes solutions de l'espace de recherche en associant, à chacune de ces solutions, une valeur réelle qui évalue sa qualité. Ainsi une fonction objectif mal définie, peut conduire à obtenir d'inacceptables solutions et ceci quelle que soit la métaheuristique utilisée.

2.3.4 Analyse des performances des métaheuristiques

Cette section présente les lignes directrices pour analyser les performances des métaheuristiques puisque une approche théorique n'est généralement pas suffisante [Bartz 06]. Pour évaluer la performance d'une métaheuristique d'une manière rigoureuse, trois étapes doivent être considérées [Talbi 09] et ceci de la manière suivante : Dans un premier temps, un plan d'expérience doit être défini, ensuite des mesures et des indicateurs de performances sont sélectionnés afin d'évaluer l'efficacité de la métaheuristique. Enfin une analyse statistique est appliquée sur les résultats obtenus une fois que les différentes expériences ont été réalisées.

2.3.4.1 Plan d'expérience

Les plans d'expériences sont utilisés pour organiser et structurer au mieux les essais et les expérimentations qui accompagnent une recherche scientifique ou des études industrielles [Goupy 01]. Durant cette étape, l'ensemble des expériences, les instances sélectionnées et les facteurs doivent être clairement définis ; car l'objectif de la conception d'une métaheuristique dépendra non seulement de ces expériences mais aussi de l'analyse statistique.

Cette dernière permet d'effectuer des tests de calcul afin d'assurer une étude statistique rigoureuse [Montgomery 84]. Elle peut être jugée en fonction de plusieurs indicateurs de performances comme le temps de calcul, la qualité des solutions, le taux d'utilisation des machines, le taux de production… ou bien en fonction d'un ensemble de combinaisons de paramètres liés à l'expérience. De ce fait, de nombreux paramètres doivent être pris en compte pour toute métaheuristique. Le réglage des paramètres représente une étape délicate et importante qui permet d'obtenir une plus grande flexibilité et robustesse, mais nécessite une initialisation bien réfléchie.

Ces paramètres peuvent avoir une grande influence sur l'efficacité et l'efficience de la recherche, cependant, il n'est pas évident de définir à priori quel paramétrage doit être utilisé et en plus un changement de paramètres peut induire un comportement complètement différent du système, en outre, les valeurs optimales pour les paramètres dépendent principalement de la nature du problème, de l'instance à traiter et du temps de recherche. Donc il est important de trouver le meilleur paramétrage de la métaheuristique utilisée afin de pouvoir analyser sa sensibilité et d'évaluer son efficacité par rapport au système considéré.

2.3.4.2 Évaluation

Une fois que le plan d'expérience est défini et les résultats expérimentaux sont obtenus pour les différents indicateurs, une analyse statistique peut être utilisée pour évaluer l'efficacité des métaheuristiques. Des indicateurs de performances, qui sont liés à la qualité des solutions doivent être envisagés et peuvent être classés en trois groupes [Barr 95] :

- ✓ la qualité des résultats
- ✓ le temps de résolution
- ✓ la robustesse.

2.3.4.3 Qualité des résultats

Une métaheuristique doit être en mesure de fournir une méthode de résolution performante du point de vue qualité de la solution. Les résultats obtenus en terme de précision peuvent être représentés par des indicateurs de performance qui mesurent l'écart entre "la solution obtenue" et "la solution optimale" ou "la meilleure solution connue" ou "des bornes inférieures".

2.3.4.4 Temps de résolution

Le temps de calcul représente un critère important qui permet de résoudre efficacement une instance donnée. Il doit être pris en compte dans l'évaluation des performances afin de déterminer les valeurs des paramètres. Le temps de calcul varie selon le critère d'arrêt choisi

comme par exemple le nombre d'itérations, le nombre de générations sans amélioration, le temps pour obtenir une solution donnée, etc.

2.3.4.5 Robustesse

La robustesse mesure la performance des algorithmes en fonction des différents types d'instances ayant des configurations différentes car une métaheuristique jugée robuste doit être capable de bien performer non seulement sur une grande variété mais aussi avec de différentes structures de problème.

2.3.4.6 Analyse statistique

Une analyse statistique peut être appliquée afin de pouvoir étudier l'effet des facteurs expérimentaux et pour procéder à l'évaluation de la performance des métaheuristiques conçues [Dean 99]. Cette étape peut être achevée une fois que les différentes expériences sont exécutées et les résultats expérimentaux sont obtenus pour les différents indicateurs en employant, dans un premier temps, un indicateur qui prend en considération certains facteurs qui résument la moyenne et les tendances d'écart. Ensuite, différents tests statistiques peuvent être menés pour analyser, comparer les métaheuristiques et pour évaluer la validité des résultats obtenus. Pour vérifier cette validité, les modèles d'analyse de variance s'avèrent des techniques bien établies [Cohen 95]. Ils permettent l'analyse simultanée de plusieurs mesures de performance. En ce qui concerne, l'interprétation des résultats, elle doit être la plus explicite possible et pilotée par les objectifs qui ont été définis en prenant en compte les mesures de performance considérées pour permettre une meilleure compréhension de l'évaluation de la performance des résultats obtenus.

Dans cette section, nous avons cité les critères les plus importants qui doivent être pris en compte pour l'évaluation des performances des métaheuristiques. Les indicateurs de performances tel que la qualité de la solution, la robustesse pour mesurer l'efficacité et le temps de calcul doivent être bien ajustés afin de déterminer les valeurs des paramètres de la métaheuristique pour résoudre d'autres instances. Cependant, il existe d'autres critères tels que la simplicité de mise en œuvre et la flexibilité qui peuvent aussi être employés. La section suivante est consacrée à la classification des métaheuristiques.

2.3.5 Classification des métaheuristiques

Au-delà du fait que chaque métaheuristique possède sa propre stratégie de recherche, il existe plusieurs façons de les classer selon des critères différents. Dans cette section, nous résumons brièvement les classifications les plus répandues [Stiitzle 99] [Blum 03].

2.3.5.1 Métaheuristique inspirée de la nature ou non inspirée de la nature

La façon qui semble la plus intuitive de classer les métaheuristiques est celle qui se base sur leurs origines, en les séparant en deux catégories. Celles qui ont été inspirées par des analogies avec la réalité : comme le recuit simulé avec la physique, les algorithmes génétiques avec la biologie et les colonies fourmis avec l'éthologie, de celles qui ne l'ont pas été comme la méthode de descente, ou la recherche tabou. Cependant, une telle classification ne semble pas très utile du fait qu'il existe de nombreuses métaheuristiques récentes qui ne

correspondent à aucune des deux classes (ou elles s'inscrivent dans les deux en même temps). Ou bien il s'avère parfois difficile d'attribuer un algorithme à l'une des deux catégories.

2.3.5.2 Métaheuristique avec ou sans mémoire

Une caractéristique très importante pour classer les métaheuristiques réside dans l'utilisation de l'historique de recherche (le passé) ou pas. Dans cette classification, il est nécessaire de distinguer les métaheuristiques, qui fonctionnent sans mémoire c'est-à-dire que l'action à réaliser est totalement déterminée par la situation courante et qu'il est aussi possible de revenir sur des solutions déjà examinées, de celles qui ont la capacité de mémoriser des informations à mesure que leur recherche avance, dans cette classe, il existe des méthodes qui utilisent une mémoire à court terme et d'autres qui utilisent une mémoire à long terme. Le représentant le plus courant des métaheuristiques "avec mémoire" est la recherche tabou et celui des "sans mémoire" est entre autres le recuit simulé.

2.3.5.3 Métaheuristique déterministe ou stochastique

Une métaheuristique est dite déterministe, si elle a la faculté de résoudre un problème d'optimisation en prenant des décisions déterministes c'est-à-dire que la même solution initiale va conduire à la même solution finale, comme c'est le cas pour la recherche locale ou la recherche tabou. En ce qui concerne les métaheuristiques stochastiques, il peut y avoir différentes solutions définitives qui sont obtenues à partir de la même solution initiale comme par exemple, les algorithmes génétiques ou le recuit simulé.

2.3.5.4 Métaheuristique itérative ou gloutonne

Le principe dans les algorithmes itératifs c'est qu'initialement, nous possédons une solution et qu'à chaque itération nous la transformons en utilisant différents mécanismes et stratégies de recherche, alors que le principe dans les algorithmes gloutons c'est que nous commençons avec une solution vide, et à chaque étape une variable de décision du problème est affectée jusqu'à ce que nous obtenions une solution complète. La plupart des métaheuristiques suivent le premier principe.

2.3.5.5 Nombre de structure de voisinages

Cette classification est réalisée en fonction du nombre de structures de voisinages utilisées. Nous pouvons trouver des algorithmes métaheuristiques qui travaillent sur une seule structure de voisinage c'est-à-dire qu'elles ne changent pas au cours de l'algorithme, par contre d'autres métaheuristiques, utilisent un ensemble de structures de voisinage, ce qui permet de diversifier la recherche entre ces différentes structures.

2.3.5.6 Métaheuristique dynamique ou statique

Certaines métaheuristiques dites dynamiques font usage d'une nouvelle fonction obtenue à partir de la fonction du problème primal en la modifiant en fonction des informations collectées au cours de l'exploration ce qui permet de changer la topologie de l'espace de solutions comme c'est le cas dans la recherche locale guidée, alors que d'autres, dites statiques laissent la fonction objectif telle qu'elle, et ce du début jusqu'à la fin du processus de calcul. Ceci étant dit, le but est de toujours éviter de tomber dans le minimum local, pour avoir plus de chances de trouver le minimum global.

2.3.5.7 Métaheuristique à une solution ou une population de solutions

La performance la plus remarquable des métaheuristiques est due à la façon dont elles imitent les meilleures fonctionnalités des phénomènes naturels. De nombreux chercheurs préfèrent classer les métaheuristiques selon leurs principes de fonctionnement durant le processus de recherche d'une solution, dans ce sens, deux principales catégories peuvent être distinguées, à savoir : les métaheuristiques qui manipulent une seule solution à la fois et les métaheuristiques qui travaillent avec une population de solutions. En ce qui concerne la première catégorie, les algorithmes manipulent une seule solution durant le processus de recherche par des déplacements successifs dans un voisinage constitué d'un ensemble de solutions afin de la transformer ou de l'améliorer, tel est le cas des algorithmes de recherche locale, de recherche tabou, de recuit simulé, etc. En outre, les métaheuristiques dans la seconde catégorie travaillent sur un ensemble de points de l'espace de recherche et ceci en se basant sur une population d'individus qui interagissent entre eux afin de produire de nouveaux individus plus performants, comme c'est le cas des algorithmes évolutionnaires.

Nous avons présenté plusieurs systèmes de classification des métaheuristiques, basés sur diverses propriétés. Cependant, les systèmes qui restent les plus significatifs sont ceux qui s'appuient sur le nombre de solutions à traiter (les techniques à base d'une seule solution et à base de population de solutions) [Fatos 08]. C'est sur cette base que nous allons effectuer la classification des métaheuristiques. Cette catégorisation permet une description plus précise des algorithmes. En outre, une hybridation peut être obtenue en combinant les caractéristiques de ces deux méthodes ce qui permet d'assurer une diversification dans l'ensemble de l'espace de recherche, d'exploiter le caractère de complémentarité qui existe entre elles et de tirer les avantages de chacune de ces approches.

2.4 Présentation des métaheuristiques les plus répandues

Les métaheuristiques continuent à attirer l'attention de différents chercheurs à travers le monde. Elles sont largement basées sur un ensemble de principes communs et la majorité d'entre elles s'inspire toujours de processus naturels ou de comportements sociaux d'animaux, ce qui conduit à obtenir une grande variété de métaheuristiques. Parallèlement, il existe d'autres métaheuristiques qui sont basées sur d'autres sources d'inspirations comme le processus d'improvisation des musiciens pour trouver la meilleure harmonie, etc. Dans cette section, présentons les principes des métaheuristiques les plus répandues. En effet, nous nous intéressons aux métaheuristiques basées sur une seule solution, qui permettent d'intensifier la recherche puisqu'elles sont orientées "exploitation", telles que le recuit simulé et la recherche tabou et les métaheuristiques basées sur une population de solutions, puisqu'elles sont orientées "exploration", telles que les colonies de fourmis, les essaims particulaires et les algorithmes génétiques.

2.4.1 Les métaheuristiques à base de solution unique

2.4.1.1 Le recuit simulé (simulated annealing)

Le recuit simulé est une technique de recherche qui s'inspire du phénomène physique du recuit utilisé par les métallurgistes dont les principes ont été proposés pour la première fois

par Kirkpatrick et al. [Kirkpatrick 83] pour obtenir un état solide bien ordonné de basse énergie. Cette métaheuristique est dite communément être la plus ancienne parmi toutes les métaheuristiques et fut aussi l'un des premiers algorithmes à avoir eu une stratégie explicite pour s'échapper des minima locaux. Le recuit simulé a eu un impact majeur sur le domaine de la recherche heuristique pour sa simplicité et son efficacité dans la résolution de problèmes d'optimisation combinatoire [Talbi 2009]. Bien que cette méthode ait été essentiellement développée pour la résolution des problèmes d'optimisation discrets, elle peut être utilisée pour traiter des problèmes d'optimisation continus.

Le processus de recuit nécessite de chauffer le matériau à une haute température puis le refroidir lentement pour atteindre un équilibre thermodynamique à chaque palier de température et obtenir ainsi une structure cristalline solide. La résistance de cette dernière dépend de la vitesse de refroidissement des métaux. Toutefois, une structure métastable et des imperfections sont obtenues, si la température initiale n'est pas suffisamment élevée ou un refroidissement rapide est appliqué. La méthode du recuit simulé exploite l'algorithme de *Métropolis* qui simule les variations d'énergie dans un système soumis à un procédé de refroidissement jusqu'à ce qu'elle converge vers un état d'équilibre et permet de décrire l'évolution d'un système physique vers son équilibre thermodynamique à une température *T*. L'analogie entre le système physique et le problème d'optimisation est illustrée dans le tableau suivant :

Tableau 2.2 : Analogie entre un problème d'optimisation et un système physique [Dréo 03] [Talbi 09] [Souier 2012]

Problème d'optimisation	**Système physique**
Solution	État du système
Fonction objectif	Énergie libre (*E*)
Paramètres du problème	Coordonnées des particules
Trouver une bonne configuration	Trouver les états à basse énergie
Optimum global	État stable ordonné
Optimum local	État métastable
Recherche locale	Trempe rapide
Le paramètre *T*	La température

Dans cet algorithme, Nous partons d'une configuration donnée et un paramètre *T* de contrôle de l'algorithme afin de minimiser la fonction objectif du problème qui est analogue à l'état d'énergie du système E. Nous faisons subir au système une perturbation d'une manière aléatoire pour générer une autre solution. Si cette modification élémentaire a pour effet d'améliorer la fonction objectif, donc la solution est acceptée immédiatement, sinon, elle est acceptée avec une probabilité égale à $exp(\Delta E/T)$ afin d'éviter de converger vers un minimum local. Une fois que l'équilibre thermodynamique est atteint, la température baisse doucement pour passer à l'itération suivante. Notons que ΔE représente la variation de la fonction objectif. L'organigramme du recuit simulé peut être représenté comme suit (figure 2.2) :

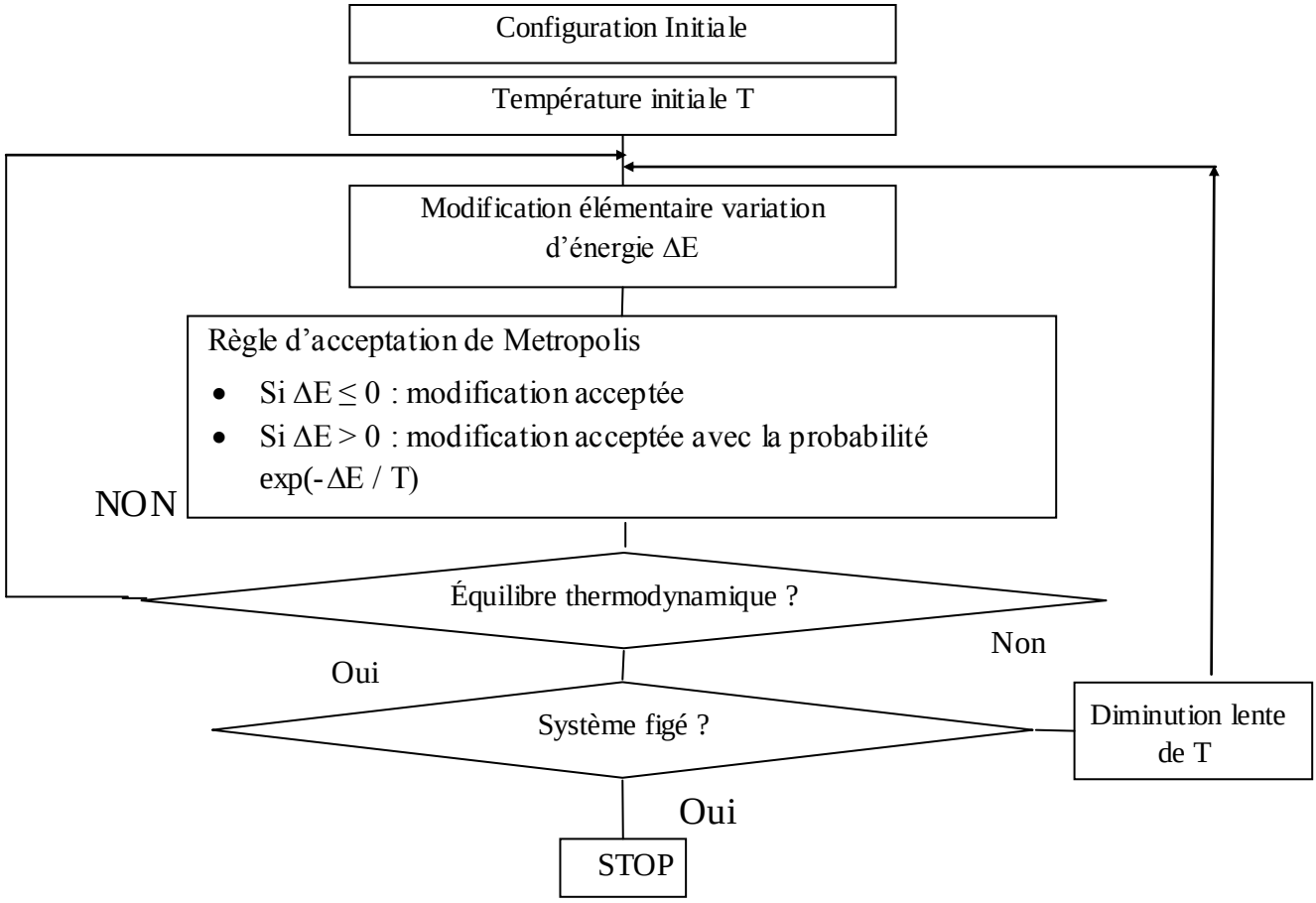

Figure 2.1 : Organigramme du recuit simulé

Le recuit simulé présente l'avantage d'être facile à mettre en œuvre et offre de bonnes solutions, cependant, l'inconvénient majeur de cette méthode réside dans la difficulté de réglage des paramètres ce qui peut induire l'algorithme à converger vers un minimum local. De plus, le temps de calcul peut devenir très significatif. L'algorithme du recuit simulé a été largement étudié et a montré son efficacité sur différents problèmes en particulier sur le problème du voyageur de commerce [Dorigo 97] [Geng 11]. En outre, la tendance s'amplifie ces dernières années pour les problèmes d'ordonnancement [Sevaux 04] [Elmi 11] [Laha 12] [Sezgin 13] [Rojas 13].

2.4.1.2 La recherche tabou (tabu search)

La méthode de recherche tabou inspirée de la mémoire humaine a été formalisée par Glover [Glover 86]. La recherche tabou est une procédure développée pour faire face à des problèmes d'optimisation discrets. Cependant, elle peut être appliquée à des problèmes continus. Cette méthode est devenue très classique en optimisation combinatoire et très populaire, grâce aux succès qu'elle a eu ces dernières années dans la résolution de plusieurs problèmes difficiles. Plusieurs investigations de cette métaheuristique ont été appliquées sur des problèmes d'ordonnancement dans des environnements de machines parallèles, job shop, etc., comme celle de Dell'Amico et Trubian [Dell'Amico 93], Li et al. [Li 11], Lara et al. [Lara 12], Bo_zejko et al. [Bo_zejko 13] et Lee et al. [Lee 13]. Contrairement au recuit simulé qui est totalement dépourvu de mémoire et qui ne génère qu'une seule solution voisine à

chaque itération, la méthode tabou a pour avantage d'examiner un échantillonnage de solutions et d'utiliser la notion de mémoire pour éviter de tomber sur des optima locaux. Le procédé de la méthode de recherche tabou consiste en une séquence de mouvements d'exploration sur l'espace de recherche, qui démarre à partir d'une solution possible afin d'atteindre l'optimum en remplaçant à chaque itération la solution courante par la meilleure solution trouvée dans son voisinage. Pour passer d'une solution à une autre, le processus crée un ensemble de solutions voisines de la solution courante en un seul mouvement élémentaire, cependant, l'algorithme explore le voisinage de la solution courante et sélectionne la meilleure solution possible en évaluant la fonction objectif du problème. Cette configuration est actualisée au cours des itérations successives jusqu'à ce qu'un critère d'arrêt soit satisfait. En partant de ce principe, la méthode peut revenir à une configuration déjà retenue. Pour palier à cette situation, la méthode enregistre les dernières solutions explorées dans une liste dite tabou. Cette liste est une mémoire à court terme qui sert à empêcher tout déplacement vers une solution de cette liste. Le concept original de cette liste n'est pas d'empêcher un mouvement précédent de se répéter, mais plutôt pour s'assurer qu'il n'a pas été inversée. En introduisant le concept de fonctionnalité de la solution, nous pouvons perdre des informations qui concernent la mémoire de recherche, soit en rejetant des solutions qui n'ont pas encore été générés, soit en interdisant des mouvements qui peuvent aboutir à une amélioration de la fonction objectif. Pour éviter ces inconvénients, nous intégrons dans l'algorithme des critères d'aspiration ce qui permet de lever l'interdiction sur certains mouvements qui ont été interdits par la liste tabou. Aussi, certains mécanismes sont souvent introduits dans la recherche tabou pour donner la priorité à l'ensemble des meilleures solutions et explorer ainsi les zones non visités de l'espace de solution, comme l'intensification et la diversification de la recherche. Le fonctionnement de cet algorithme est illustré par la figure suivante :

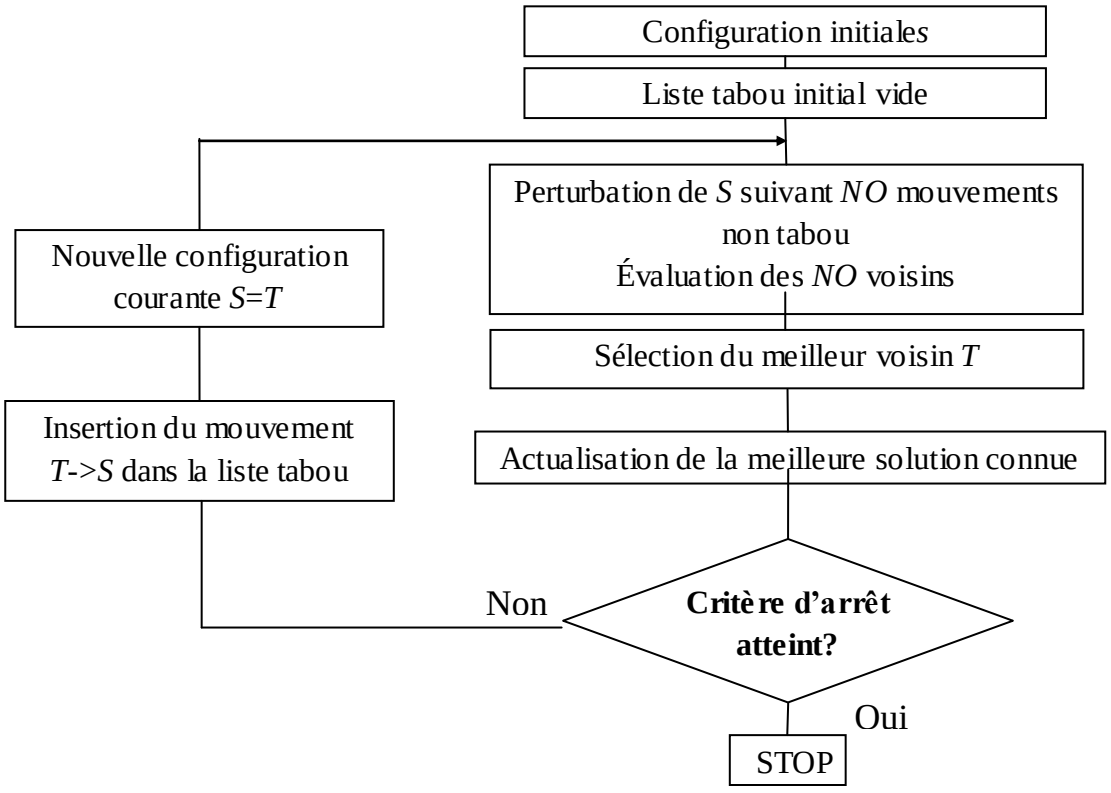

Figure 2.2 : L'organigramme de la recherche tabou [Dréo 03]

Algorithme 2.1 Modèle de l'**Algorithme de la recherche tabou**
Initialiser la liste tabou
Répéter
Pour chaque voisin **faire**
Mettre à jour le meilleur voisin
Mettre à jour la liste tabou
Fin pour
Jusqu'à satisfaction d'un critère d'arrêt
Sauvegarder la meilleure solution trouvée.

2.4.2 Les métaheuristiques à base de population de solutions

2.4.2.1 Les colonies de fourmis (Ant Colony Optimization)

Les algorithmes de colonies de fourmis appartiennent à la famille des métaheuristiques destinées à résoudre les problèmes d'optimisation difficile. Ils se basent sur une technique de recherche coopérative qui imite les comportements collectifs de fourmis réelles pour la résolution de ce genre de problèmes. Les algorithmes de colonies de fourmis ont été introduits par Dorigo et al. [Dorigo 92], ils ont été conçus pour le problème du voyageur de commerce et depuis cette technique a considérablement évolué et a pris un large éventail d'applications sur des domaines variés : [Dorigo 99], **[**Rajendran 04], [Berrichi 13], [Low 13] et [Neto 13].

Les fourmis sont capables de sélectionner collectivement le chemin le plus court entre une source de nourriture et leurs nids bien qu'elles ont individuellement des capacités cognitives très limitées. Elles commencent à chercher la zone entourant leurs nids de manière aléatoire, elles peuvent construire aussi le chemin le plus court à l'aide des mécanismes de communications indirectes via une substance volatile chimique appelée phéromone, qui est le principal facteur de la coordination des activités collectives chez les fourmis. Quand les fourmis rencontrent un obstacle, dans un premier temps, il y a une probabilité égale pour toutes les fourmis pour aller sur les chemins possibles, mais après un certain temps, les fourmis choisissent le chemin le plus court en raison de l'augmentation du pourcentage de la phéromone sur ce chemin, donc la concentration de phéromones peut être considérée comme indicateur de solutions de qualité à un problème d'intérêt qui leur permet de retrouver le chemin vers leurs nids lors du retour. D'autre part, les odeurs peuvent être utilisées par d'autres fourmis pour retrouver les sources de nourriture détectées par leurs consœurs [Jourdan 03]. Les fourmis utilisent leurs pistes de phéromones comme un moyen de communication de l'information entre elles. Quand une fourmi isolée tombe sur une source de nourriture au hasard, elle dépose une quantité de phéromone sur cet emplacement. Cependant, les fourmis qui se déplacent de façon aléatoire peuvent détecter cette piste de phéromone marquée. En outre, ces fourmis peuvent suivre cette trajectoire avec un haut degré de probabilité et améliorer simultanément la piste en déposant leurs phéromones sur leurs trajectoires. À cet effet, les fourmis utilisent le chemin le plus court car il sera le plus imprégné par la phéromone [Khalouli 10]. Parallèlement, les mouvements d'une fourmi sont contrôlés par des phéromones qui s'évaporent au fil du temps et par conséquent les chemins les moins utilisés auront tendance à disparaître avec l'évaporation de la phéromone. Ce procédé est basé sur le mécanisme de la rétroaction positive.

L'algorithme de colonie de fourmis consiste à travailler sur une population de solutions, chacune d'entre elle correspond à une fourmi artificielle. Le principe de l'algorithme est d'utiliser une structure de donnée commune, qui permet de renfermer des informations sur les quantités de phéromones artificielles accumulées dans l'espace de recherche afin de coordonner le fonctionnement de la population**.** Au début du processus d'optimisation, tous les bords sont initialisés avec la même quantité de phéromone. Cependant, les fourmis commencent à partir du nœud de début en choisissant au hasard un nœud dans chaque couche pour arriver au nœud de la nourriture. De petites quantités de phéromone sont déposées au cours de la phase de construction, tandis que les plus importantes sont déposées à la fin de chaque itération, en proportion de l'efficacité de la solution. Ce processus s'arrête si l'une des conditions d'arrêt est satisfaite. Les valeurs des variables de conception désignées par les nœuds sur le chemin d'accès avec la plus grande quantité de phéromone sont considérées comme les composantes du vecteur de la solution optimale. La procédure générale de l'algorithme de colonies de fourmis se compose de trois grandes étapes : l'initialisation de l'algorithme, la construction de la solution, et la mise à jour de la phéromone.

- ✓ Initialisation de l'algorithme des colonies de fourmis : cette étape comprend deux parties, la première consiste à initialiser la phéromone. Dans la deuxième partie, un certain nombre de fourmis sont placés arbitrairement sur les nœuds choisis au hasard. Ensuite, chacune des fourmis distribuées effectuera une tournée en construisant un chemin en fonction de la règle de transition de nœuds.
- ✓ Production d'une solution : la construction d'une solution se fait selon une règle de transition d'état probabiliste qui dépend principalement de l'état de la phéromone et la visibilité des fourmis. La visibilité est une capacité supplémentaire utilisée pour rendre cette méthode plus efficace. Les fourmis artificielles peuvent être considérées comme des procédures stochastiques qui construisent une solution d'une manière probabiliste.
- ✓ Mise à jour de la phéromone : l'intensité de la phéromone sur chaque bord est mise à jour par un mécanisme simulant son évaporation et cela après avoir construit l'ensemble des solutions, puis une phase de renforcement est mis à jour en fonction des solutions générées, des décisions sont ensuite prises par le processus probabiliste en fonction des quantités de phéromones artificielles obtenues et selon la vision locale du problème en utilisant une heuristique d'information appelée aussi visibilité [Khalouli, 2010].

Algorithme 2.2 Modèle de l'**Algorithme de colonie de fourmis**
Initialiser les traces de phéromone
Répéter
Pour chaque fourmi **faire**
Construction de solutions utilisant la phéromone
Mettre à jour les traces de phéromone
Évaporation
Renforcement
Fin pour
Jusqu'à satisfaction d'un critère d'arrêt
Sauvegarder la meilleure solution trouvée.

2.4.2.2 Les essaims particulaires (Particle Swarm Optimization)

Les essaims particulaires sont une famille des métaheuristiques qui s'appuie sur les observations des comportements sociaux des animaux. Elle s'inspire des interactions entre individus au sein d'une population pour atteindre un objectif donné dans un espace de recherche telle que les nuées d'oiseaux, les bancs de poissons et les essaims d'abeilles [Reynolds 87] et Heppner et Grenander [Heppner 90]. L'algorithme des essaims particulaires, inspiré par la simulation du comportement social, a été initialement proposé par Kennedy et Eberhart [Kennedy 95]. Cette métaheuristique utilise l'idée du partage social de l'information entre les membres d'un groupe qui peut avoir un avantage évolutif [Kennedy 01]. Le groupe possède une organisation globale et une collaboration entre ses individus ce qui peut conduire à la création de solutions à des problèmes complexes. La collaboration des individus permet de tirer les avantages de cette population et d'éviter le fait que chaque individu a une intelligence limitée. Selon Tfaili [Tfaili 08] ce type de stratégies collectives donne lieu à d'autres métaheuristiques comme les colonies de fourmis, cela est dû au travail collectif des entités qui interagissent entre elles, faisant révéler un comportement complexe global. La méthode d'optimisation par essaim particulaire est l'inspiration pour un nouveau domaine de recherche appelé intelligence en essaim, elle est développée pour les problèmes d'ordonnancement continus, mais plusieurs auteurs l'ont adapté aux cas discret [Lei 08], [Zhang 09b]. Cette métaheuristique reste une technique très générale, elle est appliquée à de nombreux problèmes du monde réel, nous citons les travaux présentés dans [Xia 2006], [Tseng 08], [Kashan 09], [Torabi 13] et [Chen 13].

Comme cité précédemment, cette métaheuristique simule le comportement d'un groupe, comme celui des oiseaux, chaque individu de cette population cherche la nourriture dans un champ d'une manière aléatoire sans qu'il n'ait une information sur l'endroit de la nourriture. La seule donnée disponible c'est la distance qu'il y a entre eux. Le but est de suivre l'oiseau le plus proche de la nourriture. Un algorithme d'essaims particulaires s'inspire de ce principe qui est facile à mettre en œuvre, il est initialisé avec une population (essaim) de solutions potentielles aléatoires (particules). Chaque particule possède une distance par rapport à la solution qui peut être évaluée afin de mettre à jour les meilleures positions connues. Cette particule se déplace itérativement selon une vitesse utilisée pour guider son mouvement et sa position dans l'espace de recherche, son déplacement est attiré par la position qui a donné la meilleure position réalisée par la particule elle même (best local) et celle trouvée par la particule et ses voisins (best global). Notons qu'à chaque itération de l'algorithme, la nouvelle vitesse et la nouvelle position de chaque particule sont calculées.

Le déplacement de chaque particule s'effectue selon une certaine vitesse qui dépend de plusieurs facteurs qui pondèrent les tendances de la particule. Le premier facteur est l'écart par rapport à leurs meilleures positions connues et à celle de leurs voisines, pondérées par des coefficients de confiance qui sont respectivement c_1 et c_2 (c_1 est un coefficient cognitif *et* c_2 est un coefficient social). Le second est le facteur de vitesse actuelle pondérée par un coefficient d'inertie w. En fait, dans les essaims particulaires, au lieu d'utiliser les opérateurs génétiques plus traditionnels, chaque particule ajuste son vol en fonction de sa propre expérience et celle de ses compagnons. En effet, ces dernières années plusieurs variantes de l'algorithme des essaims particulaires ont été proposées et ses paramètres ont été réglés. Ceci

à conduit à l'amélioration de ses performances en déterminant ses coefficients à l'aide d'une étude de convergence et en réglant ses principales variables aléatoires.

Algorithme 2.3 Modèle de l'**Algorithme des essaims particulaires**
Initialiser l'essaim
Répéter
Pour chaque particule **faire**
Mettre à jour le best local
Mettre à jour le best global
Mettre à jour la particule
Fin pour
Jusqu'à satisfaction d'un critère d'arrêt
Sauvegarder la meilleure solution trouvée

2.4.2.3 Les algorithmes génétiques (Genetics Algorithms)

Le paradigme des algorithmes évolutionnaires proposé dans les années 1960, consiste à s'inspirer des mécanismes de l'évolution naturelle [De Jong 00], [Back 00] et à utiliser le concept de populations d'individus pour faire face à des problèmes qui émergent du secteur industriel en particulier et du monde réel en général [Dupas 04]. Les algorithmes évolutionnaires s'inspirent du comportement d'évolution des espèces en exploitant les principes de variation et de sélection effectués dans les processus évolutifs naturels. L'évolution de l'organisme représente une suite successive d'améliorations afin que l'espèce s'adapte au mieux dans son environnement ou dans le milieu dans lequel elle évolue. Cette adaptation est fondée sur deux principaux mécanismes qui sont la sélection naturelle et la reproduction [Rebreyend 99]. Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnaires qui sont des techniques d'optimisation itératives et stochastiques visant à résoudre des problèmes d'optimisation NP-difficiles.

- **Concepts de base d'un algorithme génétique**

Les algorithmes génétiques ont été développés par Holland dans les années 1975 [Holland 75] puis approfondis par Goldberg en 1989 [Goldberg 89]. Ce sont des méthodes de recherche stochastiques conçues pour explorer des espaces de problèmes complexes afin de guider le processus de recherche et de trouver des solutions optimales en utilisant un minimum d'informations sur le problème. Contrairement à d'autres techniques d'optimisation, les algorithmes génétiques sont caractérisés par l'utilisation d'une population d'individus pour effectuer la recherche sur différentes régions de l'espace de recherche afin d'optimiser une fonction prédéfinie, appelée fonction objectif, ou fitness.

Une population est un ensemble de chromosomes ou individus. Chaque chromosome est une représentation d'une solution potentielle d'un problème d'optimisation, il est composé d'une chaîne de symboles appelée "gènes" qui peuvent prendre plusieurs valeurs, appelées "allèles". Dans un algorithme génétique, chaque solution est représentée par un chromosome artificiel, qui est défini par deux caractéristiques qui sont : le génotype et le phénotype. Le premier est une représentation des gènes de l'individu et le second est une représentation naturelle d'une solution pour évaluer sa qualité après le codage du génotype. Les solutions

sont manipulées en leur appliquant certains opérateurs stochastiques pour trouver la solution optimale ou une solution satisfaisante. La population évolue durant une succession de générations, jusqu'à ce qu'un critère d'arrêt soit satisfait.

- **Principe de fonctionnement de l'algorithme génétique**

L'inspiration est de favoriser la survie et la reproduction des individus les mieux adaptés à l'environnement. Cette adaptation est évaluée par une fonction fitness qui est directement liée à la valeur de la fonction objectif de cet individu. Le processus de fonctionnement de l'algorithme génétique commence par représenter la solution dans l'espace de recherche. Ensuite, nous générons aléatoirement une population initiale constituée d'individus pour lesquels nous calculons leurs fitness. Ces individus seront manipulés par une stratégie de sélection, suivi par une stratégie de reproduction qui consiste à concevoir un opérateur de mutation afin de garder une certaine diversité au sein de la population et un autre de croisement afin de générer de nouveaux individus qui seront mieux adaptés. Les individus issus du processus d'amélioration par les opérateurs génétiques vont passer par une étape de remplacement qui consiste à conserver que les meilleures solutions en se basant sur leurs fonctions fitness pour engendrer la nouvelle population de la génération suivante. Le processus permet de créer, de génération en génération, une population d'individus de mieux en mieux adaptés et il s'arrête lorsqu'un critère d'arrêt est satisfait.

- **Génération de la population initiale**

Une étape importante lors du fonctionnement du processus de l'algorithme génétique est la génération de la population initiale car elle affecte la qualité de la solution et le nombre de générations nécessaires pour l'obtenir [Sait 99]. Le niveau de diversité présente au sein de la population représente une caractéristique importante pour que l'algorithme ne reste pas bloqué dans un optimum local. Habituellement, la première population est générée soit d'une manière aléatoire, soit par des heuristiques ou des techniques spécifiques au problème. Finalement, la taille de la population qui agit sur la qualité des solutions obtenues et sur la performance de l'algorithme doit être générée d'une manière optimale pour assurer une bonne exploration de l'espace de recherche.

- **Sélection**

Le mécanisme de sélection est l'une des étapes principales dans le processus de recherche dans l'algorithme. Le principe de cette phase est de définir quels seront les individus de la population actuelle qui vont servir de parents et vont être dupliqués dans la nouvelle population. Les meilleurs individus ont plus de chances d'être parents. Une telle technique de sélection permet d'identifier les individus susceptibles d'être croisés et peut conduire la population à de meilleures solutions. Cependant, les mauvais individus en termes de fitness auront quand même une petite probabilité d'être sélectionnée. Cela peut conduire à un processus d'amélioration performant. Les parents sont sélectionnés en fonction de leurs fonctions fitness au moyen de l'une des stratégies suivantes [Dréo 03], nous citons les plus importantes :

- ✓ **Sélection par roulette (wheel) :** elle consiste à attribuer à chaque individu une probabilité de sélection qui est proportionnelle à sa performance, donc les individus

les plus adaptés au problème ont plus de chances d'être sélectionnés. Cette stratégie de sélection peut rencontrer des problèmes lorsque la valeur d'adaptation des chromosomes varie énormément et par conséquent, nous pouvons tomber sur des chromosomes qui auront très peu de chance d'être sélectionnés.

- ✓ **Sélection par rang :** c'est l'une des stratégies les plus courantes de sélection, elle consiste à ordonner les individus de la population dans un ordre croissant ou décroissant, selon leurs fonctions d'évaluation ce qui favorise les individus ayant un rang élevé et donc le hasard n'entre pas dans ce mode de sélection. Cette technique utilise les performances d'un rang d'individus plutôt qu'un seul individu (comme la sélection précédente). Donc, avec cette stratégie de sélection, tous les chromosomes ont une chance d'être sélectionnés.
- ✓ **Sélection aléatoire :** cette méthode de sélection suit une loi de probabilité uniforme et sans intervention de la fonction fitness. Donc tous les individus ont la même probabilité d'être sélectionnés. La sélection aléatoire reste une technique peu utilisée.
- ✓ **Sélection par tournoi :** elle consiste à sélectionner aléatoirement k individus et ensuite leurs appliquer un tournoi afin de choisir celui qui a la meilleure performance. Les individus impliqués dans le tournoi peuvent être retirés ou remis selon le choix des décideurs. Nous répétons ce processus n fois de manière à obtenir les n individus de la nouvelle population.

➢ **Croisement**

L'opérateur de croisement est un opérateur stochastique qui manipule la structure des chromosomes, il consiste à sélectionner deux individus parmi les parents potentiels à l'aide d'une des méthodes de sélection et échanger des parties de leurs chaînes selon une probabilité de croisement, pour donner de nouveaux descendants appelés enfants qui héritent certaines caractéristiques des deux parents afin d'enrichir la population. Cet opérateur ne s'applique pas systématiquement à chaque génération mais selon une probabilité appelée probabilité de croisement. Selon la littérature, nous pouvons distinguer plusieurs processus de croisement [Nearchou 04], nous présentons les plus utilisés :

- ✓ Croisement à un point : consiste à choisir aléatoirement deux parents et les diviser en deux parties à la même position, ensuite échanger les deux fragments situés à droite du point choisi. Le point de coupure s'effectue à n'importe quel niveau des gènes, donc tous les points possèdent la même probabilité d'être sélectionnés.
- ✓ Croisement à deux points : consiste à choisir deux points de coupure aléatoirement pour dissocier chaque parent en trois parties, ensuite il prend les deux parties en extrémités pour le premier parent afin de les remettre au premier enfant. Nous complétons la partie restante du premier enfant par les gènes du deuxième parent. Nous appliquons le même raisonnement pour le deuxième parent et le deuxième enfant.

➢ **Mutation**

Les individus de la population issue du croisement vont subir un processus de mutation qui agit sur un seul individu avec une probabilité très faible appelée probabilité de mutation

afin de créer un autre nouvel individu qui n'existait pas auparavant. L'opérateur de mutation permet d'éviter à l'algorithme génétique de converger vers des optimums locaux à travers la diversification de la population au cours des générations et de l'exploration de l'espace de recherche. Selon la littérature, plusieurs opérateurs de mutation sont proposés, les plus importants sont cités par Nearchou [Nearchou 04].

- ✓ Opérateur d'échange bit flip : appelé aussi opérateur à un point, il consiste à choisir au hasard un bit de l'individu et l'inverser indépendamment avec une certaine probabilité.
- ✓ Opérateur d'échange déterministe : consiste à définir un nombre fixé de bits de l'individu et de les inverser.
- ✓ Opérateur d'échange réciproque : permet de choisir deux gènes sur un chromosome et de les changer.
- ✓ Opérateur d'échange aléatoire : consiste à sélectionner deux gènes au hasard et inverser leurs positions.

La performance de l'algorithme génétique dépend de ses caractéristiques qui dépendent du problème. Le fonctionnement d'un algorithme génétique est résumé dans la figure suivante :

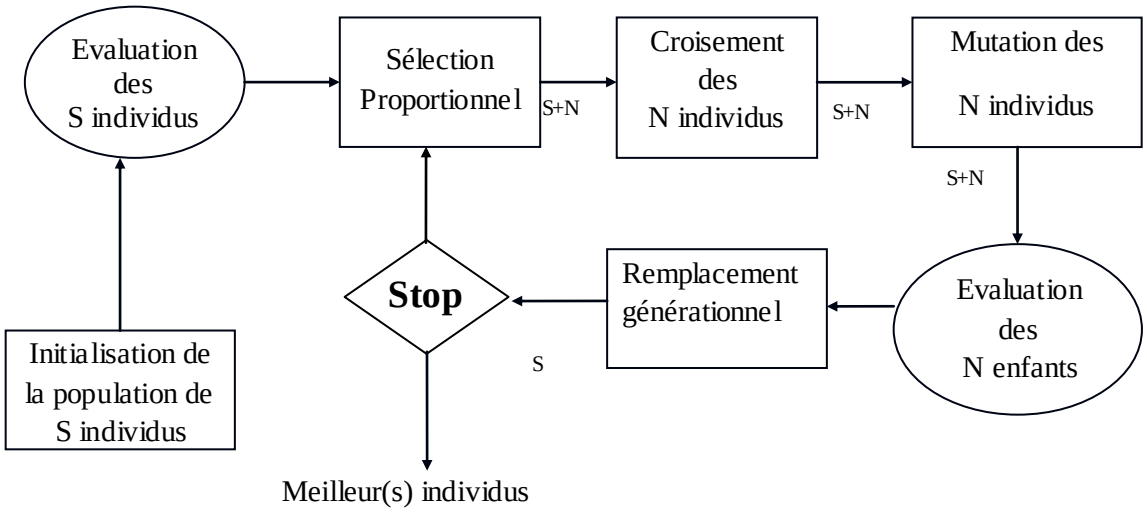

Figure 2.3 : Fonctionnement d'un algorithme génétique [Dréo 04]

L'algorithme génétique a connu beaucoup d'intérêt par la communauté d'optimisation à cause de sa performance dans la résolution des problèmes complexes, sa simplicité et sa facilité d'hybridation avec d'autres métaheuristiques et de son application avec succès sur une très large gamme de problèmes, telle que :

- ✓ Le problème de planification d'un système de production [Chen 02]
- ✓ La sécurité des systèmes de communication [Karras 04]
- ✓ L'optimisation de réseaux de chaînes logistiques [Ding 06]
- ✓ Le dimensionnement des stocks tampon d'un système de production [Dolgui 07]
- ✓ La gestion des stocks d'une usine de production [Pasandideh 08]
- ✓ La gestion des approvisionnements [Borisovsky 08].

En particulier, il a été implémenté avec succès sur plusieurs problèmes d'ordonnancement tels que : l'ordonnancement de projet [Torres 10], les problèmes à une seule machine [Rahmani 13], les problèmes à machines parallèles [Balin 11], le flow shop [Yalaoui 13], le job shop [Zhang 12], l'open shop [Low 12] et les systèmes flexibles de production [Souier 13]. Cela peut être expliqué par la structure simple de cet algorithme, sa facilité d'adaptation, sa rapidité, la possibilité d'acquérir plusieurs solutions en même temps, sa flexibilité pour des applications pratiques et finalement sa robustesse. Toutes ces raisons, nous motive à implémenter cette métaheuristique que nous allons détailler dans le chapitre suivant.

2.5 Conclusion

Dans ce chapitre, nous avons présenté dans un premier temps des généralités sur les problèmes d'optimisation et dans un second temps les principales métaheuristiques, leurs phénomènes d'inspiration, leurs principes de fonctionnement et leurs algorithmes de bases. Un intérêt particulier est apporté à la méthode de résolution principale de cette thèse à savoir les algorithmes génétiques. Ces derniers forment une classe de métaheuristiques très populaire pour proposer des solutions aux problèmes que les algorithmes d'optimisation classiques n'ont pas pu résoudre. Néanmoins, à travers le chapitre précédent, nous avons constaté qu'il y a peu d'étude qui s'intéresse à l'adaptation des métaheuristiques et plus particulièrement sur les algorithmes génétiques pour la résolution des problèmes d'ordonnancement sur machines parallèles avec la présence des ressources non renouvelables.

Cela nous motive à étudier ce problème dans le chapitre suivant qui sera consacré à l'adaptation de l'algorithme génétique pour la résolution d'un problème d'ordonnancement sur machines parallèles et ressources consommables. Une recherche locale sera proposée afin d'améliorer le fonctionnement de l'algorithme, plusieurs expérimentations seront ensuite réalisées sur le problème étudié et pour finir une étape de comparaison sera effectuée, en premier avec un modèle mathématique et en second avec plusieurs heuristiques, dans le but d'évaluer le comportement de la métaheuristique.

Chapitre 3

Algorithme génétique pour l'ordonnancement sur machines parallèles avec ressources consommables

3.1. Introduction

Actuellement, la dynamique des marchés a évolué et les environnements de production sont devenus complexes et caractérisés par des conditions en perpétuel changement, ceci pousse les industries à s'adapter pour rester compétitifs. Pour y arriver, ils sont contraints à améliorer leurs systèmes de production tout en exploitant les différentes ressources pour minimiser les coûts de production et les délais de livraison.

L'environnement de machines parallèles représente un axe important, il fournit de multiples routes alternatives pour produire un ensemble de pièces de manière efficace et avec une capacité de réagir aux pannes de machines. La majorité des études sur les machines parallèles considèrent, généralement la machine comme la seule ressource. Cependant, les tâches peuvent nécessiter des ressources non-renouvelables, telles que la matière première, les outils, etc. Par conséquent, une mauvaise gestion des ressources peut dégrader les performances du système. Nous pouvons remarquer que dans un système de production, plusieurs difficultés sont rencontrées durant l'ordonnancement, car un ensemble de paramètres et de contraintes doit être considéré. En outre, les problèmes d'ordonnancement avec ressources consommables sont eux aussi relativement complexes, cela explique le fait qu'il existe peu de méthodes analytiques qui sont développées. Cependant, les modèles mathématiques permettent d'exploiter les propriétés polyèdres de certains modèles d'optimisation, y compris les modèles d'ordonnancement. Ceci permet de conduire à l'élaboration d'une stratégie de solution efficace. De plus, il peut exister un autre type d'approche algorithmique qui permet aussi d'instaurer des techniques d'optimisation robustes.

Cela nous motive à développer dans ce travail, un programme linéaire en nombres entiers pour résoudre un problème d'ordonnancement d'atelier composé de plusieurs machines parallèles identiques avec des ressources consommables. Toutefois, la majorité des situations rencontrées dans les entreprises manufacturières sont complexes et les industriels doivent

faire face à des situations difficiles et sont dans la nécessité de résoudre des problèmes de grandes tailles. Vu que les méthodes exactes ne peuvent s'appliquer qu'à des problèmes de petite taille alors elles sont dans l'incapacité de résoudre ce type de problèmes de façon optimale et efficace. Pour y remédier, il serait nécessaire d'utiliser des méthodes approchées afin de pouvoir traiter des problèmes de taille importante et d'assurer un compromis entre le temps de résolution et la qualité de la solution. Les métaheuristiques représentent une classe importante de méthodes d'optimisation approchées qui sont une famille d'algorithmes de types stochastiques dédiée à la résolution de problèmes d'optimisation difficile.

Dans ce chapitre, la technique suggérée est une métaheuristique à base d'algorithme génétique dont le but est d'affecter dans un premier temps les tâches aux machines et dans un second temps de déterminer la séquence des tâches assignées à la même machine afin de minimiser le makespan tout en respectant les différentes contraintes de ressources. Une analyse de sensibilité est appliquée pour définir les meilleures valeurs des paramètres de cet algorithme et pour analyser leurs variations. Ensuite, une procédure de recherche locale est proposée dans le but de surmonter les limitations inhérentes aux composants individuels de l'algorithme génétique et d'améliorer son fonctionnement. Un autre objectif de ce chapitre est l'investigation sur les performances des heuristiques sur le système étudié. Ces heuristiques sont basées sur des règles qui sont en rapport soit avec le temps de traitement (*Longest Processing Time first, Shortest Processing Time first*), soit avec la consommation des composants (*Largest Resource Consumption first, Smallest Resource Consumption first*) ou bien les deux (*Longest processing-time-to-resources-consumption ratios first, Shortest processing-time-to-resources-consumption ratios first).*

Le modèle linéaire en nombres entiers et résolu avec le logiciel de programmation linéaire CPLEX. En outre, le reste des résultats obtenus dans cette thèse sont effectués par le langage Java sur un Core (TM) i3 CPU - 2,13 GHz - 4,00 Go de Ram. Ces résultats sont présentés après avoir effectué dix réplications pour chaque cas. Deux paramètres sont utilisés pour effectuer cette comparaison : le premier est le temps d'exécution et le second est l'écart entre la solution courante et la solution optimale. Plus de détails sur l'environnement des tests seront présentés dans la section des expérimentations.

Le reste de ce chapitre est structuré comme suit : dans un premier temps, nous présentons l'environnement du travail et ses caractéristiques. Dans un second temps, nous proposons un modèle linéaire en nombres entiers pour la modélisation du problème sur machines parallèles avec ressources consommables. Ensuite dans la troisième section, nous commencerons notre présentation de la mise en œuvre de l'algorithme génétique proposé, puis nous passons à l'application des différentes heuristiques. Parallèlement, une procédure de recherche locale est proposée pour améliorer le fonctionnement de la métaheuristique. Après, dans la quatrième section, nous établissons une analyse de sensibilité de l'algorithme génétique proposé afin de définir ses meilleurs paramètres. Ensuite, nous entamons la cinquième section qui est dédiée à la présentation des résultats par des expérimentations sur le modèle mathématique que nous enrichissons avec des comparaisons entre l'algorithme génétique, l'algorithme génétique hybride et les différentes heuristiques. Nous clôturons ce chapitre par une brève synthèse sur les résultats obtenus et une conclusion.

3.2 Description du problème

Dans ce travail, nous nous intéressons à ordonnancer un ensemble de n tâches sur m machines parallèles identiques avec la présence de ressources non-renouvelables. Le système étudié possède les caractéristiques suivantes :

- L'atelier contient plusieurs machines parallèles identiques.
- Chaque tâche *j* est caractérisée par un temps de traitement p_j et elle peut être exécutée sur une machine *i*, lorsque tous les composants nécessaires sont disponibles.
- Le processus de production pourra être lancé si tôt qu'une tâche peut être exécutée.
- Chaque tâche consomme *k* composants au début de son exécution.
- Les composants sont livrés par les fournisseurs à des instants différents.
- L'arrivée de chaque composant est représentée par une courbe sous forme d'escalier.
- La préemption n'est pas autorisée.
- Les machines peuvent traiter qu'une seule tâche à la fois et elles sont disponibles à l'instant initial $t = 0$.
- Chaque machine possède une file d'attente d'entrée avec une capacité illimitée et les tâches seront dirigées vers la file d'attente qui comporte le moins de tâches, voir la figure 3.1.
- L'objectif est de choisir dans un premier temps la machine la plus appropriée pour chaque opération et de déterminer dans un second temps la séquence des tâches assignées à la même machine afin de minimiser le makespan.

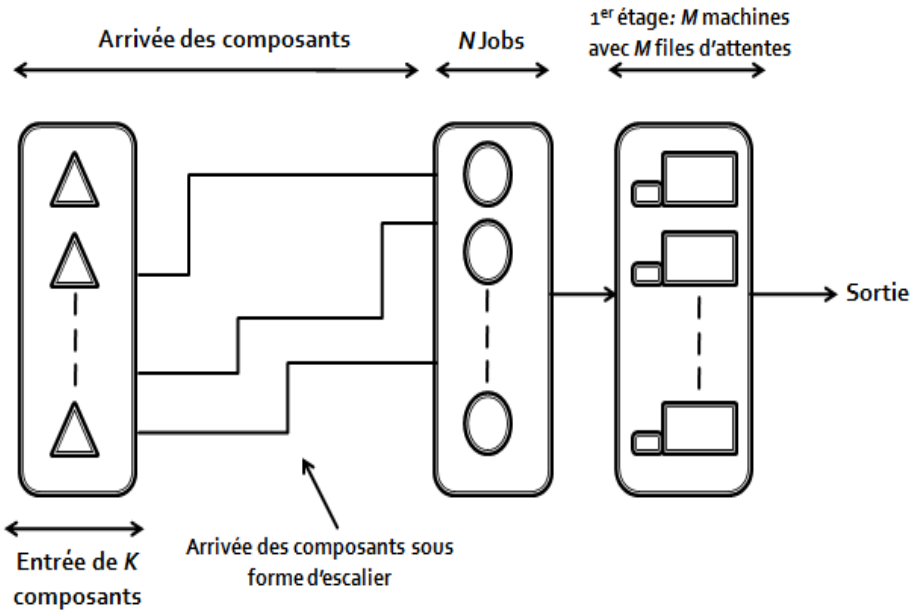

Figure 3.1 : Représentation du problème [Belkaid 12] [Belkaid 13a]

3.3 Modèle de programmation linéaire en nombres entiers

Dans cette section, nous développons un modèle mathématique linéaire en nombres entiers pour ordonnancer un ensemble de tâches sur plusieurs machines parallèles identiques

avec des ressources non-renouvelables. La mesure de performance choisie est la minimisation du makespan.

3.3.1 Paradigmes de modélisation d'ordonnancement de la production

Le modèle mathématique en nombres entiers est un moyen qui permet d'obtenir le meilleur résultat. Ce modèle peut être décrit selon sa fonction objectif ainsi que ses contraintes, comme il peut être classé en fonction des variables de décision dont il dépend. L'objectif du modèle étudié est d'identifier un paradigme d'optimisation qui pourra ensuite être, soit utilisé pour résoudre d'une manière exacte les grands problèmes d'ordonnancement de taille industriels, soit pour établir une série de règles de priorités et de bornes inférieures pour les différentes instances considérées. Il existe plusieurs façons pour modéliser un problème d'ordonnancement en utilisant différents types de variables de décision. Cependant, chaque modèle peut avoir ses propres avantages et inconvénients en fonction du type de problème à étudier à l'aide d'un paradigme donné. Selon Unlu et Mason [Unlu 10] et Blazewicz et al. [Blazewicz 91] les modèles de programmation les plus importants peuvent être résumés comme suit :

3.3.1.1 Variables de position des tâches

Cette formulation représente une mesure clé pour évaluer la qualité d'un ordonnancement de production proposée, elle indique :

- l'affectation des variables de décision aux tâches
- la position des tâches dans l'ordonnancement (indiquant si un job est à une position de l'ordonnancement ou non).
- l'instant de début de traitement des tâches

Cette catégorie de modélisation semble bien adaptée aux problèmes d'ordonnancement complexes, nous pouvons orienter le lecteur vers les papiers suivants : [Lasserre 92], [Dauzère 97], [Dauzère-Pérès 03], [Carrera 10].

3.3.1.2 Variables d'achèvement des tâches

Cette classe de modélisation représente un point important d'évaluation, indiquant le temps nécessaire pour exécuter une tâche, mais elle reste peu utilisée. Dans le cadre de résolution des problèmes d'ordonnancement, quelques chercheurs ont proposé ce type de modélisation [Balas 85], [Queyranne 93], [Queyranne 94].

3.3.1.3 Variables indexées par le temps

Cette classe de programmation est caractérisée par des variables de décision indexées par le temps, elle consiste généralement à affecter des tâches à des périodes de temps. Cette formulation a été introduite par Sousa et Wolsey [Sousa 92], depuis elle a pris un large éventail d'application [Blazewicz 91], [Van den Akker 99]. Dans cette section, nous avons opté pour la modélisation des variables de position pour la présentation du programme linéaire en nombres entiers pour éviter l'utilisation d'un grand nombre de variables et de contraintes indexées par le temps, donc le modèle s'exprime de la manière suivante :

3.3.2 Présentation du programme linéaire en nombres entiers

3.3.2.1 Variables utilisées

Les variables utilisées pour la représentation du modèle mathématique afin de minimiser le makespan dans un environnement de machines parallèles avec des ressources non-renouvelables sont présentées comme suit :

- n : le nombre de tâches
- m : le nombre de machines
- c : le nombre de composants
- T_1 : l'instant de la première arrivée des composants
- T_{last} : l'instant de la dernière arrivée des composants
- j : l'indice de la tâche, où $j=1,...,n$
- i : l'indice de la machine, où $i=1,...,m$
- k : l'indice de la ressource, où $i=1,...,c$
- t : l'indice d'arrivé des composants, où $t= T_1,..., T_{last}$
- n_i : le nombre de tâches affectées à la machine i
- p : la position de la tâche dans une machine, où $p=1,...n_i$
- p_j : le temps opératoire de la tâche j
- d_{i0} : la date de début de l'ordonnancement
- d_{ip} : la date de début de traitement du job sur la machine i en position p
- p_{ip} : la somme des temps de traitement des jobs jusqu'à atteindre la machine i en position p
- f_{ip} : la date de fin de traitement du job sur la machine i en position p
- c_{jk} : la quantité de composant k que le job j consomme
- c_{ipk} : le nombre de composant k que le job qui est sur la machine i en position p consomme
- A_{tk} : le nombre total de composant k arrivée jusqu'à l'instant t
- Z : est un nombre positif très grand
- C_{max} : le makespan
- X_{jip} et Y_{ipt} : les variables binaires

3.3.2.2 Fonction objectif et contraintes

Cette partie est réservée à la présentation de la fonction objectif à minimiser et les contraintes à satisfaire, qui peuvent être données de la façon suivante :

- **Fonction objectif :**

$$Min\ C_{max} \tag{1}$$

- **Contraintes :**

$$\sum_{j=1}^{n} X_{jip} = 1 \quad \forall \mathrm{i} = 1,2,\dots,\mathrm{m} \quad \forall \mathrm{p} = 1,2,\dots,n_i \tag{2}$$

$$\sum_{\mathrm{i}=1}^{\mathrm{m}} \sum_{\mathrm{p}=1}^{n_i} X_{jip} = 1 \quad \forall \mathrm{j} = 1,2,\dots,\mathrm{n} \tag{3}$$

$$p_{ip} = \sum_{j=1}^{n} X_{jip}\, p_j \quad \forall\, i = 1,2,\dots,m \quad \forall\, p = 1,2,\dots,n_i \tag{4}$$

$$d_{i0} = 0 \quad \forall\, i = 1,2,\dots,m \tag{5}$$

$$d_{i(p-1)} + p_{ip} \leq f_{ip} \quad \forall\, i = 1,2,\dots,m \quad \forall\, p = 1,2,\dots,n_i \tag{6}$$

$$d_{ip} = f_{ip} - p_{ip} \quad \forall\, i = 1,2,\dots,m \quad \forall\, p = 1,2,\dots,n_i \tag{7}$$

$$c_{ipk} = \sum_{j=1}^{n} X_{jip}\, c_{jk} \quad \forall\, i = 1,2,\dots,m \quad \forall\, p = 1,2,\dots,n_i \quad \forall\, k = 1,2,\dots,c \tag{8}$$

$$\sum_{i=1}^{v}\sum_{p=1}^{w} c_{ipk} \leq \sum_{t=T_1}^{T_{last}} A_{tk} * Y_{iwt} \quad \forall\, v = 1,2,\dots,m \quad \forall\, w = 1,2,\dots,n_i \quad \forall\, k = 1,2,\dots,c \tag{9}$$

$$Z * (Y_{ipt} - 1) \leq d_{ip} - t \quad \forall\, v = 1,2,\dots,m \quad \forall\, t = t_1,\dots,T_{last} \tag{10}$$

$$C_{max} = \max_i f_{ip} \quad \forall\, i = 1,2,\dots,m \quad \forall\, p = 1,2,\dots,n_i \tag{11}$$

$$X_{jip} = \begin{cases} 1 \text{ si le job j est ordonnancé à la position p sur la machine i} \\ 0\ sinon \end{cases} \tag{12}$$

$$Y_{ipt} = \begin{cases} 1\ si\ d_{ip} \geq t \\ 0\ sinon \end{cases} \tag{13}$$

- La fonction (1) est la fonction objectif. Elle représente la minimisation du temps d'exécution maximale (minimisation du makespan).
- La contrainte (2) garantit qu'il y a seulement une tâche sur la machine *i* et à la position *p*.
- La contrainte (3) assure que chaque tâche est affectée seulement une fois sur ces machines.
- La contrainte (4) calcule la durée d'opération de la tâche qui est en position *p* sur la machine *i*.
- La contrainte (5) assure que le début de l'ordonnancement est à l'instant 0.
- La contrainte (6) a pour objectif le calcule des temps de fin de traitement des jobs à chaque position p.
- La contrainte (7) calcule la date de début du job qui est sur la machine *i et* en position *p*.
- La contrainte (8) permet de calculer le nombre de composants consommés par la tâche qui est en position *p* sur la machine *i*.
- La contrainte (9) vérifie que la quantité consommée par un job en position p est inférieure ou égale au nombre qu'on dispose et cela pour chaque ressource. Et elle assure que la date de début de traitement du job (s'il n'y a pas suffisamment de composants pour qu'il soit traité) sur la machine i en position *p* est supérieure à l'arrivée du composant.
- La contrainte (10) met le lien entre la variable Y_{ipt} et l'instant de début de traitement du job en position p. Elle a pour objectif de conserver sa linéarité du modèle mathématique.
- La contrainte (11) décrit le makespan, il est égal au maximum des dates de fin de traitement des derniers jobs sur les machines.

- La contrainte (12) indique que la variable binaire X_{jip} est égale à 1, si la tâche *j* est en position *p* sur la machine *i* et 0 sinon.
- La contrainte (13) est une variable binaire, elle est égale à 1 si la date de début de traitement du job sur la machine i en position est supérieur à l'instant d'arrivé des composants et 0 sinon.

3.4 Résolution du problème

Cette section vise à décrire les approches utilisées pour la résolution du problème considéré. Nous commençons par présenter la métaheuristique proposée et nous terminons par décrire les heuristiques appliquées.

3.4.1 Algorithme génétique hybride

L'algorithme génétique hybride est une combinaison entre l'algorithme génétique et une méthode de recherche locale dont le but d'améliorer les résultats de la métaheuristique.

3.4.1.1 Algorithme génétique

L'algorithme génétique appliqué pour la résolution du problème décrit dans la section 3.2 (le lecteur intéressé peut se rapporter au chapitre précédent) et les différentes techniques adoptées sont présentés de la manière suivante :

- **Codage**

Le codage des individus consiste à construire un modèle mathématique du problème considéré afin de l'étudier. Ensuite, encoder les individus sous la forme de chromosomes. Chaque chromosome définit un système de configuration particulière qui reflète une succession d'affectations des tâches aux machines. Quand une machine devient disponible, alors la tâche suivante est choisie conformément avec le chromosome correspondant qui sera évalué et modifié pour minimiser le makespan. Pour mieux illustrer le codage utilisé prenons l'exemple de six tâches qui doivent être exécutées sur deux machines. Le chromosome est codé dans une table qui se compose de deux lignes, la première représente les tâches à exécuter et la deuxième illustre l'affectation des tâches aux différentes machines, ainsi, les tâches peuvent avoir l'affectation suivant (figure 3.2).

j_1	j_2	j_3	j_4	j_5	j_6
1	2	1	2	2	1

Figure 3.2 : Représentation du chromosome

En outre, les tâches affectées aux machines peuvent être clairement illustrées comme le montre la figure 3.3.

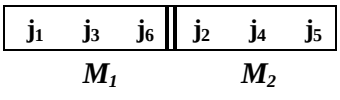

Figure 3.3 : Représentation des tâches affectées aux machines

- **Population**

Les individus de la population sont crées d'une manière aléatoire qui est la technique la plus classique pour réaliser cette étape. Les valeurs des gènes sont réparties de façon aléatoire, c'est-à-dire que : pour chaque chromosome ; la probabilité associée à affecter une tâche j sur une machine i est la même pour tous les gènes. Enfin la taille N de la population est un compromis entre la qualité de la solution et le temps de calcul.

- **Évaluation**

Chaque chromosome est évalué par sa fonction fitness qui illustre le temps d'exécution maximale. Notons que plusieurs réplications sont effectuées pour obtenir une solution comme nous le présentons dans la section des expérimentations.

- **Sélection des individus**

La phase de sélection est basée sur la fonction fitness des individus, deux parents sont sélectionnés pour générer des enfants. Cet opérateur désigne les individus qui participent à la reproduction. Dans cette étude, la technique de sélection appliquée est la technique élitiste. Cette phase est suivie par l'étape de création de nouveaux individus qui s'effectue principalement à l'aide des opérateurs de croisement et de mutation.

- **Croisement**

L'opérateur de croisement est un opérateur stochastique qui combine entre deux parents afin de produire des enfants. La technique choisie pour effectuer cette étape est un croisement de deux points qui est adapté à la structure de notre environnement d'ordonnancement sur machines parallèles avec ressources consommables. Pour illustrer le fonctionnement de croisement, nous l'appliquons sur l'exemple précédent (figure 3.4) :

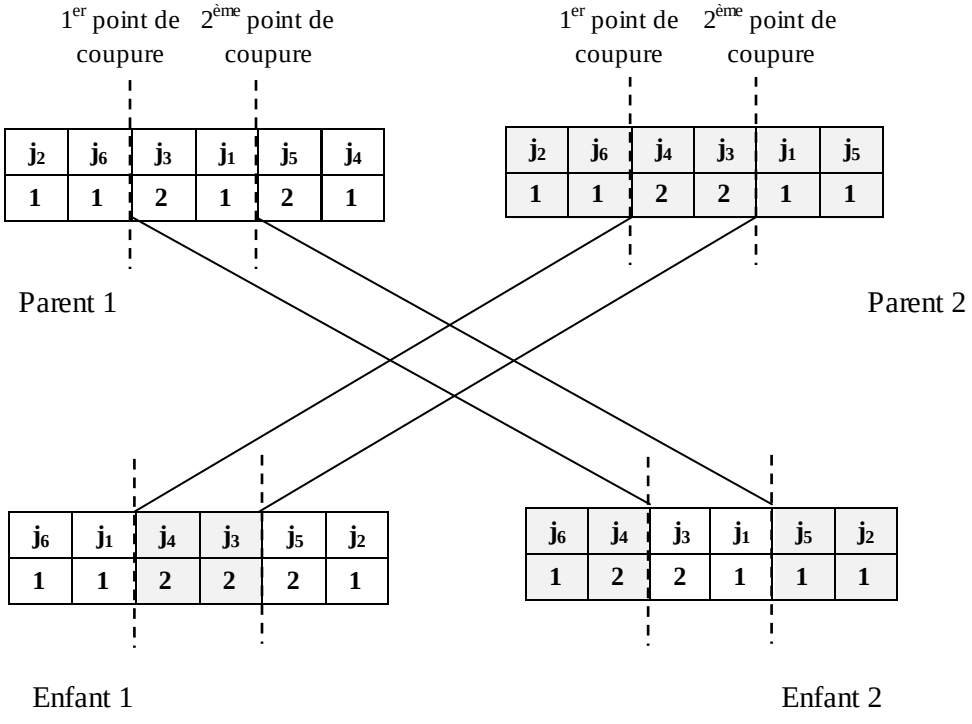

Figure 3.4 : Représentation de l'opérateur de croisement à deux points

- **Mutation**

L'opérateur de mutation est un opérateur stochastique qui modifie un individu pour créer une autre. La mutation permet l'exploration des différentes régions de l'espace de solution, ce qui conduit à maintenir la diversité de la population et par conséquent, d'éviter de tomber sur un minimum local. L'opérateur de mutation choisie est la mutation à un point qui est effectué en modifiant l'affectation des tâches aux machines. Notons que cette perturbation se produit aléatoirement. La figure 5 illustre un exemple de mutation dont lequel l'affectation de la tache trois a été changé de la machine une vers la machines deux.

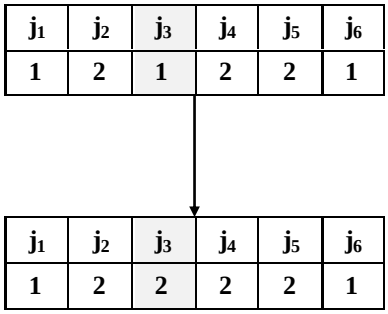

Figure 3.5 : Représentation de l'opérateur de mutation

- **Critère d'arrêt**

Un critère souvent utilisé est le nombre d'itérations dans laquel aucune amélioration n'est survenue consécutivement, si ce critère est atteint, nous sauvegardons la meilleure solution obtenue.

3.4.1.2 Recherche locale

Une technique de recherche doit exploiter et explorer l'espace de recherche. Or, une méthode de recherche comme les algorithmes génétiques peut ne pas assurer une intensification suffisante, à cause de ses opérateurs standards de croisement et de mutation [Hoos 04]. Toutefois, les algorithmes génétiques ont montré leurs performances et leurs facilités d'hybridation avec d'autres méthodes. L'hybridation consiste à combiner plusieurs algorithmes de nature différente pour améliorer les performances. C'est pour cette raison que les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale [Hernandez 08]. L'objectif principal de la recherche locale est d'accélérer la recherche pour accélérer la convergence de l'algorithme. Elle commence par une solution initiale donnée. À chaque itération, cette technique remplace la solution considérée par une solution voisine à travers une simple modification qui entraine de profonds changements dans le comportement de l'algorithme pour améliorer les résultats. Cette stratégie de voisinage est appliquée à tout nouvel individu obtenu au cours de la recherche.

Dans ce contexte, plusieurs travaux sont réalisés dans le but de montrer qu'il est nécessaire d'intégrer une procédure de recherche locale afin d'améliorer le fonctionnement de l'algorithme [Gonçalves 05], [Valls 08], [Sioud 12]. Les méthodes de recherche locale sont très variées et multiples, elles peuvent être des approches simples comme la méthode de descente où des approches développées comme la recherche tabou. Pour la suite de notre

travail, nous avons opté pour la méthode de descente car elle peut être facilement implémentée et s'articule autour d'un principe simple.

- **Méthode de descente**

Le principe de cette méthode consiste à chercher à partir d'une solution initiale donnée x, une solution voisine x_0 dans le voisinage de x. Si cette solution est meilleure que x, ($f(x_0) < f(x)$ dans le cas de la minimisation) alors la solution sera actualisée et la nouvelle solution sera sauvegarder, sinon la solution générée au début est conservée. La solution voisine est obtenue en modifiant l'affectation de certaines tâches aux machines (voir figure 1.5). Le processus est répété jusqu'à la satisfaction d'un critère d'arrêt.

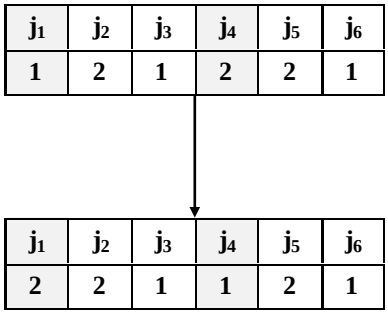

Figure 3.6 : Représentation de l'opérateur de mutation

Le pseudo-code de la méthode de descente est représenté par l'algorithme 3.1 :

Algorithme 3.1 : Modèle de l'algorithme de descente
Initialisation de la solution initiale
Répéter
Pour chaque individu **faire**
Rechercher une solution voisine
Mettre à jour la solution
Fin pour
Jusqu'à satisfaction d'un critère d'arrêt
Sauvegarder la meilleure solution trouvée.

Nous avons combiné entre deux méthodes qui sont complémentaires, l'algorithme génétique car il permet de détecter les régions prometteuses de l'espace de recherche et la recherche locale puisqu'elle se focalise de manière intensive à explorer ces zones de recherches [Moscato 99]. Le fonctionnement de l'algorithme génétique hybride peut être résumé comme suit (algorithme 3.2) :

Algorithme 3.2 : Modèle de l'algorithme génétique hybride

S'il y a un ensemble de tâches non ordonnancées **alors**
Générer une population initiale d'une manière aléatoire.
Pour chaque chromosome
Calculer la fonction fitness du chromosome qui est représenté par le makespan.
Évaluer la fonction fitness du chromosome.
Si cette solution est plus petite que la meilleure solution **donc**

Actualiser la meilleure solution.
Fin si
Fin pour
Tant que le critère d'arrêt n'est pas satisfait
Appliquer l'opérateur de sélection.
Appliquer l'opérateur de croisement.
Appliquer la recherche locale.
Appliquer l'opérateur de mutation.
Pour chaque chromosome
Calculer la fonction fitness du chromosome (le makespan).
Évaluer la fonction fitness du chromosome.
Si cette solution est plus petite que la meilleure solution **donc**
Actualiser la meilleure solution.
Fin si
Fin pour
Constituer la nouvelle génération.
Fin tant que
Fin si

L'algorithme génétique, ainsi que la procédure de recherche locale associée sont mis au point puis comparés aux résultats obtenus par différentes méthodes dans la section des expérimentations.

3.4.2 Heuristiques

Dans l'environnement industriel, plusieurs entreprises manufacturières affrontent les problèmes d'ordonnancement par l'utilisation d'heuristiques. Ces heuristiques sont des règles qui consistent à affecter les tâches disponibles dans la file d'attente à la machine libre en suivant un critère de priorité [Bhaskaran 91]. Les heuristiques sont très populaires en raison de leur facilité de mise en œuvre, car une application d'une méthode exacte est généralement impossible parce que les différentes contraintes et les perturbations imprévues rendent l'ordonnancement irréalisable [Aytug 05]. Bien que, les heuristiques ont été largement étudiées pour la résolution des problèmes d'ordonnancement, quelques chercheurs ont tenté de les utiliser pour les problèmes d'ordonnancement avec des ressources consommables. Par conséquent, dans cette section, nous faisons une brève description des heuristiques que nous allons adapter pour minimiser le temps d'exécution maximal du système étudié.

3.4.2.1 Longest Processing Time first (LPT)

Longest processing time first est une heuristique qui consiste à ordonnancer les tâches dans l'ordre décroissant de leurs temps de traitement. Cette heuristique est connue comme la règle de répartition la plus appropriée pour le problème d'ordonnancement de machines parallèles. Elle a été utilisée par de nombreux chercheurs qui la suggèrent comme une bonne heuristique en raison de ses bons résultats révélés dans la littérature. Par conséquent, LPT retient les petites tâches en termes de temps de traitement à la fin de l'ordonnancement afin d'équilibrer la charge du système.

3.4.2.2 Shortest Processing Time first (SPT)

Cette heuristique est l'une des règles les plus connues de la théorie d'ordonnancement. Elle consiste à classer les tâches dans l'ordre croissant de leurs temps de traitement. Ainsi, cette heuristique conserve les tâches les plus longues vers la fin de l'ordonnancement.

3.4.2.3 Largest Resource Consumption first (LRC)

Largest Resource Consumption first est une heuristique qui consiste à ordonner les tâches dans l'ordre décroissant de la somme des composants qu'elles consomment, c'est-à-dire à chaque fois qu'une machine est libre, le job qui consomme le plus de composants est choisi pour être traité. Ainsi, cette heuristique vise à conserver les tâches qui consomment le moins de ressources vers la fin de l'ordonnancement.

3.4.2.4 Smallest Resource Consumption first (SRC)

Smallest Resource Consumption first est une heuristique qui consiste à ordonner les tâches dans l'ordre croissant de la somme des composants qu'elles utilisent. Le principe est que lorsqu'une machine se libère, la tâche qui consomme le moins de ressources est choisie pour être traitée. Le but de cette heuristique est de faire placer les tâches sélectionnées au début de l'ordonnancement.

3.4.2.5 Longest processing-time-to-resources-consumption ratios first (L-PT/RC)

Cette heuristique est proposée par Carrera [Carrera 10] pour étudier un problème sur une seule machine avec ressource non-renouvelables. Elle consiste à ordonnancer les tâches dans l'ordre décroissant du rapport de leurs temps de traitement et consommation des ressources. Nous avons généralisé cette heuristique sur notre problème qui contient plusieurs machines parallèles identiques, les résultats obtenus sont présentés dans [Belkaid 13a].

3.4.2.6 Shortest processing-time-to-resources-consumption ratios first (S-PT/RC)

Cette technique est inspirée par la dernière heuristique présentée, elle consiste à ordonner les tâches dans l'ordre inverse de l'heuristique *L-PT/RC*.

3.5 Analyse de sensibilité de l'algorithme génétique proposé

Les algorithmes génétiques ont gagné l'intérêt de différentes communautés scientifiques ou industrielles à cause de leur capacité à être appliquées avec succès dans de nombreux problèmes en ayant un minimum d'informations. Cependant, leur simplicité d'adaptation pour la résolution des problèmes d'optimisation difficile est contrebalancée par le fait qu'ils n'offrent aucune preuve d'optimalité. Cela est dû au fait que les métaheuristiques partagent des difficultés à savoir leurs sensibilités aux paramétrages qui compliquent leurs adaptations en vue d'aboutir à une solution quasi-optimale. Par conséquent, la performance de chaque métaheuristique est influencée par le choix et le réglage de ses paramètres qui influe le comportement du système. Pour cela, nous réalisons une analyse de sensibilité qui consiste à faire varier les paramètres de l'algorithme génétique pour trouver son meilleur paramétrage, puis pour pouvoir analyser leurs impacts sur le système.

L'analyse de sensibilité est jugée par un indicateur de performance représenté par le makespan pour avoir une idée sur la qualité de la solution de cette métaheuristique. Les études en simulations comprennent trois axes d'analyses, le premier concerne l'effet de la taille de population, le deuxième consiste à expérimenter la variation du paramètre de croisement qui manipule la structure des chromosomes et le dernier concerne la mutation qui diversifie la recherche. En outre nous avons varié les valeurs du critère d'arrêt représenté par le nombre d'itérations sans amélioration. Les paramètres étudiés peuvent être résumée comme suit :

- ✓ Paramètre de mutation(P_{mut})
- ✓ Paramètre de croisement(P_{cross})
- ✓ Nombre d'itération sans amélioration ($N_{ité}$)
- ✓ Taille de population (T_{pop})

Rappelons que, lors de l'étape des tests, nous devons tester un seul paramètre à la fois, c'est-à-dire, nous fixons les autres paramètres et varions le paramètre que nous voulons évaluer. Parallèlement, le nombre de réplications pour chaque solution est égal à dix.

3.5.1 L'effet du paramètre de mutation

Dans cette étude, nous analysons l'effet du paramètre de mutation sur les performances de la métaheuristique. À cet effet, le comportement de l'algorithme génétique est décrit en termes des paramètres (P_{cross}, $N_{ité}$, T_{pop}) dont les valeurs ont été fixées respectivement à (0.9, 50, 100). De plus, nous affectons au paramètre de mutation P_{mut}, les variables de l'ensemble P_{mut} ={0.05, 0.1, 0.15, 0.2}.

Tableau 3.1 : Effets de la mutation sur le makespan

Paramètre de mutation	0.05	0.1	0.15	0.2
$V_{moy}C_{max}$	84,377	84,755	84,522	84,777

Nous remarquons dans le tableau 3.1 que la meilleure valeur du $V_{moy}C_{max}$ est obtenue lorsque la probabilité de mutation est égale à 0.05 et par conséquent, nous retenons cette valeur pour effectuer les expérimentations.

3.5.2 L'effet du paramètre de croisement

Pour analyser l'influence du paramètre de croisement sur l'algorithme étudié, nous attribuons aux paramètres suivants (P_{mut}, $N_{ité}$, T_{pop}) les valeurs suivantes (0.05, 50, 100). Parallèlement, nous testons le paramètre de croisement P_{cross} lorsqu'il prend respectivement les valeurs qui sont décrites dans l'ensemble P_{cross}={0.05, 0.1, 0.15, 0.2}.

Tableau 3.2 : Effets du croisement sur le makespan

Paramètre de croisement	0.7	0.8	0.9	1
$V_{moy}C_{max}$	84,811	84,577	84,277	84,611

Le tableau 3.2 présente les différentes probabilités concernant le makespan. Nous constatons, à partir des résultats obtenus que la meilleure valeur pour cette mesure de performance est de 84,27 ce qui correspond à une probabilité de 90 %.

3.5.3 L'effet de nombre d'itérations

Le critère d'arrêt représenté par le nombre d'itérations successives, dans lequel une amélioration n'a pas eu lieu, nous avons testé plusieurs valeurs qui se trouvent dans l'ensemble suivant $N_{ité}$={50, 100, 150, 200} afin de déterminer la meilleure d'entre elles. Notons que les paramètres (P_{cross}, P_{mut}, T_{pop}) ont pris les valeurs suivantes (0.9, 0.05, 100).

Tableau 3.3 : Effets du nombre d'itérations sur le makespan

Nombre d'itération	25	50	75	100
$V_{moy}C_{max}$	84,733	84,277	84,655	84,522

À travers les résultats du tableau 3.3, nous pouvons remarquer que la meilleure valeur a été obtenue pour le critère du makespan, après avoir effectué 50 itérations.

3.5.4 L'effet de taille de population

Dans cette section nous étudions la variation du makespan en fonction de la taille de population dont le but d'analyser la sensibilité de l'algorithme génétique pour l'ordonnancement sur machines parallèles avec des ressources consommables. Pour régler la taille de population, nous fixons les paramètres (P_{mut}, P_{cross}, $N_{ité}$) respectivement à (0.05, 0.9, 50) avec une variation de la taille de population à partir de l'ensemble suivant T_{pop} ={50, 100, 150, 200}

Tableau 3.4 : Effets de la taille de population sur le makespan

Taille de la population	50	100	150	200
$V_{moy}C_{max}$	85,788	84,277	83,666	83,588

Le tableau 3.4 montre que la taille de la population optimale doit être fixée à 200, mais nous ne retenons pas cette valeur en raison du fait que le temps de calcul est très important, pour cela nous choisissons la taille de population de 150 vu que l'erreur n'est pas significative entre les deux résultats (voir tableau 3.4)

À partir des différents protocoles de test que nous venons de présenter, nous pouvons constater que le réglage adéquat des paramètres de l'algorithme génétique est obtenu en retenant les valeurs qui sont résumées dans le tableau 3.5.

Tableau 3.5 : Paramètres de l'algorithme génétique

Paramètre	**Valeur**
Mutation	**0.05**
Croisement	**0.9**
Nombre d'itérations sans amélioration	**50**
Taille de population	**150**

3.6 Expérimentations et résultats

Cette partie est consacrée, dans un premier temps à la description des expérimentations et l'interprétation des résultats obtenus respectivement par :

- ✓ Le programme Linéaire en Nombres Entiers (PLNE)
- ✓ L'algorithme Génétique (AG)
- ✓ L'algorithme Génétique Hybride (AGH)
- ✓ Les heuristiques (décrites dans la section 3.4.2)

Et dans un second temps, à l'analyse des performances du système lorsque les règles de séquencement de tâches sont combinées avec l'algorithme génétique.

3.6.1 Environnement de tests

Les expériences ont été effectuées sur le langage Java sur un Core (TM) i3 CPU - 2,13 GHz - 4,00 Go de Ram. En outre, le modèle mathématique est résolu avec le solveur de programmation linéaire CPLEX.

Nous générons des instances afin que le total des ressources nécessaires soit inférieur ou égal au nombre de ressources disponibles dans le système. Les performances du système sont évaluées en termes de mesures telles que :

- ➢ GAP qui représente l'erreur relative entre la solution courante et la meilleure solution obtenue pour chaque instance.
- ➢ CPU_{time} est le temps d'exécution, c'est-à-dire le temps nécessaire pour qu'une méthode fournisse une solution à une instance donnée.

Le GAP reporté dans cette section peut être calculé par :

- ➢ $GAP = (C_{max}^{cur} - C_{max}^{best})/C_{max}^{best}$, où :
  - ✓ C_{max}^{best} est la meilleure solution
  - ✓ C_{max}^{cur} est la solution courante

Le temps de traitement de chaque tâche j est généré par une loi de distribution uniforme U [1, 50]. Chaque tâche j consomme entre 1 et k composants. Le nombre d'arrivées des composants est égal au nombre de machines.

3.6.2 Comparaison entre l'AG et le PLNE

Afin de prouver l'efficacité de l'algorithme génétique et d'évaluer les limites du modèle mathématique proposé pour le critère du makespan, nous réalisons plusieurs tests dont le but de définir les points critiques.

La durée d'exécution d'un modèle linéaire en nombres entiers étant généralement trop importante pour la déployer entièrement, nous avons limité le temps de simulation à 1500 secondes.

Notons que, nous effectuons des tests lorsque le nombre de composants *k* varie dans l'intervalle [1 - 4].

Tableau 3.6 : Evaluation des limites de PLNE lorsque k=1

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
2	4	1.21	0	0.13	0
	8	1.06	0	0.15	0
	12	1.14	0	0.19	0
	16	40.18	0	0.21	0
	18	>1500	/	0.23	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
3	6	1.38	0	0.14	0
	9	1.49	0	0.16	0
	12	147.3	0	0.19	0
	15	>1500	/	0.22	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
4	8	1.06	0	0.15	0
	12	1.14	0	0.18	0
	16	106.19	0	0.21	0
	20	>1500	/	0.23	0

Tableau 3.7 : Evaluation des limites de PLNE lorsque k=2

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
2	4	1.45	0	0.15	0
	8	33.09	0	0.16	0
	12	108.72	0	0.17	0
	16	156.17	0	0.18	0
	18	>1500	/	0.19	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
3	6	1.40	0	0.16	0
	9	10.48	0	0.17	0
	12	4.93	0	0.18	0
	15	94.83	0	0.19	0
	18	>1500	/	0.21	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
4	8	37.25	0	0.17	0
	12	38.01	0	0.19	0
	16	>1500	/	0.20	0

Tableau 3.8 : Evaluation des limites de PLNE lorsque k=3

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
2	4	1.66	0	0.13	0
	8	10.06	0	0.15	0
	12	165.45	0	0.18	0
	16	>1500	/	0.20	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
3	6	35.23	0	0.17	0
	9	61.77	0	0.17	0
	12	59.22	0	0.18	0
	15	49.17	0	0.19	0
	18	>1500	/	0.21	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
4	8	37.25	0	0.18	0
	12	73.01	0	0.20	0
	16	122.68	0	0.22	0
	20	>1500	/	0.21	0

Tableau 3.9 : Evaluation des limites de PLNE lorsque k=4

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
2	4	1.94	0	0.13	0
	8	11.76	0	0.15	0
	12	9.01	0	0.18	0
	16	>1500	/	0.21	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
3	6	45.70	0	0.16	0
	9	41.38	0	0.18	0
	12	56.59	0	0.19	0
	15	106.96	0	0.23	0
	18	>1500	/	0.24	0

Instances		PLNE		AG	
m	n	CPU_T	GAP	CPU_T	GAP
4	8	39.33	0	0.18	0
	12	73.64	0	0.19	0
	16	197.7	0	0.23	0
	20	>1500	/	0.25	0

Les tableaux suivants représentent les résultats obtenus par l'AG et le PLNE. La première colonne des tableaux contient la famille d'instances qui est divisée elle-même en deux autres, la première sous-colonne indique le nombre de machines et la deuxième, le nombre de jobs. Les deux dernières colonnes indiquent les résultats obtenus par respectivement le PLNE et l'AG. Nous remarquons que la meilleure solution est obtenue par une des deux méthodes AG ou PLNE cela implique que toutes les instances résolues, ont été résolues à l'optimalité.

Cependant, le modèle mathématique a nécessité des temps de calcul assez importants par rapport à l'AG. Où il n'a pas pu résoudre des instances supérieures à 16 tâches pour

différentes tailles de problème. Cela est dû à la contrainte des ressources non-renouvelable qui a augmenté la complexité du problème et par conséquent une croissance exponentielle au temps de calcul CPU_T. En effet, prendre en considération la contrainte de ressource consommable complique la résolution du modèle mathématique qui devient NP-difficile. Afin de prouver cette affirmation, nous relaxons cette contrainte dans le tableau 3.10, c'est-à-dire que nous supposons que les tâches ne consomment pas de ressources, ensuite nous analysons les résultats obtenus par le modèle.

À travers le tableau 3.10, nous pouvons remarquer que le PLNE a pu résoudre des instances jusqu'à 500 tâches lorsque le nombre de machines est égal à 2 dans un délai qui varie de l'ordre de 360s. En outre, il a pu résoudre jusqu'à 270 tâches lorsque le nombre de machines égal à 3 dans un délai de 147.9 s, et finalement, il a pu atteindre les 40 tâches dans 288.1s. Donc, cela prouve bien que le modèle mathématique rencontre des problèmes lorsqu'il s'agit de la résolution d'instances avec ressources consommables**.**

Tableau 3.10 : Evaluation des limites de PLNE lorsque k=0

Instances		PLNE
m	**n**	**CPU_T**
2	20	0.11
	60	0.47
	100	20.63
	200	49.95
	300	158.8
	400	230.2
	500	360.1
	600	>1500

Instances		PLNE
m	**n**	**CPU_T**
3	45	236.5
	120	151.5
	180	254.1
	240	106.4
	270	147.9
	300	>1500

Instances		PLNE
m	**n**	**CPU_T**
4	12	0.04
	16	0.09
	20	5.66
	40	288.1
	60	>1500

Pour conclure, nous pouvons dire pour le problème de minimisation du makespan sur machines parallèles avec ressources consommables, que le modèle linéaire en nombres entiers se révèle performant pour les problèmes de petites tailles mais il rencontre des difficultés pour résoudre les problèmes de moyennes et grandes tailles. Cela peut être dû au coût des bornes inférieures calculées par CPLEX qui sont très pénalisantes pour les performances du programme mathématique.

3.6.3 Comparaison entre l'AG et les heuristiques

Cette section présente une étude comparative entre l'AG et les heuristiques proposées afin d'en déduire la meilleure et permettre ainsi d'analyser son comportement sur le système. Les expériences effectuées sur le problème posé sont reparties en trois grandes familles de tests qui sont indiquées dans les tableaux (3.11-3.13) qui représentent respectivement les problèmes pour les petites, moyennes et grandes instances. La première colonne contient l'instance étudiée qui est composée elle-même de 3 autres colonnes représentant le nombre de tache (n), le nombre de machines (m) et le nombre de composants (k). Le reste des colonnes synthétise les résultats obtenus par les différentes heuristiques et métaheuristiques utilisées, chaque colonne est divisée en deux sous-colonnes. La première indique le temps d'exécution (CPU_T) et la deuxième représente la variation relative moyenne entre la solution courante et

la meilleure solution obtenue pour chaque instance (GAP). Pour chaque échelle, les résultats sont obtenus après avoir effectué 10 simulations.

- Pour les petites instances :
  - ✓ n= {4, 6, 8}
  - ✓ m= {2, 3}
  - ✓ k= {2, 3, 4}
- Pour les moyennes instances :
  - ✓ n= {10, 20, 30}
  - ✓ m= {6, 8}
  - ✓ k= {2, 3, 4}
- Pour les grandes instances :
  - ✓ n= {50, 75, 100}
  - ✓ m= {10, 15}
  - ✓ k= {2, 3, 4}

D'autre part, les contraintes suivantes doivent être respectées :

- ✓ Chaque machine doit exécuter qu'une seule tâche à la fois.
- ✓ Chaque tâche doit être traitée sans interruption.
- ✓ Tous les travaux doivent être traités pour achever le processus de fabrication.

Tableau 3.11 : Résultats expérimentaux pour les petites instances

Problèmes			AG		SPT		LPT		SRC		LRC		S-PT/RC		L-PT/RC	
n	M	k	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP
4	2	2	0.13	0.000	0.01	0.091	0.01	0.091	0.01	0.000	0.01	0.182	0.01	0.091	0.01	0.000
		3	0.12	0.000	0.01	0.167	0.01	0.000	0.01	0.333	0.01	0.000	0.01	0.167	0.01	0.000
		4	0.13	0.000	0.01	0.167	0.01	0.000	0.01	0.167	0.01	0.000	0.01	0.000	0.01	0.000
	3	2	0.13	0.000	0.01	0.167	0.01	0.000	0.01	0.167	0.01	0.000	0.01	0.167	0.01	0.000
		3	0.13	0.000	0.01	0.125	0.01	0.000	0.01	0.125	0.01	0.000	0.01	0.000	0.01	0.000
		4	0.13	0.000	0.01	0.100	0.01	0.000	0.01	0.100	0.01	0.000	0.01	0.100	0.01	0.000
6	2	2	0.14	0.000	0.01	0.059	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.000
		3	0.14	0.000	0.01	0.111	0.01	0.000	0.01	0.222	0.01	0.000	0.01	0.111	0.01	0.000
		4	0.14	0.000	0.01	0.167	0.01	0.083	0.01	0.083	0.01	0.000	0.01	0.250	0.01	0.083
	3	2	0.14	0.000	0.01	0.182	0.01	0.091	0.01	0.091	0.01	0.273	0.01	0.273	0.01	0.091
		3	0.14	0.000	0.01	0.300	0.01	0.200	0.01	0.000	0.01	0.300	0.01	0.200	0.01	0.100
		4	0.14	0.000	0.01	0.083	0.01	0.000	0.01	0.167	0.01	0.167	0.01	0.167	0.01	0.167
8	2	2	0.15	0.000	0.01	0.040	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.040	0.01	0.040
		3	0.15	0.000	0.01	0.043	0.01	0.000	0.01	0.087	0.01	0.000	0.01	0.130	0.01	0.000
		4	0.15	0.000	0.01	0.000	0.01	0.000	0.01	0.118	0.01	0.000	0.01	0.176	0.01	0.000
	3	2	0.16	0.000	0.01	0.167	0.01	0.083	0.01	0.417	0.01	0.417	0.01	0.417	0.01	0.083
		3	0,16	0.000	0.01	0.267	0.01	0.067	0.01	0.133	0.01	0.200	0.01	0.200	0.01	0.133
		4	0,17	0.000	0.01	0.385	0.01	0.154	0.01	0.308	0.01	0.154	0.01	0.231	0.01	0.231

Le makespan est un critère de performance important pour mesurer les capacités du système. Dans ce contexte, le tableau 3.11 représente les résultats obtenus pour les petites

instances. Nous remarquons que le CPU_T reste faible pour les différentes heuristiques ainsi que pour la métaheuristique et que le GAP pour l'AG est toujours égal à 0 ce qui montre que ce dernier donne les meilleurs résultats. De plus, l'heuristique LPT qui consiste à ordonner las tâches dans l'ordre décroissant par rapport à leurs temps de traitement et l'heuristique L-PT/RC qui a comme but d'ordonnancer les tâches dans l'ordre décroissant en fonction non seulement à leurs temps de traitement mais aussi à leurs consommations des ressources fournissent des résultats satisfaisants par rapport aux autres heuristiques et donnent à plusieurs reprises la même solution que la métaheuristique. Cela peut être expliqué par le fait que LPT donne l'avantage aux taches possédant des temps d'exécution important et L-PT/RC donne la priorité et favorisent les tâches ayant un temps de traitement important ainsi qu'une faible consommation de composant ce qui conduit ainsi à conserver ces tâches vers la fin de l'ordonnancement pour équilibrer la charge (du point de vue temps de traitement et consommation). Pour le SPT et S-PT/RC, ces dernières donnent de mauvais résultats lorsqu'il s'agit des problèmes de petites tailles.

Tableau 3.12 : Résultats expérimentaux pour les moyennes instances

Problèmes			AG		SPT		LPT		SRC		LRC		S-PT/RC		L-PT/RC	
n	M	k	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP
10	6	2	0.39	0.000	0.01	0.221	0.01	0.105	0.01	0.279	0.01	0.337	0.01	0.337	0.01	0.279
		3	0.20	0.000	0.01	0.200	0.01	0.000	0.01	0.250	0.01	0.150	0.01	0.200	0.01	0.200
		4	0.21	0.000	0.01	0.409	0.01	0.000	0.01	0.227	0.01	0.364	0.01	0.409	0.01	0.091
	8	2	0.20	0.000	0.01	0.261	0.01	0.000	0.01	0.130	0.01	0.174	0.01	0.174	0.01	0.000
		3	0.21	0.000	0.01	0.125	0.01	0.000	0.01	0.100	0.01	0.125	0.01	0.125	0.01	0.000
		4	0.21	0.000	0.01	0.115	0.01	0.115	0.01	0.115	0.01	0.115	0.01	0.115	0.01	0.000
20	6	2	0.37	0.000	0.01	0.382	0.01	0.005	0.01	0.256	0.01	0.281	0.01	0.432	0.01	0.000
		3	0.34	0.000	0.01	0.232	0.01	0.065	0.01	0.232	0.01	0.127	0.01	0.253	0.01	0.065
		4	0.34	0.000	0.01	0.316	0.01	0.005	0.01	0.077	0.01	0.077	0.01	0.340	0.01	0.029
	8	2	0.35	0.000	0.01	0.502	0.01	0.069	0.01	0.231	0.01	0.592	0.01	0.471	0.01	0.021
		3	0.34	0.000	0.01	0.292	0.01	0.000	0.01	0.188	0.01	0.146	0.01	0.229	0.01	0.000
		4	0.42	0.000	0.01	0.350	0.01	0.053	0.01	0.121	0.01	0.396	0.01	0.350	0.01	0.007
30	6	2	0.52	0.000	0.01	0.249	0.01	0.013	0.01	0.154	0.01	0.237	0.01	0.307	0.01	0.013
		3	0.53	0.000	0.01	0.202	0.01	0.048	0.01	0.090	0.01	0.111	0.01	0.230	0.01	0.013
		4	0.51	0.000	0.01	0.242	0.01	0.030	0.01	0.093	0.01	0.148	0.01	0.266	0.01	0.014
	8	2	0.56	0.000	0.01	0.275	0.01	0.016	0.01	0.090	0.01	0.267	0.01	0.311	0.01	0.016
		3	0.56	0.000	0.01	0.402	0.01	0.037	0.01	0.354	0.01	0.476	0.01	0.415	0.01	0.012
		4	0.64	0.000	0.01	0.436	0.01	0.074	0.01	0.398	0.01	0.286	0.01	0.498	0.01	0.024

Le tableau 3.12 expose les résultats obtenus pour les moyennes instances. Nous remarquons que les heuristiques fournissent des solutions dans un temps très réduit. En outre, le CPU_T de l'AG reste acceptable, car il ne dépasse pas les 0.64 secondes. Par ailleurs, dans la majorité des cas, les mauvais résultats ont été obtenus par SPT et S-PT/RC, car l'heuristique SPT consiste à favoriser les tâches qui ont un temps de traitement petit ce qui diminue l'efficacité de l'ordonnancement. D'autre part, l'heuristique S-PT/RC donne la priorité aux tâches qui ont un temps de traitement réduit avec une large consommation de composants ce

qui perturbe le comportement du système car, les tâches qui prennent plus de temps seront conservées vers la fin de l'ordonnancement.

L'heuristique SRC se révèle plus efficace que l'heuristique LRC, car elle fait passer en premier les tâches ayant une faible consommation de composants ce qui conduit à une amélioration des performances du système. De plus, l'heuristique LPT surpasse l'heuristique SPT en raison du fait qu'elle permet un équilibre de charge en conservant les tâches les plus petites en termes de temps de traitement. Notons que pour les problèmes de moyenne taille, la solution optimale est obtenue en utilisant l'AG ce qui démontre l'efficacité de cette métaheuristique.

Tableau 3.13 : Résultats expérimentaux pour les grandes instances

Problèmes			AG		SPT		LPT		SRC		LRC		S-PT/RC		L-PT/RC	
N	m	k	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP	CPU_T	GAP
50	10	2	1.10	0.000	0.01	0.269	0.01	0.000	0.01	0.085	0.01	0.256	0.01	0.295	0.01	0.006
		3	1.08	0.000	0.01	0.313	0.01	0.000	0.01	0.076	0.01	0.292	0.01	0.340	0.01	0.008
		4	1.11	0.000	0.01	0.200	0.01	0.000	0.01	0.154	0.01	0.232	0.01	0.200	0.01	0.018
	15	2	1.20	0.000	0.01	0.168	0.01	0.048	0.01	0.115	0.01	0.188	0.01	0.188	0.01	0.055
		3	1.29	0.000	0.01	0.173	0.01	0.052	0.01	0.160	0.01	0.186	0.01	0.233	0.01	0.046
		4	1.37	0.000	0.01	0.298	0.01	0.050	0.01	0.130	0.01	0.258	0.01	0.298	0.01	0.030
75	10	2	2.09	0.000	0.01	0.146	0.01	0.040	0.01	0.071	0.01	0.128	0.01	0.203	0.01	0.000
		3	1.95	0.000	0.01	0.135	0.01	0.030	0.01	0.087	0.01	0.170	0.01	0.178	0.01	0.000
		4	2.40	0.000	0.01	0.157	0.01	0.030	0.01	0.016	0.01	0.180	0.01	0.171	0.01	0.000
	15	2	2.13	0.000	0.01	0.279	0.01	0.020	0.01	0.104	0.01	0.230	0.01	0.377	0.01	0.006
		3	2.16	0.000	0.01	0.139	0.01	0.014	0.01	0.064	0.01	0.201	0.01	0.207	0.01	0.008
		4	2.16	0.000	0.01	0.205	0.01	0.029	0.01	0.101	0.01	0.216	0.01	0.283	0.01	0.000
100	10	2	2.91	0.000	0.01	0.169	0.01	0.000	0.01	0.069	0.01	0.103	0.01	0.175	0.01	0.000
		3	3.11	0.000	0.01	0.151	0.01	0.028	0.01	0.079	0.01	0.072	0.01	0.145	0.01	0.022
		4	3.02	0.000	0.01	0.144	0.01	0.022	0.01	0.072	0.01	0.097	0.01	0.119	0.01	0.003
	15	2	3.59	0.000	0.01	0.153	0.01	0.005	0.01	0.072	0.01	0.089	0.01	0.149	0.01	0.000
		3	3.22	0.000	0.01	0.173	0.01	0.012	0.01	0.076	0.01	0.088	0.01	0.156	0.01	0.003
		4	3.57	0.000	0.01	0.176	0.01	0.011	0.01	0.083	0.01	0.121	0.01	0.163	0.01	0.000

À travers les résultats obtenus dans tableau 3.13, nous pouvons remarquer que pour les problèmes de grande taille, l'heuristique SRC donne de meilleurs résultats que LRC, car la SRC consiste à ordonner les tâches dans l'ordre croissant de consommation des ressources et donc elle favorise les tâches qui consomment le moins de composants ce qui induit à une accélération de l'ordonnancement. Notons aussi que, les heuristiques L-PT/RC et LPT donnent des résultats proches et presque similaires à ceux obtenus par la métaheuristique et ceci pour la majorité des instances.

Un autre point intéressant est que l'heuristique L-PT/RC que nous avons proposée pour résoudre notre problème s'améliore lorsque le nombre de tâches devient supérieur à 75. De plus, elle se comporte mieux que S-PT/RC, car elle donne l'avantage aux tâches possédant un temps de traitement important et une consommation relativement faible des ressources. Pour ce qui est des heuristiques qui en un rapport avec le temps de traitement, la règle LPT

surpasse la règle SPT cela est dû à sa capacité d'équilibrer l'ordonnancement en favorisant les tâches les plus longues. Finalement, nous pouvons conclure que la métaheuristique proposée reste la plus efficace en la comparant avec ces heuristiques et elle permet de résoudre les problèmes de différentes tailles et ceci dans un temps acceptable.

3.6.4 Analyse de l'effet de règles de séquencement combinées avec l'algorithme génétique

L'objectif de cette section est d'analyser l'impact de l'interaction des décisions de séquencement des tâches avec l'algorithme génétique sur les performances du système étudié.

Dans ce cas, les décisions de séquencement sont appliquées sur les solutions obtenues par l'algorithme génétique afin de définir, la prochaine tâche à exécuter parmi toutes les tâches en attente du traitement dans la file d'attente et cela à chaque fois qu'une machine se libère. En d'autre terme, l'affectation est faite par l'algorithme génétique et le séquencement est fait par les heuristiques qui sont en rapport avec soit avec le temps de traitement (LPT, SPT), la consommation des composants (LRC, SRC) ou bien les deux (L-PT/RC, S-PT/RC).

Les résultats concernant l'effet des règles de séquencement pour les petites, moyennes et grandes instances sont illustrés dans les tableaux (3.14-3.16)

Tableau 3.14 : Effet des règles de séquencement pour les petites instances

Problèmes			GAP						
n	m	k	GA	GA+SPT	GA+LPT	GA+SRC	GA+LRC	GA+S-PT/RC	GA+L-PT/RC
4	2	2	0.000	0.255	0.236	0.000	0.255	0.255	0.000
		3	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	3	2	0.000	0.226	0.016	0.258	0.032	0.274	0.000
		3	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	2	2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		3	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	3	2	0.000	0.000	0.142	0.000	0.208	0.217	0.025
		3	0.000	0.400	0.400	0.000	0.400	0.400	0.100
		4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	2	2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		3	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		4	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	3	2	0.000	0.086	0.070	0.016	0.078	0.070	0.000
		3	0.000	0.023	0.006	0.006	0.023	0.023	0.000
		4	0.000	0.097	0.097	0.000	0.097	0.097	0.058

Nous pouvons remarquer, pour les problèmes de petites tailles (tableau 3.14), que lorsque nous avons ajouté les différentes heuristiques que ce soit en rapport avec le temps de traitement, la consommation des ressources ou celles qui sont en rapport avec les deux, cela a diminué considérablement les performances obtenues par l'AG particulièrement en ce qui concerne respectivement SPT, LPT, LRC et S-PT/RC.

Tableau 3.15 : Effet des règles de séquencement pour les moyennes instances

Problèmes			GAP						
n	m	k	GA	GA+SPT	GA+LPT	GA+SRC	GA+LRC	GA+S-PT/RC	GA+L-PT/RC
10	6	2	0.000	0.000	0.113	0.029	0.167	0.157	0.069
		3	0.000	0.225	0.023	0.171	0.068	0.194	0.108
		4	0.000	0.430	0.000	0.240	0.434	0.430	0.154
	8	2	0.000	0.250	0.039	0.121	0.207	0.198	0.000
		3	0.000	0.148	0.018	0.133	0.165	0.145	0.000
		4	0.000	0.260	0.000	0.135	0.240	0.329	0.000
20	6	2	0.000	0.233	0.077	0.149	0.193	0.229	0.029
		3	0.000	0.210	0.217	0.202	0.227	0.198	0.181
		4	0.000	0.156	0.088	0.102	0.145	0.131	0.104
	8	2	0.000	0.521	0.117	0.298	0.601	0.641	0.092
		3	0.032	0.166	0.004	0.148	0.176	0.189	0.000
		4	0.000	0.337	0.201	0.231	0.343	0.363	0.140
30	6	2	0.022	0.087	0.000	0.003	0.089	0.119	0.001
		3	0.000	0.125	0.109	0.045	0.121	0.124	0.087
		4	0.000	0.124	0.123	0.058	0.118	0.134	0.097
	8	2	0.000	0.243	0.145	0.089	0.259	0.245	0.116
		3	0.000	0.187	0.147	0.043	0.190	0.209	0.083
		4	0.000	0.283	0.233	0.219	0.281	0.301	0.166

En ce qui concerne les problèmes de moyenne taille, nous pouvons constater, à partir du tableau 3.15 que nous avons obtenus la solution optimale lorsque nous avons combiné avec l'AG les deux heuristiques : LPT et LPT-RC, mais seulement pour les deux instances qui sont respectivement (20-8-3) et (30-6-2), quant aux résultats obtenus par les autres heuristiques, ces dernières n'ont fait que décroitre les performances obtenues par l'AG seul.

Tableau 3.16 : Effet des règles de séquencement pour les grandes instances

Problèmes			GAP						
n	m	k	GA	GA+SPT	GA+LPT	GA+SRC	GA+LRC	GA+S-PT/RC	GA+L-PT/RC
50	10	2	0.000	0.204	0.173	0.165	0.222	0.199	0.135
		3	0.000	0.258	0.129	0.171	0.257	0.273	0.106
		4	0.000	0.108	0.051	0.046	0.085	0.102	0.042
	15	2	0.000	0.279	0.291	0.234	0.322	0.318	0.223
		3	0.000	0.299	0.270	0.265	0.283	0.332	0.252
		4	0.000	0.403	0.252	0.261	0.379	0.383	0.242
75	10	2	0.000	0.115	0.090	0.079	0.133	0.118	0.067
		3	0.000	0.154	0.136	0.110	0.172	0.186	0.091
		4	0.000	0.139	0.130	0.067	0.135	0.142	0.117
	15	2	0.000	0.479	0.150	0.178	0.423	0.502	0.108
		3	0.000	0.200	0.121	0.127	0.267	0.283	0.143
		4	0.000	0.376	0.226	0.209	0.376	0.367	0.214
100	10	2	0.000	0.157	0.130	0.089	0.158	0.161	0.113
		3	0.000	0.110	0.105	0.101	0.111	0.119	0.092
		4	0.000	0.116	0.111	0.097	0.132	0.128	0.078
	15	2	0.000	0.241	0.163	0.156	0.205	0.267	0.143
		3	0.000	0.264	0.186	0.201	0.255	0.282	0.173
		4	0.000	0.252	0.189	0.150	0.242	0.294	0.156

Pour ce qui est des problèmes de grande taille, nous pouvons remarquer qu'aucune heuristique qui a été ajoutée à l'AG n'a fait améliorer les résultats de ce dernier, au contraire cela les a diminué considérablement et éloigné de la solution optimale trouvée lorsque nous avons employé l'AG seul, de plus, cela concerne toutes les instances.

À travers tous les faits qui viennent d'être énoncés, nous pouvons constater que, dans la majorité des cas, lorsque nous avons combiné les différentes heuristiques avec l'AG, les résultats obtenus nous éloignent de la solution optimale trouvée par l'AG seul et ceci que ce soit pour les problèmes de petite, moyenne et grande taille. Par conséquent, il serait judicieux, pour obtenir de bonnes performances d'utiliser l'AG seul ou de le combiner avec d'autres techniques comme la recherche locale.

3.6.5 Comparaison entre l'AG et l'AGH

Cette section a pour objectif d'analyser l'impact de la recherche locale sur le système à travers une étude comparative entre l'AG et l'AGH. Pour cela, nous réalisons plusieurs simulations afin de pouvoir équilibrer la charge entre les machines et offrir un taux élevé d'utilisation des machines.

Problèmes			AG		AGH	
n	M	k	CPU_T	GAP	CPU_T	GAP
4	2	2	0.13	0.000	0.13	0.000
		3	0.12	0.000	0.12	0.000
		4	0.13	0.000	0.13	0.000
	3	2	0.13	0.000	0.13	0.000
		3	0.13	0.000	0.13	0.000
		4	0.13	0.000	0.13	0.000
6	2	2	0.14	0.000	0.14	0.000
		3	0.14	0.000	0.14	0.000
		4	0.14	0.000	0.14	0.000
	3	2	0.14	0.000	0.15	0.000
		3	0.14	0.000	0.14	0.000
		4	0.14	0.000	0.15	0.000
8	2	2	0.15	0.000	0.15	0.000
		3	0.15	0.000	0.15	0.000
		4	0.15	0.000	0.16	0.000
	3	2	0.16	0.000	0.16	0.000
		3	0.16	0.000	0.16	0.000
		4	0.17	0.000	0.17	0.000

Tableau 3.14 : Résultats de l'AGH pou les petites instances

Problèmes			AG		AGH	
n	m	k	CPU_T	GAP	CPU_T	GAP
10	6	2	0.39	0.012	0.39	0.000
		3	0.20	0.000	0.20	0.000
		4	0.21	0.000	0.21	0.000
	8	2	0.20	0.000	0.20	0.000
		3	0.21	0.000	0.21	0.000
		4	0.21	0.000	0.20	0.000
20	6	2	0.37	0.003	0.37	0.000
		3	0.34	0.002	0.34	0.000
		4	0.34	0.002	0.34	0.000
	8	2	0.35	0.003	0.36	0.000
		3	0.34	0.000	0.35	0.000
		4	0.42	0.009	0.42	0.000
30	6	2	0.52	0.002	0.53	0.000
		3	0.53	0.004	0.53	0.000
		4	0.51	0.002	0.52	0.000
	8	2	0.56	0.003	0.56	0.000
		3	0.56	0.005	0.57	0.000
		4	0.64	0.009	0.65	0.000

Tableau 3.15 : Résultats de l'AGH pour les moyennes instances

Problèmes			AG		AGH	
n	m	k	CPU_T	GAP	CPU_T	GAP
50	10	2	1.10	0.005	1.10	0.000
		3	1.08	0.008	1.08	0.000
		4	1.11	0.000	1.12	0.000
	15	2	1.20	0.000	1.20	0.000
		3	1.29	0.005	1.30	0.000
		4	1.37	0.006	1.38	0.000
75	10	2	2.09	0.007	2.1	0.000
		3	1.95	0.003	1.96	0.000
		4	2.40	0.003	2.05	0.000
	15	2	2.13	0.000	2.14	0.000
		3	2.16	0.000	2.17	0.000
		4	2.16	0.000	2.17	0.000
100	10	2	2.91	0.000	2.95	0.000
		3	3.11	0.000	3.12	0.000
		4	3.02	0.000	3.04	0.000
	15	2	3.59	0.003	3.60	0.000
		3	3.22	0.006	3.24	0.000
		4	3.57	0.002	3.59	0.000

Tableau 3.16 : Résultats de l'AGH pour les grandes instances

Les tableaux ci-dessus présentent les performances de l'AG et l'AGH pour les problèmes de petite, moyenne et grande taille. Nous remarquons que les performances des deux métaheuristiques sont similaires pour les problèmes de petites instances. En outre, le GAP de l'AG commence à augmenter pour les moyennes et grandes instances, ceci peut être

expliqué par la complexité croissante du problème lorsque le nombre de tâches, machines et composants augmentent.

Concernant les résultats obtenus de l'AGH, ces derniers se sont améliorés de manière significative et ont surpassé même ceux de l'AG, cela est dû à la phase de recherche local qui a conduit à une amélioration significative en atteignant un pourcentage de performance de 1.2%. De plus, le temps de calcul, n'a pas dépasse les 0.01 sec, en ajoutant cette procédure et ceci pour les problèmes de petites, moyennes et grandes tailles, ce qui représente un temps moins important que celui de l'algorithme génétique pour fournir une solution.

3.7 Synthèse

L'étude menée dans ce chapitre nous a permis de dégager quelques résultats préliminaires qui méritent une plus grande attention en termes de ses implications sur le processus décisionnel des activités pour la gestion optimale de l'ordonnancement. Les points les plus importants peuvent être résumés comme suit :

- Le modèle linéaire en nombres entiers proposé se révèle efficace pour la minimisation du makespan mais il a des difficultés pour résoudre des instances de taille supérieure à 16 tâches nécessitant un temps de calcul assez important.
- En ce qui concerne les heuristiques qui en un rapport avec le temps de traitement, LPT donne de meilleurs résultats que le SPT car elle favorise les tâches qui ont un long temps de traitement.
- Concernant les heuristiques qui ont un rapport avec la consommation de composants, la règle SRC se révèle plus performante que la règle LRC, car elle favorise les tâches qui ont une faible consommation de composants ce qui conduit à une amélioration des performances du système.
- L'heuristique proposée L-PT/RC se comporte mieux que S-PT/RC et les autres heuristiques et son fonctionnement s'améliore lorsque la taille du problème augmente, car elle permet d'offrir l'avantage aux tâches possédant un temps de traitement important et une consommation relativement faible des ressources.
- Les règles de séquencement ont un impact négatif sur les performances du système lorsqu'elles sont combinées avec l'algorithme génétique.
- L'algorithme génétique proposé permet d'avoir la prise de décision concernant l'ordonnancement des tâches dans un environnement de machines parallèles avec ressources consommables.
- Les meilleurs résultats sont obtenus par l'algorithme génétique, par conséquent, ce dernier reste le plus efficace parmi toutes les heuristiques utilisées.
- L'algorithme génétique hybride surpasse l'algorithme génétique en termes de qualité de solution, cela est dû à l'effet positif de la procédure de recherche locale sur les performances de ce dernier.

3.8 Conclusion

Dans ce chapitre, nous nous somme intéressés à l'adaptation d'un algorithme génétique pour l'ordonnancement sur machines parallèles avec ressources consommables. Nous avons tout d'abord commencé par décrire le modèle linéaire en nombres entiers que nous avons développé pour minimiser le makespan. En suite, nous avons présenté la structure de l'algorithme génétique proposée et la procédure de recherche locale appliquée pour améliorer le fonctionnement de ce dernier. Nous avons enrichi ce travail par une description des heuristiques que nous avons adaptées et généralisées. Aussi, nous avons présenté une analyse de sensibilité sur la métaheuristique proposée pour définir les meilleures valeurs de ses paramètres et pour voir leurs influences sur les performances de l'algorithme génétique. De plus, nous avons effectué une étude basée sur la simulation pour analyser les performances du système, lorsque les différentes techniques sont appliquées.

D'une part, cette étude nous a permis de montrer que la métaheuristique proposée est capable de résoudre des problèmes de grandes tailles dans un laps de temps raisonnable toute en maintenant la qualité de la solution à un niveau acceptable. L'efficacité de l'algorithme génétique est démontrée en le comparant avec un modèle linéaire en nombres entiers pour les petites instances et avec des heuristiques pour les grandes instances. De plus, cette recherche nous a permis de constater que cet algorithme est plus performant lorsqu'il est combiné avec la procédure de recherche locale pour minimiser le makespan sur machines parallèles avec des ressources non-renouvelables.

D'autre part, l'analyse de la revue de la littérature effectuée dans le premier chapitre, nous a montré un manque d'études concernant l'implémentation des métaheuristiques pour la résolution des problèmes d'ordonnancement avec processus réentrant dans un environnement manufacturier caractérisé par des contraintes de ressources. Donc, il apparaît qu'il soit nécessaire de continuer à développer et étendre la recherche menée dans ce chapitre en ce qui concerne l'analyse de l'impact de cette contrainte sur le comportement du système. Le chapitre suivant est un prolongement des travaux conduits dans ce chapitre où nous allons proposer une métaheuristique à base d'algorithme génétique pour minimiser le temps d'exécution maximal dans un environnement de machines parallèles avec des ressources consommables et processus réentrant.

Chapitre 4

Algorithme génétique pour l'ordonnancement sur machines parallèles avec ressources consommables et processus réentrant

4.1. Introduction

Dans un contexte industriel, avec une féroce concurrence et des circonstances en perpétuel changement, la gestion opérationnelle est devenue un facteur concurrentiel important. L'ordonnancement efficace de l'usine peut être un outil essentiel pour les entreprises afin d'atteindre des performances élevées. Cela est particulièrement vrai dans les industries de haute technologie (exemple : usine de semi-conducteurs), qui sont caractérisées par des flux de produits réentrants complexes et des équipements coûteux avec différentes caractéristiques d'ordonnancement. Un système de production réentrant, qui est différent des modes de production traditionnels, est caractérisé par un acheminement de produit qui se compose de plusieurs visites à un poste de travail ou d'un groupe de postes de travail au cours du processus de fabrication.

Le fait que ce problème est couramment rencontré dans l'environnement de la production actuelle, les entreprises manufacturières ont adopté des systèmes de production réentrants afin d'améliorer leurs performances en minimisant les coûts de production et en augmentant la productivité. D'autre part, le développement de méthodologies d'ordonnancement efficaces est devenu précieux pour le bon fonctionnement des industries. La résolution de ces modèles en considérant plusieurs contraintes devient de plus en plus difficile. En outre, la façon de traduire les objectifs de l'entreprise en des mesures de performances appropriées, l'augmentation du nombre de machines ainsi que le nombre de cycles de traitement, tout ceci augmente la complexité du système. Généralement, dans les industries de fabrication (cas discret en particulier), l'ordonnancement est effectué par des heuristiques ou des règles de priorités sans tenir compte de ces caractéristiques. Bien que ces règles soient très populaires en raison de leurs facilités d'application, la plupart utilisent uniquement des informations locales sur l'état du système de production.

Si l'apport du chapitre précédent se limite à l'étude de ressources consommables sur machines parallèles, ce chapitre aborde une autre piste à savoir la considération du processus réentrant du système étudié tout en tenant compte de différentes contraintes et de différents critères. Nous proposons un algorithme génétique capable de s'intégrer à cet environnement et de réagir aux divers changements. La prochaine section vise à présenter une description détaillée du problème étudié avec processus réentrant. La troisième section est destinée à la présentation de l'algorithme génétique proposé, les recherches locales adaptées et l'ensemble des heuristiques utilisées. La quatrième section est consacrée à une analyse de sensibilité de l'algorithme génétique proposé. Enfin, la dernière section de ce chapitre expose les résultats obtenus par une étude comparative entre les différents algorithmes proposés pour l'ordonnancement sur machines parallèles avec ressources consommables et processus réentrant. Nous commençons par présenter l'environnement des tests, ensuite nous établissons une comparaison entre l'algorithme génétique et les différentes heuristiques lorsqu'elles sont appliquées pour le premier cycle ou pour tous les cycles. Nous clôturons cette section par étudier l'impact des recherches locales sur les résultats de l'AG. Notons qu'à la fin de chaque section des résultats, des synthèses sont présentées afin d'expliquer le comportement des approches appliquées et leurs impacts sur le système.

4.2. Description du problème

Le problème considéré dans ce chapitre est l'ordonnancement sur machines parallèles identiques avec processus réentrant dans le but d'analyser l'impact des approches d'ordonnancement sur les performances du système. Le système étudié est composé d'un seul étage constitué de m machines parallèles identiques afin d'ordonnancer un ensemble de n tâches. Les caractéristiques de ce système peuvent être résumées comme suit :

- Chaque tâche doit effectuer un nombre maximal de L cycles (*layer*). L représente le nombre de cycles à effectuer (par exemple, si L = 2, ceci signifie que chaque tâche doit réaliser une première opération, puis une seconde opération sur le système).
- Chaque tâche j a un temps de traitement p_{jL} au cours du cycle L et peut avoir des chemins différents, à savoir, la même opération peut être traité durant le premier cycle sur une machine et dans le second cycle sur une autre machine. Pour cela, il nous faut préciser quelles sont les machines empruntées par les tâches. Pour éviter le chevauchement des tâches et limiter la variation de la durée du cycle de travail, des files d'attente sont situées devant chaque machine et chaque tâche sera affectée à la première machine disponible (qui contient le moins de tâches dans sa file d'attente), même si le nombre de cycles est différent pour les tâches.
- Les machines ne peuvent traiter qu'une seule tâche à la fois et elles sont disponibles à partir de l'instant t = 0 alors que la préemption n'est pas autorisée.
- Chaque tâche consomme k composantes au début de son exécution pour chaque cycle et peut être exécutée sur une machine i, lorsque tous les composants nécessaires sont disponibles.
- Le processus de production pourra être lancé si tôt qu'une tâche peut être exécutée.

- Les composants sont livrés par des fournisseurs à des moments différents. L'arrivée de ressources est représentée par courbe sous forme d'escalier. La configuration du système étudié est spécifiée dans la figure 4.1.
- L'objectif de cette étude est la minimisation du temps d'exécution maximale (makespan).

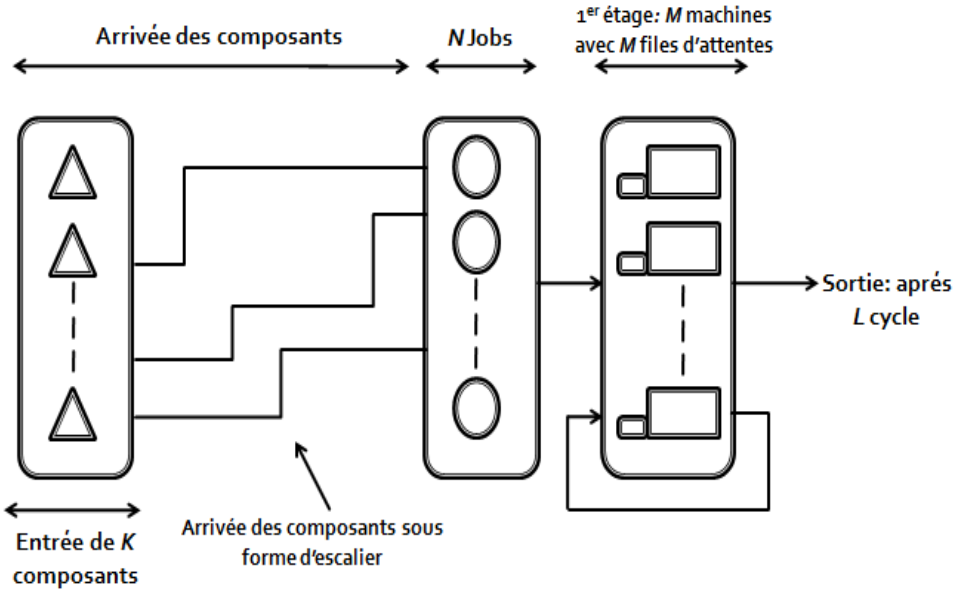

Figure 4.1 : Représentation du problème avec processus réentrant [Belkaid 13c]

4.3. Résolution du problème

4.3.1 Algorithme génétique

Dans cette section, nous rappelons brièvement l'algorithme adapté pour la résolution du problème avec processus réentrant, nous insistons principalement sur les spécificités et différences par rapport au chapitre précédent. Toutefois, son principe et mode de fonctionnement sont identiques à celui de l'algorithme génétique présenté dans le chapitre 3.

4.3.1.1 Codage

Pour illustrer le codage appliqué, nous prenons l'exemple suivant : nous considérons quatre tâches qui doivent être exécutées sur deux machines et performer deux cycles (c'est-à-dire que chaque tâche doit passer deux fois sur les machines). Le chromosome est composé d'une table de deux lignes, la première contient les tâches et la deuxième leurs affectations aux machines. De plus il est divisé en deux parties (selon le nombre de cycles). La première partie représente l'affectation des tâches aux machines pour le premier cycle et la seconde partie pour le second cycle. Par conséquent, le chromosome aura la structure suivante (Figure 4.2).

j_1	j_2	j_3	j_4	j_2	j_1	j_4	j_3
1	2	1	1	2	1	2	1
L=1				***L=2***			

Figure 4.2: Représentation du chromosome pour *L=2*

4.3.1.2 Croisement

Nous appliquons un croisement en deux points pour chaque cycle. La figure 4.3 illustre un exemple de quatre tâches qui doivent être exécutées sur deux machines où le nombre de cycles à effectuer est égal à 2.

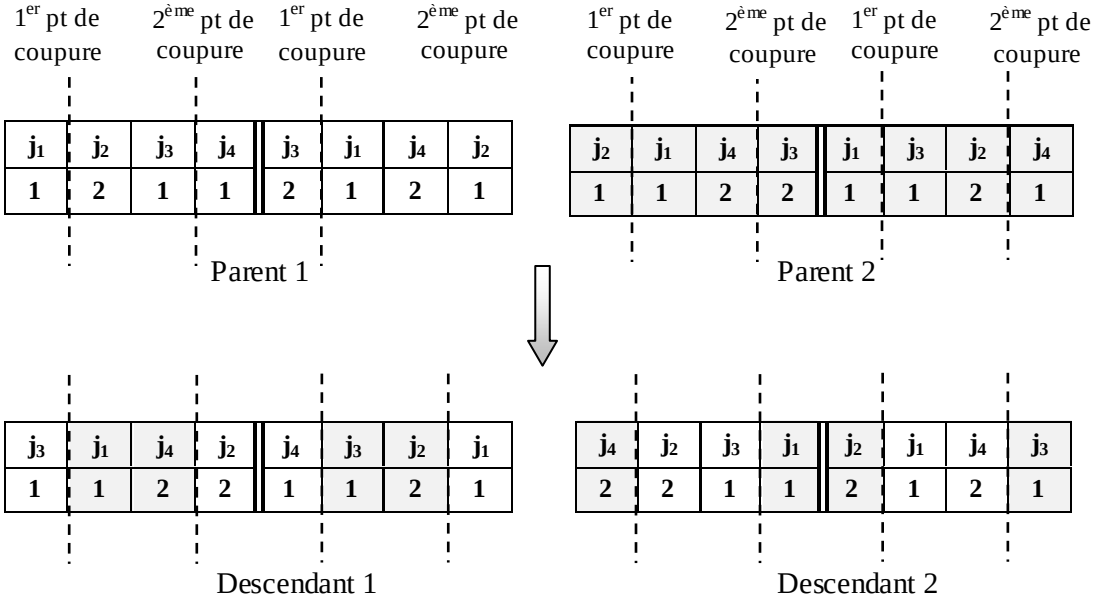

Figure 4.3 : Représentation de l'opérateur de croisement à deux points pour L=2

4.3.1.3 Mutation

L'opérateur de mutation choisi est un opérateur à un point qui s'applique de la manière suivante (voir figure 4.4).

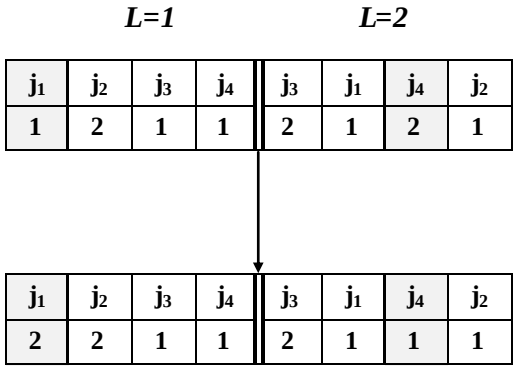

Figure 4.4 : Représentation de l'opérateur de mutation pour L=2

4.3.2 Recherche locale

La recherche locale est une technique qui consiste à améliorer les performances de la métaheuristique adaptée. Dans ce contexte, nous appliquons dans un premier temps l'approche de recherche locale utilisée dans le chapitre précédent et dans un second temps deux autres structures de voisinages basées sur des opérateurs SWAP notés API (Adjacent Pairwise Interchange) et NAPI (Non Adjacent Pairwise Interchange**)**. Ces deux opérateurs ont

été appliqués récemment par Dugardin et al. [Dugardin 12] afin d'améliorer une nouvelle méthode dans un environnement de flow shop hybride caractérisé par un processus réentrant.

4.3.2.1 L'opérateur DM (Descent Methods)

Cet opérateur consiste à changer l'affectation d'une tache j et cela $\forall j \in \{1, \dots, N\}$ comme le montre la figure 4.5.

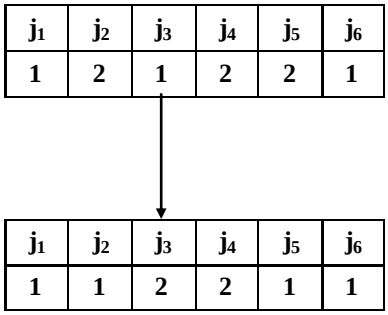

Figure 4.5 Exemple de l'opérateur API

4.3.2.2 L'opérateur API (Adjacent Pairwise Interchange)

Cet opérateur consiste à permuter entre la tâche j et la tâche j+1 et cela $\forall j \in \{1, \dots, (N-1)\}$ comme le montre la figure 4.6.

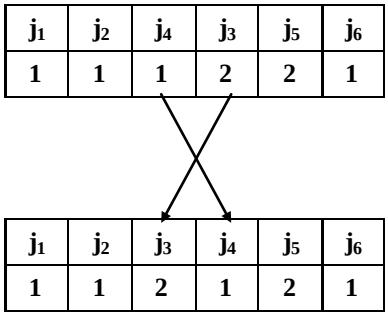

Figure 4.6 Exemple de l'opérateur API

4.3.2.3 L'opérateur NAPI (Non Adjacent Pairwise Interchange)

Cet opérateur consiste à permuter entre la tâche j et la tâche j+2 $\forall j \in \{1, \dots, (N-2)\}$ comme le montre la figure 4.7.

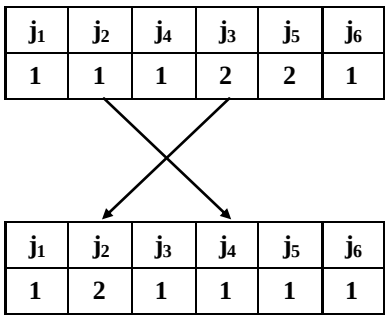

Figure 4.7 Exemple de l'opérateur NAPI

Algorithme 4.1 : Modèle de l'algorithme génétique hybride

S'il y a un ensemble de tâches non ordonnancé **alors**
Générer une population initiale d'une manière aléatoire.
Pour chaque chromosome
Calculer la fonction fitness du chromosome qui est représenté par le makespan.
Évaluer la fonction fitness du chromosome.
Si cette solution est plus petite que la meilleure solution **donc**
Actualiser la meilleure solution.
Fin si
Fin pour
Tant que le critère d'arrêt n'est pas satisfait
Appliquer l'opérateur de sélection.
Appliquer l'opérateur de croisement.
Appliquer une méthode de recherche locale (DM, API ou NAPI).
Appliquer l'opérateur de mutation.
Pour chaque chromosome
Calculer la fonction fitness du chromosome (le makespan).
Évaluer la fonction fitness du chromosome.
Si cette solution est plus petite que la meilleure solution **donc**
Actualiser la meilleure solution.
Fin si
Fin pour
Constituer la nouvelle génération.
Fin tant que
Fin si

4.4. Analyse de sensibilité de l'algorithme génétique proposé

Dans cette section, nous réalisons la même étude conduite au chapitre 3 en utilisant les mêmes protocoles des tests afin de régler les paramètres qui influe sur la métaheuristique pour l'ordonnancement réentrant sur machines parallèles avec ressources consommables. Rappelons que, les paramètres considérés sont les suivant :

- ✓ Paramètre de mutation(P_{mut})
- ✓ Paramètre de croisement(P_{cross})
- ✓ Nombre d'itération sans amélioration ($N_{ité}$)
- ✓ Taille de population (T_{pop})

4.4.1 L'effet du paramètre de mutation

Dans cette section nous étudions la variation du makespan en fonction du paramètre de mutation afin d'analyser l'impact de ce dernier sur les performances du système et d'analyser la sensibilité de l'algorithme génétique. Nous affectons aux paramètres suivants (P_{cross}, $N_{ité}$, T_{pop}) les valeurs suivantes (0.9, 50, 100). Le paramètre de mutation P_{mut}, varie comme suit P_{mut} ={0.05, 0.1, 0.15, 0.2}.

Tableau 4.1 : Effets de la mutation sur le makespan

Paramètre de mutation	0.05	0.1	0.15	0.2
$V_{moy}C_{max}$	165.944	166.111	166.277	166.277

Le tableau 4.1 présente les différentes probabilités de mutation concernant le makespan. Nous remarquons que la variation moyenne du C_{max} est inversement proportionnelle au paramètre de mutation, cependant la meilleure valeur moyenne est obtenue pour un faible taux de mutation égal à 0.05 ce qui correspond à une probabilité de 5 %.

4.4.2 L'effet du paramètre de croisement

Dans cette étude, nous analysons l'effet du paramètre de croisement sur les performances de l'algorithme génétique. À cet effet, le comportement de ce dernier est décrit en termes des paramètres (P_{mut}, $N_{ité}$, T_{pop}) qui prennent les valeurs suivantes (0.05, 50, 100). Notons que nous testons le paramètre de croisement P_{cross} lorsqu'il prend respectivement les valeurs qui sont décrites dans l'ensemble P_{cross}={0.7, 0.8, 0.9, 1}.

Tableau 4.2 : Effets du croisement sur le makespan

Paramètre de croisement	0.7	0.8	0.9	1
$V_{moy}C_{max}$	166.40	166.16	165.94	166.33

Nous remarquons dans le tableau 4.2 que la métaheuristique atteint ses meilleures performances lorsque la probabilité de croisement est égale à 0.09 ce qui correspond à une probabilité de 90% avec une valeur moyenne du temps d'exécution maximale égale à 165.94.

4.4.3 L'effet de nombre d'itérations

Pour définir la meilleure valeur du critère d'arrêt, nous l'avons varié dans l'ensemble suivant $N_{ité}$={50, 100, 150, 200} afin de déterminer la meilleure d'entre elles tout en précisant que les paramètres (P_{cross}, P_{mut}, T_{pop}) ont pris les valeurs suivantes (0.9, 0.05, 100).

Tableau 4.3 : Effets du nombre d'itérations sur le makespan

Nombre d'itérations	25	50	75	100
$V_{moy}C_{max}$	166.17	165.94	165.78	166.06

Le tableau 4.3, nous indique que la solution optimale est obtenue pour une valeur égale à 75 itérations successives sans amélioration.

4.4.4 L'effet de taille de population

Dans cette partie, l'objectif est de définir la taille de population optimale de l'algorithme génétique. Dans ce sens, nous attribuons aux paramètres (P_{mut}, P_{cross}, $N_{ité}$) les valeurs suivantes (0.05, 0.9, 50). En outre, la taille de population varie dans l'ensemble suivant T_{pop} = {50, 100, 150, 200}.

Tableau 4.4 : Effets de la taille de population sur le makespan

Taille de la population	50	100	150	200
$V_{moy}C_{max}$	167.17	165.94	165.82	165.82

À travers les résultats du tableau 4.4, nous remarquons que la meilleure valeur est obtenue pour une taille de la population égale à 150 (voir tableau 4.4) par conséquent nous retenons cette valeur pour effectuer les expérimentations.

À partir des différents protocoles de test que nous venons d'effectuer, nous obtenons les réglages du tableau 4.5.

Tableau 4.5 : Paramètres de l'algorithme génétique proposé

Paramètre	**Valeur**
Mutation	**0.05**
Croisement	**0.9**
Nombre d'itérations sans amélioration	**75**
Taille de population	**150**

4.5 Expérimentations et résultats

L'objectif des expérimentations menées dans cette section est d'analyser l'impact de la caractéristique réentrante du système sur les performances de ce dernier et d'évaluer le comportement de la métaheuristique proposée lorsque les différentes recherches locales sont appliquées.

4.5.1 Environnement des tests

Dans cette section, plusieurs approches sont adaptées pour performer les différentes expériences. Elles sont codées en langage Java et testées sur un ordinateur avec un Core (TM) i3 CPU - 2,13 GHz - 4,00 Go de Ram. Les performances du système sont évaluées par deux mesures de performances (les mêmes que celles du chapitre précédent) qui sont :

- CPU_{time} est le temps d'exécution c'est-à-dire le temps nécessaire pour qu'une méthode fournisse une solution à une instance donnée.
- GAP est l'erreur relative entre la solution courante et la meilleure solution obtenue pour chaque instance. Il peut être calculé par : $GAP = (C_{max}^{cur} - C_{max}^{best})/C_{max}^{best}$, où :
  - ✓ C_{max}^{best} est la meilleure solution
  - ✓ C_{max}^{cur} est la solution courante

La métaheuristique et les autres algorithmes sont testés sur différentes configurations du problème pour évaluer leurs performances. Dans ce sens, les variables utilisées pour effectuer cette investigation sont les suivantes :

N : Le nombre de tâches appartient à :

- {4, 6, 8} pour les problèmes de petites tailles
- {10, 20, 30} pour les problèmes de moyennes tailles
- {50, 75, 100} pour les problèmes de grandes tailles

M : Le nombre de machines appartient à :

- {2, 3} pour les problèmes de petites tailles
- {6, 8} pour les problèmes de moyennes tailles
- {10, 15} pour les problèmes de grandes tailles

K : Le nombre de composants appartient à {2, 3, 4}

L : Le nombre de cycles appartient à {2, 3, 4}

p_{jl} : Le temps de traitement de la tâche *j* au cycle *L*. Il est généré par une loi de distribution uniforme *U [1, 50]*.

Notons que pour effectuer les différentes simulations, nous avons pris les mêmes considérations que le chapitre précédent. De plus chaque tâche doit achever le nombre de cycles nécessaire pour son exécution afin de terminer le processus de production.

4.5.2 Comparaison entre l'AG et les heuristiques

Cette section est réservée à la comparaison entre les résultats obtenus par les différentes approches lorsqu'elles sont appliquées pour l'ordonnancement sur machines parallèles avec ressources non-renouvelable et processus réentrant. Les heuristiques qui sont en rapport avec le temps de traitement (LPT et SPT), la consommation des composants (LRC et SRC) ou bien les deux (L-PT/RC et S-PT/RC) sont proposées pour performer cette étude, elles sont appliquées que pour le premier cycle ; cependant, les tâches doivent accomplir le nombre de cycles nécessaire (restant) afin d'achever le processus de production. Dans ce sens, les tâches peuvent être assignées, suivant différentes priorités, basées sur plusieurs paramètres. Dans ce travail, nous avons établi ces décisions à l'aide d'une règle de priorité a portée locale FIFO utilisant la disponibilité des tâches comme un paramètre de priorité, par conséquent, chaque tâche sera dirigée vers la file d'attente de la machine qui comporte le moins de tâches.

4.5.2.1 Résultats des comparaisons pour un nombre de cycles égal à 2

Tableau 4.6 : Résultats expérimentaux pour les petites instances (L=2)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
4	**2**	**2**	0.26	0.000	0.00	0.118	0.00	0.118	0.00	0.000	0.00	0.176	0.00	0.118	0.00	0.059
		3	0.28	0.000	0.00	0.100	0.00	0.000	0.00	0.200	0.00	0.000	0.00	0.100	0.00	0.000
		4	0.29	0.000	0.00	0.077	0.00	0.000	0.00	0.077	0.00	0.000	0.00	0.000	0.00	0.000
	3	**2**	0.26	0.000	0.00	0.250	0.00	0.000	0.00	0.125	0.00	0.125	0.00	0.250	0.00	0.000
		3	0.28	0.000	0.00	0.083	0.00	0.000	0.00	0.083	0.00	0.000	0.00	0.000	0.00	0.000
		4	0.30	0.000	0.00	0.071	0.00	0.071	0.00	0.071	0.00	0.000	0.00	0.071	0.00	0.071
6	**2**	**2**	0.29	0.000	0.00	0.037	0.00	0.000	0.00	0.037	0.00	0.037	0.00	0.000	0.00	0.037
		3	0.31	0.000	0.00	0.125	0.00	0.000	0.00	0.188	0.00	0.000	0.00	0.125	0.00	0.000
		4	0.34	0.000	0.01	0.043	0.01	0.043	0.01	0.043	0.01	0.000	0.01	0.130	0.01	0.000
	3	**2**	0.30	0.000	0.01	0.167	0.01	0.111	0.01	0.111	0.01	0.167	0.01	0.222	0.01	0.056
		3	0.33	0.000	0.00	0.176	0.00	0.118	0.00	0.000	0.00	0.176	0.00	0.118	0.00	0.059
		4	0.35	0.000	0.01	0.053	0.01	0.000	0.01	0.053	0.01	0.000	0.01	0.053	0.01	0.000
8	**2**	**2**	0.34	0.000	0.00	0.023	0.00	0.023	0.00	0.023	0.00	0.023	0.00	0.023	0.00	0.023
		3	0.36	0.000	0.00	0.023	0.00	0.000	0.00	0.045	0.00	0.000	0.00	0.091	0.00	0.000
		4	0.39	0.000	0.00	0.032	0.00	0.000	0.00	0.097	0.00	0.000	0.00	0.097	0.00	0.000
	3	**2**	0.34	0.000	0.00	0.130	0.00	0.043	0.00	0.217	0.00	0.174	0.00	0.217	0.00	0.000
		3	0.37	0.000	0.00	0.115	0.00	0.038	0.00	0.077	0.00	0.077	0.00	0.115	0.00	0.038
		4	0.40	0.000	0.01	0.208	0.01	0.042	0.01	0.167	0.01	0.083	0.01	0.083	0.01	0.125

Tableau 4.7 : Résultats expérimentaux pour les moyennes instances (L=2)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
10	**6**	**2**	0.45	0.000	0.00	0.170	0.00	0.099	0.00	0.206	0.00	0.206	0.00	0.206	0.00	0.206
		3	0.47	0.000	0.01	0.179	0.01	0.036	0.01	0.250	0.01	0.143	0.01	0.143	0.01	0.036
		4	0.54	0.000	0.00	0.333	0.00	0.000	0.00	0.100	0.00	0.267	0.00	0.333	0.00	0.033
	8	**2**	0.47	0.000	0.01	0.111	0.01	0.000	0.01	0.083	0.01	0.028	0.01	0.083	0.01	0.000
		3	0.61	0.000	0.01	0.111	0.01	0.022	0.01	0.067	0.01	0.111	0.01	0.111	0.01	0.000
		4	0.58	0.000	0.01	0.121	0.01	0.000	0.01	0.069	0.01	0.086	0.01	0.103	0.01	0.000
20	**6**	**2**	1.03	0.000	0.00	0.254	0.00	0.042	0.01	0.169	0.00	0.183	0.00	0.254	0.00	0.000
		3	1.08	0.000	0.00	0.124	0.00	0.041	0.00	0.160	0.00	0.077	0.00	0.148	0.00	0.053
		4	1.36	0.000	0.01	0.207	0.01	0.009	0.01	0.037	0.01	0.051	0.01	0.207	0.01	0.023
	8	**2**	1.28	0.000	0.00	0.333	0.00	0.037	0.00	0.167	0.00	0.370	0.00	0.352	0.00	0.000
		3	0.88	0.000	0.00	0.219	0.00	0.016	0.00	0.089	0.00	0.147	0.00	0.219	0.00	0.016
		4	0.98	0.000	0.00	0.257	0.01	0.018	0.00	0.108	0.01	0.272	0.01	0.243	0.01	0.018
30	**6**	**2**	1.17	0.000	0.00	0.129	0.00	0.009	0.00	0.078	0.00	0.129	0.00	0.160	0.00	0.003
		3	1.32	0.000	0.00	0.114	0.00	0.024	0.00	0.047	0.00	0.058	0.00	0.114	0.00	0.005
		4	1.45	0.000	0.00	0.135	0.00	0.021	0.00	0.055	0.00	0.080	0.00	0.135	0.00	0.008
	8	**2**	1.16	0.000	0.00	0.156	0.00	0.011	0.00	0.059	0.00	0.164	0.00	0.172	0.00	0.019
		3	1.33	0.000	0.00	0.247	0.00	0.032	0.00	0.187	0.00	0.268	0.00	0.241	0.00	0.005
		4	1.57	0.000	0.00	0.273	0.00	0.028	0.00	0.219	0.00	0.150	0.00	0.253	0.00	0.014

Tableau 4.8 : Résultats expérimentaux pour les grandes instances (L=2)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
50	**10**	**2**	2.38	0.000	0.04	0.133	0.03	0.000	0.00	0.050	0.04	0.140	0.04	0.161	0.04	0.007
		3	2.50	0.000	0.04	0.158	0.00	0.000	0.00	0.054	0.00	0.165	0.04	0.194	0.04	0.011
		4	2.81	0.000	0.01	0.127	0.01	0.003	0.01	0.093	0.01	0.143	0.01	0.127	0.01	0.018
	15	**2**	2.70	0.000	0.00	0.126	0.00	0.029	0.00	0.084	0.00	0.109	0.00	0.135	0.00	0.037
		3	2.62	0.000	0.01	0.128	0.01	0.030	0.01	0.102	0.01	0.123	0.01	0.149	0.01	0.026
		4	2.74	0.000	0.01	0.167	0.01	0.024	0.01	0.074	0.01	0.129	0.01	0.171	0.01	0.019
75	**10**	**2**	4.07	0.000	0.00	0.075	0.00	0.021	0.00	0.040	00.00	0.068	0.00	0.105	0.00	0.000
		3	4.10	0.000	0.00	0.075	0.00	0.021	0.01	0.051	0.01	0.093	0.01	0.100	0.00	0.000
		4	4.80	0.000	0.01	0.096	0.01	0.024	0.01	0.022	0.01	0.104	0.01	0.099	0.01	0.000
	15	**2**	4.17	0.000	0.00	0.167	0.00	0.007	0.00	0.054	0.00	0.116	0.01	0.188	0.00	0.000
		3	4.24	0.000	0.01	0.095	0.01	0.006	0.02	0.040	0.02	0.040	0.02	0.105	0.01	0.006
		4	5.03	0.000	0.01	0.158	0.02	0.025	0.01	0.062	0.01	0.134	0.02	0.177	0.02	0.000
100	**10**	**2**	6.01	0.000	0.00	0.097	0.00	0.004	0.00	0.044	0.00	0.064	0.00	0.097	0.00	0.000
		3	6.69	0.000	0.00	0.073	0.00	0.013	0.01	0.043	0.01	0.038	0.01	0.075	0.01	0.010
		4	7.02	0.000	0.00	0.075	0.00	0.015	0.00	0.043	0.01	0.056	0.00	0.071	0.00	0.005
	15	**2**	6.29	0.000	0.00	0.099	0.00	0.014	0.00	0.053	0.00	0.063	0.00	0.096	0.00	0.000
		3	7.01	0.000	0.00	0.106	0.00	0.002	0.00	0.040	0.00	0.049	0.00	0.082	0.00	0.000
		4	6.82	0.000	0.00	0.116	0.00	0.012	0.00	0.055	0.00	0.076	0.00	0.097	00.0	0.000

L'investigation concernant le makespan lorsque chaque opération doit effectuer deux cycles pour les problèmes de petite, moyenne et grande taille est présentée dans les tableaux (4.6-4.8). Selon ces résultats, nous remarquons que le temps d'exécution est presque similaire pour toutes les heuristiques et il est directement proportionnel aux nombres de tâches pour ce qui concerne l'AG. Entre autre, nous constatons qu'il y a une différence significative entre le makespan obtenu par la métaheuristique et la majorité des heuristiques, en particulier si le nombre de tâches varie entre 10 et 30, le nombre de machines entre 10 et 15 et le nombre de composants est supérieur à 2. Il convient de noter que les heuristiques LPT et L-PT/RC peuvent trouver des solutions proches de l'algorithme génétique, surtout si le nombre de tâches est supérieur ou égal à 50. En outre, SPT donne les plus mauvais résultats, en particulier pour des problèmes de taille moyenne. En ce qui concerne les heuristiques qui sont basés sur le temps de traitement, nous observons que LPT qui est connu comme la règle de séquencement la plus appropriée pour les problèmes d'ordonnancement sur machines parallèles se comporte mieux que SPT dans tous les cas. En outre, pour les heuristiques qui ont un rapport avec la consommation de composants, SRC a presque le même comportement que LRC pour les petites instances et la dépasse pour les moyennes et grandes instances. Finalement, sur la base des résultats obtenus par les heuristiques qui ont un rapport avec le temps de traitement et consommation de composants, nous pouvons confirmer que l'heuristique L-PT/RC est plus performante que S-PT/RC pour toutes les tailles de problèmes. Pour conclure, la méthode d'optimisation proposée qui est basée sur l'algorithme génétique surpasse les autres approches approximatives sur la base de makespan.

4.5.2.2 Résultats des comparaisons pour un nombre de cycles égal à 3

Tableau 4.9 : Résultats expérimentaux pour les petites instances (L=3)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
4	**2**	**2**	0.28	0.000	0.01	0.083	0.01	0.083	0.01	0.000	0.01	0.125	0.01	0.083	0.01	0.042
		3	0.32	0.000	0.00	0.071	0.00	0.000	0.00	0.143	0.00	0.000	0.00	0.071	0.00	0.000
		4	0.33	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00	0.000
	3	**2**	0.30	0.000	0.00	0.182	0.00	0.000	0.00	0.091	0.00	0.000	0.00	0.091	0.00	0.000
		3	0.32	0.000	0.00	0.133	0.00	0.000	0.00	0.133	0.00	0.000	0.00	0.067	0.00	0.000
		4	0.34	0.000	0.00	0.056	0.00	0.056	0.00	0.056	0.00	0.000	0.00	0.056	0.00	0.056
6	**2**	**2**	0.34	0.000	0.01	0.026	0.01	0.000	0.01	0.026	0.01	0.000	0.01	0.000	0.01	0.000
		3	0.37	0.000	0.00	0.042	0.00	0.000	0.00	0.083	0.00	0.000	0.00	0.042	0.00	0.000
		4	0.40	0.000	0.00	0.059	0.00	0.029	0.00	0.029	0.00	0.000	0.00	0.088	0.00	0.029
	3	**2**	0.35	0.000	0.00	0.115	0.00	0.077	0.00	0.038	0.00	0.115	0.00	0.154	0.00	0.038
		3	0.39	0.000	0.00	0.083	0.00	0.083	0.00	0.000	0.00	0.125	0.00	0.042	0.00	0.042
		4	0.41	0.000	0.01	0.040	0.01	0.040	0.01	0.080	0.01	0.080	0.01	0.080	0.01	0.080
8	**2**	**2**	0.40	0.000	0.00	0.016	0.00	0.000	0.00	0.000	0.00	0.016	0.00	0.016	0.00	0.000
		3	0.43	0.000	0.01	0.015	0.01	0.000	0.01	0.030	0.01	0.000	0.01	0.045	0.01	0.000
		4	0.47	0.000	0.00	0.000	0.00	0.000	0.00	0.043	0.00	0.000	0.00	0.065	0.00	0.000
	3	**2**	0.41	0.000	0.01	0.059	0.01	0.029	0.01	0.147	0.01	0.118	0.01	0.147	0.01	0.029
		3	0.44	0.000	0.00	0.111	0.00	0.056	0.00	0.083	0.00	0.083	0.00	0.083	0.00	0.056
		4	0.48	0.000	0.00	0.147	0.00	0.029	0.00	0.118	0.00	0.059	0.00	0.059	0.00	0.088

Tableau 4.10 : Résultats expérimentaux pour les moyennes instances (L=3)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
10	**6**	**2**	0.55	0.000	0.00	0.122	0.00	0.097	0.00	0.148	0.00	0.173	0.00	0.173	0.00	0.122
		3	0.61	0.000	0.00	0.172	0.00	0.035	0.00	0.172	0.00	0.117	0.00	0.090	0.00	0.035
		4	0.71	0.000	0.01	0.231	0.01	0.000	0.01	0.051	0.01	0.179	0.01	0.231	0.01	0.000
	8	**2**	0.60	0.000	0.01	0.116	0.01	0.023	0.00	0.070	0.00	0.047	0.01	0.070	0.01	0.047
		3	0.70	0.000	0.00	0.134	0.01	0.021	0.01	0.040	0.01	0.096	0.01	0.115	0.01	0.002
		4	0.76	0.000	0.00	0.127	0.00	0.000	0.00	0.079	0.00	0.095	0.00	0.111	0.00	0.016
20	**6**	**2**	1.26	0.000	0.00	0.183	0.00	0.029	0.00	0.115	0.00	0.115	0.00	0.173	0.00	0.000
		3	1.57	0.000	0.00	0.078	0.00	0.021	0.00	0.111	0.00	0.046	0.00	0.095	0.00	0.021
		4	1.96	0.000	0.01	0.149	0.01	0.008	0.01	0.028	0.01	0.038	0.01	0.149	0.01	0.018
	8	**2**	1.64	0.000	0.00	0.231	0.00	0.000	0.00	0.103	0.00	0.244	0.00	0.231	0.00	0.013
		3	1.12	0.000	0.00	0.161	0.00	0.011	0.00	0.086	0.00	0.108	0.00	0.151	0.00	0.011
		4	1.25	0.000	0.00	0.173	0.00	0.017	0.00	0.084	0.00	0.207	0.00	0.184	0.00	0.017
30	**6**	**2**	1.43	0.000	0.00	0.094	0.00	0.004	0.00	0.051	0.00	0.085	0.00	0.107	0.00	0.000
		3	1.70	0.000	0.00	0.070	0.00	0.016	0.00	0.031	0.00	0.039	0.00	0.078	0.00	0.003
		4	1.75	0.000	0.00	0.093	0.00	0.010	0.00	0.033	0.00	0.056	0.00	0.091	0.00	0.004
	8	**2**	1.53	0.000	0.00	0.104	0.00	0.004	0.00	0.032	0.00	0.110	0.00	0.110	0.00	0.004
		3	1.84	0.000	0.00	0.160	0.00	0.020	0.00	0.132	0.00	0.183	0.00	0.169	0.00	0.002
		4	1.81	0.000	0.01	0.183	0.00	0.023	0.01	0.155	0.01	0.108	0.01	0.178	0.01	0.019

Tableau 4.11 : Résultats expérimentaux pour les grandes instances (L=3)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
50	**10**	**2**	2.87	0.000	0.00	0.091	0.00	0.000	0.00	0.034	0.00	0.098	0.00	0.111	0.00	0.005
		3	3.02	0.000	0.00	0.111	0.00	0.000	0.00	0.034	0.00	0.111	0.00	0.131	0.00	0.005
		4	3.32	0.000	0.02	0.086	0.01	0.001	0.02	0.065	0.02	0.099	0.02	0.088	0.02	0.012
	15	**2**	3.33	0.000	0.00	0.080	0.00	0.019	0.00	0.056	0.00	0.074	0.00	0.092	0.00	0.025
		3	3.37	0.000	0.00	0.083	0.00	0.021	0.01	0.070	0.00	0.089	0.00	0.104	0.00	0.012
		4	3.51	0.000	0.00	0.112	0.02	0.023	0.02	0.057	0.02	0.103	0.02	0.121	0.02	0.014
75	**10**	**2**	4.99	0.000	0.00	0.053	0.00	0.015	0.00	0.029	0.00	0.046	0.00	0.074	0.00	0.000
		3	5.18	0.000	0.00	0.051	0.00	0.013	0.00	0.037	0.00	0.064	0.00	0.068	0.00	0.000
		4	5.34	0.000	0.01	0.065	0.01	0.016	0.02	0.015	0.02	0.068	0.01	0.067	0.02	0.000
	15	**2**	4.87	0.000	0.01	0.105	0.01	0.005	0.00	0.037	0.00	0.081	0.02	0.127	0.01	0.000
		3	5.38	0.000	0.01	0.058	0.01	0.004	0.01	0.027	0.01	0.072	0.01	0.074	0.01	0.004
		4	6.39	0.000	0.00	0.107	0.00	0.015	0.01	0.044	0.01	0.094	0.01	0.123	0.01	0.000
100	**10**	**2**	7.41	0.000	0.01	0.064	0.01	0.002	0.00	0.028	0.00	0.043	0.00	0.066	0.00	0.000
		3	8.76	0.000	0.00	0.050	0.01	0.008	0.00	0.027	0.00	0.024	0.00	0.050	0.00	0.005
		4	8.26	0.000	0.00	0.051	0.01	0.008	0.00	0.027	0.00	0.035	0.00	0.046	0.00	0.001
	15	**2**	8.51	0.000	0.00	0.068	0.01	0.008	0.00	0.035	0.00	0.043	0.00	0.065	0.00	0.000
		3	7.97	0.000	0.01	0.073	0.01	0.003	0.00	0.028	0.00	0.034	0.00	0.057	0.00	0.000
		4	8.59	0.000	0.00	0.079	0.00	0.008	0.01	0.036	0.01	0.051	0.01	0.067	0.01	0.000

Lorsque le nombre de cycles à effectuer est égal à 3, les résultats concernant l'impact des approches appliquées sur les indicateurs de performances étudiées pour différents tests - tailles sont synthétisés dans les tableaux (4.9-4.11). Les performances des heuristiques ne peuvent pas être mesurées par le temps de traitement, en raison du fait qu'il n'a pas un impact important dans les résultats. D'autre part, ce critère d'évaluation s'accroit rapidement pour les métaheuristiques lorsque la taille du problème augmente.

De plus, nous remarquons qu'il existe une différence significative dans l'erreur moyenne relative entre les solutions obtenues par l'algorithme génétique développé et les heuristiques adaptées. Les résultats montrent que la meilleure solution pour ce problème peut être obtenue par la métaheuristique, tout en notant qu'elle surpasse toutes les heuristiques principalement pour les problèmes de taille moyenne. En outre, SPT donne les plus mauvais résultats. Cependant, le temps de calcul reste faible pour toutes les méthodes. Une observation importante doit être citée qui est que LPT et L-PT/RC peuvent conduire généralement à une solution intéressante par rapport à l'algorithme génétique.

Selon les heuristiques qui sont basées sur le temps de traitement, il est remarqué que LPT donne de meilleurs résultats que SPT pour différentes tailles de problèmes. En outre, LRC et SRC ont pratiquement le même comportement avec un simple avantage pour SRC, sur la base de composants. Enfin, L-PT/RC surpasse S-PT/RC en particulier pour les moyennes instances sur la base des ratios temps de traitement par rapport à la consommation des ressources.

4.5.2.3 Résultats des comparaisons pour un nombre de cycles égale à 4

Tableau 4.12 : Résultats expérimentaux pour les petites instances (L=4)

Problèmes			***AG***		***SPT***		***LPT***		***SRC***		***LRC***		***S-PT/RC***		***L-PT/RC***	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
4	**2**	**2**	0.32	0.000	0.01	0.065	0.01	0.065	0.01	0.000	0.01	0.097	0.01	0.065	0.01	0.032
		3	0.34	0.000	0.01	0.056	0.01	0.000	0.01	0.111	0.01	0.000	0.01	0.056	0.01	0.000
		4	0.37	0.000	0.01	0.048	0.01	0.000	0.01	0.048	0.01	0.000	0.01	0.000	0.01	0.000
	3	**2**	0.32	0.000	0.01	0.154	0.01	0.000	0.01	0.154	0.01	0.077	0.01	0.154	0.01	0.077
		3	0.36	0.000	0.01	0.105	0.01	0.000	0.01	0.105	0.01	0.000	0.01	0.000	0.01	0.000
		4	0.39	0.000	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.000	0.01	0.000
6	**2**	**2**	0.38	0.000	0.01	0.021	0.01	0.000	0.01	0.021	0.01	0.000	0.01	0.000	0.01	0.021
		3	0.43	0.000	0.01	0.065	0.01	0.000	0.01	0.097	0.01	0.000	0.01	0.065	0.01	0.000
		4	0.46	0.000	0.01	0.022	0.01	0.022	0.01	0.022	0.01	0.000	0.01	0.067	0.01	0.000
	3	**2**	0.41	0.000	0.01	0.059	0.01	0.059	0.01	0.029	0.01	0.059	0.01	0.088	0.01	0.000
		3	0.49	0.000	0.01	0.075	0.01	0.042	0.01	0.010	0.01	0.075	0.01	0.042	0.01	0.010
		4	0.47	0.000	0.01	0.031	0.01	0.000	0.01	0.063	0.01	0.031	0.01	0.031	0.01	0.031
8	**2**	**2**	0.46	0.000	0.01	0.013	0.01	0.013	0.01	0.013	0.01	0.013	0.01	0.013	0.01	0.013
		3	0.50	0.000	0.01	0.011	0.01	0.000	0.01	0.023	0.01	0.000	0.01	0.046	0.01	0.000
		4	0.54	0.000	0.01	0.017	0.01	0.000	0.01	0.050	0.01	0.000	0.01	0.050	0.01	0.000
	3	**2**	0.48	0.000	0.01	0.067	0.01	0.022	0.01	0.111	0.01	0.089	0.01	0.111	0.01	0.000
		3	0.51	0.000	0.01	0.085	0.01	0.043	0.01	0.064	0.01	0.043	0.01	0.064	0.01	0.021
		4	0.58	0.000	0.01	0.114	0.01	0.045	0.01	0.091	0.01	0.068	0.01	0.045	0.01	0.068

Tableau 4.13 : Résultats expérimentaux pour les moyennes instances (L=4)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
10	**6**	2	0.64	0.000	0.01	0.109	0.01	0.050	0.01	0.109	0.01	0.109	0.01	0.129	0.01	0.089
		3	0.71	0.000	0.01	0.109	0.01	0.042	0.01	0.153	0.01	0.086	0.01	0.086	0.01	0.064
		4	0.88	0.000	0.01	0.191	0.01	0.000	0.01	0.064	0.01	0.170	0.01	0.191	0.01	0.021
	8	2	0.76	0.000	0.01	0.078	0.01	0.020	0.01	0.059	0.01	0.039	0.01	0.039	0.01	0.020
		3	0.88	0.000	0.01	0.107	0.01	0.008	0.01	0.058	0.01	0.074	0.01	0.091	0.01	0.008
		4	0.92	0.000	0.01	0.101	0.01	0.014	0.01	0.072	0.01	0.087	0.01	0.101	0.01	0.014
20	**6**	2	1.57	0.000	0.01	0.131	0.01	0.022	0.02	0.088	0.02	0.088	0.02	0.139	0.02	0.000
		3	2.19	0.000	0.01	0.057	0.01	0.020	0.01	0.089	0.01	0.038	0.01	0.076	0.01	0.020
		4	3.26	0.000	0.02	0.102	0.01	0.008	0.02	0.023	0.02	0.031	0.02	0.117	0.02	0.016
	8	2	1.45	0.000	0.02	0.168	0.01	0.000	0.01	0.079	0.01	0.178	0.01	0.168	0.01	0.000
		3	1.35	0.000	0.01	0.130	0.01	0.010	0.01	0.062	0.01	0.087	0.01	0.122	0.01	0.010
		4	1.52	0.000	0.02	0.140	0.02	0.007	0.02	0.060	0.01	0.157	0.01	0.140	0.02	0.007
30	**6**	2	1.71	0.000	0.02	0.065	0.01	0.003	0.01	0.039	0.01	0.065	0.01	0.081	0.01	0.000
		3	2.21	0.000	0.02	0.055	0.01	0.014	0.01	0.024	0.02	0.030	0.01	0.057	0.01	0.004
		4	2.18	0.000	0.01	0.069	0.01	0.010	0.01	0.025	0.02	0.043	0.01	0.069	0.01	0.005
	8	2	1.92	0.000	0.01	0.081	0.01	0.004	0.01	0.030	0.01	0.081	0.01	0.089	0.01	0.004
		3	2.08	0.000	0.01	0.123	0.01	0.014	0.01	0.099	0.01	0.137	0.01	0.127	0.01	0.000
		4	2.43	0.000	0.01	0.139	0.01	0.017	0.01	0.117	0.01	0.082	0.01	0.135	0.01	0.014

Tableau 4.14 : Résultats expérimentaux pour les grandes instances (L=4)

Problèmes			*AG*		*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**	**CPU_T**	**GAP**
50	**10**	2	3.30	0.000	0.01	0.069	0.01	0.000	0.01	0.037	0.01	0.075	0.01	0.084	0.01	0.004
		3	3.91	0.000	0.01	0.084	0.01	0.000	0.01	0.027	0.01	0.086	0.01	0.099	0.01	0.005
		4	3.91	0.000	0.01	0.067	0.01	0.000	0.01	0.051	0.01	0.078	0.01	0.070	0.01	0.010
	15	2	3.76	0.000	0.01	0.064	0.01	0.014	0.01	0.043	0.01	0.060	0.01	0.072	0.01	0.021
		3	4.37	0.000	0.01	0.066	0.01	0.015	0.01	0.056	0.01	0.071	0.01	0.086	0.01	0.010
		4	4.49	0.000	0.01	0.091	0.01	0.016	0.01	0.048	0.01	0.079	0.01	0.096	0.01	0.014
75	**10**	2	5.10	0.000	0.01	0.040	0.01	0.011	0.01	0.021	0.01	0.035	0.01	0.056	0.01	0.000
		3	5.92	0.000	0.01	0.039	0.01	0.010	0.01	0.028	0.01	0.048	0.01	0.052	0.01	0.000
		4	6.14	0.000	0.01	0.047	0.01	0.012	0.01	0.010	0.01	0.052	0.01	0.052	0.01	0.000
	15	2	6.10	0.000	0.01	0.079	0.01	0.004	0.01	0.028	0.01	0.061	0.01	0.096	0.01	0.000
		3	6.52	0.000	0.01	0.045	0.01	0.002	0.02	0.020	0.02	0.054	0.01	0.056	0.01	0.002
		4	8.29	0.000	0.01	0.083	0.01	0.012	0.01	0.034	0.01	0.073	0.01	0.095	0.01	0.000
100	**10**	2	7.81	0.000	0.02	0.048	0.02	0.001	0.01	0.021	0.01	0.032	0.01	0.050	0.01	0.000
		3	8.70	0.000	0.02	0.039	0.02	0.008	0.01	0.023	0.02	0.020	0.01	0.039	0.01	0.006
		4	9.60	0.000	0.02	0.038	0.02	0.005	0.01	0.020	0.01	0.027	0.01	0.034	0.01	0.001
	15	2	9.11	0.000	0.02	0.051	0.02	0.008	0.02	0.028	0.02	0.033	0.02	0.051	0.02	0.001
		3	9.47	0.000	0.02	0.058	0.02	0.003	0.01	0.021	0.01	0.026	0.02	0.045	0.02	0.000
		4	9.98	0.000	0.02	0.061	0.02	0.006	0.01	0.029	0.01	0.039	0.01	0.051	0.01	0.000

Les tableaux (4.12 - 4.14) présentent les résultats concernant les indicateurs de performances obtenus pour les problèmes de différentes tailles quand chaque tâche doit effectuer quatre cycles. Comme il est remarqué dans ces tableaux, le temps d'exécution est pratiquement le même dans toutes les heuristiques, Cependant, une augmentation qui atteint presque les dix secondes pour l'algorithme génétique.

Les résultats obtenus montrent que L-PT/RC et LPT donnent presque les mêmes résultats pour le problème de petites et moyennes tailles. L'heuristique L-PT/RC donne des solutions proches de la solution optimale obtenue par l'AG. D'autre part, la meilleure solution peut être obtenue par la métaheuristique pour les problèmes de grande taille. Les valeurs indiquées dans les tableaux concernant les heuristiques qui ont un rapport avec le temps de traitement démontrent que LPT surpasse SPT pour tous les cas.

D'autre part, les valeurs concernant les heuristiques qui se basent sur la consommation des ressources montrent que SRC et LRC ont un comportement presque similaire pour les petites instances (une petite différence entre ces deux heuristiques) mais SRC commence à creuser l'écart pour les moyennes instances et la surpasse largement pour les grandes instances. En outre, L-PT/RC dépasse S-PT/RC en particulier pour les problèmes de moyennes et grandes tailles sur la base du rapport temps de traitement et consommation des ressources. Notons que, SPT et S-PT/RC donnent les pires résultats.

- **Synthèse**

L'étude menée dans cette section, nous a permis de donner un aperçu sur l'impact des ces approches sur le système étudié et de dégager un certain nombre de conclusions, qui sont résumées ci-dessous :

- Le temps de calcul reste faible pour les heuristiques qui sont en rapport avec le temps de traitement (*LPT* et *SPT*), la consommation des composants (*LRC* et *SRC*) ou bien les deux (*L-PT/RC* et *S-PT/RC)* sur tous les cycles.
- Le temps de traitement de l'AG est directement proportionnel au nombre de cycles, plus ce dernier s'accroit, plus le temps de calcul augmente.
- L'erreur relative moyenne entre les différentes approches diminue lorsque le nombre de cycles augmente. Cela peut être expliqué par le fait que le fonctionnement des heuristiques s'améliore lorsque la taille du problème augmente.
- LPT donne de meilleurs résultats par rapport à SPT, cela est dû au fait que : LPT conserve les petits jobs en termes de temps de traitement à la fin de l'ordonnancement afin d'équilibrer la charge du système.
- SRC dépasse LRC en termes de makespan, pour la majorité des cas; car SRC maintient les tâches qui consomment plusieurs composants à la fin de l'ordonnancement ce qui accélère le processus de production.
- Vu que, L-PT/RC est une heuristique qui vise à promouvoir d'un coté les jobs qui ont un temps de traitement long et de l'autre les jobs qui ont une faible consommation des ressources, ainsi cela permet d'obtenir un compromis entre ces deux critères, par conséquent, elle mène à un équilibre de la charge moyenne.

- L'algorithme génétique est le plus performant pour des problèmes de structures différentes. Cependant, pour le reste des heuristiques, il n'y a pas une approche qui surclasse les autres dans tous les cas. Sauf si *L-PT/RC* permet de fournir des solutions acceptables. Dans ce cas, il serait judicieux, de combiner ces deux approches pour améliorer le fonctionnement de l'algorithme génétique.

4.5.3 Amélioration des performances des heuristiques

Dans la section précédente, nous avons appliqué les heuristiques seulement pour le premier cycle et pour ce qui est du reste, nous avons adapté la règle FIFO dont le but d'affecter les tâches aux machines.

Cependant, le développement de méthodologies d'ordonnancement simples et efficaces est devenu précieux pour le bon fonctionnement des industries. Dans ce contexte, nous utilisons les heuristiques qui sont respectivement LPT et SPT, LRC et SRC, L-PT/RC et S-PT/RC sur l'ensemble des cycles afin d'essayer d'améliorer les performances du système. Cela implique que chacune de ces méthodes est appliquée indépendamment pour chaque cycle.

Pour effectuer les différentes expérimentations, nous utilisons l'appellation suivante :

- ✓ $C_{max}H_1$: représente le makespan obtenu lorsque l'heuristique est appliquée que pour le premier cycle.
- ✓ $C_{max}H_L$: représente le makespan obtenu lorsque l'heuristique en question est appliquée pour tous les cycles.

4.5.3.1 Résultats des comparaisons entre les différentes heuristiques lorsque L=2

Tableau 4.15 : Résultats des heuristiques pour les petites instances (L=2)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
4	**2**	**2**	19	20	19	18	17	16	20	20	21	19	17	17
		3	12	10	10	10	10	12	10	10	11	12	10	10
		4	15	14	13	13	14	14	14	13	13	14	13	13
	3	**2**	12	10	8	8	9	11	8	9	9	10	9	10
		3	14	12	12	11	14	12	14	12	12	12	11	12
		4	17	15	13	14	14	15	15	14	15	15	13	13
6	**2**	**2**	29	26	26	26	26	26	27	27	29	26	26	26
		3	19	17	15	15	18	16	15	15	18	15	15	15
		4	26	23	22	22	23	21	22	23	24	22	21	21
	3	**2**	24	21	18	18	22	20	21	20	19	19	18	20
		3	20	19	19	18	20	18	18	18	18	19	18	16
		4	21	20	18	18	19	19	18	19	23	20	19	19
8	**2**	**2**	45	41	40	40	40	40	43	43	41	40	41	41
		3	44	42	40	38	43	43	42	40	44	41	39	40
		4	32	30	28	28	30	30	29	29	30	30	28	28
	3	**2**	25	24	22	22	25	23	27	23	24	24	21	21
		3	29	28	25	25	26	26	30	27	28	28	26	26
		4	30	25	23	23	24	23	25	24	25	25	23	23

Tableau 4.16 : Résultats des heuristiques pour les moyennes instances (L=2)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
10	**10**	**2**	35	32	28	28	34	30	31	34	35	34	32	30
		3	36	31	26	26	31	34	33	30	34	30	26	27
		4	22	18	18	18	20	20	20	18	20	20	18	17
	15	**2**	47	38	36	34	41	36	40	37	36	37	35	37
		3	23	21	19	19	23	17	19	19	20	19	19	19
		4	67	65	58	62	65	64	65	63	64	65	61	63
20	**10**	**2**	91	77	65	64	88	71	69	70	84	75	67	70
		3	96	85	78	78	93	86	83	83	94	81	79	85
		4	87	74	63	64	71	69	71	67	73	70	67	65
	15	**2**	84	66	45	49	74	55	71	60	67	63	60	53
		3	95	75	66	61	81	73	85	69	72	77	70	65
		4	89	74	64	62	84	73	80	75	80	75	62	67
30	**10**	**2**	178	152	139	138	146	143	183	154	168	147	141	141
		3	280	254	232	229	256	240	254	243	282	238	234	237
		4	243	226	206	205	238	203	257	211	223	211	205	208
	15	**2**	143	125	109	109	116	109	140	118	139	118	109	113
		3	188	158	129	132	142	135	192	159	161	149	148	135
		4	188	155	125	132	176	147	170	152	179	148	138	132

Tableau 4.17 : Résultats des heuristiques pour les grandes instances (L=2)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
50	**10**	**2**	307	271	238	236	264	254	296	268	285	266	249	238
		3	312	277	236	233	257	262	284	270	281	258	239	241
		4	288	258	230	230	262	240	257	253	291	264	254	232
	15	**2**	269	236	221	214	236	217	243	225	268	243	232	210
		3	272	236	221	214	235	214	251	237	248	228	228	227
		4	292	236	221	214	233	233	271	239	247	227	227	213
75	**10**	**2**	428	392	372	371	375	384	389	398	410	393	374	364
		3	424	393	372	371	383	379	397	402	385	377	370	365
		4	417	386	361	360	366	364	409	373	378	378	362	363
	15	**2**	336	255	238	234	292	247	278	259	299	251	237	230
		3	336	271	256	252	270	262	303	273	283	267	259	252
		4	362	312	287	285	331	305	343	307	316	298	306	286
100	**10**	**2**	564	499	478	475	490	495	498	500	524	490	497	477
		3	580	530	516	515	518	526	537	533	582	533	526	511
		4	589	530	516	515	545	536	532	527	576	533	520	510
	15	**2**	437	376	361	360	363	371	395	378	407	376	364	356
		3	450	381	365	363	371	381	390	386	411	380	378	360
		4	463	384	365	363	405	384	396	378	394	389	369	358

Les tableaux (4.15 - 4.17) présentent les résultats concernant les indicateurs de performances obtenus pour des problèmes de tailles différentes quand les heuristiques sont appliquées dans un premier temps sur le premier cycle et dans un second temps lorsque nous rajoutons les cycles qui restent, l'un après l'autre. À travers ces tableaux, nous pouvons remarquer que pour les problèmes de petites tailles, les résultats sont presque similaires et

qu'il n'y a pas une grande différence quand nous appliquons l'ensemble des heuristiques pour le premier cycle ou lorsque nous rajoutons le deuxième. En outre, les résultats obtenus par les heuristiques qui ont un rapport avec le temps de traitement démontrent que lorsque ces approches sont appliquées sur tous les cycles, elles donnent de meilleurs résultats que lorsqu'elles sont effectuées que sur le premier, l'amélioration peut être d'autant plus remarquée par le LRC. Ce fait est constaté, que ce soit pour les problèmes de moyennes tailles et pour les grandes tailles.

De plus, lorsque nous testons les heuristiques qui ont un rapport avec la consommation des ressources sur les problèmes de moyennes tailles. Nous pouvons remarquer, à travers les valeurs qui se trouvent dans le tableau qu'il y a une différence significative dans les résultats obtenus, et que ces derniers sont meilleurs en les appliquant pour les deux cycles (puisque le nombre de cycles est égal à 2, L=2), cette amélioration concerne non seulement le SRC mais aussi le LRC. Aussi, nous pouvons constater qu'en appliquant les heuristiques S-PT/RC ainsi que L-PT/RC sur les problèmes de grandes tailles, il y a un écart important qui s'est creusé, et que les résultats obtenus sont meilleurs lorsque ces règles ordonnancent les tâches de chaque cycle. Ce fait peut être aussi remarqué pour les problèmes de moyennes tailles, mais avec un écart moins significatif.

À travers les conclusions qui viennent d'être faites à partir des tableaux (4.15 - 4.17), qui concerne le nombre de cycles L=2, nous pouvons constater que lorsque nous avons appliqué les heuristiques sur tous les cycles, cela nous a donné de meilleurs résultats qu'en effectuant qu'un seul et ceci dans presque la majorité des cas.

4.5.3.2 Résultats des comparaisons entre les différentes heuristiques lorsque L=3

Tableau 4.18 : Résultats des heuristiques pour les petites instances (L=3)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
4	**2**	**2**	25	25	25	25	24	23	25	26	28	25	23	23
		3	15	14	14	14	13	16	15	14	14	16	14	16
		4	20	18	17	17	17	18	18	17	19	18	17	18
	3	**2**	15	13	11	11	13	15	11	13	13	13	12	16
		3	19	16	15	15	19	16	19	16	16	16	15	16
		4	20	19	19	18	19	19	20	18	20	19	18	18
6	**2**	**2**	36	36	34	33	35	36	36	36	39	37	34	33
		3	27	22	21	21	23	23	21	21	19	21	21	21
		4	35	32	31	30	32	31	30	31	34	31	31	31
	3	**2**	30	27	24	25	23	26	28	28	29	27	24	26
		3	26	25	23	23	22	24	27	25	25	25	25	22
		4	26	26	24	23	29	25	25	24	26	25	24	25
8	**2**	**2**	60	55	54	53	56	54	53	55	57	54	55	54
		3	54	54	55	53	62	58	55	54	57	56	58	57
		4	39	39	40	41	44	42	39	41	37	38	40	40
	3	**2**	34	30	31	31	35	31	41	33	35	31	31	31
		3	37	35	35	34	38	35	39	35	40	34	33	34
		4	31	34	32	32	35	33	36	32	32	32	33	31

Tableau 4.19 : Résultats des heuristiques pour les moyennes instances (L=3)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
10	**10**	**2**	45	40	42	35	45	39	43	42	51	37	38	39
		3	42	38	36	35	42	36	39	37	39	36	36	35
		4	29	26	23	23	32	25	28	25	28	27	22	24
	15	**2**	50	44	47	40	51	45	51	41	47	43	43	43
		3	28	31	28	30	30	30	33	27	30	30	30	28
		4	72	69	62	77	71	73	72	69	73	69	67	74
20	**10**	**2**	111	96	119	86	107	87	119	90	96	92	85	88
		3	115	108	105	100	109	103	113	103	126	107	112	104
		4	101	90	84	83	93	86	96	85	102	86	84	80
	15	**2**	102	77	62	66	80	67	87	79	101	73	74	69
		3	112	92	79	77	110	86	99	81	94	87	88	80
		4	99	89	81	78	96	89	101	90	101	87	81	82
30	**10**	**2**	212	196	182	183	194	189	230	195	204	193	193	188
		3	330	324	303	300	345	304	312	316	328	312	302	307
		4	301	283	270	267	303	269	342	276	350	269	267	273
	15	**2**	175	156	146	142	162	146	199	150	166	154	148	147
		3	234	199	176	172	184	181	221	192	199	182	176	174
		4	234	197	210	171	207	182	220	184	252	186	172	176

Tableau 4.20 : Résultats des heuristiques pour les grandes instances (L=3)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*S-PT/RC*		*L-PT/RC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
50	**10**	**2**	379	339	314	309	338	333	372	328	386	332	322	315
		3	385	345	316	310	347	341	353	343	364	344	334	322
		4	352	322	299	294	371	318	342	314	341	324	299	300
	15	**2**	328	280	270	265	279	274	316	274	338	273	282	260
		3	336	280	270	265	296	271	322	279	348	271	268	272
		4	348	280	270	265	313	275	345	278	339	266	287	270
75	**10**	**2**	537	505	490	487	490	498	507	516	516	499	485	488
		3	535	507	490	487	507	493	508	513	527	508	494	495
		4	530	497	479	475	473	476	519	492	537	488	477	480
	15	**2**	432	327	314	311	387	320	375	337	359	329	313	316
		3	443	345	332	329	399	348	372	353	398	341	351	332
		4	453	385	366	363	401	372	445	382	435	379	376	373
100	**10**	**2**	717	655	642	635	647	642	676	661	689	654	683	636
		3	742	697	682	678	677	687	695	683	722	697	686	671
		4	742	698	682	678	749	681	696	694	733	689	685	677
	15	**2**	542	481	469	466	477	468	519	490	518	483	475	458
		3	557	492	477	472	501	483	493	478	529	491	480	467
		4	566	492	477	472	520	480	519	489	531	487	481	470

Nous pouvons constater, à partir des valeurs se trouvant dans le tableau 4.18, que quand nous appliquons les différentes approches, il n'y a pas un écart significatif entre ces valeurs et que les résultats obtenus sont presque les mêmes que ce soit lorsque nous appliquons ces heuristiques que pour le premier cycle ou lorsque nous les appliquons à tous les cycles, mais cela s'observe juste pour les problèmes de petites tailles.

Pour ce qui est des problèmes de moyennes tailles (tableau 4.19), nous pouvons remarquer qu'en appliquant les heuristiques qui ont un rapport avec la consommation des ressources, les résultats obtenus s'améliorent en effectuant plusieurs cycles au lieu d'un seul, et cette amélioration s'accroit davantage lorsqu'il s'agit de l'heuristique LRC.

Quant aux approches qui concernent le temps de traitement, nous pouvons remarquer, à partir des résultats obtenus, qu'en les appliquant pour les trois cycles cela donnent de meilleurs résultats que lorsque nous les utilisons uniquement pour le premier, et ceci pour les problèmes respectivement de moyennes et grandes tailles (tableau 4.20). Ce fait se confirme d'autant plus pour l'heuristique SPT.

En ce qui concerne les heuristiques S-PT/RC ainsi L-PT/RC, les résultats obtenus démontrent qu'il y a une différence significative et une amélioration importante en les appliquant pour tous les cycles, sur les problèmes de grandes tailles. Cette différence existe aussi, mais pas tout aussi importante quand nous appliquons ces approches sur les problèmes de moyennes tailles.

Nous pouvons conclure, à partir des faits qui viennent d'être constatés, que le que se soit pour le nombre de cycles L=2 ou L=3, le raisonnement est le même à savoir qu'en appliquant les différentes approches sur tous les cycles, l'un après l'autre, ces dernières donnent de meilleurs résultats que lorsque nous les appliquons que sur le premier cycle, sauf pour les problèmes de petites tailles où il n'y a pas une grande différence dans les résultats.

4.5.3.3 Résultats des comparaisons entre les différentes heuristiques lorsque L=4

Tableau 4.21 : Résultats des heuristiques pour les petites instances (L=4)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*SRC*		*LRC*	
n	**m**	**k**	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
4	**2**	**2**	33	32	30	30	28	29	34	32	33	32	28	29
		3	22	19	16	16	18	20	17	17	19	20	18	20
		4	21	22	22	20	24	22	23	21	21	22	21	22
	3	**2**	16	16	15	12	15	19	14	16	15	16	15	20
		3	21	20	18	17	21	20	21	18	20	20	19	20
		4	23	23	21	21	22	23	25	23	25	23	22	21
6	**2**	**2**	46	44	43	44	46	47	43	44	39	44	43	46
		3	22	30	29	26	24	29	29	26	23	28	28	29
		4	41	41	46	40	44	42	42	44	39	41	40	44
	3	**2**	39	32	34	32	42	31	31	34	31	32	31	34
		3	29	29	31	30	33	30	32	32	31	31	29	31
		4	29	31	30	29	33	30	31	32	33	31	31	29
8	**2**	**2**	71	71	68	67	70	68	73	70	78	71	68	69
		3	74	71	70	68	76	70	74	74	70	72	69	73
		4	54	52	50	49	52	52	52	49	46	50	51	48
	3	**2**	42	40	39	38	44	41	45	39	44	40	37	35
		3	50	46	49	41	45	43	54	46	49	44	44	44
		4	37	43	39	39	43	41	40	39	42	42	42	39

Tableau 4.22 : Résultats des heuristiques pour les moyennes instances (L=4)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*SRC*		*LRC*	
n	m	k	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
10	10	2	57	48	45	43	52	45	48	49	52	47	46	46
		3	48	44	43	40	55	41	50	42	47	49	44	40
		4	36	31	29	29	39	29	32	29	30	31	30	30
	15	2	54	53	48	44	57	53	57	45	54	53	51	50
		3	35	34	35	34	43	34	38	34	36	36	34	34
		4	76	75	68	87	84	77	81	75	75	73	75	78
20	10	2	138	119	138	105	128	110	114	110	150	120	104	107
		3	134	130	141	120	138	129	138	124	146	128	123	126
		4	116	108	102	100	108	102	114	101	119	108	107	99
	15	2	114	97	93	77	112	81	112	91	111	85	96	79
		3	120	102	95	92	115	101	110	106	114	99	95	95
		4	122	104	103	91	116	105	122	104	106	99	92	96
30	10	2	272	237	266	225	244	228	294	239	259	240	238	227
		3	389	376	366	359	416	370	396	373	435	373	368	365
		4	368	333	324	325	365	320	353	337	345	326	346	329
	15	2	220	185	211	173	195	171	216	182	207	182	180	175
		3	248	227	249	209	238	223	284	216	246	223	216	217
		4	248	227	241	207	270	213	273	225	285	223	210	210

Tableau 4.23 : Résultats des heuristiques pour les grandes instances (L=4)

Problèmes			*SPT*		*LPT*		*SRC*		*LRC*		*SRC*		*LRC*	
n	m	k	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$	$C_{max}H_1$	$C_{max}H_L$
50	10	2	433	397	417	374	391	392	446	405	402	402	381	376
		3	458	406	415	374	417	396	431	402	433	402	400	384
		4	415	371	357	348	395	366	398	366	416	371	356	359
	15	2	377	322	319	308	364	311	357	328	425	320	348	310
		3	395	322	319	308	355	311	395	325	391	315	318	319
		4	393	322	319	308	367	323	375	322	413	317	316	310
75	10	2	628	596	584	581	581	585	616	595	642	592	580	572
		3	651	596	584	581	584	580	596	597	643	590	589	581
		4	628	587	575	570	574	585	614	571	622	586	574	569
	15	2	497	389	381	374	450	380	454	384	463	382	375	369
		3	499	405	399	392	444	404	444	400	474	397	412	388
		4	514	444	431	426	496	428	504	440	513	432	470	424
100	10	2	873	789	778	776	794	789	793	806	863	793	786	774
		3	893	830	819	817	814	832	825	834	915	837	816	813
		4	890	830	819	817	895	827	828	822	895	836	817	807
	15	2	625	572	562	554	561	568	603	576	660	563	553	551
		3	657	576	566	565	579	579	585	582	636	579	568	560
		4	654	576	566	565	657	577	613	576	616	580	566	557

Nous pouvons remarquer, que le makespan obtenu dans le tableau 4.21 pour les problèmes de petites tailles est presque similaire que ce soit pour un seul cycle ou pour tous les cycles et ceci pour l'ensemble des heuristiques utilisées. En outre, nous pouvons constater, à travers les résultats obtenus dans les tableaux (4.22 - 4.23) qu'en adaptant les approches qui concernent le temps de traitement, nous obtenons de meilleurs résultats lorsque ces

heuristiques sont adaptées à chaque cycle par rapport à lorsque nous effectuons que sur le premier. Une amélioration significative est obtenue pour le SPT, que ce soit pour les problèmes de moyennes ou grandes tailles. De plus, nous observons, qu'il commence à y avoir un écart dans les résultats obtenus en appliquant les heuristiques qui sont en rapport avec la consommation des ressources, et que ces résultats sont en amélioration lorsque nous les exécutons pour les quatre cycles, mais cela s'applique pour les problèmes de moyennes tailles. Aussi, une amélioration s'observe pour les problèmes de grandes tailles mais en appliquant cette fois-ci les deux heuristiques qui sont respectivement S-PT/RC ainsi L-PT/RC, et en les effectuant sur tous cycles surtout lorsqu'il s'agit du L-PT/RC. Ceci est de même pour les problèmes de moyenne tailles, mais avec une amélioration moins avantageuse.

- **Synthèse**

À partir des différentes conclusions qui ont été énoncées, nous pouvons déduire à travers l'ensemble des résultats obtenus, que ces derniers sont en amélioration significative que ce soit pour le nombre de cycle L=2, L=3 et L=4, et ceci lorsque nous les appliquons sur tous les cycles au lieu d'un seul, mis-à-part pour les problèmes de petites tailles ou les résultats sont presque identiques.

4.5.4 Amélioration des performances de l'AG par recherche locale

Cette section est réservée à la présentation des différents résultats obtenus par l'AG et les recherches locales. Tout d'abord, nous présentons les résultats des tests de l'AG qui sont effectués sans utiliser la procédure de recherche locale ensuite nous conduisons une étude comparative de l'algorithme génétique avec recherche locale (AG+RL) afin d'analyser l'impact de l'ajout des recherches locales DM, API et NAPI (décrites dans la section 4.3.2). Dans ce contexte, les tableaux (4.24 - 4.26) présentent les performances moyennes de l'AG sans et avec procédure de recherche locale. Les mesures de performances utilisées pour effectuer cette étude sont :

- CPU_T : le temps de calcul moyen
- $AdP = (C_{max}^{AG} - C_{max}^{AG+RL})/C_{max}^{AG+RLt}$,
  - ✓ C_{max}^{AG} : est le makespan obtenu par l'AG.
  - ✓ C_{max}^{AG+RL} : est le makespan obtenu par l'AG lorsqu'il est hybridé avec une méthode de recherche locale.

En d'autre terme, l'AdP représente l'Amélioration des Performances conduite par les recherches locales à l'AG. Notons que l'AG est évalué que par un seul critère qui est le CPU_T (car son AdP est toujours égal à zéro). Finalement, pour achever cette étude, nous utilisons les notations suivantes :

- AG+DM : Algorithme génétique combiné avec la recherche locale DM
- AG+API : Algorithme génétique combiné avec la recherche locale API
- AG+NAPI : Algorithme génétique combiné avec la recherche locale NAPI

Tableau 4.24 Résultats des comparaisons entre les différentes recherches locales (L=2)

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
4	**2**	**2**	0.26	0.26	0.000	0.27	0.000	0.27	0.000
		3	0.28	0.28	0.000	0.27	0.000	0.27	0.000
		4	0.29	0.30	0.000	0.29	0.000	0.29	0.000
	3	**2**	0.26	0.26	0.000	0.26	0.000	0.26	0.000
		3	0.28	0.28	0.000	0.29	0.000	0.28	0.000
		4	0.30	0.30	0.000	0.30	0.000	0.30	0.000
6	**2**	**2**	0.29	0.30	0.000	0.30	0.000	0.30	0.000
		3	0.31	0.32	0.000	0.32	0.000	0.32	0.000
		4	0.34	0.35	0.000	0.35	0.000	0.34	0.000
	3	**2**	0.30	0.31	0.000	0.31	0.000	0.32	0.000
		3	0.33	0.33	0.000	0.33	0.000	0.33	0.000
		4	0.35	0.36	0.000	0.36	0.000	0.35	0.000
8	**2**	**2**	0.34	0.34	0.000	0.34	0.000	0.34	0.000
		3	0.36	0.37	0.000	0.37	0.000	0.36	0.000
		4	0.39	0.40	0.000	0.40	0.000	0.40	0.000
	3	**2**	0.34	0.35	0.000	0.35	0.000	0.35	0.000
		3	0.37	0.38	0.000	0.38	0.000	0.38	0.000
		4	0.40	0.41	0.000	0.41	0.000	0.41	0.000

Petites instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
10	**6**	**2**	0.45	0.45	0.011	0.46	0.000	0.47	0.007
		3	0.47	0.48	0.000	0.48	0.004	0.50	0.007
		4	0.54	0.55	0.003	0.53	0.000	0.52	0.007
	8	**2**	0.47	0.48	0.000	0.47	0.000	0.46	0.000
		3	0.61	0.62	0.000	0.47	0.000	0.47	0.000
		4	0.58	0.59	0.000	0.52	0.000	0.53	0.000
20	**6**	**2**	1.03	1.04	0.007	0.80	0.003	0.80	0.012
		3	1.08	1.09	0.001	0.82	0.001	0.87	0.002
		4	1.36	1.37	0.003	0.93	0.006	0.88	0.000
	8	**2**	1.28	1.30	0.012	0.89	0.002	0.82	0.012
		3	0.88	0.90	0.001	0.85	0.001	0.86	0.000
		4	0.98	0.99	0.005	0.99	0.005	0.98	0.006
30	**6**	**2**	1.17	1.18	0.003	1.10	0.000	1.18	0.002
		3	1.32	1.34	0.000	1.34	0.001	1.30	0.003
		4	1.45	1.47	0.001	1.34	0.001	1.35	0.000
	8	**2**	1.16	1.17	0.000	1.27	0.000	1.24	0.005
		3	1.33	1.35	0.000	1.42	0.001	1.33	0.003
		4	1.57	1.59	0.000	1.52	0.004	1.51	0.010

Moyennes instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
50	**10**	**2**	2.38	2.39	0.000	2.37	0.001	2.23	0.002
		3	2.50	2.52	0.003	2.50	0.001	2.41	0.004
		4	2.81	2.84	0.001	2.81	0.002	2.76	0.001
	15	**2**	2.70	2.72	0.012	2.54	0.000	2.47	0.009
		3	2.62	2.65	0.005	2.84	0.005	2.73	0.003
		4	2.74	2.77	0.000	3.00	0.003	2.92	0.003
75	**10**	**2**	4.07	4.10	0.002	4.16	0.001	4.06	0.004
		3	4.10	4.13	0.003	4.23	0.000	4.31	0.000
		4	4.80	4.84	0.000	4.65	0.000	4.56	0.001
	15	**2**	4.17	4.20	0.001	4.19	0.002	4.09	0.000
		3	4.24	4.27	0.002	4.70	0.001	4.23	0.000
		4	5.03	5.07	0.004	4.98	0.001	4.90	0.001
100	**10**	**2**	6.01	6.05	0.000	6.17	0.001	6.31	0.003
		3	6.69	6.74	0.005	7.12	0.005	7.13	0.000
		4	7.02	7.08	0.005	7.13	0.000	6.95	0.004
	15	**2**	6.29	6.32	0.000	6.45	0.002	6.47	0.004
		3	7.01	7.06	0.000	6.97	0.003	7.12	0.002
		4	6.82	6.87	0.000	7.85	0.001	7.22	0.006

Grandes instances

À partir des valeurs qui sont dans le tableau (4.24), nous pouvons remarquer que l'ajout des trois méthodes de recherche locale à l'algorithme génétique n'améliore pas les résultats obtenus. Mais ceci s'applique juste pour les problèmes de petites tailles.

En effet, pour les problèmes de moyennes tailles et de grandes tailles, nous constatons qu'il commence à y avoir une légère amélioration dans les résultats en rajoutant, à l'algorithme génétique les deux approches qui sont respectivement DM et API, mais cette amélioration s'accroit davantage et devient significative en utilisant la méthode NAPI, en atteignant une amélioration de performance de 1.2 % pour l'instance (20, 8, 2).

En ce qui concerne le temps de calcul, nous pouvons constater, que pour les problèmes de petites tailles, il reste similaire et très faible, que ce soit en appliquant l'AG seul, ou en rajoutant les trois techniques de recherche locale.

Pour ce qui est des problèmes de moyennes tailles, nous pouvons remarquer, que le CPU_T augmente légèrement par rapport aux problèmes de petites tailles, mais il est presque aussi le même en utilisant l'AG seul ou en lui rajoutant les trois autres techniques.

Quant aux problèmes de grandes tailles, nous pouvons observer que le CPU_T a augmenté que ce soit par l'AG ou en appliquant les autres techniques (AG+RL), et particulièrement lorsque nous avons combiné l'AG avec les deux approches de recherche locale API et NAPI.

Tableau 4.25 Résultats des comparaisons entre les différentes recherches locales (L=3)

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	m	k	CPU_T	CPU_T	AdP	CPU_T	AdP	CPU_T	AdP
4	2	2	0.28	0.29	0.000	0.29	0.000	0.29	0.000
		3	0.32	0.32	0.000	0.32	0.000	0.33	0.000
		4	0.33	0.34	0.000	0.34	0.000	0.33	0.000
	3	2	0.30	0.30	0.000	0.30	0.000	0.30	0.000
		3	0.32	0.32	0.000	0.33	0.000	0.32	0.000
		4	0.34	0.35	0.000	0.35	0.000	0.35	0.000
6	2	2	0.34	0.35	0.000	0.35	0.000	0.35	0.000
		3	0.37	0.37	0.000	0.38	0.000	0.37	0.000
		4	0.40	0.41	0.000	0.41	0.000	0.40	0.000
	3	2	0.35	0.36	0.000	0.36	0.000	0.35	0.000
		3	0.39	0.40	0.000	0.40	0.000	0.39	0.000
		4	0.41	0.42	0.000	0.43	0.000	0.42	0.000
8	2	2	0.40	0.40	0.000	0.40	0.000	0.40	0.000
		3	0.43	0.43	0.000	0.44	0.000	0.43	0.000
		4	0.47	0.47	0.000	0.48	0.000	0.47	0.000
	3	2	0.41	0.41	0.000	0.41	0.000	0.41	0.000
		3	0.44	0.45	0.000	0.47	0.000	0.46	0.000
		4	0.48	0.49	0.000	0.50	0.000	0.49	0.000

Petites instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	m	k	CPU_T	CPU_T	AdP	CPU_T	AdP	CPU_T	AdP
10	6	2	0.55	0.55	0.003	0.57	0.000	0.56	0.005
		3	0.61	0.62	0.000	0.59	0.003	0.60	0.003
		4	0.71	0.72	0.000	0.63	0.000	0.62	0.000
	8	2	0.60	0.61	0.000	0.56	0.000	0.56	0.000
		3	0.70	0.70	0.000	0.62	0.000	0.59	0.000
		4	0.76	0.77	0.000	0.67	0.000	0.68	0.000
20	6	2	1.26	1.28	0.000	0.94	0.002	0.99	0.006
		3	1.57	1.59	0.006	1.07	0.004	1.07	0.000
		4	1.96	2.0	0.000	1.16	0.006	1.17	0.000
	8	2	1.64	1.66	0.000	1.07	0.008	1.03	0.009
		3	1.12	1.14	0.005	1.11	0.000	1.13	0.002
		4	1.25	1.27	0.002	1.30	0.003	1.21	0.000
30	6	2	1.43	1.45	0.005	1.46	0.001	1.46	0.006
		3	1.70	1.72	0.000	1.74	0.001	1.71	0.002
		4	1.75	1.78	0.002	2.04	0.004	1.88	0.001
	8	2	1.53	1.55	0.001	1.56	0.000	1.57	0.006
		3	1.84	1.86	0.005	1.75	0.004	1.70	0.000
		4	1.81	1.84	0.002	2.14	0.006	1.97	0.000

Moyennes instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
50	**10**	**2**	2.87	2.90	0.003	2.80	0.001	3.13	0.000
		3	3.02	3.05	0.004	3.06	0.002	3.37	0.000
		4	3.32	3.36	0.002	3.69	0.003	3.24	0.000
	15	**2**	3.33	3.36	0.002	3.23	0.002	3.17	0.002
		3	3.37	3.40	0.003	3.63	0.006	3.62	0.000
		4	3.51	3.55	0.002	3.73	0.008	3.57	0.000
75	**10**	**2**	4.99	5.04	0.002	5.22	0.000	4.76	0.001
		3	5.18	5.23	0.001	5.23	0.000	5.42	0.003
		4	5.34	5.40	0.003	6.34	0.001	6.06	0.000
	15	**2**	4.87	4.91	0.001	5.54	0.002	5.26	0.000
		3	5.38	5.44	0.000	5.94	0.002	5.73	0.002
		4	6.39	6.45	0.005	6.12	0.003	6.59	0.004
100	**10**	**2**	7.41	7.46	0.001	6.93	0.000	6.95	0.001
		3	8.76	8.83	0.001	8.37	0.004	9.53	0.000
		4	8.26	8.34	0.002	10.37	0.004	8.61	0.001
	15	**2**	8.51	8.57	0.003	7.97	0.002	8.25	0.000
		3	7.97	8.04	0.004	8.24	0.000	8.74	0.004
		4	8.59	8.67	0.000	8.92	0.005	9.08	0.003

Grandes instances

Nous pouvons remarquer, à travers les trois tableaux ci-dessus que le temps de traitement est relativement faible, ne dépassant pas les deux secondes pour les problèmes de petites et moyennes tailles. Par contre, il atteint les neuf secondes pour les grandes tailles.

En ce qui concerne les approches de recherche locale utilisées, nous pouvons constater que lorsqu'il s'agit des problèmes de petites tailles, l'ajout de ces techniques à l'algorithme génétique n'améliore pas les résultats. Par contre, pour les problèmes de moyennes et grandes tailles, une amélioration s'est observée et plus particulièrement, elle augmente et devient significative lorsque nous avons utilisons l'approche de recherche locale NAPI avec une performance qui a atteint les 0.9 %.

En ce qui concerne le temps de calcul, nous pouvons remarquer, dans le tableau des petites instances, qu'en employant l'AG seul ou avec les autres approches, le temps reste faible est similaire et ceci pour les problèmes de petites tailles. En outre, pour les problèmes de moyennes tailles, nous pouvons constater qu'il a légèrement augmenté lorsque nous avons appliqué l'AG ou lorsque nous lui avons ajouté les recherches locales DM, API ou NAPI. À travers le tableau 4.25 (moyennes instances), nous pouvons remarquer, que pour les problèmes de grandes tailles, le temps de calcul commence à devenir plus important, surtout en ajoutant les techniques NAPI et API atteignant, pour cette dernière, 10 secondes pour l'instance (100 tâches, 10 machines et 3 composants).

Nous pouvons remarquer aussi que le temps a aussi augmenté par rapport au cycle précédent lorsqu'il s'agit des problèmes de grandes tailles.

Tableau 4.26 Résultats des comparaisons entre les différentes recherches locales (L=4)

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
4	**2**	**2**	0.32	0.32	0.000	0.32	0.000	0.32	0.000
		3	0.34	0.35	0.000	0.35	0.000	0.40	0.000
		4	0.37	0.38	0.000	0.38	0.000	0.38	0.000
	3	**2**	0.32	0.33	0.000	0.33	0.000	0.33	0.000
		3	0.36	0.36	0.000	0.37	0.000	0.36	0.000
		4	0.39	0.36	0.000	0.40	0.000	0.39	0.000
6	**2**	**2**	0.38	0.39	0.000	0.40	0.000	0.40	0.000
		3	0.43	0.44	0.000	0.43	0.000	0.42	0.000
		4	0.46	0.47	0.000	0.47	0.000	0.47	0.000
	3	**2**	0.41	0.41	0.006	0.41	0.009	0.43	0.012
		3	0.49	0.50	0.017	0.54	0.013	0.51	0.013
		4	0.47	0.48	0.000	0.51	0.000	0.49	0.000
8	**2**	**2**	0.46	0.46	0.000	0.46	0.000	0.46	0.000
		3	0.50	0.51	0.000	0.51	0.000	0.51	0.000
		4	0.54	0.55	0.000	0.55	0.000	0.56	0.000
	3	**2**	0.48	0.49	0.000	0.49	0.000	0.49	0.000
		3	0.51	0.53	0.000	0.54	0.000	0.53	0.000
		4	0.58	0.59	0.000	0.65	0.000	0.57	0.000

Petites instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
10	**6**	**2**	0.64	0.65	0.002	0.68	0.000	0.69	0.008
		3	0.71	0.72	0.002	0.75	0.004	0.69	0.000
		4	0.88	0.89	0.000	0.84	0.000	0.78	0.002
	8	**2**	0.76	0.77	0.000	0.68	0.000	0.70	0.000
		3	0.88	0.89	0.012	0.73	0.000	0.72	0.003
		4	0.92	0.94	0.000	0.83	0.000	0.80	0.000
20	**6**	**2**	1.57	1.59	0.005	1.18	0.000	1.19	0.003
		3	2.19	2.22	0.002	1.34	0.000	1.32	0.000
		4	3.26	3.31	0.002	1.41	0.000	1.44	0.002
	8	**2**	1.45	1.47	0.007	1.26	0.004	1.32	0.006
		3	1.35	1.38	0.004	1.43	0.002	1.34	0.001
		4	1.52	1.54	0.004	1.51	0.000	1.48	0.004
30	**6**	**2**	1.71	1.73	0.002	1.73	0.000	1.96	0.002
		3	2.21	2.24	0.001	2.06	0.002	2.29	0.001
		4	2.18	2.22	0.000	2.18	0.000	2.43	0.000
	8	**2**	1.92	1.95	0.002	1.82	0.000	1.96	0.002
		3	2.08	2.11	0.003	2.06	0.002	1.96	0.000
		4	2.43	2.46	0.009	2.59	0.000	2.41	0.005

Moyennes instances

Problèmes			*AG*	*AG+DM*		*AG+API*		*AG+NAPI*	
n	**m**	**k**	**CPU_T**	**CPU_T**	**AdP**	**CPU_T**	**AdP**	**CPU_T**	**AdP_1**
50	**10**	**2**	3.30	3.34	0.000	3.90	0.002	3.58	0.002
		3	3.91	3.96	0.000	3.79	0.002	3.73	0.002
		4	3.91	3.96	0.002	4.77	0.000	4.21	0.000
	15	**2**	3.76	3.80	0.002	3.92	0.003	3.94	0.001
		3	4.37	4.41	0.004	4.37	0.001	4.70	0.002
		4	4.49	4.55	0.000	4.72	0.005	4.69	0.000
75	**10**	**2**	5.10	5.15	0.000	5.77	0.001	5.65	0.001
		3	5.92	5.99	0.001	6.36	0.000	6.53	0.001
		4	6.14	6.21	0.000	6.70	0.000	7.57	0.002
	15	**2**	6.10	6.15	0.002	6.77	0.001	6.41	0.000
		3	6.52	6.58	0.002	6.79	0.002	6.43	0.000
		4	8.29	8.36	0.000	7.60	0.002	7.98	0.002
100	**10**	**2**	7.81	7.90	0.002	8.01	0.000	8.07	0.001
		3	8.70	8.78	0.001	9.97	0.000	9.44	0.002
		4	9.60	9.71	0.000	10.44	0.001	10.71	0.002
	15	**2**	9.11	9.19	0.002	9.36	0.001	9.26	0.001
		3	9.47	9.57	0.001	10.22	0.002	10.90	0.002
		4	9.98	10.08	0.002	11.55	0.002	12.09	0.002

Grandes instances

Nous pouvons constater, que pour les approches de recherches locales, nous pouvons remarquer qu'en ajoutant ces méthodes à l'AG, ces dernières n'améliorent pas les résultats, et ceci juste pour les problèmes de petites tailles. Mais pour ce qui est des problèmes de moyennes et grandes tailles, les résultats s'améliorent en utilisant les trois techniques surtout lorsque nous ajoutons à l'AG l'approche de recherche locale NAPI.

Les résultats obtenus, dans le tableau (4.26), relatifs au temps de traitement montrent que ce dernier est le même et qu'il reste très faible pour l'ensemble des techniques, et cela pour les problèmes de petites tailles, cependant il commence à augmenter légèrement lorsqu'il s'agit des problèmes de moyenne tailles avec un temps de calcul de 2 secondes mais c'est surtout pour les problèmes de grandes tailles qu'il augmente considérablement atteignant les 12 secondes. Quant aux problèmes de grandes tailles, nous pouvons constater que le temps de calcul est devenu plus important non seulement par rapport aux problèmes de petites et moyennes tailles puisqu'il atteint les 12 secondes lorsque nous avons ajouté à l'AG l'approche NAPI, mais aussi, il a augmenté par rapport au nombre de cycles respectivement L=2 et L=3.

À travers les résultats obtenus dans les tableaux (4.24-4.26) nous ne pouvons pas affirmer quelle est la meilleure méthode de recherche locale (DM, API ou NAPI) qu'il faudrait utiliser et à ajouter pour améliorer les résultats de l'AG, pour cela, il serait judicieux de calculer la moyenne des différents résultats relatifs au CPU_T et AdP qui ont été obtenus afin de déterminer quelle serait la meilleure à ajouter, nous appliquons ce principe pour l'ensemble des cycles. Les moyennes obtenues de chacune des trois méthodes sont résumées dans le tableau (4.27).

Notons que l'AdPM représente l'Amélioration des Performances Moyenne en pourcentage, conduite par les recherches locales à l'AG, elle est calculée de la manière suivante :

$$AdPM = [(C_{max}^{AG} - C_{max}^{AG+RL})/C_{max}^{AG+RL}] * 100$$

Tableau 4.27 Performances moyennes des différentes recherches locales

Nombre de cycles	AG	AG+DM		AG+API		AG+NAPI	
	CPU_T	CPU_T	AdPM	CPU_T	AdPM	CPU_T	AdPM
L=2	1.95	1.97	0.166	1.97	0.106	1.94	0.226
L=3	2.41	2.44	0.130	2.46	0.160	2.45	0.113
L=4	2.83	2.86	0.190	2.93	0.113	2.94	0.163
Moyenne	**2.39 s**	**2.42 s**	**0.162%**	**2.453 s**	**0.126%**	**2.443 s**	**0.167%**

À partir du tableau 4.27, nous pouvons remarquer, que la moyenne du temps de traitement reste la même ne dépassant pas les trois secondes que ce soit en appliquant l'AG seul ou en lui rajoutant les trois approches de recherche locale, et ceci pour l'ensemble des cycles. Mais en ce qui concerne les performances moyennes de l'AG avec procédure de recherche locale, nous pouvons remarquer que les résultats commencent à s'améliorer lorsque

nous avons utilisé la recherche locale API avec une augmentation égale à 0.126 %, et que cette amélioration est devenue significative (par rapport à cette dernière) dés que nous ajoutons les deux approches DM et NAPI à l'AG, surtout de la part de cette dernière, ces techniques ont atteint un pourcentage de performance respectivement de 0.162% et 0.167%.

- **Synthèse**

  À travers toutes les conclusions qui viennent d'être faite, nous pouvons déduire que :

  - ✓ Il y a une amélioration dans les résultats que ce soit en rajoutant l'approche DM, API ou NAPI mais que le système a atteint ses meilleures performances par la recherche locale NAPI qui mène à une amélioration moyenne de 0.167%.
  - ✓ Le temps de calcul reste le même pour les problèmes de petites tailles que ce soit en appliquant l'AG seul ou en lui rajoutant les autres techniques de recherche locale.
  - ✓ Le temps de calcul commence à augmenter et devient important surtout pour les problèmes de grandes tailles en rajoutant à l'AG les deux approches API et NAPI.
  - ✓ La recherche locale DM n'a pas un impact significatif sur le temps de calcul de l'AG.
  - ✓ Nous pouvons aussi déduire que plus le nombre de cycles augmente, plus le temps de traitement s'accroit, cela s'est observé particulièrement pour les problèmes de grandes tailles ou le temps de traitement à nécessité 7 secondes au début et a atteint les 12 secondes à la fin.

4.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'adaptation des algorithmes génétiques pour résoudre le problème de minimisation de makespan dans un environnement de machines parallèles avec processus réentrant tout en prenant en compte les contraintes de ressources qui compliquent le processus décisionnel d'ordonnancement des tâches qui peuvent nécessiter plusieurs composants en même temps de différents types. Nous avons montré l'efficacité de l'algorithme génétique sur un grand nombre de configurations en le comparant à plusieurs heuristiques en termes de qualité de solution. En outre, nous avons conduit une étude concernant l'impact de trois méthodes de recherches locales qui sont respectivement DM, API et NAPI sur les performances de l'algorithme génétique en terme de makespan. L'étude menée dans ce chapitre nous permet de dégager quelques résultats préliminaires qui nous donnent la possibilité d'évaluer le système décisionnel d'ordonnancement utilisant l'algorithme génétique, nous présentons l'essentiel des conclusions :

- À travers les différentes études qui ont été réalisées, nous avons constaté, que plus le nombre de cycles augmente, l'AG prend plus de temps pour effectuer le traitement que ce soit pour les problèmes de petites, de moyennes ou de grandes tailles. Par contre, les heuristiques nécessitent moins de temps de calcul.

- Concernant les heuristiques qui ont un rapport avec le temps de traitement, nous avons remarqué que dans la majorité des cas le LPT donne de meilleurs résultats que le SPT et pour les heuristiques de consommation, c'est le SRC qui surpasse le LRC et finalement pour les heuristiques qui sont en rapport avec les deux, le L-PT/RC

surpasse S-PT/RC. En outre, nous avons pu obtenir une amélioration des résultats lorsque nous avons appliqué les différentes heuristiques sur tous les cycles au lieu d'un seul.

- Nous pouvons constater, qu'en ce qui concerne les nombres de cycles L=2, L=3 ou L=4 et qu'en appliquant les différentes heuristiques sur tous les cycles, les résultats ont été meilleurs par rapport à ceux obtenus lorsque nous les avons utilisé que sur le premier cycle.
- L'approche qui reste la plus performante, c'est l'algorithme génétique, particulièrement lorsque nous avons ajouté les méthodes de recherches locales. En effet, les résultats obtenus ont montré que le rajout de ces procédures que ce soit DM, API ou NAPI a conduit à des améliorations importantes des performances du système, en termes de la minimisation de son temps d'exécution maximal et par conséquent, la maximisation de sa productivité. La meilleure performance a été obtenue par la recherche locale NAPI. Ces améliorations ont été contrebalancées par une augmentation en temps de calcul qui devient important surtout pour les problèmes de grandes tailles en rajoutant à l'AG les deux approches API et NAPI. Ce fait peut être expliqué par la présence d'un nombre important de cycles et par l'existence de la contrainte des ressources consommables, qui compliquent la prise de décision.

Conclusion générale

Ce travail s'intéresse à l'ordonnancement sur machines parallèles avec ressources consommables et processus réentrant dans les systèmes de production.

Tout d'abord, nous avons présenté, dans le premier chapitre, un aperçu sur les systèmes de production, ensuite, nous avons donné des généralités sur l'ordonnancement, par la suite nous nous sommes intéressés aux différents problèmes engendrés par l'ordonnancement qui deviennent de plus en plus fréquents dans les systèmes de production. Après, nous avons brièvement illustré, les approches de résolution utilisées pour résoudre ces problèmes. Enfin, nous avons synthétisé un état de l'art sur les thématiques de recherche considérées. Ce dernier nous a permis d'exposer la problématique d'ordonnancement dans les systèmes de production, et de rassembler les travaux fondamentaux sur: les problèmes d'ordonnancement avec ressources consommables et les problèmes d'ordonnancement réentrant.

Cependant, nous avons constaté, que les approches de résolution doivent être en mesure de fournir des solutions acceptables dans un environnement caractérisé par des circonstances imprévues tout en satisfaisant les contraintes relatives aux ressources (consommables ou renouvelables), aux tâches (préemptives, indépendantes…), aux processus (réentrant ou non)... Donc, l'ordonnancement s'avère être une tâche extrêmement difficile, puisque ces systèmes sont caractérisés par leurs complexités fonctionnelles et l'évolution rapide de leurs états. Par conséquent, les approches d'optimisation traditionnelles sont souvent inefficaces face à une telle situation et incapables de fournir des résultats satisfaisants.

Dans ce contexte et pour faciliter la prise de décision aux gestionnaires et décideurs, il est devenu nécessaire de proposer des stratégies de résolutions qui permettent de prendre à très court terme des décisions d'allocation et d'ordonnancement des tâches et d'établir des plans de processus de production en fonction de la disponibilité des ressources et des machines ainsi que d'autres situations difficiles ou événements imprévus. Cette stratégie d'aide à la décision repose sur les métaheuristiques, ces dernières font partie des algorithmes approximatifs et constituent un moyen efficace pour la résolution des problèmes NP-difficiles. Dans ce sens, le deuxième chapitre a été destiné de manière approfondie aux métaheuristiques. Nous avons commencé par mettre l'accent sur les notions de bases liées aux problèmes d'optimisation. Ensuite, nous avons mis le point sur les principes les plus connus de cette approche en incluant ses différentes définitions, caractéristiques, principaux concepts et classifications. Enfin, nous avons présenté les métaheuristiques les plus répandues qui sont

soit à base de solution unique ou bien à base de population de solutions en rappelant leurs origines, leurs mode de fonctionnement et leurs algorithmes de base.

Dans le troisième chapitre, nous nous somme intéressés à l'adaptation d'un algorithme génétique pour l'ordonnancement dans un environnement de machines parallèles avec ressources consommables. L'objectif est d'affecter dans un premier temps les tâches aux machines et dans un second temps de déterminer la séquence des tâches assignées à la même machine afin de minimiser le makespan tout en respectant les différentes contraintes de ressources. Nous avons tout d'abord commencé par décrire le modèle linéaire en nombres entiers afin de minimiser le makespan. Nous avons montré que le modèle proposé se révèle efficace pour cette mesure de performance néanmoins, il nécessite un temps de calcul assez important pour résoudre les problèmes de moyennes et grandes tailles. Un autre objectif de ce chapitre est l'investigation sur les performances des heuristiques qui sont respectivement en rapport soit avec le temps de traitement (LPT, SPT), soit avec la consommation des composants (LRC, SRC) ou bien les deux (L-PT/RC, S-PT/RC) sur le système étudié. Les résultats obtenus ont montré qu'en ce qui concerne le temps de traitement, c'est le LPT qui a donné de meilleurs résultats que le SPT en raison du fait qu'elle favorise les tâches qui ont un temps de traitement long. Pour ce qui est de la consommation des ressources, c'est la règle SRC qui s'est comportée mieux que la LRC car elle a favorisé les tâches qui ont une faible consommation de composants ce qui a conduit à une amélioration des performances du système, quant aux dernières heuristiques que nous avons utilisé, c'est la règle L-PT/RC qui s'est révélée non seulement plus performante par rapport à la S-PT/RC parce qu'elle permet d'offrir l'avantage aux tâches possédant un temps de traitement important et une consommation relativement faible des ressources mais aussi, cette règle s'est montrée la plus efficace parmi toutes les heuristiques utilisées (LPT, SPT, LRC, SRC et S-PT/RC). Nous avons ensuite présenté, dans ce chapitre, la structure de l'algorithme génétique proposé, en lui définissant les paramètres adéquats à travers l'analyse de sensibilité appliquée. Il s'est avéré performant quant à sa prise de décision concernant l'ordonnancement des tâches dans un environnement de machines parallèles avec ressources consommables. L'algorithme génétique s'est révélé le plus performant parmi toutes les approches utilisées. De plus, nous avons montré que l'interaction entre les heuristiques et la métaheuristique à base des algorithmes génétiques diminuait considérablement les performances du système. Dans ce contexte, la dernière partie de ce chapitre à été consacré à l'application d'une méthode de recherche locale, car cette dernière présente l'avantage de se focaliser et ceci de manière intensive à explorer et exploiter l'espace de recherche en vue d'améliorer les performances de l'algorithme génétique. Les résultats obtenus s'avèrent très prometteurs puisque l'algorithme génétique hybride a surpassé l'algorithme génétique en termes de qualité de solution, cela est dû à l'effet positif de la procédure de recherche locale sur les performances de ce dernier.

Si l'apport du chapitre précédent se limite à l'étude de ressources consommables sur machines parallèles. Dans le quatrième chapitre, nous avons abordé une autre piste à savoir la considération du processus réentrant du système étudié tout en tenant compte de la contrainte des ressources non renouvelables pour pouvoir analyser l'effet de la variation de l'augmentation du nombre de cycles sur les performances du système. Afin de faire face à cette situation, nous avons proposé un algorithme génétique capable de s'intégrer à cet

environnement et de réagir aux divers changements. Les résultats obtenus en intégrant cette approche ont montré que l'AG a pris plus de temps pour effectuer le traitement au fur et à mesure que le nombre de cycles augmenté, que ce soit pour les problèmes de petites, moyennes ou de grandes tailles. Contrairement aux heuristiques qui quant à elles ont nécessité moins de temps. Les performances obtenues de la part de ces heuristiques ont démontré que pour celles qui avaient un rapport avec le temps de traitement, c'est l'heuristique LPT qui a surpassé l'heuristique SPT, concernant les heuristiques de consommation, c'est la règle SRC qui a obtenu de meilleurs résultats que la règle LRC, quant aux heuristiques S-PT/RC et L-PT/RC, c'est finalement cette dernière qui s'est avérée la plus performante. Nous avons aussi conclu que lorsqu'on applique les différentes heuristiques sur tous les cycles, les résultats sont meilleurs par rapport à ceux obtenus lorsque nous les avons utilisé que sur le premier cycle. Enfin, nous avons montré que les performances du système se sont considérablement améliorées lorsque nous avons utilisé l'algorithme génétique, particulièrement lorsque nous l'avons hybridé avec des méthodes de recherches locales à savoir DM, API ou NAPI, car cela a permis de minimiser le temps d'exécution, et du coup de maximiser la productivité. Cette amélioration s'est accrue davantage de part de la méthode de recherche locale NAPI ; cependant pour les problèmes de grandes tailles, l'amélioration est devenue moins importante et le temps a augmenté particulièrement en rajoutant à l'AG les deux approches API et NAPI cela s'explique par le fait que plus le nombre de cycles est important, plus les contraintes relatives aux ressources consommables augmentent, et par conséquent la prise de décision devient compliquée.

Perspectives

L'étude rapportée dans cette thèse représente un potentiel considérable en termes de ses implications sur le processus décisionnel des opérations pour l'ordonnancement avec ressources consommables et processus réentrant. Ce dernier reste un domaine très vaste et ne cesse d'attirer sensiblement l'intérêt des chercheurs non seulement en raison de l'importance de ce type de problème dans les entreprises manufacturières, mais aussi en fonction de la forte augmentation du nombre de publications. Cependant, la contrainte des ressources consommables mérite une réflexion plus approfondie quant à ses implications sur la gestion opérationnelle et l'utilisation optimale des capacités de production. Par conséquent, des efforts doivent être fournis pour enrichir ce travail qui peut faire l'objet de nouvelles pistes de recherches et des implications importantes dans les domaines suivants :

- **Exploitation du modèle mathématique**

Dans le travail présenté dans [Belkaid 13b], nous avons développé un modèle mathématique linéaire en nombres entiers pour le cas d'ordonnancement sur machines parallèles identique avec des arrivées sous forme d'escalier, ce programme est résolu avec un logiciel de programmation linéaire (ILOG CPLEX). Cependant, ce modèle peut être exploité d'une manière plus efficace, car plusieurs résultats théoriques peuvent y être dégagés. Pour y remédier, nous proposons d'utiliser les caractéristiques du problème afin d'essayer de dégager des règles générales et de trouver des résultats optimaux.

- **Développement d'une méthode de résolution exacte**

Un algorithme génétique est adapté afin de minimiser le makespan mais nous n'avons aucune idée sur la qualité de la solution obtenue et de la solution optimale, c'est pourquoi nous proposons de développer une méthode de résolution exacte comme la procédure par séparation et évaluation. D'autre part, nous proposons de:

- ✓ Développer des bornes inférieures en utilisant des relaxations ou des techniques de *splitting* tout en proposant des phases de corrections, etc.
- ✓ Créer des règles de dominance qui dépendent des caractéristiques du problème.
- ✓ Concevoir des ordonnancements (actifs et/ou sans délais) afin de construire des méthodes approchées.
- ✓ Construire des ordonnancements avec des algorithmes de liste, etc.

- **Combinaison avec d'autres règles de priorités**

Dans cette étude, chaque tâche est affectée à la machine qui contient le moins de tâches dans les files d'attentes. Cependant, l'efficacité du système dépend fortement de la manière de séquencement des tâches pour le traitement sur chaque machine. Ainsi, nous proposons de combiner la métaheuristique proposée avec des règles de priorités. En outre, nous pouvons essayer de donner la possibilité au système de choisir la règle adéquate à une situation donnée.

- **Extension du problème à l'optimisation multi-objectif**

Les recherches menées se concentrent sur la réduction du makespan dans un environnement de machines parallèles. Cependant, de nombreux objectifs peuvent être bien justifiés dans la pratique, ce qui est plus réaliste, minimisant ainsi le retard est aussi intéressant parce que la livraison tardive d'un produit par rapport à sa date d'échéance est pénalisée. Par conséquent, nous remarquons qu'une étude d'optimisation multi-objectif s'avère nécessaire et représente une bonne orientation future. Pour ce, nous pouvons essayer d'adapter un algorithme du type *Non-dominated Sorting Genetic Algorithm* qui est reconnu par la communauté scientifique comme une référence parmi les algorithmes génétiques.

- **Création d'un algorithme d'optimisation robuste**

Les résultats des simulations dépendent étroitement des caractéristiques du système et des paramètres appliqués. Puisque les systèmes de production sont caractérisés par leur évolution rapide et par la présence d'événements imprévus comme la rupture des ressources ou la présence de pannes, l'approche proposée peut ne pas convenir et mène à de nouvelles circonstances. Cependant, les entreprises manufacturières doivent répondre en temps opportun au cours de la fabrication, pour faire face à ces inconvénients. Nous suggérons alors de considérer le système avec d'autres spécifications et d'autres variantes afin de pouvoir mieux cerner le problème. Aussi, nous recommandons de proposer un algorithme robuste pour faire face à diverses perturbations qui surgissent des ateliers de production tout en essayant de dégager des conclusions générales.

Références bibliographiques

[Adan 06] Adan, I. and Weiss, G. (2006). Analysis of a simple Markovian re-entrant line with infinite supply of work under the LBFS policy. *Queueing Systems, 54 (3), 169–183.*

[Alfieri 09] Alfieri, A. (2009). Workload simulation and optimisation in multi-criteria hybrid flow shop scheduling: a case study. *International Journal of Production Research, 47 (18), 5129–5145.*

[Amorim 13] Amorim, R., Dias, B., de Freitas, R., and Uchoa, E. (2013). A hybrid genetic algorithm with local search approach for E/T scheduling problems on identical parallel machines. *In Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion* (pp. 63-64). ACM.

[Arzi 98] Arzi, Y. and Raviv, D. (1998). Dispatching in a workstation belonging to a re-entrant production line under sequence-dependent set-up times. *Production Planning & Control, 9 (7), 690–699.*

[Aytug 05] Aytug, H. Lawley, M.A. McKay, K.N. Mohan, S. and Uzsoy, R. (2005). Executing production schedules in the face of uncertainty: a review and some future directions. *Euro. J. Oper. Res.*161, 86–110.

[Azem 10] Azem, S. (2010). Ordonnancement des systèmes flexibles de production sous contraintes de disponibilité des ressources, *Thèse de doctorat, École Nationale Supérieure des Mines de Saint-Étienne, France.*

[Bahaji 08] Bahaji, N. and Kuhl, M. E. (2008). A simulation study of new multi-objective composite dispatching rules, CONWIP, and push lot release in semiconductor fabrication. *International Journal of Production Research, 46 (14), 3801–3824.*

[Baker 74] Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. Wiley, New York.

[Balas 85] Balas, E. (1985). On the facial structure of scheduling polyhedra. *Mathematical Programming, 24, 179–218.*

[Balasubramanian 04] Balasubramanian, H. Mönch, L. Fowler, J. and Pfund, M. (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research, 42*(8), 1621-1638.

[Balin 11] Balin, S. (2011). Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation. *Information Sciences, 181*(17), 3551-3569.

[Baranger 87] Baranger, P. (1987). *Gestion de la production Editions Vuibert entreprise*.

[Bard 99] Bard, J. F. and Devanath, K. S. (1999). An optimisation approach to capacity expansion in semiconductor manufacturing facilities. *International Journal of Production Research, 37 (15), 3359–3382.*

[Barr 95] Barr, B. S. Golden, B. L. Kelly, J. P. Resende, M. G. C. and Stewart, W. R. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 46:9–32.

[Bartz 06] Bartz-Beielstein. T. (2006). Experimental Research in Evolutionary Computation. *Springer*.

[Barua 05] Barua, A., Raghavan, N., Upasani, A., and Uzsoy, R. (2005). Implementing global factory schedules in the face of stochastic disruptions. *International Journal of Production Research*, *43*(4), 793-818.

[Belkaid 12] Belkaid, F. Yalaoui F. and Sari, Z. 2012). A genetic algorithm for parallel machine scheduling with consumable resources to minimize makespan. *The 4th International Conference on Metaheuristics and Nature Inspired Computing ,Tunisia.*

[Belkaid 13a] Belkaid, F. Sari, Z. and Souier, M. (2013). A Genetic Algorithm for the Parallel Machine Scheduling Problem with Consumable Resources. *International Journal of Applied Metaheuristic Computing (IJAMC) 4 (2),17-30.*

[Belkaid 13b] Belkaid, F. Yalaoui, F. and Sari, Z. (2013). A hybrid genetic algorithm for parallel machine scheduling problem with consumable resources. *IEEE International Conference on Control Decision and Information Technologies CoDIT'13.*

[Belkaid 13c] Belkaid, F. Yalaoui, F. and Sari, Z. (2013). Ordonnancement sur machines parallèles avec ressources consommables et processus réentrant. *14ème conférence* Roadef *de la société Française de Recherche Opérationnelle et Aide à la Décision, Roadef, Troyes, France.*

[Bellman 57] Bellman, R. (1957). Dynamic Programming. *Princeton University Press, Princeton, NJ.*

[Bellman 82] Bellman, R. and Ernest, R. (1982). Mathematical aspects of scheduling and applications. *Oxford: Pergamon Press*.

[Ben Hmida 09] Ben Hmida Sakly, A. (2009). Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible. *Thèse de doctorat, Université de Toulouse, France*.

[Ben Othman 09] Boukef Ben Othman H. (2009). Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques Optimisation par algorithmes génétiques et essaims particulaires. *Thèse* de d*octorat délivré conjointement par l'École Centrale de Lille et l'École Nationale d'Ingénieurs de Tunis*.

[Berrichi 13] Berrichi, A. and Yalaoui, F. (2013). Efficient bi-objective ant colony approach to minimize total tardiness and system unavailability for a parallel machine scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 1-16.

[Bhaskaran 91] Bhaskaran, K. and Pinedo, M. Chapter 83. (1991). dispatching. *In Handbook of Industrial Engineering, edited by G. Salvendy*, (John Wiley: New York).

[Birbil 03] Birbil, S. I. and Fang, S. (2003). An Electromagnetism-like Mechanism for Global Optimization. *Journal of Global Optimization, 25 (3), 263-282*.

[Blazewicz 83] Blazewicz, J. Lenstra J. K. and Rinnooy Kan A. H. G. (1983). Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24.

[Blazewicz 86] Blazewicz, J. Cellary, W. Slowinski R. and Weglarz; J. (1986). Scheduling under resource constraints – deterministic models. *In Annals of Operations Research, volume 7, pages 989–1006. Kluwer Academic Publishers, The Netherlands.*

[Blazewicz 91] Blazewicz, J. Dror, M. and Weglarz, J. (1991). Mathematical programming formulations for machine scheduling : A survey. *European Journal of Operational Research, 51(3):283–300, 1991*

[Blazewicz 93] Blazewicz, J. Kubiak, W. and Martello, S. (1993). Algorithms for minimizing maximum lateness with unit length tasks and resource constraints. *Discrete Applied Mathematics* 42, 123–138.

[Blazewicz 07] Blazewicz, J. Ecker, H. Pesch, E. Schmidt, G. and Weglarz, J. (2007). *Handbook on Scheduling From Theory to Applications*; Springer-Verlag.

[Blum 03] Blum, C. Roli, A. (2003) Meta-heuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput Surv* 35:268–308.

[Blum 05] Blum, C., Roli, A. and Alba, E. (2005). An Introduction to Metaheuristic Techniques. *Book chapter in: Parallel Metaheuristics: A New Class of Algorithms, Edited by E. Alba, Wiley series on parallel and distributed computing, John Wiley & Sons, 1-42.*

[Blum 08] Blum, C. Roli, A. (2008). Hybrid Metaheuristics: An Introduction. *Book chapter in:* Hybrid Metaheuristics An Emerging Approach to Optimization, *Edited by Blum, C., Aguilera, M. J. B., Roli, A. and Sampels, M., Studies in Computational Intelligence, Springer-Verlag Berlin Heidelberg, 1-30.*

[Bo_zejko 13]Bo_zejko, W. Pempera, J. and Smutnicki, C. (2013). Parallel tabu search algorithm for the hybrid flow shop problemq. *Computers & Industrial Engineering, 65,* 466-474.

[Borisovsky 07] Borisovsky P. Dolgui A. and Eremeev A. (2007). "Genetic algorithms for a supply management problem: MIP-recombination vs greedy decoder". *European Journal of Operational Research,* In Press, Corrected Proof, Available online 19 November 2007.

[Briand 09] Briand, C. (2009). A new any-order schedule generation scheme for resource-constrained project scheduling. *RAIRO - Operations Research* 43, 297-308.

[Brucker 98] Brucker, P (1998). Scheduling Algorithms. *Springer-Verlag, Berlin Heidelberg*.

[Caggiano 08] Caggiano, K. E. and Jackson, P. L. (2008). Finding minimum flow time cyclic schedules for nonidentical, multistage jobs. *IIE Transactions*, 40 (1), 45–65.

[Cakar 08] Çakar, T. Köker, R. and Demir, H. I. (2008). Parallel robot scheduling to minimize mean tardiness with precedence constraints using a genetic algorithm. *Advances in Engineering Software* 39, 47–54.

[Carlier 82] Carlier, J. and Rinnooy Kan A. H. G. (1982). Scheduling subject to nonrenewable resource constraints. *Operations Research Letters*, 1:52–55.

[Carlier 88] Carlier, J.and Chretienne, P. (1988). Problèmes d'ordonnancement: modélisation/ complexité/algorithmes, *Masson*, Paris.

[Carlier 09] Carlier, J. Moukrim A. and Xu, H. (2009). The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. Discrete Applied Mathematics 157 (17), 3631-3642.

[Carrera 10] Carrera, S. (2010). Planification et Ordonnancement de Plateformes Logistiques. *Thèse de doctorat, Institut National Polytechnique de Lorraine,* France.

[Chang 05] Chang, P. C. Chen, S. H. and Lin, K. L. (2005). Two-phase sub population genetic algorithm for parallel machine-scheduling problem. Expert Systems with Applications 29, 705–712.

[Chen 02] Chen, R. S. Lu, K. Y. and Yu , S. C. (2002). "A hybrid genetic algorithm approach on multiobjective of assembly planning problem". *Engineering Applications of Artificial Intelligence*, 15 (5), pp. 447-457.

[Chen 04] Chen, J. C. Chen, C. W. Tai, C. Y. and Tyan, J. C. (2004). Dynamic state-dependent dispatching for wafer fabrication. *International journal of production research*, *42*(21), 4547-4562.

[Chen 06a] Chen, J. A. (2006). Branch and bound procedure for the reentrant permutation flow-shop scheduling problem. *International Journal of Advanced Manufacturing Technology* 29 (11-12) ,1186–1193.

[Chen 06b] Chen, J. S. and Pan, J. C. H. (2006). Integer programming models for the re-entrant shop scheduling problems. *Engineering Optimisation*, 38 (5), 577–592

[Chen 08a] Chen, J. S. Pan, J. C. H., and Lin, C. M., (2008a). A hybrid genetic algorithm for the re-entrant flowshop scheduling problem. *Expert Systems with Applications*, 34 (1), 570–577.

[Chen 08b] Chen, J. S., Pan, J. C. H., and Wu, C. K. (2008b). Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. Expert Systems with Applications, 34 (3), 1924–1930

[Chen 13] Chen, Y. Y. Cheng, C. Y. Wang, L. C. and Chen, T. L. (2013). A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems—A case study for solar cell industry. *International Journal of Production Economics*, *141*(1), 66-78.

[Cheng 95] Cheng, R. Gen, M. and Tozawa, T, (1995). Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. *Computers and Industrial Engineering* 29, 513–517.

[Chiu 99] Chiu, N. C. Fang, S. C. and Lee, Y. S. (1999). Sequencing parallel machining operations by genetic algorithms. Computers and Industrial Engineering 36, 259–280.

[Choi 09a] Choi, S. W. and Kim, Y. D., (2009 a). Minimising total tardiness on a two-machine re-entrant flowshop. *European Journal of Operational Research*, 199 (2), 375–384.

[Choi 09b] Choi, H. S. Kim, H. W. Lee, D. -H. Yoon, J. Yun, C. Y. and Chae, K. B. (2009 b). Scheduling algorithms for two-stage reentrant hybrid flow shops : minimizing makespan under the maximum allowable due dates. *The International Journal of Advanced Manufacturing Technology*, 34 (3) 42(9-10) ,963-973.

[Choi 09c] Choi, J. Y. and Ko, S. S. (2009 c). Simulation-based two-phase genetic algorithm for the capacitated reentrant line scheduling problem. *Computers and Industrial Engineering*, 57 (3), 660–666.

[Choi 11] Choi, H. S. Kim, J. S. and Lee, D. H. (2011). Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line. *Expert Systems with Applications*, *38*(4), 3514-3521

[Chu 10] Chu, F., Chu, C., and Desprez, C. (2010). Series production in a basic re-entrant shop to minimize makespan or total flow time. *Computers & Industrial Engineering*, *58*(2), 257-268.

[Chung 09] Chung, S. H., Tai, Y. T., and Pearn, W. L. (2009). Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *International Journal of Production Research*, 47 (18), 5109–5128.

[Cigolini 99] Cigolini, R. (1999). Implementing new dispatching rules at SGS-Thomson Microelectronics. *Production Planning & Control*, *10*(1), 97-106.

[Clerc 04] Clerc, M. and Siarry, P. (2004), Une nouvelle métaheuristique pour l'optimisation difficile: la méthode des essaims particulaires, *J3eA, Vol. 3–7.*

[Cochand 89] Cochand, M. Werra D. and Slowinski R. (1989). Preemptive scheduling with staircase and piecewise linear resource availability. *Methods and Models of Operations Research*, 33:297–313.

[Cohen 95] Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge.

[Cochran 03] Cochran, J. K. and Shunk, D. L. (2003). Two-stage simulation optimisation for agile manufacturing capacity planning. *International Journal of Production Research*, 41 (6), 1181–1197.

[Courtois 95] Courtois, A. Pillet, M. and Martin, C. (1995). *Gestion de Production Editions d'organisation*, 2ème édition.

[Damodaran 06] Damodaran, P. Manjeshwar, P. K. and Srihari, K. (2006). Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics* 103, 882–891.

[Daniels 96] Daniels, R. L. Hoopes, B. J. and Mazzola, J. B. (1996). Scheduling parallel manufacturing cells with resource flexibility. Management Science 42 (9), 1260–1276.

[Daniels 97] Daniels, R. L. Hoopes, B. J.,and Mazzola, J. B. (1997). An analysis of heuristics for the parallel-machine flexible-resource scheduling problem. *Annals of Operations Research* 70, 439–472.

[Daniels 99] Daniels, R. L., Hua, S. Y.and Webster, S. (1999). Heuristics for parallel-machine flexible resource scheduling problems with unspecified job assignment. *Computers and Operations Research* 26, 143–155.

[Dantzig 54] Dantzig, G. Fulkerson, R. and Johnson, S. (1954). Solution of a large-scale travelingsalesman problem. *Journal of the Operations Research Society of America*, 2 (4): pp. 393_410, 1954.

[Dauzère-Pérès 97] Dauzère-Pérès, S. (1997). An efficient formulation for minimizing the number of late jobs in single-machine scheduling. *In IEEE symposium on emerging technologies & factory automation (ETFA), pp. 442–445.*

[Dauzère-Pérès 03] Dauzère-Pérès, S. and Sevaux, M. (2003). Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics, 50(3), 273–288.*

[Dean 99] Dean, A. and Voss, D.(1999). *Design and Analysis of Algorithms*. Springer.

[Dell'Amico 93] Dell'Amico, M. and Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research, 41*(3), 231-252.

[Ding 06] Ding, H. Benyoucef, L. and Xie, X. (2006). "A simulation-based multi-objective genetic algorithm approach for networked enterprises optimization". *Engineering Applications of artificial Intelligence*, 19, pp. 609-623.

[Dolgui 07] Dolgui, A. Eremeev, A.V. and Sigaev, V.S. (2007). "HBBA: Hybrid Algorithm for Buffer Allocation in Tandem Production Lines". *Journal of Intelligent Manufacturing*, 18 (3), pp.411-420.

[Dorigo 92] Dorigo, M. (1992). Optimization, learning and natural algorithms. *PhD thesis, Politecnico di Milano, Italy.*

[Dorigo 97] Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *BioSystems*, *43*(2), 73-81.

[Dorigo 99] Dorigo, M. Di Caro, G. and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial life*, *5*(2), 137-172.

[Dréo 03] Dréo, J. Pétrowski, A. Siarry, P. and Thaillard, E. (2003). Métaheuristiques pour l'optimisation difficile. *Eyrolles*.

[Dugardin 09a] Dugardin, F. Amodeo, L. and Yalaoui, F. (2009). Méthodes multi-objectif pour l'ordonnancement de lignes réentrantes. *Journal of Decision Systems* (2008) 231–255.

[Dugardin 09b] Dugardin, F. Amodeo, L. and Yalaoui, F. (2009). Multiobjective scheduling of a reentrant hybrid flowshop. *In Proceedings of the 39th International Conference on Computers and Industrial Engineering* (2009) 193-198.

[Dugardin 10] Dugardin, F. Yalaoui, F. and Amodeo, L. (2010). New multiobjective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research* 203 (1), 22–31.

[Dugardin 12] Dugardin, F. Yalaoui F. and Amodeo, L. (2012). Hybrid Multi-Objective Methods to Solve Reentrant Shops. *International Journal of Applied Logistics (IJA*L) 3 (4) , 15-32.

[Dupont 02] Dupont, L. and Dhaenens-Flipo, C. (2002). Minimising the makespan on a batch machine with non-identical job sises: an exact procedure. *Computers and Operations Research*, 29 (7), 807–819.

[Eberhart 95] Eberhart, R. C and Kennedy, J. (1995). New optimizer using particle swarm theory. *Proceedings of the sixth IEEE international symposium on micro machine and human science, Nagoya, Japan, 39-43*.

[Edis 11] Edis, E. B., and Ozkarahan, I. (2011). A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *Engineering Optimization*, *43*(2), 135-157.

[Edis 12a] Edis, E.B. and Ozkarahan, I. (2012). Solution approaches for a real-life resource constrained parallel machine scheduling problem. *The International Journal of Advanced Manufacturing Technology* 58, 1141–1153.

[Edis 12b] Edis, E.B. and Oguz, C. (2012). Parallel machine scheduling with flexible resources. *Computers and Industrial Engineering* 63, 433–447.

[Edis 13] Edis, E. B., Oguz, C., and Ozkarahan, I. (2013). Parallel Machine Scheduling with Additional Resources: Notation, Classification, Models and Solution Methods. *European Journal of Operational Research*.

[Elmi 11] Elmi, A. Solimanpur, M. Topaloglu, S. and Elmi, A. (2011). A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & Industrial Engineering*, *61*(1), 171-178.

[Esquirol 99] Esquirol, P. and Lopez, P (1999). *L'ordonnancement*. Economica, Paris, France.

[Farmer 86] Farmer, J. D., Packard, N. and Perelson, A (1986). The immune system, adaptation and machine learning. *Physica D, 2 (1-3), 187-204*.

[Fatos 08] Fatos Xhafa, and Ajith Abraham (Eds.) (2008). Metaheuristics for Scheduling in Industrial and Manufacturing Applications. *Springer , Studies in Computational Intelligence*, ISBN 978-3-540-78984-0.

[Feo 89] Feo, T. A. and Resende M. G. C. (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71.

[Fleury 93] FLEURY, G. (1993). Méthode s stochastiques e t déterministes pour les problèmes NP-difficiles. *Thèse. Univ. Blaise Pascal. Clermont-Ferrand* (France).

[Foulds 81] Foulds, L. R. (1981). Optimization Techniques. *Springer-Verlag*, New York, NY.

[Fowler 03] Fowler, J. W. Horng S. M. and Cochran, J. K. (2003). A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. *International Journal of Industrial Engineering: Theory Applications and Practice* 10 (3), 232–243.

[Galante 07] Galante, G. and Passannanti, G. (2007). Integrated approach to part scheduling and inspection policies for a job shop manufacturing system. *International Journal of Advanced Manufacturing Technology*, 45 (22), 5177–5198.

[Garey 79] Garey, M. R. and Johnson, M.R. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman

[Gafarov 10] Gafarov E. R. and Lazarev, A. A. (2010). Single machine scheduling with a non-renewable financial resource" *Working paper*. 2010.

[Geem 01] Geem, Z. W. Kim, J. H. and Loganathan, G. V. (2001). A new heuristic optimization: harmony search. Simulation. 76, 60 68.

[Geng 11] Geng, X. Chen, Z. Yang, W. Shi, D. and Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, *11*(4), 3680-3689.

[Giard 88] Giard V. (1988). Gestion de la Production, 2ème édition, Economica, Paris, France.

[Giard 03] Giard, V. (2003). Gestion de la production et des flux, *Economica*.

[Glover 86] Glover, F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research, 13(5), 533-549.*

[Goldberg 89] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison Wesley Professional*.

[Gomes 13] Gomes, M. C. Barbosa-Póvoa, A. P. and Novais, A. Q. (2013). Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. *International Journal of Production Research*, (ahead-of-print), 1-22.

[Gonçalves 05] Gonçalves, J. F. de Magalhães Mendes, J. J. and Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, *167*(1), 77-95.

[Goupy 01] GOUPY Jacques (2001). Introduction aux Plans d'expériences. *Dunod*. Paris. 303 pages.

[Gousty 88] Gousty, Y. and Kieffer, J.P. (1988). Une nouvelle typologie des systèmes industriels de production. *Revue française de gestion*, vol.juin-juillet-août, n°, pp. 104-112.

[Graves 83] Graves, S. C. Meal, H. C. Stefek, D. and Hamid, A. (1983). Scheduling of reentrant flow-shop. *Journal of Operations Management* 3 (4), 197-207.

[Grigoriev 05] Grigoriev, A. Sviridenko, M. and Uetz, M. (2005). Unrelated parallel machine scheduling with resource dependent processing times. *In: Jünger, M., Kaibel, V. (Eds.), LNCS, Integer Programming and Combinatorial Optimization*, vol. 3509. *Springer-Verlag, Berlin*, Germany, pp. 182–195.

[Gupta 01] Gupta, J. N. D. and Ho, J. C. (2001). Minimizing makespan subject to minimum flowtime on two identical parallel machines. *Computers and Operations Research* 28, 705–717.

[Gupta 06] Gupta, A. K. and Sivakumar, A. I. (2006). Job shop scheduling techniques in semi conductor manufacturing. *International Journal of Advanced Manufacturing Technologies* 27, 1163-1169.

[Heppner 90] Heppner, F. and Grenander, U. (1990). A stochastic nonlinear model for coordinated bird flocks. *Book chapter in : The Ubiquity of Chaos, Edited by: Krasner, S., 233-238, AAAS, Washington DC.*

[Hernandez 08] Hernandez, J.C.H. (2008). Algorithmes Métaheuristiques hybrides pour la sélection de gènes et la classification de données de biopuces, *Thèse de doctorat, Université d'Angers.*

[Holland 75] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press, Ann Arbor, MI, 1975.*.

[Hoos et Stüzle 04] Hoos, H.H. and Stüzle, T. (2004). Stochastic local search: Foundations and applications, *Morgan Kauffman*. pp. 43-85.

[Horst 95] Horst, R. and Pardolos, P. M., editors (1995). Handbook of Global Optimization. *Kluwer Academic Publishers*.

[Hwang 98] Hwang, H. and Sun, J. U. (1998). Production sequencing problem with re–entrant work flows and sequence dependent setup times. *International Journal of Production Research*, 36 (9),2435–2450.

[Ignisio 08] Ignisio, J. P. (2008). Cycle time reduction via machine-to-operation qualification. *International Journal of Production Research*, 46 (1), 1–8.

[Jain 03] Jain, V. Swarnkar, R. and Tiwari, M. K. (2003). Modelling and analysis of wafer fabrication scheduling via generalised stochastic Petri net and simulated annealing. International Journal of Production Research, 41 (15), 3501–3527.

[Janiak 07] Janiak, A. Janiak, W. and Lichtenstein, M. (2007). Resource management machine scheduling problems: a survey. *Decision Making in Manufacturing and Services* 1 (2), 59–89.

[Jing 11] Jing, C. Huang, W. and Tang, G. (2011). Minimizing total completion time for re-entrant flow shop scheduling problems. *Theoretical Computer Science*, *412*(48), 6712-6719.

[Jou 05] Jou, C. (2005). A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines. *Computers and Industrial Engineering* 48, 39–54.

[Jourdan 03] Jourdan, L. (2003). Métaheuristiques pour l'extraction de connaissances : Application à la génomique. *Thèse de doctorat, Université de Lille*, France.

[Kai 11] Kai Li, Ye Shi, Y. Shan-lin and Ba-yi, C. (2011). Parallel machine scheduling problem to minimize the makespan with resource dependent processing times, *Applied Soft Computing*: pp, 5551-5557.

[Karp 72] Karp, R. M. (1972). Reducibility Among Combinatorial Problems. *In Miller, R.E. and Thatcher, J. W. (eds). Complexity of Computer Computations, Plenum Press*, New York, 85-104.

[Karaboga 05] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. *Technical Report TR06, Computer Engineering Department, Erciyes 451 University, Turkey*

[Kashan 08] Kashan, A. H. Karimi, B. and Jenabi, M. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers and Operations Research* 35, 1084–1098.

[Kashan 09] Kashan, A. H. and Karimi, B. (2009). A discrete particle swarm optimization algorithm for scheduling parallel machines. *Computers & Industrial Engineering*,*56*(1), 216-223.

[Kaspi 04] Kaspi M. and Shabtay, D. (2004). Convex resource allocation for minimizing the makespan in a single machine with job release dates," *Computers and Operations Research*, 31:1481-1489.

[Kaveh 10] Kaveh, A. and Talatahari, S. (2010). A novel heuristic optimization method: charged system search. *Acta Mech*. 213 (3 4), 267 289.

[Kellerer 08] Kellerer, H. and Strusevisch, V. A.(2008). Scheduling parallel dedicated machines with the speeding-up resource. *Naval Research Logistics* 55 (5), 377–389.

[Kennedy 95] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks, Piscataway*, NJ, pp. 1942 2 1948.

[Kennedy 01] Kennedy, J. and Eberhart, R. C. Shi, Y. (2001). Swarm Intelligence. *Morgan Kaufman Publishers*, San Francisco, CA.

[Khalouli, 10] Khalouli, S. (2010). Métaheuristiques à base de modèles : applications à l'ordonnancement d'atelier flow-shop hybride monocritère, *Thèse de doctorat, Université de Reims Champagne-Ardenne*

[Kirkpatrick 83] Kirkpatrick, S. Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220(4598), 671-680.*

[Kumar 93] Kumar, P. R. (1993). Reentrant lines, Queueing Systems: Theory and Applications. *Special Issue on Queueing Networks* 13 1-3 (1993) 87-110.

[Kumar 04] Kumar, R. Tiwari, M. and Allada, V. (2004). Modelling and rescheduling of a re-entrant wafer fabrication line involving machine unreliability. *International Journal of Production Research,* 42 (21), 4431–4455.

[Kumar 06] Kumar, S. and Omar, M. K. (2006). Stochastic re–entrant line modeling for an environment stress testing in a semiconductor assembly industry. *Applied Mathematics and Computation*, 173 (1), 603–615.

[Laha 12] Laha, D. (2012). A simulated annealing heuristic for minimizing makespan in parallel machine scheduling. In *Swarm, Evolutionary, and Memetic Computing*(pp. 198-205). Springer Berlin Heidelberg.

[Lara 12] Lara, B. Felipe, A. Yalaoui, F. and Dugard F. (2012), May. A Tabu Search Algorithm to Minimize the Total Tardiness in the Parallel Machines Scheduling Problem. In *Information Control Problems in Manufacturing* (Vol. 14, No. 1, pp. 1371-1376).

[Lasserre 92] Lasserre, J. B. and Queyranne, M. (1992). Generic scheduling polyhedral and a new mixed integer formulation for single-machine scheduling. In Proceedings of the second IPCO conference, *Carnegie-Mellon University Pittsburgh, pp. 136–149.*

[Lazarev 09] Lazarev A. A. and Werner F. (2009). Algorithms for Special Cases of the Single Machine Total Tardiness Problem and an Application to the Even-Odd Partition Problem. *Mathematical and Computer Modelling*, Vol. 49, No. 9-10, 2061 – 2072.

[Le Moigne 90] Le Moigne , J. L. (1990). La théorie du système général : théorie de la modélisation", *2ème édition*, Paris, France.

[Lee 03] Lee, Y. H. and Lee, B. (2003). Push–pull production planning of the re-entrant process. *International Journal of Advanced Manufacturing Technology*, 22 (11), 922–931

[Lee 06] Lee, H. Y. and Lee, T. E. (2006). Scheduling single-armed cluster tools with reentrant wafer flows. *IEEE Transactions on Semiconductor Manufacturing*, 19 (2), 226–240

[Lee 07] Lee, T. E. Lee, H. Y. and Lee, S. J. (2007). Scheduling a wet station for wafer cleaning with multiple job flows and multiple wafer-handling robots. *International Journal of Production Research*, 45 (3), 487–507.

[Lee 10] Lee, C . K. M., and Lin, D. (2010). Hybrid genetic algorithm for bi-objective flow shop scheduling problems with re-entrant jobs. In *Industrial Engineering and Engineering Management (IEEM), 2010 IEEE International Conference on* (pp. 1240-1245). IEEE.

[Lee 13] Lee, J. H. Yu, J. M. and Lee, D. H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 1-9.

[Lei 08] Lei, D. (2008). A Pareto archive particle swarm optimization for multi-objective job shop scheduling. *Computers and Industrial Engineering*, 54, 960-971.

[Lenstra 77] Lenstra, J. K. Rinnooy Kan, A. H. G. and Brucker, P. (1977). Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics, Vol. 1, pp.342-362.

[Letouzey 01] Letouzey, A. (2001). Ordonnancement interactif basé sur des indicateurs : Applications à la gestion de commandes incertaines et à l'affectation des opérateurs. *Thèse de doctorat, L'Institut National Polytechnique de Toulouse, France.*

[Leung 04] Leung, J. Y. T. (2004). *Models and Performance Analysis*. *Handbook of scheduling : Algorithms*, Chapman and Hall / CRC.

[Li 03] Li, Y. Wang, F.and Lim, A. (2003). Resource constraints machine scheduling: a genetic algorithm approach. *Congress on Evolutionary Computation* 1–4, 1080–1085.

[Li 11] Li, J. Q. Pan, Q. K. Suganthan, P. N. and Chua, T. J. (2011). A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 52*(5-8), 683-697.

[Li 12] Li, X. Yalaoui, F. Amodeo L. and Chehade H.(2012), Metaheuristics and exact methods to solve a multi objective parallel machines scheduling problem. *Journal of Intelligent Manufacturing* 23 (4) (2012) 1179-1194.

[Li 13a] Li, Z. C. Qian, B. Hu, R. Zhang, C. S. and Li, K. (2013). A Self-adaptive Hybrid Population-Based Incremental Learning Algorithm for M-Machine Reentrant Permutation Flow-Shop Scheduling. In *Intelligent Computing Theories* (pp. 8-20). Springer Berlin Heidelberg.

[Li 13b] Lin, D. Lee, C. K. M. and Ho, W. (2013). Multi-level genetic algorithm for the resource-constrained re-entrant scheduling problem in the flow shop. *Engineering Applications of Artificial Intelligence*.

[Lin 04] Lin, J. T. Wang, F. K. and Lee, W. T. (2004). Capacity-constrained scheduling for a logic IC final test facility. *International Journal of Production Research*, 42 (1), 79–99.

[Lin 11] Lin Danping and Carman, K. M. Lee (2011). A review of the research methodology for the re-entrant scheduling problem, *International Journal of Production Research*, 49:8, 2221-2242.

[Lin 12] Lin, D. Lee, C. K. M. and Wu, Z. (2012). Integrating analytical hierarchy process to genetic algorithm for re-entrant flow shop scheduling problem. *International Journal of Production Research*, *50*(7), 1813-1824.

[Ling 09] Ling H. S. and Chun, Y. L. (2009). Scheduling parallel machines with resource

dependent processing times, *International Journal of Production Economics*, 2009; Volume 117, Pages 256-266.

[Liu 04] Liu, M. and Wu, C. (2004). Genetic algorithm using sequence rule chain for multi-objective optimisation in re-entrant micro-electronic production line. *Robotics and computer integrated manufacturing*, 20 (3), 225–236.

[Liu 07] Liu, H. Jiang, Z. and Fung, R. Y. K. (2007). The infrastructure of the timed EOPNs-based multiple objective real-time scheduling system for 300mm wafer fab. *International Journal of Production Research*, 45 (21), 5017–5056.

[Liu 09] Liu, C. H. (2009).A genetic algorithm based approach for scheduling of jobs containing multiple orders in a three machine flowshop. *International Journal of Production Research*, DOI: 10.1080/ 00207540902933163.

[Lopez 01] Lopez, P. and Roubellat, F. (2001). *Ordonnancement de la Production*. Hermes Science, France.

[Low 05] Low, C. Wu, T. H. and Hsu, C. M. (2005). Mathematical modelling of multi-objective job shop scheduling with dependent setups and re-entrant operations. *International Journal of Advanced Manufacturing Technology*, 27 (1), 181–189

[Low 09] Low, C. and Yeh, Y. (2009). Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. *Robotics and Computer-Integrated Manufacturing*, *25*(2), 314-322.

[Low 13] Low, C. Li, R. K. and Wu, G. H. (2013) , January. Ant Colony Optimization Algorithms for Unrelated Parallel Machine Scheduling with Controllable Processing Times and Eligibility Constraints. In *Proceedings of the Institute of Industrial Engineers Asian Conference 2013* (pp. 79-87). Springer Singapore..

[Luu 02] Luu, D. T. Bohez, E. L. J. and Techanitisawad, A. (2002) . A hybrid genetic algorithm for the batch sequencing problem on identical parallel machines. *Production Planning and Control* 13(3) , 243–252.

[Malvea 07] Malvea, S. and Uzsoy, R. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers and Operations Research* 34 (10) ,3016–3028.

[Manikas 08] Manikas A. and Chang, Y. L. (2008). Multi-criteria sequence-dependent job shop scheduling using genetic algorithms. Computers and Industrial Engineering 56 (1) 179–185.

[Mason 03] Mason, S. J. and Oey, K. (2003). Scheduling complex job shops using disjunctive graphs: a cycle elimination procedure. *International Journal of Production Research*, 41 (5), 981–994.

[Min 99] Min, L. and Cheng, W. (1999). A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering* 13 , 399–403.

[Min 03] Min, L. and Cheng, W. (2003). Scheduling algorithm based on evolutionary computing in identical parallel machine production line. *Robotics and Computer-Integrated Manufacturing*, 19, 401–407.

[Min 06] Min, L. and Cheng, W. (2006). Genetic algorithms for the optimal common due date assignment and the optimal scheduling policy in parallel machine earliness/ tardiness scheduling problems. *Robotics and Computer-Integrated Manufacturing* 22, 279–287.

[Miragliotta 05] Miragliotta, G. and Perona, M. (2005). Decentralised, multi-objective driven scheduling for reentrant shops: A conceptual development and a test case. *European Journal of Operational Research*, 167 (3), 644–662.

[Moghaddam 12a] Moghaddam, A. Yalaoui, F. and Amodeo, L. (2012). An efficient meta-heuristic based on self-control dominance concept for a bi-objective re-entrant scheduling problem with outsourcing. In *Learning and Intelligent Optimization* (pp. 467-471). Springer Berlin Heidelberg.

[Mok 07] Mok, P. Y. Kwong, C. Y. and Wong, W. K. (2007). Optimization of fault-tolerant fabriccutting schedules using genetic algorithms and fuzzy set theory. *European Journal of Operations Research* 177 ,1876–1893.

[Montgomery 84] Montgomery. D. (1984). *Design and Analysis of Experiments*. Wiley.

[Moon 99] Moon, S. and Hrymak, A. N. (1999). Scheduling of the batch annealing process – deterministic case. *Computers & Chemical Engineering*, 23 (9), 1193–1208.

[Moscato 89] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Technical Report C3P 826, Cal-teech Concurrent Computation Program*.

[Moumene 08] Moumene, K. and Ferland, J. A. (2008). New representation to reduce the search space for the resource-constrained project scheduling problem. *RAIRO - Operations Research* 42, 215-228.

[Mulkens 93] Mulkens, H. (1993). Les nouvelles organisations productives. *Revue française de gestion industrielle* N°3.

[Nazarathy 09] Nazarathy, Y. and Weiss, G. (2009). Near optimal control of queueing networks over a finite time horizon. *Annals of Operations Research*, 170 (1), 233–249

[Nearchou 04] Nearchou, A. C. (2004). The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88 :191–203

[Neto 13] Neto, R. F. T. Godinho Filho, M. and da Silva, F. M. (2013). An ant colony optimization approach for the parallel machine scheduling problem with outsourcing allowed. *Journal of Intelligent Manufacturing*, 1-12.

[Odrev 01] Odrey, N. G. Green, J. D. and Appello, A. (2001). A generalised Petri net modeling approach for the control of re-entrant flow semiconductor wafer fabrication. *Robotics and computer integrated manufacturing*, 17 (1–2), 5–11.

[Olafsson 00] Shi, S. L (2000). A method for scheduling in parallel manufacturing systems with flexible resources. *IIE Transactions* 32, 135–146.

[Omar 09] Omar, M. K. Kumar, S. and Suppiah, Y. (2009). Performance analysis in a re-entrant operation with combinational routing and yield probabilities. *Applied Mathematical Modelling*, 33 (3),1601–1612.

[Osman 96] Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research, 63(1996), 513-623.*

[Ozalp 06] Özalp, A. (2006). A genetic algorithm for scheduling of jobs on lines of press machines, I. Lirkov, S. Margenov, and J. Wasniewski (Eds.). *Springer-Verlag Berlin Heidelberg* , 535–543.

[Pan 03] Pan, J. C. H., and Chen, J. S. 2003. Minimizing makespan in re-entrant permutation flow-shops. *Journal of the Operational Research Society*, 54(6), 642–653

[Pan 04] Pan J. C. H. and Chen, J. S. (2004). A comparative study of schedule generation.

[Park 02] Park, Y. Kim, S. and Jun, C. H. (2002). Mean value analysis of re-entrant line with batch machines and multi-class jobs. *Computers and Operations Research*, 29 (8), 1009–1024.

[Pasandideh 08] Pasandideh, S. H. R.and Niaki, S. T. A. (2008). "A genetic algorithm approach to optimize a multi-products EPQ model with discrete delivery orders and constrained space". *Applied Mathematics and Computation,* 195, pp. 506-514.

[Pinedo 08] Pinedo, M. (2008). Scheduling : Theory, algorithms and systems (2nd edition). *Prentice Hall.*

[Pinedo 12] Pinedo, M. (2012). *Scheduling: theory, algorithms, and systems*. Springer.

[Portmann 88] Portmann, M. C. (1988). *Méthodes de décomposition spatiales et temporelles en ordonnancement*. RAIRO-APII, 22, 5, 439-451.

[Queyranne 93] Queyranne, M. (1993). Structure of a simple scheduling polyhedron. *Mathematical Programming, 58, 263–285.*

[Queyranne 94] Queyranne, M. and Schulz, A. S. (1994). Polyhedral approaches to machine scheduling. Technical report 408/1994, Department of Mathematics, Technical *University of Berlin, Berlin, Germany.*

[Quilliot 12] Quilliot A. and Toussaint, H. (2012). Flow Polyhedra and Resource Constrained Project Scheduling Problems. *RAIRO - Operations Research* 46, 373-409

[Rahmani 13] Rahmani, K. and Mahdavi, I. (2013). A genetic algorithm for the single machine preemptive scheduling problem with linear earliness and quadratic tardiness penalties. *The International Journal of Advanced Manufacturing Technology*, 1-8.

[Rajendran 04] Rajendran, C. and Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, *155*(2), 426-438.

[Rao 09] Rao, S. S. (2009). Optimization Theory and Application. Fourth ed. *John Wiley & Sons*, Hoboken, NJ.

[Rau 09] Rau, H. and Cho, K. H. (2009). Genetic algorithm modeling for the inspection allocation in reentrant production systems. *Expert Systems with Applications*, 35 (8), 11287–11295.

[Reynolds 87] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM.*

[Rinnooy Kan 76] Rinnooy Kan, A.H.G. (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Haque.

[Rojas 13] Rojas-Santiago, M. Damodaran, P. Muthuswamy, S. and Vélez-Gallego, M. C. (2013). Makespan minimization in a job shop with a BPM using simulated annealing. *The International Journal of Advanced Manufacturing Technology*, 1-9.

[Ruiz-Torres 07] Ruiz-Torres, A. J. and Centeno, G. (2007). Scheduling with flexible resources in parallel workcenters to minimize maximum completion time. *Computers and Operations Research* 34, 48–69.

[Sahab 13] Sahab M. G. Toropov V. V. and Gandomi A. H. (2013). A Review on Traditional and Modern Structural Optimization: Problems and Techniques. *Metaheuristic Applications in Structures and Infrastructures*, *2013*, *Pages 25-47.*

[Sait 99] Sait, S. M. and H. Youssef (1999). Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems, *IEEE Computer Society Press Los Alamitos*, CA, USA.

[Sethi 77] Sethi, R. (1977). On the complexity of mean flow time scheduling, Mathematics of Operations Research, 2(4), 320-330.

[Sevaux 04] Sevaux M. (2004) Metaheuristiques : Stratégies pour l'optimisation de la production de biens et services. *Rapport HDR*, Université de Valenciennes et du Hainaut- Cambresis.

[Sezgin 13] Sezgin Kaplan and Ghaith Rabadi (2013) Simulated annealing and metaheuristic for randomized priority search algorithms for the aerial refuelling parallel machine scheduling problem with due date-to-deadline windows and release times, *Engineering Optimization*, 45:1, 67-87.

[Shabtay 06] Shabtay, D. and Kaspi, M. (2006). Parallel machine scheduling with a convex resource consumption function, *European Journal of Operational Research* 173 (1), 92–107.

[Sioud 12] Sioud, A. Gravel, M. and Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times.*Computers & Operations Research, 39*(10), 2415-2424.

[Sivrikaya 99] Sivrikaya, F. and Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research* 26, 773–787.

[Slowinski 84] Slowinski, R. (1984). Preemptive scheduling of independent tâches on parallel machines subject to financial constraints. *European Journal of Operational Research*;15:366-373 .

[Souier 12] Souier, M. (2012). Investigations sur la sélection de routages alternatifs en temps réel basées sur les métaheuristiques -les essaims particulaires-. *Thèse de doctorat, Université de Tlemcen, Algérie*.

[Sousa 92] Sousa, J. P. and Wolsey, L. A. (1992). A time-indexed formulation of non preemptive single machine scheduling problems. *Mathematical programming, 54, 353–367*.

[Stiitzle 99] Stiitzle, T. (1999). *Local Search Algorithms for Combinatorid Problems - Analysis, Algorithms and New Applications.* DISK1 - Dissertationen zur Kiinstliken Intelligenz. infix, Sankt Augustin, Germany.

[Stockton 08a] Stockton, D. Ardon-Finch, J. and Khalil, R. (2008a). Control point policy: Part 1 – Efficiency within make-to-order environments. *International Journal of Production Research*, 46 (11), 2927–2943.

[Stockton 08b] Stockton, D. J. Khalil, R. and Ardon-Finch, J. (2008b). Control point policy optimisation using genetic algorithms. *International Journal of Production Economics*, 46 (21), 2785–2795.

[Sue 09] Sue, L. H. and Lien, C. Y. (2009). Scheduling parallel machines with resource-dependent processing times. *International Journal of Production Economics* 117, 256–266.

[Sun 09] Sun, J. U. (2009). A genetic algorithm for a re-entrant job-shop scheduling problem with sequence dependent setup times. *Engineering Optimisation*, 41 (6), 502–520

[Talbi 09] Talbi, E. (2009). Metaheuristics: From Design to Implementation. *JohnWiley and Sons, Inc.*

[Tamani 92] Tamani, K. (1992). Développement d'une méthodologie de pilotage intelligent par régulation de flux adaptée aux systèmes de production. *Thèse de doctorat, Université de Savoie, France*

[Tanaev 94] Tanaev, V. S. Gordon, V.S. and Shafransky, Y.M. (1994). *Scheduling theory. Single-stage systems*. Kluwer Academic Publishers. Dordrecht / Boston / London.

[Tangour 07] Tangour Toumi, F. (2007). Ordonnancement Dynamique dans les Industries Agroalimentaires. *Thèse de doctorat, Université des Sciences et Technologies de Lille,* France.

[Tavakkoli 09] Tavakkoli M, R. Taheri, F. Bazzazi, M. Izadi, M. and Sassani, F. (2009).

Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints.*Computers & Operations Research, 36*(12), 3224-3230.

[Tfaili 08] Tfaili, W. (2008). Conception d'un algorithme de colonie de fourmis pour l'optimisation continue dynamique. *Thèse de doctorat, Université Paris 12 Val De Marne,* France.

[Toker 91] Toker, A. Kondakci S. and Erkip, N. (1991). Scheduling under a non-renewable resource constraint, *Journal of the Operational Research Society*, 42(9):811–814, 1991.

[Torabi 13] Torabi, S. A. Sahebjamnia, N. Mansouri, S. A. and Bajestani, M. A. (2013). A Particle Swarm Optimization for a Fuzzy Multi-objective Unrelated Parallel Machines Scheduling Problem. *Applied Soft Computing*.

[Torres 10] Torres, J. R. M. Franco, E. G. and Mayorga, C. P. (2010). Project scheduling with limited resources using a genetic algorithm. *International Journal of Project Management, 28 (6), 619-628.*

[Tseng 08] Tseng, C. T. and Liao, C. J. (2008). A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *International Journal of Production Research, 46*(17), 4655-4670.

[Unlu 10] Unlu, Y. and Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering, 58*(4), 785-800.

[Upasani 08] Upasani, A. and Uzsoy, R. (2008). Integrating a decomposition procedure with problem reduction for factory scheduling with disruptions: a simulation study. *International Journal of Production Research*, 46 (21), 5883–5905

[Vallada 11] Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612-622.

[Valls 08] Valls, V. Ballestin, F. and Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 185*(2), 495-508.

[Van den Akker 99] Van den Akker, J. M. Van Hoesel, C. P. M., and Savelsbergh, M. W. P. (1999). A Polyhedral approach to single machine scheduling problems. *Mathematical Programming, 85, 541–572*

[Vanderplaats 99] Vanderplaats, G. N. (1999). Numerical Optimization Techniques for Engineering Design. Third ed. Vanderplaats Research Inc., *Colorado Springs, CO.*

[Vargas-Villamil 00] Vargas-Villamil, F .D. and Rivera, D. E. (2000). Multilayer optimisation and scheduling using model predictive control: application to reentrant semiconductor manufacturing lines. *Computers and Chemical Engineering*, 24 (8), 2009–2021.

[Voss 99] Voss, S, Martello, S. Osman, I. H. and Roucairol, C. (1999). Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. *Kluwer Academic Publishers, Dordrecht, The Netherlands.*

[Wang 97] Wang, M. Y. Sethi, S. P. and Van De Velde, S. L. (1997). Minimizing makespan in a class of reentrant shops. *Operation Research* 45, 702-712.

[Widmer 91] Widmer , M. (1991). Modèles mathématiques pour une gestion efficace des ateliers flexibles. *Editions Presses Polytechniques Romandes*.

[Wilson 04] Wilson, A. D. King, R. E. and Hodgson T. J. (2004). Scheduling non-similar groups on a flow line: Multiple group setups. *Robotics and Computer-Integrated Manufacturing* 20, 505-515.

[Woodward 65] Woodward, J. (1965). Industrialisation organisation: practice and theory. *Editions Oxford University Press*.

[Xia 06] Xia, W. J. and Wu, Z. M. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *29*(3-4), 360-366.

[Xie 97] Xie, J. (1997). Polynomial algorithms for a single machine scheduling problems with financial constraints. *Operations Research Letters*, 21:39–42.

[Xie 11] Xie, X. Tang, L. and Li, Y. (2011). Scheduling of a hub reentrant job shop to minimize makespan. *The International Journal of Advanced Manufacturing Technology*, *56*(5-8), 743-753.

[Yalaoui 13] Yalaoui, N. Ouazene, Y. Yalaoui, F. Amodeo, L. and Mahdi, H. (2013). Fuzzy-metaheuristic methods to solve a hybrid flow shop scheduling problem with pre-assignment. *International Journal of Production Research*, (ahead-of-print), 1-16.

[Yan 12] Yan, Y. and Wang, Z. (2012). A two-layer dynamic scheduling method for minimising the earliness and tardiness of a re-entrant production line. *International Journal of Production Research*, *50*(2), 499-515.

[Yang 08a] Yang, D. L. Kuo, W. H., and Chern, M. S. (2008). Multi-family scheduling in a two-machine reentrant flow shop with setups. *European Journal of Operational Research*, 187 (3), 1160–1170.

[Yang 08b] Yang, X. S. (2008). Nature-Inspired Metaheuristic Algorithms. *First ed. Luniver Press, Frome*.

[Yang 09] Yang, X. S. and Deb, S. (2009). Cuckoo search via Lévy flights. In: *Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBic). IEEE Publications*, USA, pp. 210 214.

[Yang 10] Yang, X. S. (2010). Nature-inspired metaheuristic algorithms. *Luniver Press*.

[Yang 11] Yang, T., Hsieh, C. H., and Cheng, B. Y. (2011). Lean-pull strategy in a re-entrant manufacturing environment: a pilot study for TFT-LCD array manufacturing. *International Journal of Production Research*, *49*(6), 1511-1529.

[Yip hoi 96] Yip-Hoi, D. and Dutta, D. (1996). A genetic algorithm application for sequencing operations in process planning for parallel machining. *IIE Transaction* 28(1), 55–68.

[Zamani 12] Zamani, R. (2012). A polarized adaptive schedule generation scheme for the resource-constrained project scheduling problem. *RAIRO - Operations Research* 46, 23-39.

[Zhang 07] Zhang, J. Zhai, W. and Yan, J. (2007). Multiagent-based modeling for re-entrant manufacturing system. *International Journal of Production Research*, 45 (13), 3017–3036.

[Zhang 09a] Zhang, H. Jiang, Z. and Guo, C. (2009). Simulation-based optimisation of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology. *International Journal of Advanced Manufacturing Technology*, 41 (1), 110–121.

[Zhang 09b] Zhang, G. Shao, X. Li, P. and Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers and Industrial Engineering*, 56 (4), 1309-1318.

[Zhang 12] Zhang, R. Chang, P. C. and Wu, C. (2012). A hybrid genetic algorithm for the job shop scheduling problem with practical considerations for manufacturing costs: investigations motivated by vehicle production. *International Journal of Production Economics*.

Résumé :

La dynamique des marchés a évolué et les environnements de production sont devenus complexes, caractérisés par des conditions en perpétuel changement. Cette complexité dépend non seulement de l'évolution des systèmes de production mais aussi du processus, ce qui rend, par conséquent les problèmes d'ordonnancement dépendant d'un grand nombre de contraintes, relatives aux ressources (consommables ou renouvelables), aux processus (réentrant ou non), etc. Cela conduit les industries à s'adapter à ces contraintes et à améliorer leurs systèmes de production en exploitant les différentes ressources pour rester compétitives. Dans ce contexte, notre thèse s'intéresse à l'ordonnancement sur machines parallèles avec ressources consommables et processus réentrant. Nous développons, tout d'abord, un programme linéaire en nombres entiers pour la résolution du problème d'ordonnancement sur machines parallèles avec ressource consommables. Ensuite, nous proposons une métaheuristique à base d'algorithme génétique. Cette technique est utilisée pour résoudre le problème mais cette fois ci dans un environnement caractérisé par un processus réentrant. De plus, plusieurs procédures de recherches locales sont proposées pour surmonter les limitations inhérentes aux composants individuels de l'algorithme génétique afin de pouvoir améliorer davantage son fonctionnement.

Mots clés : Ordonnancement, Système de production, Algorithme génétique, Recherche locale, Ressources consommables, Processus réentrant.

Abstract:

Market dynamic has evolved and manufacturing environments have become complex and characterized by conditions in perpetual change. This complexity depends not only on the development of production systems but also of the process, making it therefore scheduling problems depending on a large number of constraints related to resources (consumable or renewable), processes (reentrant or no), etc. This leads industries to adapt to these constraints and to improve their manufacturing systems by exploiting different resources to remain competitive. In this context, our thesis focuses on scheduling on parallel machines with consumable resources and reentrant process. First, we develop an integer linear program to solve the scheduling problem on parallel machines with consumable resource. Subsequently, we propose a metaheuristic based on genetic algorithm. This technique is used to solve the problem but this time in an environment characterized by a reentrant process. In addition, several local search procedures are proposed to overcome the inherent limitations to the individual components of the genetic algorithm in order to further improve its functioning.

Keywords: Scheduling, Manufacturing System, Genetic Algorithm, Local search, Consumables resources Reentrant process.

ملخص :

لقد عرفت حركة الاسواق تحول ملحوظا اذ اصبحت اشد تعقيدا نتيجة للتغيرات التي تطرا باستمرار ,اذ لا يقتصر هذا التعقيد على تطور أنظمة الإنتاج بل في كيفية الانتاج و الاساليب المتبعة لتحقيق ذلك بما فيها مشاكل الجدولة التي تتضمن مجموعة القيود المفروضة من قبل الموارد (المستهلكة و المتجددة) و العمليات (المتكررة و عدم المتكررة) , هذا يدفع بالمؤسسات للتكيف مع هذه القيود و تطوير نظمها الانتاجية بتحسين الموارد المتوفرة لمواكبة التقدم والمنافسة الراهنة , في هذا السياق نناقش في هذه الأطروحة اساليب الجدولة المطبقة على الالات المتوازية بوجود موارد استهلاكية و عمليات متكررة في بادئ الامر نعمل على تطوير برنامج خطي ذا عدد طبيعي لحل اشكالية الالات المتوازية مع الموارد الاستهلاكية بعد ذلك نقترح كاشف يعتمد على الخوارزمية الجينية هذه التقنية تستخدم لحل هذا المشكل في بيئة تستخدم العمليات المتكررة اضافة لذلك اقترحت العديد من الاستدلالات للتغلب على القيود الملازمة للمركبات الفردية الخاصة بالخوارزمية الجينية من اجل مواصلة تحسين ادائها.

الكلمات المفتاحية: جدولة، نظام الإنتاج، الخوارزميات الجينية، البحث الموضعي ، الموارد المستهلكة، العمليات المتكررة.