الجـــــــــــــــــــــمهـوريـــــة الجـــــــــــــــزائـريـة الديمـــــــقـراطيـة

الـشـــــــــــعبيـة

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التـــــــــــــــــــــــــــعـليـم العـــــــالي و البـــــــــــحث العـــــلـمـــي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جـــــــــــــــامعة أبي بكـــر بـلـقـايد– تـلمســـــــــــــــان –

Université Aboubakr Belkaïd– Tlemcen –
Faculté de TECHNOLOGIE

THESE

Présentée pour l'obtention du **grade** de **DOCTEUR ès SCIENCES**

En : Génie Industriel

Spécialité : Productique

Par : Mme KHEDIM Née OUIS Amaria

Sujet

Métaheuristique à base de colonies d'abeilles pour l'ordonnancement des ateliers de type Job shop

Soutenue publiquement le 10 juin 2020, devant le jury composé de :

Mr MELIANI Sidi Mohamed	MCA	Université de Tlemcen	Président du jury
Mr SOUIER Mehdi	MCA	Ecole Supérieure de Mangement de Tlemcen	Directeur de thèse
Mr BELKADI Khaled	Prof	Université des Sciences et Technologie d'Oran	Examinateur
Mme GHOMRI Latéfa	MCA	Université de Tlemcen	Examinateur
Mr SARI Zaki	Prof	Ecole Supérieure en Sciences Appliquées de Tlemcen	Invité

À tous ceux qui me sont chers …

MERCI !
C'est un petit mot tout simple
Mais qui pèse lourd
Un grand Merci, un petit Merci
Peu importe sa taille
Il n'a pas de dimension …
Que ce soit dans la joie ou dans la tristesse !
C'est un signe de reconnaissance
Qui ne connaît pas l'indifférence !
Merci !
Un petit mot qui fait du bien quand on le prononce
Un petit mot gracieux qui calme et réjouit
Merci ! Merci !
Merci de m'avoir permis de te dire,
De vous dire :
Merci !

Auteur anonyme.

Remerciements

Tous les mots ne pourraient exprimer ma gratitude envers un grand Monsieur, le Professeur Zaki SARI. Sa compétence scientifique, sa culture autant que ses conseils constructifs et ses encouragements ont toujours été très précieux pour moi. Je lui adresse mes sincères remerciements et mon profond respect, pour m'avoir fait l'honneur d'être membre de son laboratoire de recherche, Pour m'avoir accordé le privilège d'être mon premier Directeur de thèse, pour l'avantage de l'avoir eu comme excellent professeur, pour le plaisir de travailler avec un très sympathique collègue comme lui, pour la chance qu'il me donne en étant toujours présent pour me soutenir et me conseiller comme un grand frère et finalement, pour avoir bien voulu accepter notre invitation pour ainsi rehausser par sa présence le niveau de ma soutenance.

J'exprime mes remerciements les plus vifs, empreints d'une grande reconnaissance à Monsieur Mehdi SOUIER, Maitre de conférences à l'école supérieure de mangement de Tlemcen, pour avoir accepté de diriger ma thèse et sans qui je n'aurais jamais pu envisager de la finir. Il a toujours été présent pour m'aider et m'orienter avec sa grande compétence, ses pertinentes remarques et ses judicieux conseils. Toutes ses qualités n'ont d'égale que la patience qu'il m'a accordée lors de la réalisation de ce travail. Pour tout cela, je tiens à lui exprimer encore, ma profonde et sincère gratitude.

Je présente mes respectueux remerciements à Monsieur Sidi Mohamed MELIANI, Maître de Conférences à l'université de Tlemcen et Directeur du laboratoire du Génie industriel de Tlemcen, pour l'honneur qu'il me fait en acceptant de présider notre jury de thèse.

J'exprime toute ma reconnaissance à Monsieur Khaled BELKADI, Professeur à l'université des sciences et technologie d'Oran, qui me fait l'honneur de faire partie du Jury et de participer à l'évaluation de cette thèse.

Que Madame Latéfa GHOMRI, Maître de Conférences à l'université de Tlemcen, trouve ici le témoignage de mes sincères remerciements pour l'intérêt qu'elle porte à ce travail et pour avoir bien voulu accepter de le juger.

Arrivée à ce moment, et après plus de vingt ans passés à la faculté de Technologie, je ne peux faire passer cette occasion pour présenter mes respectueux remerciements à tous son personnel qui a toujours contribué au bon déroulement de mon travail.

Et encore… Pour la sympathie qu'ils m'ont toujours témoignée et pour le soutien moral qu'ils ont su me donner, et la bonne ambiance avec laquelle ils m'ont bien entourée, je remercie vivement tous mes cher(e)s ami(e)s et mes collègues.

Enfin, je témoigne mon ineffable reconnaissance à mes proches pour tous leurs encouragements, leur infaillible soutien, et leur grande patience ! Je tiens alors à présenter mes chaleureux et perpétuels remerciements à toute mon adorable famille et plus particulièrement à ma mère, mon père, mes filles, mon mari, mes frères, mes sœurs, mes beaux-parents, mes nièces, mes neveux, … - et la liste est longue ! - …Que le bon dieu me donne la sagesse et le pouvoir de leur rendre tout l'amour le bien qu'ils m'ont octroyé pour faire de moi la personne que je suis…

Si nous prenons la nature pour guide,
nous ne nous égarerons jamais.

Cicéron

Résumé

Dans cette thèse, nos investigations sont orientées vers le développement d'une méthode efficace pour la résolution du problème d'ordonnancement des ateliers du type Job shop (JSP: Job shop Scheduling Problem). Il s'agit d'un problème pratique très important dans le domaine de la gestion de production et de l'optimisation combinatoire. En effet, en raison de sa grande applicabilité et de son inhérente difficulté, le JSP est considéré comme l'un des problèmes les plus courants mais aussi les plus complexes.

De ce fait, pour résoudre le JSP qui est classé comme étant un problème NP-difficile au sens fort, on a forcément besoin d'avoir recours aux métaheuristiques. Nous avons choisi la métaheuristique des colonies d'abeilles qui s'inspire du comportement intelligent des abeilles butineuses dans la recherche d'une source de nourriture de bonne qualité. Cette méthode a été récemment introduite et formulée par l'algorithme ABC (Artificial Bee Colony) pour résoudre des problèmes d'optimisation difficile mais de nature continue. Ainsi, pour résoudre le JSP qui est de nature combinatoire, notre première contribution dans cette thèse fut d'adapter la version continue de l'algorithme ABC au problème combinatoire du Job shop. L'algorithme proposé est noté : « CABC : Combinatorial Artificial Bee Colony ». Il s'articule sur trois phases : la phase des abeilles actives et la phase des abeilles spectatrices qui assurent l'exploitation, et la phase des abeilles scoutes qui assure une bonne exploration et permet d'échapper aux optima locaux.

En deuxième lieu, afin d'améliorer encore le côté exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale a été introduite. La procédure proposée est appelée "Simple Iterated Local Search (SILS)". C'est une métaheuristique simple qui applique la recherche locale de manière itérative pour affiner la meilleure solution de l'itération actuelle de l'algorithme CABC. La procédure de recherche locale s'appuie sur la notion de voisinage généré par un opérateur d'insertion. La version hybride est notée « CABC_SILS ».

Les approches proposées sont testées sur de nombreux benchmarks de Job shop extraits de la bibliothèque de recherche opérationnelle (OR-Library). Les simulations montrent que les deux algorithmes proposés donnent des résultats très satisfaisants dans la plupart des cas étudiés. De plus, on a noté que la version hybride CABC_SILS améliore efficacement l'exploitation de l'algorithme combinatoire proposé, car elle offre de meilleurs résultats en termes de qualité de la solution évaluée par le Makespan et en termes de vitesse de convergence.

Mots clés : Ordonnancement, Job shop, métaheuristique, colonies d'abeilles artificielles, ILS.

Title

Metaheuristic based on bee colony for Job shop scheduling problems

Abstract

The Job shop Scheduling Problem (JSP) is known to be one of the most difficult scheduling problems. This is a very important practical problem in the field of production management and combinatorial optimization. To solve an NP-hard JSP, we need to use methods based on metaheuristics. Among these methods, we have the Artificial Bee Colony (ABC) algorithm which belongs to the class of metaheuristics proposed for difficult optimization. This algorithm - very recent - is inspired from the behavior of bees observed in nature. In this thesis, we first established a combinatorial version of the ABC algorithm for solving the JSP which is combinatorial in nature. Secondly, we looked to further improve the combinatorial version of the algorithm by its sequential hybridization with a new version for the iterated local search method called "Simple Iterated Local Search (SILS)". The proposed approaches are tested on numerous benchmarks extracted from the operational research library (OR-Library). The simulations showed that the both proposed approaches give very satisfying results in most of the cases studied. Furthermore, it was noted that the hybrid release CABC_SILS effectively improves the exploitation of the proposed combinatorial algorithm, because it offers better results in terms of quality of the solution evaluated by Makespan and in terms of speed of convergence.

Keywords: Scheduling, Job shop, metaheuristics, Artificial Bee Colonies, ILS.

Table des matières

Liste des figures

Liste des Tableaux

Introduction générale

Depuis la nuit des temps, l'être humain a toujours aspiré à une vie meilleure, c'est pourquoi, il serait perpétuellement, à la recherche de son bien-être physique, psychique, social et matériel. De ce fait, les entreprises de production de biens et de services se trouveraient toujours obligées de répondre à ses exigences sans cesse de plus en plus croissantes.

De nos jours, pour qu'une entreprise puisse assurer sa pérennité et réaliser des bénéfices, il est claire qu'elle doit être très compétitive dans la satisfaction des exigences et des contraintes imposées par ses clients en termes de qualité, de coûts de production, de flexibilité et de délais de livraison. Pour se faire, elle doit considérer parmi ses priorités l'optimisation de tous ses moyens de gestion et de production et accorder de l'importance au développement des techniques et des outils lui permettant d'atteindre cet objectif. De notre part, dans cette thèse, nos investigations sont orientées en particulier, vers l'un de ces outils, où l'on cherchera à développer une méthode efficace pour la résolution des problèmes d'ordonnancement des systèmes de production.

L'ordonnancement consiste à organiser dans le temps l'exécution d'un ensemble de tâches au moyen d'un ensemble de ressources, de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation. La richesse d'interprétation des termes « tâche » et « ressource » témoigne de l'importance de l'ordonnancement dans différents secteurs de l'économie : En *industrie*, particulièrement pour la gestion de production, l'ordonnancement consiste à déterminer les séquences d'opérations à réaliser sur les différentes machines de l'atelier. En domaine de *construction*, pour une bonne gestion de projet, il faut ordonnancer, et donc déterminer les dates d'exécution des activités constituant le projet. En *informatique*, ordonnancer revient à décider de l'ordre d'exécution des processus (des tâches) et de leur attribution à des processeurs ou à la mémoire. En *administration*, l'ordonnancement est par exemple, très efficace pour la préparation d'emplois du temps. En *logistique*, l'ordonnancement permet d'établir l'ordre des livraisons aux clients et leurs affectations aux différentes ressources de transport.

En effet, l'ordonnancement est un processus de prise de décision qui se trouve au cœur de nombreux types de systèmes de production, Dans cette thèse, nous nous intéressons particulièrement aux systèmes de production industrielle, en l'occurrence aux problèmes d'ordonnancement d'ateliers de production.

Un atelier de production est composé de différentes ressources, telles que les machines et les ressources humaines, etc. Dans les problèmes d'ateliers, l'ensemble des opérations (tâches) qui doivent être exécutées sur les différentes machines (ressources) constitue un travail appelé communément « *job* ». En fonction de l'ordre de passage des opérations sur les machines, on distingue différents types d'ateliers (Flow shop, Job shop, Open shop,…).

Dans le cadre de cette thèse, on s'intéresse au problème d'ordonnancement d'ateliers de type Job shop (JSP : *Job shop Scheduling Problem*). Ce dernier représente un cas particulier du problème d'ordonnancement général. Il est le plus étudié dans la littérature. En effet, il appartient à la catégorie des organisations à ressources multiples et plus particulièrement il constitue l'ordonnancement des ateliers à cheminements multiples.

Les ateliers de type Job shop sont considérés comme des systèmes fortement combinatoires. Ils sont composés d'un ensemble de n jobs, chacun composé d'un ensemble de m_i opérations qui peuvent être exécutées sur m machines en respectant certaines contraintes technologiques. Les cheminements multiples et le besoin d'une utilisation commune des ressources dans un Job shop le rendent plus complexe à gérer par rapport aux autres types d'ateliers.

Les problèmes d'ordonnancement en général et en particulier ceux des ateliers de type Job shop, sont considérés parmi les problèmes les plus difficiles et les plus complexes. Leur difficulté relative se manifeste en fonction des données telles que leurs tailles (nombre de machines et nombre de tâches) ainsi que la multitude de leurs contraintes (conjonctives, disjonctives, disponibilité, etc.). En effet, un problème d'ordonnancement de Job shop (JSP) de petite taille peut être résolu par des méthodes exactes qui fournissent des solutions optimales. Mais à partir de certaines instances, ces méthodes s'avèrent inefficaces, car le JSP se trouve classé parmi les problèmes NP-difficiles [Lenstra et Rinnooy Kan, 1979]. Pour remédier à ce genre de limitations dans la recherche de solutions, les chercheurs se sont alors orientés vers l'utilisation de méthodes approchées appelées « métaheuristiques ». L'objectif d'une métaheuristique est de trouver une solution acceptable en un temps de calcul assez raisonnable.

Durant ces cinq dernières décennies, plusieurs investigations ont été faites sur les méthodes de résolution du JSP, aussi bien sur les méthodes exactes, que sur les méthodes approchées et encore même sur les méthodes hybrides. Dans le papier de Jain et Meerran [1999], on trouve un bon état de l'art concernant le Job shop, de son début jusqu'à la fin des années 90. On trouve aussi dans l'article de Çaliş et Bulkan [2015] un autre état de l'art concernant tous les travaux récents sur les méthodes basées sur l'intelligence artificielle pour l'ordonnancement d'un Job shop.

Dans le cadre de notre travail, nous nous intéressons à une métaheuristique qui s'inspire du comportement naturel des colonies d'abeilles. Cette approche est basée sur le comportement intelligent des abeilles butineuses dans la recherche d'une source d'alimentation de bonne qualité. Ce choix a été motivé par plusieurs raisons :

- Introduite en 2005 par Karaboga [2005], cette métaheuristique assez récente, représente un nouvel outil pour la résolution du JSP.
- La structure de cette métaheuristique permet un bon équilibrage entre l'exploitation et l'exploration pour la recherche de bonnes solutions.
- Il a été montré dans une étude très détaillée [Karaboga et al. 2014], qu'en peu de temps, cette métaheuristique a attiré plusieurs chercheurs et a fait l'objet de plusieurs investigations.
- Cette métaheuristique a été appliquée avec succès sur plusieurs types de problèmes d'optimisation très complexes, mais très rarement sur le JSP.

Cette métaheuristique a été formulée par l'algorithme ABC (Artificial Bee Colony algorithm) qui est un algorithme à base de colonies d'abeilles artificielles [Karaboga et Basturk, 2007]. Cependant, l'algorithme ABC fondamental, tel qu'il a été conçu à l'origine pour résoudre des problèmes d'optimisation de nature continue, ne peut pas être utilisé directement pour le cas combinatoire.

Dans ce travail, afin de résoudre le problème du JSP qui est de nature fortement combinatoire, certaines modifications doivent être apportées à l'algorithme ABC de base. Nous avons donc proposé une nouvelle version appelée « *Combinatorial Artificial Bee Colony*» (CABC) algorithme.

De plus, pour encore améliorer le côté exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale est effectuée. Nous avons appelé

la procédure proposée: “Simple Iterated Local Search (SILS)”. On notera « CABC_SILS », la nouvelle version combinatoire et hybride des colonies d’abeilles artificielles.

Pour tester l’efficacité des approches utilisées pour la résolution du problème d’ordonnancement de Job shop, nous avons réalisé nos expériences sur des Benchmarks de Job shop. Un problème benchmark est un problème de nature très difficile à résoudre, cependant, il peut être un bon moyen pour effectuer la comparaison entre les performances de différentes méthodes. En effet tous nos tests ont été faits sur plusieurs instances de Job shop prises de la Bibliothèque de Recherche Opérationnelle (OR-Library) [Beasley, 1990, 2014].

Cette thèse est structurée en cinq chapitres :

Dans le premier chapitre, nous situons notre travail dans le cadre de l’ordonnancement des systèmes de production. Pour cela, nous commençons en premier lieu par donner des définitions, des exemples et des classifications pour les systèmes de productions en général. Par la suite, nous présentons la problématique de l’ordonnancement. Nous rappelons d’abords les différents éléments qui composent un problème d’ordonnancement, ainsi que les notations utilisées et permettant de le caractériser. Nous présentons également, une typologie des problèmes d’ordonnancement qui permet de distinguer les différents types d’ateliers. Cette classification nous amène à étudier la complexité des problèmes pour spécifier le degré de difficulté de notre problème à résoudre.

Le deuxième chapitre est dédié spécialement au problème d’ordonnancement de Job shop. Nous présentons dans ce chapitre un recueil de toutes les informations nécessaires pour étudier et traiter un tel problème. Nous commençons en premier lieu, par donner une définition de l’atelier Job shop avec toutes les notations nécessaires, les hypothèses considérées, les extensions possibles et quelques exemples d’ateliers Job shop. En deuxième lieu, nous présentons le problème d’ordonnancement de Job shop (JSP) avec toutes les notations nécessaires et la taxonomie des représentations des solutions. Nous abordons aussi la notion de complexité pour situer le degré de complexité d’un JSP. Nous n’avons pas manqué de donner un aperçu sur l’historique du Job shop. Puis en dernière section de ce chapitre, nous faisons un petit passage sur les Benchmarks des ateliers Job shop.

Dans le troisième chapitre nous présentons quelques notions de base concernant l’optimisation en général et l’optimisation combinatoire en particulier. Ensuite, nous présentons quelques-unes des méthodes les plus utilisées dans la littérature et que l’on peut classer sommairement en deux grandes catégories, à savoir : les méthodes exactes et les

méthodes approchées. Pour chacune de ces méthodes, nous citons les travaux les plus pertinents pour la résolution du JSP.

Le chapitre 4 fait l'objet de l'adaptation de la métaheuristique des colonies d'abeilles au problème d'ordonnancement du Job shop qui est un problème de nature combinatoire et de plus, classé parmi les problèmes NP-difficiles. Néanmoins, avant d'aborder l'adaptation, nous avons présenté en premier lieu, le comportement d'une colonie d'abeilles *naturelles*. Puis nous avons détaillé les différentes étapes de la version fondamentale (continue) de l'algorithme ABC qui est un algorithme d'optimisation à base de colonie d'abeilles *artificielles*. Par la suite, nous avons commencé l'adaptation après avoir précisé la représentation adéquate de la solution. Puis nous avons mis point les différentes étapes de la version combinatoire pour l'algorithme des colonies d'abeilles artificielles. La version proposée est notée : « CABC : *Combinatorial Artificial Bee Colony* ». Pour établir le réglage des paramètres nécessaires et démontrer l'efficacité de cette approche, nous l'avons testé sur de nombreuses instances de benchmarks de Job shop. Des benchmarks disponibles en ligne sur la bibliothèque de recherche opérationnelle (OR-Library) [Beasley, 2014].

Le chapitre 5 a pour objectif de présenter les investigations faites dans le but d'améliorer encore le côté exploitation de l'algorithme CABC proposé en chapitre 4. Nous effectuons une hybridation séquentielle de l'algorithme CABC avec une nouvelle procédure de recherche locale. Nous avons appelé la procédure proposée: "Simple Iterated Local Search (SILS)". On notera « CABC_SILS », la nouvelle version combinatoire et hybride des colonies d'abeilles artificielles. La version hybride a été testée et validée aussi par des simulations faites sur de nombreuses instances de benchmarks de Job shop.

Le rapport de cette thèse se termine par des conclusions et une présentation des différentes perspectives de recherches envisagées.

Chapitre 1

Ordonnancement
des systèmes de production

“ Administrer,…

... c’est prévoir, organiser, commander, coordonner et contrôler ”

Henry Fayol

1.1 Introduction

Les problèmes d’ordonnancement sont présents dans tous les secteurs de l’économie car ils représentent une fonction très importante en gestion de production. L’objectif de ce chapitre introductif est de situer notre problématique de recherche qui consiste à établir une solution efficace aux problèmes d’ordonnancement d’ateliers de production.

Dans un premier temps, des définitions, quelques détails et des classifications seront donnés pour présenter les systèmes de production. Par la suite, nous ferons un petit passage sur quelques problèmes de gestion de production, et ce, pour montrer la position de la fonction d’ordonnancement par rapport aux différents niveaux décisionnels dans un système de production.

Par la suite, nous allons : définir l’ordonnancement, les différents éléments qui composent un problème d’ordonnancement, la typologie des problèmes d’ordonnancement qui permet de distinguer les différents types d’ateliers, ainsi que les notations permettant de les caractériser. À la dernière partie, nous nous intéressons à la complexité des problèmes d'ordonnancement et plus généralement, à la théorie de la complexité.

1.2 Qu'est-ce que c'est la production ?

La production est l'ensemble des activités qui transforment, avec une certaine efficacité, les *facteurs de production* pour permettre au final, la création d'un *bien* ou d'un *service*, apte à satisfaire une demande.

- **La production d'un bien** s'effectue par une succession d'opérations qui utilisent des ressources pour transformer de la matière ou assembler des composants. Des exemples classiques sont la production de voitures, des ordinateurs, de produits agro-alimentaires, etc.

- **La production d'un service**, appelée aussi "servuction" en combinant les termes service et production, elle s'obtient par une suite d'opérations consommant des ressources sans un impératif de transformation de matière. Ce qui fait qu'un service est immatériel, intangible et par conséquent ne peut pas être stocké. Comme exemples de services, nous citons : la mise à disposition de produits aux consommateurs (la vente), le traitement d'un dossier (par des administrations), la maintenance d'équipements, la desserte en eau, la couverture d'un fournisseur de réseaux téléphonique, les consultations médicales ou juridiques, les prestations bancaires, etc.

- **Les facteurs de production** peuvent se résumer aux quatre éléments suivant :

  - **Le travail**, c'est l'énergie fournie par les ressources humaines, tels que : les ouvriers, les employés et les cadres de l'entreprise. Leur travail pourrait consister à : fabriquer les produits, traiter les informations, classer, livrer, communiquer, contrôler, etc.

  - **Le capital** est représenté par les ressources financières tel que l'argent, et par les ressources matérielles représentées par des équipements de production, tels que : les machines, les outils, les terres, les bâtiments, etc.

  - **L'énergie et les matériaux**, c'est le flux de combustibles fossiles, d'électricité, de vapeur qui fait tourner les machines, et le flux de matières premières et de produits semi-finis qui servent de matériaux de départ à la fabrication ou à l'assemblage.

- **Les informations**, c'est le savoir-faire, les brevets, les licences, tous biens immatériels résultant de l'expérience des membres de l'entreprise et d'un savoir préalablement accumulé.

Aujourd'hui et depuis les années 80, sous la pression combinée des innovations technologiques et de la globalisation économique, la production doit composer avec des objectifs à priori contradictoires. Elle doit fabriquer des produits de qualité, au moindre coût et dans les meilleurs délais, tout en s'adaptant rapidement aux demandes changeantes des clients (voir section 1.3.2).

1.3 Qu'est-ce qu'un système de production?

1.3.1 Définition

Un système de production est un système artificiel. Il est composé d'éléments matériels et immatériels interconnectés et organisés pour être capables d'interagir pour la fabrication d'un bien ou la production d'un service. Un système de production est soumis à une charge qui correspond aux différents biens ou services qui doivent être produits. Pour écouler cette charge de travail, dictée par la demande, le système utilise une ou plusieurs ressources. Ces ressources sont disponibles en quantités déterminées, c'est : des opérateurs, des machines, des outils, des matières premières renouvelables ou non-renouvelables, etc.

Dans un système de production, on peut considérer que la charge est constituée d'un ensemble d'activités. Dans la littérature des systèmes de production, selon la nature et le procédé de fabrication du produit à réaliser, une *activité* peut être désignée par : une *tâche*, une *opération*, un *travail* ou un *job*. Par la suite, pour rester dans le contexte de la thématique de cette thèse, la désignation *job* sera utilisée.

Un *job* est une entité qui suit la réalisation d'un produit dans toutes ses étapes de production. Il est composé de plusieurs opérations généralement consécutives. Chaque job possède une *gamme opératoire* qui décrit la suite des opérations correspondant aux traitements nécessaires pour la réalisation d'un produit fini. La gamme fournie des informations sur les opérations à réaliser en indiquant :

- L'ordre dans lequel les opérations sont exécutées,
- La ou les ressources qui sont allouées à la réalisation d'une opération,
- La durée nécessaire pour le traitement de chaque opération,
- Les éventuelles ressources additionnelles consommées,

- Les éventuelles contraintes imposées sur les opérations.

Généralement, une gamme correspond à un type particulier de produits. Un job a donc une seule gamme. Par contre, plusieurs jobs peuvent avoir la même gamme.

C'est en réalisant toutes les opérations de la gamme opératoire d'un job que le système produit. Selon l'environnement du système de production, ces opérations peuvent être très différentes, tout comme les ressources intervenant dans leur réalisation. Dans ce cas, vu la diversité des types de systèmes de production, il serait difficile d'énumérer tous les types d'opérations et toutes les ressources possibles. Néanmoins, le Tableau 1-1 présente une liste, non exhaustive, de quelque exemples de systèmes de production de natures très différentes, ainsi que les ressources et les opérations qui leurs sont dédiées. Dans la suite de cette thèse, on se focalisera sur les systèmes industriels. Dans ce type de systèmes, les opérations peuvent être des opérations : de transformation (fraisage, tournage, perçage, moulage, …), d'assemblage, de tri, d'inspection, de stockage ou de transport.

Différents systèmes de production	**Types de ressources**	**Types d'opérations**
Systèmes industriels	Machines, opérateurs, outils, moyens de transport,…	Transformation, assemblage, transport, tri, maintenance,…
Systèmes informatiques	Microprocesseur, bus, mémoires, interface,…	Calcul, traitement, lecture d'un fichier, sauvegarde,…
Systèmes hospitaliers	Médecins, infirmiers, lits, blocs opératoires, équipements,…	Opération chirurgicale, accueil du patient, Analyses, radio, diagnostique, …
Systèmes de production de services	Caissiers, guichetiers, Hôtesses d'accueil,…	Virement, réservation, consultation juridique, orientation,…

Tableau 1-1 : Exemples de systèmes de production, leurs ressources et leurs opérations.

Si l'on a choisi d'étudier les systèmes industriels de production en particulier, cela ne démérite pas les investigations qui doivent être faites sur les autres types de systèmes de production. En effet, une bonne modélisation d'un type de système peut être assez générique (à quelques détails et contraintes près) pour représenter d'autres systèmes de production de natures différentes. Il suffit juste de bien repérer les équivalences entre les éléments des différents systèmes. Par exemple, la ressource *machine* utilisée dans un système industriel,

peut être équivalente à un *processeur* dans le domaine de l'informatique, à un *chirurgien* dans un système hospitalier ou encore à un *caissier* dans une banque. Une équivalence peut être soulevée aussi entre les opérations effectuées par les différents systèmes. Par exemple, l'opération de *fraisage* dans un système industriel peut être assimilée, à un *calcul* pour un processeur, à un *acte chirurgical* pour le système hospitalier, ou à un *virement* effectué par un caissier.

1.3.2 Les objectifs d'un système de production

Dans le cadre d'une entreprise, le système de production, outre sa finalité première qui est de produire un service ou un bien économique, il cherche à satisfaire d'autres objectifs, à savoir :

- **L'objectif en terme de quantités produites**

Le système de production doit permettre à l'entreprise *de satisfaire la demande qui lui est adressée* ce qui suppose que l'entreprise adapte sa capacité de production au volume des ventes. Ceci passe par des actions visant à maintenir les capacités productives ou par la mise au point de plans d'investissements en capacité.

- **Objectif en terme de qualité**

Les biens économiques produits doivent être de bonne qualité, c'est-à-dire *doivent permettre de satisfaire les besoins de la clientèle.* Mais la production doit aussi être de qualité en ce qui concerne l'utilisation de ressources afin de respecter le critère d'efficience attaché au système productif. Le système productif doit donc être économe en ressources et constant en terme de qualité.

- **Objectif lié au coût**

Le système productif adopté par l'entreprise doit proposer les plus faibles coûts de production possibles de *manière à garantir la compétitivité de l'entreprise*. De plus, les coûts de production calculés doivent aussi être mis en relation avec les coûts de production prévus par le centre opérationnel. Sur la longue période, cet objectif de coût se traduit par la recherche permanente de gains de productivité afin de détenir ou de conserver un avantage compétitif pour l'entreprise.

- **Objectif de délai**

Le système de production doit certes produire, mais dans *des délais raisonnables*, c'est-à-dire en conformité avec le niveau de la demande à laquelle doit faire face l'entreprise. Ceci suppose la mise en place d'un mode de production réactif qui permettra soit d'éviter des stocks de biens finaux, soit de ne pas connaître de goulets d'étranglement. En terme de productivité, l'objectif de délai signifie aussi *réduire* les délais de fabrication.

- **Objectif de flexibilité**

Certains systèmes de production ont besoin d'être flexibles pour pouvoir *s'adapter aux variations de la demande,* soit pour tenir compte des évolutions de l'environnement productif de l'entreprise (innovations technologiques,…), soit pour permettre une production simultanée de plusieurs types de produits différents en même temps.

1.3.3 Classification des systèmes de production

Les systèmes de production peuvent être classés selon différents critères, [Pillet et al. 2011] :

- **Selon la quantité produite** : production unitaire (navire, pont, bâtiment,…), par lot (vêtements, avions,…), par grande série (électroménager, automobile,…), en continu (électricité, gaz, acier, …)
- **Selon le processus technique de production** : production en continu, production en discontinu, production en hybride.
- **Selon la relation avec le client** : production pour le stock, production sur commande, Assemblage à la demande.
- **Selon la circulation des produits** dans l'atelier de production. On peut distinguer différents types d'ateliers. Les Anglophones les appellent : Project shop, Flow shop, Job shop, Open shop,…

Dans le cadre de notre travail on se focalise sur le dernier type de classifications citées ci-dessus. En effet, le procédé de fabrication d'un produit impose à celui-ci un certain mode de circulation dans l'atelier de production. Par conséquent, selon le type des produits et selon les quantités demandées, on peut distinguer différents systèmes de production implantés selon différents types d'ateliers. Plus précisément, dans cette thèse, l'atelier *Job shop* fera l'objet de notre étude.

1.3.4 Gestion des systèmes de production

La gestion des systèmes de production consiste à aborder plusieurs types de problèmes. Ces derniers peuvent être regroupés en trois catégories formant ainsi trois classes de problèmes. A savoir, les problèmes : de *conception*, de *planification* et de *contrôle*. En fonction de son horizon, chacune de ces classes possède ses propres objectifs [Pirard, 2005].

- **La conception** est l'ensemble des activités effectuées : avant la création d'un nouveau système de production, ou lorsqu'il y a un besoin de modifier un système qui existe déjà. Dans la phase de conception, les questions concernant le dimensionnement du système, le choix des machines utilisées, ou encore le routage des entités dans le système de production sont abordées.
- **La planification** consiste à prévoir la réalisation d'un ensemble d'activités (tâches/jobs), appelé « charge du système », en déterminant les différentes ressources utilisées par les opérations à réaliser pour chaque activité ainsi qu'en définissant la façon et les dates auxquelles ces opérations doivent être exécutées. On distingue trois niveaux de planification selon l'horizon temporel considéré : le long terme (stratégique), le moyen terme (tactique) et le court terme (opérationnel). C'est trois niveaux de planification coexistent et se complètent dans l'entreprise. ils ont des objectifs différents et généralement, ils ne sont pas réalisés conjointement. Néanmoins, ils doivent être cohérents afin de participer à l'optimisation d'une entreprise dans sa globalité.

- **Le contrôle** consiste à résoudre les éventuels problèmes surgissant lors de la réalisation des opérations. Ces problèmes proviennent des aléas des systèmes de production, tels que : les pannes, les nouveaux ordres de fabrication à réaliser, ou tout événement aléatoire qui entrave le bon déroulement du planning prévu.

1.3.5 Les niveaux de décisions dans un système de production

L'objectif de la gestion de production est de gérer les systèmes de production de sorte à optimiser un certain nombre de critères sur une période de temps bien déterminée. Mais, il se trouve que dans certains cas, ce temps s'étend sur plusieurs mois voire même plusieurs années, il n'est donc pas envisageable de résoudre le problème d'une entreprise sur l'ensemble de cette période. La solution classique dans le monde industriel est de prendre des décisions selon des différents horizons temporels. En gestion de production, ces décisions

sont habituellement classées selon trois catégories, à savoir : les décisions stratégiques, tactiques et opérationnelles, [Anthony, 1965] et [Van Looveren et al. 1986].

- **Décisions stratégiques**

Ces décisions définissent la politique de l'entreprise à long terme. Elles sont établies pour une durée de temps qui varie entre 2 à 5 ans et qui dépend largement des systèmes considérés. A ce niveau, les problèmes traités représentent des enjeux importants pour une entreprise et de ce fait les décisions sont prises par son plus haut niveau hiérarchique. Il s'agit généralement de problèmes de configuration et de reconfiguration des systèmes de production pour assurer une adéquation entre la capacité de production et les estimations de vente. Comme exemple, on peut décider de : la construction une nouvelle usine, l'acquisition de nouvelles ressources (Matérielles ou humaines), la modification de la charge du système en acceptant la production de nouvelles références, la modification d'une structure d'une ligne de production, etc.

- **Décisions tactiques**

Ces décisions définissent la politique de l'entreprise à moyen terme, Elles veillent à mettre en œuvres des solutions qui respectent et soutiennent les décisions stratégiques. Comme exemples de décisions tactiques, on peut citer : la planification de la production qui est une programmation prévisionnelle de la production pour une période qui varie, selon l'entreprise, entre 6 et 18 mois (c'est pour l'élaboration du plan directeur de production), les problèmes d'allocation (des fournisseurs ou des produits), la définition des niveaux de stock ainsi que le choix des modes de transport, etc.

- **Décisions opérationnelles**

C'est des décisions qui couvrent un horizon de temps variant d'une demi-journée à quelques semaines. Elles correspondent au pilotage et à l'ordonnancement du système de production. Elles sont situées au plus près de l'activité quotidienne de l'entreprise, gèrent l'allocation des commandes et déterminent le déploiement optimum des ressources et moyens de production pour satisfaire la demande immédiate. À ce niveau de décision, on élabore les plans par unités de production (plans directeurs court terme, dits aussi, programmes directeurs de production) à partir du carnet de commandes, et en cohérence avec le plan directeur à moyen terme. Plus particulièrement, les décisions opérationnelles doivent permettre de réaliser la meilleure allocation de ressources possible (séquencement) et de fixer les meilleures dates de début et de fin des ordres de fabrication

(ordonnancement). Généralement, elles sont prises par des responsables de projet, des chefs d'équipes ou des agents de maîtrise.

Dans cette thèse notre intérêt se focalise sur la prise de décisions opérationnelles, en l'occurrence sur l'odonnancement des ateliers. En effet, ce dernier a pour mission de couvrir un ensemble d'actions qui transforment les décisions de fabrication définies par le programme directeur de production en instructions d'exécution détaillées destinées à piloter et contrôler l'activité des postes de travail (ressources) dans l'atelier. Plus de détails seront fournis à ce sujet dans la section suivante.

1.4 Ordonnancement

1.4.1 Définition

L'un des problèmes récurrents dans la gestion des systèmes de production et de savoir *qui* fait *quoi*, *où* et *quand*. On parle alors de problème d'ordonnancement.

Dans la littérature et tout en restant dans le même concept, plusieurs définitions ont été proposées pour le problème de l'ordonnancement :

- Parmi les anciennes définitions, on trouve celle de Roy et Portmann [1979] :

« *Ordonnancer c'est programmer dans le temps l'exécution d'une réalisation décomposable en tâches, en attribuant des ressources à ces tâches et en fixant en particulier leurs dates de début d'exécution tout en respectant des contraintes données*. »

- Nous avons la fameuse définition de Pinedo [2016], et qui d'ailleurs se trouve aussi dans toutes les précédentes éditions de ses livres :

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

- Dans leur livre, Esquirol et Lopez [1999] définissent l'ordonnancement comme suit :

« *Résoudre un problème d'ordonnancement consiste à ordonnancer i.e. programmer ou planifier dans le temps, l'exécution des tâches en leurs attribuant les ressources nécessaires, matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation* ».

- On ajoute aussi une définition bien concise, tout en étant complète. C'est celle de Artigues [2004], où il dit que:

« *Résoudre un problème d'ordonnancement c'est résoudre un problème d'optimisation combinatoire constitué d'un ensemble de tâches, d'un ensemble de ressources, d'un ensemble d'objectifs et d'un ensemble de contraintes. La solution à ce problème est un ordonnancement qui donne des indications sur les dates de début des tâches ou qui propose une relation d'ordre total ou partiel sur la séquence de réalisation des tâches* ».

Dans sa forme générale, l'ordonnancement est un problème d'optimisation que l'on définit comme suit :

Soit :

- un ensemble J de n tâches à exécuter,
- un ensemble M de m ressources requises pour le traitement des tâches,
- un ensemble de contraintes qui doivent être satisfaites, et
- un ensemble d'objectifs permettant de juger les performances d'un ordonnancement.

➢ Quelle est la meilleure façon de séquencer et d'allouer temporellement ces tâches sur ces ressources de manière à ce que : toutes les contraintes soient satisfaites et les meilleures performances soient atteintes ?

Actuellement, l'ordonnancement est un des thèmes les plus étudiés dans la recherche opérationnelle. Effectivement, l'ordonnancement est un processus de prise de décision qui intervient dans plusieurs types de systèmes de production, en l'occurrence dans les systèmes de production industrielle. Dans cette thèse, nous nous intéressons particulièrement à ce dernier type de systèmes, où l'on se focalise sur les problèmes d'ordonnancement d'atelier que l'on va présenter dans la section suivante.

1.4.2 Problème d'ordonnancement d'ateliers

Les systèmes de production sont divers et variés et ont différentes organisations au niveau de leurs ateliers de production. Ces organisations déterminent les types des problèmes d'ordonnancements.

Un atelier de production est composé de certaines ressources, telles que les machines et les ressources humaines. Dans un atelier, une production consiste à fabriquer un produit ou une

variété de produits. Souvent ces derniers représentent des pièces à manufacturer. Communément, les termes « *produit* et *pièce* » peuvent être remplacés par le terme « *job*».

La réalisation d'un job nécessite généralement plusieurs *opérations* élémentaires appelées « *tâches* ». Les opérations seront exécutées sur les machines (ressources) en suivant la gamme opératoire propre à chaque job.

Un ordonnancement consiste à établir une organisation temporelle du fonctionnement de l'atelier. En effet, l'exécution des tâches doit être organisée et programmée dans le temps tout en : optimisant l'utilisation des ressources humaines et matérielles disponibles, et en respectant les contraintes imposées pour répondre à un ou à plusieurs objectifs fixés pour le bon rendement de l'atelier en question.

D'une manière générale, supposant que dans un atelier, on a besoin d'ordonnancer *n* jobs sur *m* machines. Chaque job pourrait nécessiter une ou plusieurs opérations à exécuter sur ces machines. *n* et *m* sont certes des données nécessaires pour la définition d'un problème d'ordonnancement d'atelier mais elles ne sont pas suffisantes. En effet, les problèmes d'ordonnancement d'atelier diffèrent et se définissent aussi selon : le *type* d'atelier, selon les technologies utilisées et selon les contraintes auxquelles ces ateliers pourraient être soumis.

On catégorise les types d'ateliers, selon :

- La nature du job (job nécessitant une ou plusieurs opérations à exécuter),
- selon l'ordre de passage des opérations sur les machines,
- ou selon le cas où un type de machine est dupliqué ou pas.

1.4.2.1 Atelier à une seule machine

Dans ce type d'ateliers, une seule machine (ressource) est disponible pour le traitement de tous les jobs, (Figure1-1). Dans ce cas, soit :

- les jobs requièrent uniquement une seule opération.
- ou la machine est polyvalente et peut réaliser plusieurs types d'opérations pour un même job.

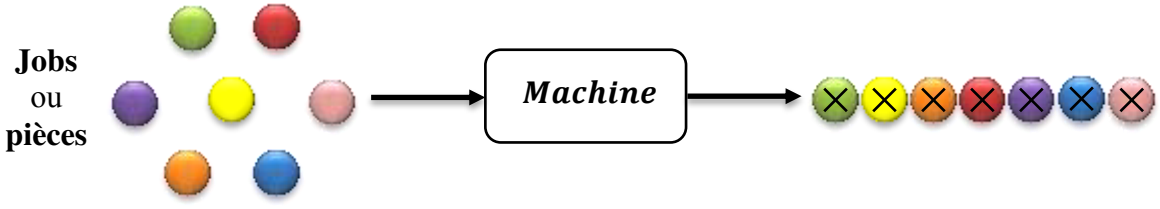

Figure 1-1 : Atelier à une seule machine.

Ce type d'atelier correspond au problème d'ordonnancement le plus basique et le plus étudié dans la littérature. D'ailleurs, dans certains cas, des problèmes d'ordonnancement complexes peuvent être décomposés et ramenés à l'étude de plusieurs problèmes à une seule machine.

Par ailleurs, dans la pratique il existe des problèmes n'utilisant qu'une seule machine, comme par exemple la programmation des atterrissages des avions. La présence d'une seule piste d'atterrissage est vue comme une machine unique et les avions comme des jobs entrants.

1.4.2.2 Atelier à machines parallèles

Le principe de ce type d'ateliers est similaire à celui d'une machine unique, les jobs sont constitués d'une seule opération, sauf qu'ils ont le choix d'exécuter cette opération sur plusieurs machines parallèles : identiques ou ayant les mêmes fonctionnalités, (Figure1-2).

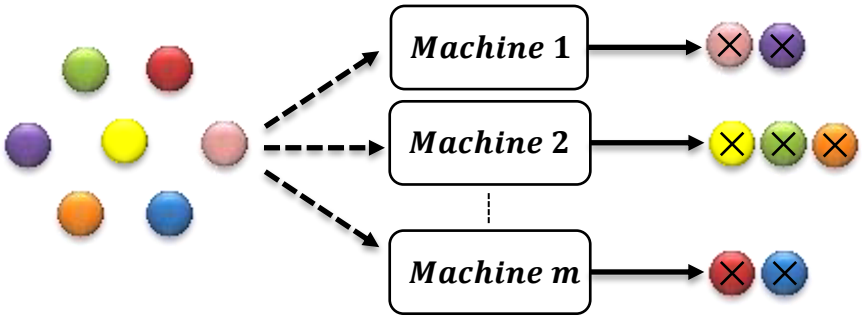

Figure 1-2 : Atelier à machines parallèles.

Dans cette catégorie d'ateliers, il est encore possible de distinguer trois sous-classes d'ateliers :

- *Machines parallèles identiques* (P) : la vitesse d'exécution d'une opération est la même sur toutes les machines.
- *Machines parallèles uniformes* (Q) : chaque machine a une vitesse d'exécution propre et constante.
- *Machines parallèles non-uniformes* (R) : la vitesse d'exécution est différente pour chaque machine et pour chaque opération.

En ordonnancement, les problèmes à machines parallèles sont considérés comme une généralisation des problèmes à une machine.

Un exemple de machines parallèles est le passage aux caisses dans un supermarché. Dans ce cas, les machines sont les caisses et les jobs sont les clients avec leurs courses qui attendent. Selon la vitesse des caissières, nous pouvons parler des trois modèles cités ci-dessus.

1.4.2.3 Atelier de type Flow shop (F)

Appelé également atelier à cheminement unique, il s'agit d'un ensemble de *m* machines disposées dans des stations de travail en séries. Le flux des jobs est unidirectionnel et linéaire. Toutes les opérations de tous les jobs passent par les machines dans le même ordre. Dans ce cas, le but de l'ordonnancement est habituellement de définir l'ordre de traitement des jobs, (Figure1-3).

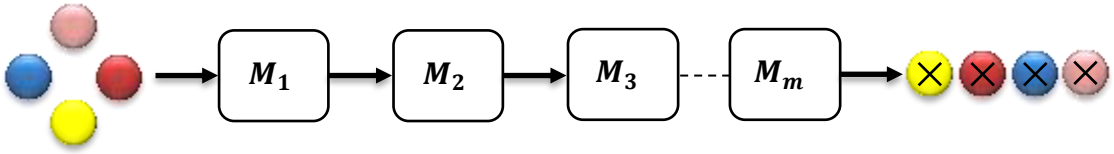

Figure 1-3 : Atelier de type Flow shop.

Un exemple typique d'atelier Flow shop est une chaîne d'assemblage. Il est à noter un cas particulier : si l'ordre de passage des opérations sur chaque machine est identique, alors on se trouve face à un problème Flow shop de permutation.

Une autre variante du Flow shop est le Flow shop flexible/hybride, où à chaque station il y a plusieurs machines en parallèle capables de traiter la même opération. Les machines sont identiques pour le cas du *Flow shop Flexible*, alors qu'elles peuvent être différentes pour le *Flow shop hybride*. Dans les deux cas, en plus du besoin d'ordonnancer les jobs, il faut prévoir l'affectation de chaque job à une des machines disponibles, station par station.

1.4.2.4 Atelier de type Job shop (J)

Les Job shop appelés aussi ateliers à cheminements multiples et une généralisation du Flow shop. Effectivement, chaque job passe sur les machines dans un ordre fixé, mais à la différence du Flow shop, cet ordre peut être différent pour chaque job. Le flux des jobs est dit multidirectionnel. De plus, dans cet atelier, un job peut revenir plusieurs fois sur la même machine. C'est le phénomène de recirculation, (Figure 1-4).

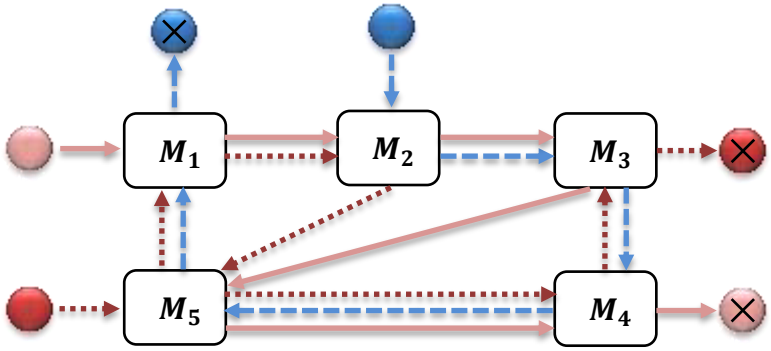

Figure 1-4 : Atelier de type Job shop.

Dans le cas du Job shop, les jobs empruntent différents routages (cheminements) mais puisqu'ils partagent des machines communes, des conflits sont susceptibles de survenir. C'est exactement dans ce cas, que l'ordonnancement va gérer ce genre de conflits en déterminant les dates de passage des jobs sur les différentes machines, tout en respectant les contraintes imposées, et en optimisant les objectifs poursuivis.

Comme pour le type d'atelier précédent, il existe aussi la variante *Job shop flexible*, avec plusieurs machines polyvalentes en parallèle par station de travail. Donc, non seulement il faut ordonnancer le mieux possible les opérations des jobs, mais aussi déterminer la meilleure affectation aux machines.

1.4.2.5 Atelier de type Open shop (O)

Cet atelier est dit à cheminement libre. L'ordre de passage des jobs sur les machines est totalement libre. Les contraintes de précédence entre les différentes opérations d'un même job ne sont pas définies. Le fait qu'il n'y ait pas d'ordre prédéterminé rend la résolution du problème d'ordonnancement de ce type plus complexe, mais offre cependant des degrés de liberté intéressants. Cet atelier est le moins étudié dans la littérature car il est rarement rencontré dans la pratique. La Figure 1-5, montre que dans un Open shop, un même job est libre de prendre différents routages.

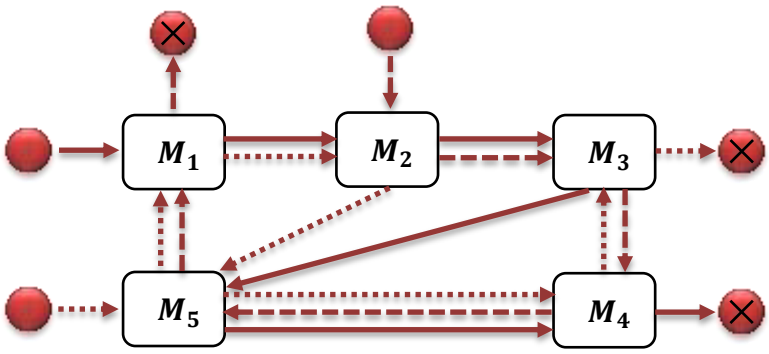

Figure 1-5 : Atelier de type Open-shop.

Dans le cas où au niveau d'une station de travail de l'Open shop, une machine est remplacée par un groupe de machines parallèles, l'atelier est appelé *Open shop généralisé*.

Dans le cadre de cette thèse, nos investigations sont faites sur le cas d'un atelier du type Job shop.

1.4.3 Les Éléments d'un problème d'ordonnancement

La définition d'un problème d'ordonnancement s'articule sur les quatre éléments suivants, [Esquirol et Lopez 2001]) :

- les tâches (activités ou opérations),
- les ressources,
- les contraintes et
- les objectifs (critères).

1.4.3.1 Tâches

Dans la littérature de l'ordonnancement, certaines nuances peuvent être constatées par rapport aux définitions d'une tâche. Il y a celles qui considèrent que *la tâche* est un travail (job) constitué d'un *ensemble d'opérations* à exécuter. D'autre part, il y a d'autres définitions qui stipulent que la tâche représente *une opération* parmi l'ensemble des opérations nécessaires pour la réalisation d'un travail (job).

Pour le cas de notre étude, on considère le deuxième type de définitions. En effet, la tâche c'est l'entité élémentaire d'un travail. Ainsi, plusieurs tâches sont groupées pour former un travail.

Dans le cas des ateliers de production classiques, des ordres de fabrication sont suggérés pour la réalisation de certains travaux. Ces travaux sont des *Jobs* (pièces ou des lots de pièces)

à réaliser pour une date donnée. Ainsi, Les ordres de fabrication représentent chacun une requête pour la fabrication d'un job.

- À chaque *job* est associée une gamme opératoire qui est une liste ordonnée d'opérations (tâches). La Figure 1-6, illustre la gamme opératoire et les principales contraintes temporelles relatives à un job i.

  Soit un job « J_i ». Il est caractérisé par les paramètres temporels suivants :

  - R_i : La date de début au plutôt ou date de disponibilité du job J_i (Release time).
  - S_i : La date de début d'exécution du job J_i (Start time).
  - C_i : La date de fin d'exécution du job J_i (Completion time).
  - D_i : La date de fin au plus tard ou date d'achèvement souhaitée du job J_i (Due date).
  - L_i : Le retard algébrique du job J_i ($L_i = C_i - D_i$).
  - T_i : Le retard réel du job J_i ($T_i = \max(L_i, 0)$).
  - E_i : L'avance réelle du job J_i ($E_i = \max(0, D_i - C_i)$).
  - F_i : La durée de séjour du job J_i, depuis sa disponibilité jusqu'à la fin de son exécution (flow time, $F_i = C_i - R_i$).

De même nous pouvons définir aussi le paramètre w_i indiquant un poids relatif (weight). Il définit la priorité de l'exécution job J_i et dénote son importance relativement aux autres jobs.

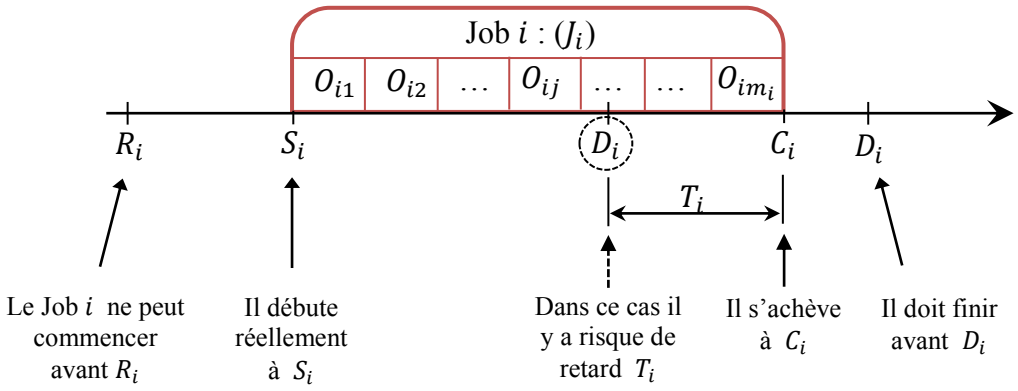

Figure 1-6 : Caractéristiques d'un job « J_i ».

Les jobs qui constituent l'un des données d'entrée de l'ordonnancement, permettent de définir, au moyen de leurs gammes opératoires, l'ensemble des tâches (opérations) que la fonction ordonnancement doit planifier. De ce fait, en sortie de cette fonction, on obtient un

planning qui restitue l'affectation des tâches fournies en entrée à des dates précises pour des durées déterminées sur les différentes ressources. Ce planning cherche à satisfaire un ou plusieurs objectifs, en respectant le plus possible les contraintes imposées.

- Les *opérations* (*tâches*) représentent donc les éléments à ordonnancer. Si l'on considère « *j-ième* » opération du job « J_i », on la note « O_{ij} » et elle doit être définie par les paramètres temporels suivants :
  - r_{ij} : La date de disponibilité de l'opération O_{ij}.
  - s_{ij} : La date de début d'exécution de l'opération O_{ij}
  - p_{ij} : La durée d'exécution de l'opération O_{ij}(processing time, $p_{ij} = c_{ij} - s_{ij}$)
  - c_{ij} : La date de fin d'exécution de l'opération O_{ij}
  - d_{ij} : La date de besoin ou date d'échéance de l'opération O_{ij}

La Figure 1-7, donne la représentation de l'opération « O_{ij} » avec ses principales caractéristiques temporelles.

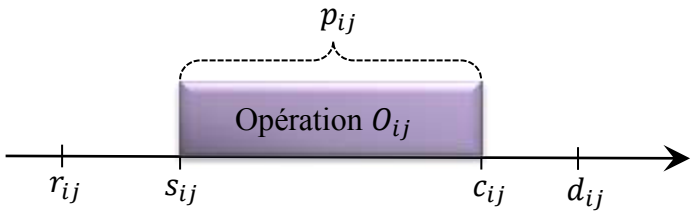

Figure 1-7 : Caractéristiques temporelles de l'opération « O_{ij} ».

Chaque opération (tâche) utilise pour son exécution une ou plusieurs ressources. On distingue deux situations :

- Si cette opération peut se réaliser par morceaux, il s'agit d'une opération préemptive (tâche morcelable).
- Par contre, si elle doit être exécutée sans interruption, on parle alors de tâche non-préemptive (indivisible, non-morcelable).

1.4.3.2 Les ressources

L'exécution des différentes tâches nécessite la mise en œuvre d'un ensemble de ressources. Dans un atelier les ressources peuvent être : des moyens techniques (machines, outils,…), des opérateurs humains, de la matière première, de l'argent, etc. Pour une ressource, la disponibilité et la capacité, limitée ou non, sont connues à priori. Plusieurs types de ressources sont distingués :

1. Selon la disponibilité de la ressource au cours du temps, on trouve :
    - *Les ressources renouvelables* : Une ressource est dite renouvelable si après avoir été allouée à une tâche, elle redevient disponible pour les autres. c'est le cas pour les machines, les équipements, les processeurs, les robots, le personnel, etc.
    - *Les ressources consommables* (non-renouvelables) : il s'agit de ressources dont la disponibilité décroît et tend à disparaitre après avoir été allouées à une ou plusieurs tâches. C'est le cas pour l'argent, la matière première, etc.

2. Selon la capacité de la ressource, on trouve :
    - *Les ressources disjonctives* (ou non-partageables) : il s'agit des ressources qui ne peuvent exécuter qu'une seule tâche à la fois. C'est le cas par exemple de machine-outil ou de robot.
    - *Les ressources cumulatives* (ou partageables) : il s'agit des ressources qui peuvent exécuter plusieurs tâches simultanément. C'est le cas par exemple d'une station de travail composée de plusieurs machines ou d'une équipe d'ouvriers.

Dans ce travail, nous nous intéressons qu'aux problèmes à ressources renouvelables et disjonctives.

1.4.3.3 Les contraintes

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement les variables de décision. Il s'agit des variables représentant les relations reliant les tâches et les ressources [Esquirol et Lopez, 1999].

La prise en compte de toutes les contraintes permet d'avoir un ordonnancement admissible et réalisable (faisable). De ce fait, la résolution des problèmes d'ordonnancement est d'autant plus compliquée que le nombre de contraintes est plus élevé.

Différents types de contraintes sont distingués :

1. **Les contraintes de temps** qui décrivent les interdépendances temporelles entre les opérations. Elles sont généralement représentées par :
    a) *Les Contraintes de dates limites* : c'est des contraintes imposées individuellement à chaque job (ou à chaque opération). Par exemple, dans un ordonnancement sans retard, l'exécution du job J_i (ou de l'opération O_{ij}) dois être terminée avant sa date limite. Et doit satisfaire la condition suivante :

$$C_i \leq D_i \qquad (c_{ij} \leq d_{ij})$$

b) *Les Contraintes de date de disponibilité* où l'opération O_{ij} ne peut commencer avant sa date de disponibilité r_{ij}. Elles peuvent être liées la disponibilité des matières premières, ou à des conditions climatiques, etc. Dans ce cas il faut que :

$$s_{ij} \geq r_{ij}$$

c) *Les contraintes de localisation temporelle* qui concernent le suivi de l'exécution des opérations qui se succèdent. l'opération O_{ij} ne peut commencer avant que l'opération qui la précède $O_{i(j-1)}$ ne soit terminée, donc il faut que :

$$s_{ij} \geq c_{i(j-1)}$$

d) *Les contraintes de calendrier* qui sont liées au respect d'horaires de travail, etc.

2. **Contraintes d'enchaînement** qui sont généralement des contraintes de cohérence technologique, décrivent le positionnement relatif de certaines tâches par rapport à d'autres. Par exemple l'opération de fermeture d'un bouchon de bouteille ne doit pas se faire avant l'opération de remplissage de celle-ci.

   Dans l'atelier de production, ces contraintes reflètent la prise en compte de la gamme opératoire du job à réaliser.

   La contrainte de précédence qui impose que l'opération O_{ij} doit se faire avant l'opération $O_{i(j+1)}$ peut être formulée par une contrainte temporelle et représentée par l'inégalité suivante :

$$s_{i(j+1)} \geq s_{ij} + p_{ij}$$

3. **Contraintes de ressources** : elle représente la nature, la quantité et la capacité de la ressource utilisée pour l'exécution d'une opération au cours du temps. Ainsi, deux types de contraintes pour les ressources renouvelables peuvent être envisagés [Esquirol et Lopez, 1999] :

   a) *Les contraintes de partages de ressources* (comme le cas de partage de ressources dans l'atelier Job shop qui fait l'objet de notre étude dans cette thèse). Ces contraintes permettent de limiter la capacité des ressources. A ce niveau deux types de contraintes liées à la nature cumulative ou disjonctive des ressources peuvent être distingués :

      - Les contraintes disjonctives qui imposent la non-réalisation simultanée de deux opérations sur la même ressource.

- Les contraintes cumulatives qui stipulent qu'un certain nombre limité d'opérations peuvent être exécutées simultanément sur la même ressource.

b) *Les contraintes de disponibilité de ressources* : elles se traduisent surtout par la nature et la quantité des moyens disponibles au cours du temps. Les pannes des machines (arrêts imprévu des machines) ou bien les périodes de maintenances préventives ou correctives (arrêts programmés des machines) sont considérés comme des contraintes de disponibilité des machines.

4. **Contraintes intérieures :** ce sont des conditions directement liées au système de production et à ses performances. On peut citer :
   a) les capacités des machines et des moyens de transport.
   b) les dates de disponibilités de la matière première des produits et des moyens de transport.
5. **Contraintes extérieures :** ce sont des conditions imposées extérieurement. Elles sont indépendantes du système de production, telles que :
   a) les dates de besoin de produits, imposées généralement par les commandes du client.
   b) les niveaux de priorité et d'urgence de quelques commandes et de quelques clients.
   c) les retards possibles accordés pour certains produits.

Nous pouvons aussi classifier d'autres types de contraintes suivant qu'elles soient strictes ou pas. Les contraintes *strictes* sont des obligations à respecter impérativement, alors que les contraintes dites « *relâchables*» (appelées aussi préférences) peuvent éventuellement ne pas être satisfaites.

1.4.3.4 Les objectifs

Aux yeux de la production, la qualité d'un ordonnancement n'est mesurée qu'au travers de la mesure de différents indicateurs de performance. Ces indicateurs nous permettent de juger si l'ordonnancement trouvé permet bien d'atteindre les objectifs qui optimisent le triptyque « coût-qualité-délais ».

Les objectifs dits aussi, critères d'évaluation sont donc les indicateurs de performance sur lesquels se base le choix d'un ordonnancement satisfaisant. Ainsi, le choix d'un

ordonnancement parmi les ordonnancements réalisables se fait en fonction d'un ou de plusieurs critère(s) que l'on cherche à optimiser.

Les critères à optimiser sont exprimés sous forme d'une fonction appelée fonction objectif. Elle peut être une fonction de minimisation, ou de maximisation d'un ou de plusieurs critères. Si l'on considère qu'un seul critère, il s'agit d'un problème d'optimisation mono-objectif ; sinon, ça va être un problème d'optimisation multi-objectif.

Dans les problèmes d'ordonnancement d'ateliers, divers objectifs (critères) existent. Ils sont liés aux temps, aux ressources ou bien aux coûts.

1. **les objectifs liés au temps** : c'est la catégorie des objectifs les plus étudiés en ordonnancement. Pour le cas de l'ordonnancement d'un ensemble 'J' de n job, $J = \{J_1, \dots, J_i, \dots, J_n\}$, nous pouvons citer les critères les plus classiques suivants :

a) Minimisation d'une fonction maximale: critères «minimax»

- Les critères basés uniquement sur la date de fin d'exécution (*Completion time*) :
  - $C_{max} = \max_{i=1,\dots,n}\{C_i\}$, appelé communément « *Makespan* ». Il est égal à la date de fin du dernier job réalisé. Il représente la durée totale de l'ordonnancement. De ce fait, c'est un critère à minimiser.
  - $F_{max} = \max_{i=1,\dots,n}\{F_i\}$ avec $F_i = C_i - R_i$, c'est le temps maximal qu'un job peut passer dans un atelier. Appelé aussi la plus grande durée de séjour d'un job (*Flow time*).

- Les critères basés sur la date d'échéance (*Due date*) :
  - $L_{max} = \max_{i=1,\dots,n}\{L_i\}$, le décalage maximal (*Lateness*) avec $L_i = C_i - D_i$
  - $T_{max} = \max_{i=1,\dots,n}\{T_i\}$, le retard maximal (*Tardiness*) avec $T_i = \max(0, C_i - D_i)$
  - $E_{max} = \max_{i=1,\dots,n}\{E_i\}$,l'avance maximale(*Earlyness*) avec $E_i = \max(0, D_i - C_i)$

b) Minimisation d'une somme de fonctions : critères de «minisum»

Généralement, les critères *minisum* sont plus difficiles à optimiser que les critères *minimax*

- $\sum_{i=1}^{n} C_i$ La somme des dates d'achèvement de tous les jobs. « *total completion time* ».

- $\sum_{i=1}^{n} F_i$ La somme des dates de séjour de tous les jobs. Appelée aussi « *total flow time* ». Minimiser ce critère revient à minimiser les encours. Parce que la quantité d'encours est en rapport avec le temps de présence (séjour) des jobs dans le système.
- $\sum_{i=1}^{n} T_i$ La somme des retards de tous les jobs. Appelée aussi « *total tardiness*».
- $\sum_{i=1}^{n} U_i$ le nombre de jobs en retards. Avec $U_i = 0$ si $C_i \leq D_i$, sinon $U_i = 1$.

Lorsqu'on a besoin de mettre en évidence les différents degrés d'importances des jobs ou lorsqu'on a besoin de tenir compte des coûts de présence de ces jobs dans le système, différentes pondérations w_i peuvent être attachées aux grandeurs temporelles citées ci-dessus. Les précédents critères deviennent :

- $\sum_{i=1}^{n} w_i C_i$ La somme pondérée des dates de fin d'exécution de tous les jobs
- $\sum_{i=1}^{n} w_i F_i$ La somme pondérée des durées de séjour de tous les jobs.
- $\sum_{i=1}^{n} w_i T_i$ La somme pondérée des retards de tous les jobs.

2. **Les objectifs liés aux ressources** : ils correspondent par exemple à :
   - La maximisation de la charge totale ou moyenne de chaque ressource. C'est-à-dire chercher à augmenter le taux d'utilisation des ressources.
   - La minimisation de nombre maximal ou moyen de ressources nécessaires pour réaliser un ensemble d'opérations.
   - L'optimisation des changements d'outils.

3. **Les objectifs liés aux coûts** : c'est des objectifs moins « normalisés » dans la littérature. Cependant, ils peuvent s'exprimer sous des formes très variées comme par exemple la minimisation des encours, la minimisation du coût de lancement, de production, de stockage, ou de transport.

Par la suite, il ne sera essentiellement question que du « M*akespan* » comme fonction objectif. Cette fonction étant le critère le plus souvent utilisé pour évaluer le coût d'un ordonnancement.

1.4.4 Les notations

Il existe une très grande variété de problèmes d'ordonnancement. Pour faciliter leur identification et leur classification, une notation standard a été proposée par Graham et al.

[1979] et par Blazewicz et al. [2007]. Cette notation est constituée de trois champs $\alpha|\beta|\gamma$. Nous allons définir ces trois champs et plus de détails seront présentés sous forme de tableaux descriptifs, tels qu'ils ont été présenté par Laribi [2018].

- **Le champ $\boldsymbol{\alpha}$** décrit l'environnement des machines utilisées. Il est constitué de deux éléments : $\alpha = \alpha_1\alpha_2$, où :
  - α_1 : représente le type d'atelier.
  - α_2 : ce paramètre indique le nombre de machines utilisées. Si α_2 n'est pas précisé, le nombre de machines est quelconque.

Plus de détails en Tableau 1-2 et Tableau1-3.

- **Le champ $\boldsymbol{\beta}$** : $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7$, il représente l'ensemble des caractéristiques du processus. Il est utilisé pour préciser les contraintes du problème et les différentes hypothèses sur le mode d'exécution des tâches. Plus de détails en Tableau 1-2 et Tableau1-4.

- **Le champ $\boldsymbol{\gamma}$** : indique le critère d'optimisation, il peut donc prendre de nombreuses valeurs et peut être une combinaison de plusieurs critères. Plus de détails sur ces critères sont donnés en section (1.4.3.4)

Champ	Variante	Description	Notation
α	(α_1) (α_2)	Type d'atelier Nombre de machines	$\{\emptyset, 1, P, Q, R, F, J, O, FH, JH, OG\}$ $\{\emptyset, m\}$
β	(β_1) (β_2) (β_3) (β_4) (β_5) (β_6) (β_7)	Mode d'exécution des jobs Ressources supplémentaires Relation de précédence Date de disponibilité Durées opératoires Date d'échéance Propriété d'attente	$\{\emptyset, pmtn\}$ $\{\emptyset, res\}$ $\{\emptyset, prec, tree, cahins\}$ $\{\emptyset, r_i\}$ $\{\emptyset, p_i = p\}$ $\{\emptyset, d_i\}$ $\{\emptyset, nwt\}$
γ		/	$\{C_{max}, L_{max}, T_{max}, \Sigma U_i, \Sigma w_i C_i, \Sigma w_i T_i, \Sigma w_i U_i\}$

Tableau 1-2 : Classification de Graham.

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

Tableau 1-3 : Interprétation des notations du champ α_1.

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

Tableau 1-4 : Interprétation des principales notations possibles de sous-champs du champ β.

1.5 Notions sur la théorie de la complexité

Il a été constaté que certains problèmes d'ordonnancement peuvent être résolus en des temps records et ce, même pour un grand nombre de données traitées. Alors que pour d'autres, les approches existantes ne peuvent traiter, dans un temps raisonnable, qu'un nombre restreint de données. La théorie de la complexité (à ne pas confondre avec la complexité d'un algorithme) a donc été établie pour catégoriser la difficulté des problèmes. Elle a été développée à partir des travaux de Cook [1971] et de Karp [1972]. Elle représente un outil important dans la théorie de 1'ordonnancement.

La théorie de la complexité propose un ensemble de résultats et d'outils pour déterminer la difficulté intrinsèque à résoudre certains problèmes. Elle permet de classer "du point de vue mathématique" les problèmes selon leur difficulté, à partir de l'analyse des temps de résolution du problème. Elle nous permet ainsi, de juger si un problème est considéré comme « faciles » ou « difficiles » [Brucker, 2009]. chaque problème appartient à une classe de

complexité, qui nous renseigne sur la complexité du «meilleur algorithme» capable de le résoudre.

En théorie de la complexité, on peut distinguer la complexité de l'algorithme et la complexité du problème. Par la suite, nous allons présenter quelques notions élémentaires sur ces deux types de complexité. Pour plus de détails et d'informations, nous invitons le lecteur à se référer à [Garey et Johnson, 1979], [Papadimitriou, 1994] et à [Du et Ko, 2000].

1.5.1 Complexité des algorithmes

Une méthode de résolution est appelée « *algorithme* ». Il peut être défini de la manière suivante : « Un algorithme de résolution d'un problème P donné est une procédure décomposable en opérations élémentaires, transformant une chaîne de caractères représentant les données de n'importe quel exemple du problème P en une chaîne de caractères représentant les résultats de P » [Sakarovitch,1986].

Une fois qu'un algorithme de résolution est trouvé, il faut déterminer son degré de complexité et son efficacité. Cela revient à évaluer le nombre d'opérations de base de cet algorithme, en fonction de la quantité des données à traiter.

En effet, la connaissance du temps d'exécution et de la place mémoire prise par les opérations de l'algorithme pour résoudre un problème, permet de calculer cette complexité.

Les complexités en temps et en mémoire d'un algorithme s'expriment en fonction du nombre N de données appelé aussi taille du problème. N peut représenter le nombre de produits, de machines, de tâches ou encore une combinaison de ces quantités.

Le calcul de la complexité en mémoire consiste à calculer le nombre de variables ou de cases mémoires utilisées.

La complexité du temps de calcul d'un algorithme est donnée par le nombre d'étapes nécessaires pour résoudre le problème. Étant donné que ce nombre peut varier en fonction de l'instance étudiée, la complexité de l'algorithme est définie par rapport au pire cas. La notation $O(.)$ est utilisée pour représenter cette complexité.

La complexité temporelle d'un algorithme (dans le pire cas) est le temps mis pour transformer l'ensemble de données de taille N en un ensemble de résultats. Cette complexité est représentée par la fonction $T(N)$. Pour simplifier, on ne s'intéressera qu'au taux de croissance de $T(N)$. Il suffira alors de suivre seulement le comportement asymptotique de la complexité temporelle d'un algorithme qui sera décrit par la fonction $O(.)$. Par exemple, pour

un algorithme donné, si la solution est obtenue en N opérations, on dit que l'algorithme a une complexité $O(N)$.

Définition 1.1 : Un algorithme est dit polynomial si pour tout N, l'algorithme s'exécute en moins de $N^{\propto}$ opérations élémentaires ($\propto$ étant une constante).

En d'autres termes, un algorithme polynomial est un algorithme dont la complexité temporelle est bornée par un polynôme p de N, $O\big(p(N)\big)$: par exemple $O(N^{\propto})$ avec $\propto$ une constante.

Définition 1.2 : Un algorithme est dit non-polynomial si le nombre d'opérations n'est pas borné par un polynôme de N.

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

Maintenant, reste à savoir : « comment définir un algorithme efficace ? ».

Pour un problème donné, chercher un algorithme efficace, veut dire trouver un algorithme où le temps nécessaire à son exécution ne soit pas trop important. Un problème est dit facile si on peut le résoudre facilement, c'est-à-dire s'il ne nécessite pas beaucoup de temps pour arriver à la solution. Donc, s'il existe un algorithme efficace pour un problème donné, ce dernier est dit facile. Un problème pour lequel on ne connaît pas d'algorithme efficace, est dit difficile. Le degré de difficulté d'un problème est estimé selon son appartenance à l'une des classes définies par la théorie de la complexité.

1.5.2 Complexité des problèmes

En mathématiques et en informatique, plusieurs types de problèmes peuvent être distingués par les deux principales catégories de problèmes suivantes:

- *Les problèmes de décision* : qui se définissent par un nom, une instance qui est une description de tous les paramètres et une question à laquelle la réponse peut être uniquement « oui » ou « non ».
- *Les problèmes d'optimisation* : ce sont des problèmes définis par un nom, une instance et dont le but est de chercher la meilleure solution admissible qui optimise (maximise ou minimise) la valeur d'une fonction objectif donnée. ils posent une question de type « trouver une solution telle que... ».

Chaque problème d'optimisation possède un problème de décision correspondant. Par exemple, si l'on considère le problème d'optimisation $P||C_{max}$ avec n job et y un entier positif. Un problème de décision associé à ce problème d'optimisation peut s'énoncer comme suit : « existe-t-il un ordonnancement avec $C_{max} \leq y$? ».

Dans ce cas, s'il existe un algorithme polynomial permettant de donner la réponse « oui » à cette question, alors, il existera sûrement un algorithme polynomial qui permet de résoudre aussi le problème d'optimisation associé. Cette propriété nous permet de classer les problèmes d'optimisation grâce à leurs problèmes de décision. De plus, selon la complexité, les problèmes peuvent être organisés en plusieurs classes [Sakarovitch 86], comme suit :

- **Classe des problèmes les plus difficiles :**

  C'est la classe des problèmes indécidables pour lesquels il n'existe pas d'algorithme pour leur résolution.

- **Classe P (Polynomiale) :** Elle fait référence à l'ensemble de tous les problèmes de décision qui peuvent être résolus en temps polynomial, au moyen d'un algorithme déterministe et polynomial. Un algorithme est déterministe si le nombre d'étapes nécessaires pour résoudre le problème peut être calculé à l'avance. Cette classe contient tous les problèmes dits « faciles ». Malheureusement, la classe P des problèmes que l'on peut résoudre en un temps polynomial est assez limitée.
- **Classe NP (Non-deterministic Polynomial) :** En fait, cette classe a un nom trompeur, NP ne correspond pas à Non Polynomial, mais à Polynomial Non-déterministe ou à « Non-deterministic Polynomial time » en anglais. La classe NP est associée à l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par un algorithme Non-déterministe. La caractéristique d'un algorithme Non-déterministe est que l'on ne peut pas prédire avec certitude le nombre d'étapes nécessaires pour résoudre le problème. Parmi la classe des problèmes NP, on distingue deux grandes sous-classes : la classe des problèmes polynômiaux (la classe P) et la classe des problèmes NP-complet (la classe NP-complet). La relation entre les différentes classes mentionnées est illustrée dans la Figure 1-8.

- **Classe NPC :** Elle est contenue dans la classe NP et regroupe les problèmes dits **NP-Complets** qui représentent les problèmes les plus difficiles de la classe NP. C'est la classe des problèmes dont on n'a pas trouvé d'algorithme polynomial pour les résoudre, sous

l'hypothèse P≠NP. Cependant, si on pouvait résoudre un problème de NPC en un temps polynomial, on obtiendrait automatiquement des algorithmes polynômiaux pour tous les problèmes de NPC.

Quand un problème est NP-complet, il n'est pas raisonnable d'espérer construire un algorithme polynomial le résolvant. Mais dans certains cas, on peut construire des algorithmes très efficaces appelés algorithmes pseudo-polynomiaux, le problème étudié est alors dit NP-complet *au sens faible*. Dans le cas contraire, il est dit NP-complet *au sens fort*.

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

  Le problème d'optimisation correspondant à un problème de décision NP-complet est dit NP-difficile, pour lequel la résolution se fait en temps exponentiel, c'est-à-dire que le temps d'exécution augmente de façon exponentielle en fonction de la taille de l'instance.

  Nous pouvons encore distinguer entre problèmes NP-difficiles *au sens faible* et problèmes NP-difficiles *au sens fort*. Les premiers sont des problèmes pour lesquels il existe un algorithme capable de les résoudre en temps polynomial, pas en fonction de la taille de l'instance, mais en fonction de la longueur des paramètres (par exemple, les demandes et les durées opératoires). Ce type d'algorithme est appelé pseudo-polynomial. S'il n'existe pas un tel algorithme, le problème est NP-difficile *au sens fort*.

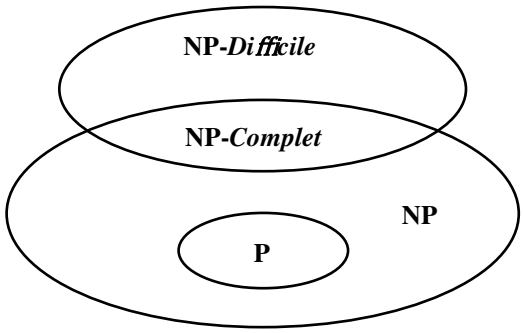

Figure 1-8 : Les principales classes de complexité.

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

1-5, donne un recueil de classes de complexité des problèmes d'ordonnancement mono-critère sans contraintes où P représente la classe P, NP : classe NP- difficile *au sens fort* et classe NP-difficile *au sens faible*.

Pour des résultats de complexité des problèmes d'ordonnancement mono-critère avec considération de contraintes, nous référons le lecteur au site web http://www2.informatik.uni-osnabrueck.de/knust/class/ développé par [Brucker et Knust, 2009]. Pour la complexité des problèmes d'ordonnancement multicritère, le lecteur intéressé peut se référer à [T'kindt et Billaut, 2006].

	Critères		C_{max}	L_{max}	ΣC_i	$\Sigma w_i C_i$	ΣT_i	$\Sigma w_i T_i$	ΣU_i	$\Sigma w_i U_i$
Problèmes	Une seule machine		P	P	P	P	NP*	NP	P	NP*
	Machine parallèles	*P*	NP	NP	P	NP	NP	NP	NP	NP
		Q	NP	NP	P	NP	NP	NP	NP	NP
		R	NP	NP	P	NP	NP	NP	NP	NP
	Flow shop		NP	NP	NP	NP	NP	NP	NP	NP
	Job shop		NP	NP	NP	NP	NP	NP	NP	NP
	Open shop		NP	NP	NP	NP	NP	NP	NP	NP
	Flow shop hybride		NP	NP	NP	NP	NP	NP	NP	NP
	Job shop flexible		NP	NP	NP	NP	NP	NP	NP	NP
	Open shop généralisé		NP	NP	NP	NP	NP	NP	NP	NP

Tableau 1-5 : Classe de complexité des problèmes d'ordonnancement mono-critère.

Pour résoudre un problème d'ordonnancement, il ne suffit pas de prouver l'existence d'une solution, il faut également la construire, il est clair que construire la solution est plus difficile que de prouver son existence. La question qui se pose maintenant : « comment définir un algorithme efficace permettant d'avoir cette solution ? »

1.6 Conclusion

Ce chapitre nous a permis d'énoncer quelques définitions relatives aux systèmes de productions et leurs différentes classes. Nous avons abordé les trois principaux problèmes de la gestion de production, à savoir : les problèmes : de conception, de planification et de contrôle. Cependant, plus de détails ont été donnés sur les trois niveaux de décisions (stratégiques, tactiques et opérationnelle), pour pouvoir ainsi, situer notre problématique par rapport aux autres problèmes de la gestion de production. En effet, notre intérêt dans cette thèse se focalise sur l'odonnancement des ateliers.

Par la suite, nous avons présenté l'ordonnancement en général et l'ordonnancement pour les ateliers de production en particulier. A ce niveau, des définitions, des notations et une classification pour les problèmes d'ordonnancement, ont été données.

Pour résoudre un problème d'ordonnancement de manière efficace, il faut prendre en compte la particularité de chaque problème. C'est pour cette raison que nous avons donné des notions sur la théorie de la complexité. Une théorie qui nous propose un ensemble de résultats et d'outils pour déterminer la difficulté intrinsèque à résoudre certains problèmes.

Généralement lors de l'étude d'un problème d'ordonnancement, on commence par chercher à classer le problème. Si on parvient à montrer qu'il est polynomial, le problème sera résolu facilement, par contre s'il s'avère plutôt NP-difficile, il restera à mettre au point des méthodes de résolution adaptées. Ceci fera l'objet de notre contribution dans cette thèse.

Chapitre 2

L’ordonnancement d’un Job shop

Le seul temps qu'on ne perd pas,

c'est le temps que l'on prend

Auteur inconnu

2.1 Introduction

Le problème d'ordonnancement de type Job shop (JSP : Job shop Scheduling Problem), est un problème très important dans le domaine de la gestion de production. Pour survivre dans un marché moderne et compétitif, qui nécessite des coûts moindres et un raccourcissement dans les cycles de vie des produits, la société doit répondre rapidement et précisément aux demandes du client. Un ordonnancement efficace aurait un rôle très important dans cette adaptation.

Notre objectif dans ce chapitre est de préparer un recueil des informations nécessaires pour étudier et traiter un problème d'ordonnancement de Job shop. Nous allons commencer en premier lieu, par donner une définition de l'atelier Job shop avec les différentes notations nécessaires, les hypothèses considérées et les extensions possibles ainsi que quelques exemples d'ateliers Job shop. En deuxième lieu, nous présenterons le problème d'ordonnancement de Job shop avec toutes les notations nécessaires et la taxonomie des représentations des solutions. Nous allons aborder aussi la notion de complexité afin de situer le degré de complexité d'un JSP. Une bonne section sera réservée à l'histoire du Job shop. Puis en dernier, on fera un petit passage sur les Benchmarks des ateliers Job shop.

2.2 L'atelier Job shop

2.2.1 Définition

Dans un atelier de type Job shop classique, le problème consiste à exécuter un ensemble de n jobs $J = \{J_1, \dots, J_n\}$ sur un ensemble de m machines $M = \{M_1, \dots, M_m\}$ (ressources). Le passage d'un job sur une machine est appelé opération. À chaque job J_i correspond une gamme opératoire composée d'un ensemble ordonné de m_i opérations noté $O_i = \{O_{i1}, O_{i2}, \dots, O_{im_i}\}$ pour lesquelles des contraintes de précédence sont décrites.

Le type et le nombre (m_i) des opérations dépendent du procédé de fabrication de chaque job J_i. Pour le passage d'un job sur les machines, on peut distinguer trois situations :

a) Le job J_i passe par toutes les machines et passe une seule fois ($m_i = m$),
b) Le job J_i ne passe pas par toutes les machines ($m_i < m$),
c) Le job J_i peut passer par toutes les machines et peut revenir plusieurs fois sur la même machine. C'est le phénomène de recirculation.

O_{ij} est la *j*-ième opération du *i*-ième job (J_i). Pour préciser que l'opération O_{ij} est exécutée sur la *k*-ième machine (M_k), on la note O_{ij}^k. Une opération O_{ij} est associée à un temps de traitement p_{ij} correspondant à sa durée d'exécution sur la machine. Chaque opération est définie alors par un couple $\langle machine, durée \rangle$, $O_{ij}^k = \{(M_k, p_{ij})\}$. Le temps nécessaire pour que toutes les opérations soient exécutées est appelé Makespan (C_{max}).

Etant donné d'un job J_i possède sa propre gamme opératoire O_i, forcément, dans l'atelier, il suivra un chemin propre à lui. De ce fait, pour l'exécution de tous les jobs, différents chemins seront empruntés (flux multidirectionnel). D'où la deuxième appellation de l'atelier Job shop qui est : « atelier à cheminements multiples ».

Un atelier Job shop composé de *n* jobs et de *m* machines, est dit : Job shop de dimension ($n \times m$), ou Job shop d'instance ($n \times m$), ou tout simplement Job shop ($n \times m$). La Figure 2-1 illustre un exemple d'un atelier Job shop à 3 jobs et 5 machines en indiquant le séquencement des jobs sur les machines. La matrice de séquencement des machines pour chaque job est donnée par le Tableau 2-1. Par exemples : le job J_1 possède le routage $(M_1, M_2, M_3, M_5, M_4)$. le job J_2 possède le routage $(M_5, M_1, M_2, M_4, M_3)$ et le routage du job J_3 est donné par la séquence $(M_2, M_3, M_4, M_5, M_1)$.

Job	Opérations				
	1	2	3	4	5
	le séquencement des Machines				
J_1	M_1	M_2	M_3	M_5	M_4
J_2	M_5	M_1	M_2	M_4	M_3
J_3	M_2	M_3	M_4	M_5	M_1

Tableau 2-1: Matrice des séquences de machines pour un Job shop (3×5).

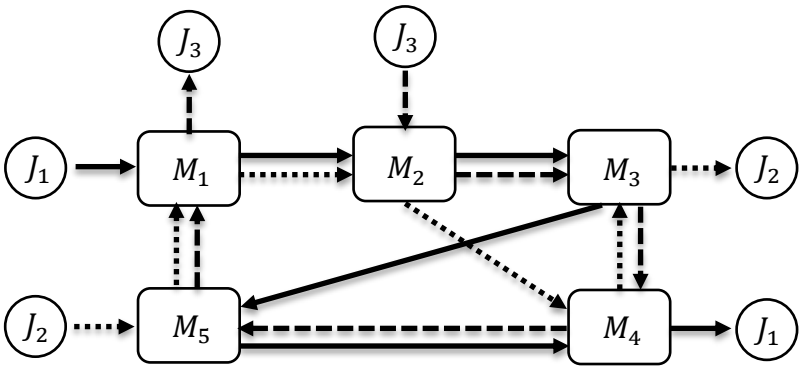

Figure 2-1 : Atelier Job shop à 3 jobs et 5 machines.

2.2.2 Caractéristiques d'un atelier Job shop

L'atelier Job shop peut se différencier des autres ateliers par les caractéristiques suivantes :

- Une grande variété de produits (adaptée aux exigences spécifiques de chaque client)
- Un volume de production peu élevé,
- Des équipements de production standards, peu automatisés et très flexibles.
- Un taux d'utilisation des équipements relativement faible,
- Un système de production par flux tirés (fabrication à la demande),
- Une main d'œuvre assez importante et qualifiée (l'ouvrier a les capacités de faire différentes opérations).

2.2.3 Exemples d'ateliers de type Job shop

Dans la pratique, plusieurs types de systèmes de production peuvent être identifiés comme étant des ateliers de type Job shop. Ainsi, l'étude de ces systèmes se basera sur le modèle théorique du Job shop qui fait l'objet de notre étude.

Parmi les exemples de systèmes que l'on pourrait considérer comme un atelier de type Job shop, on trouve :

- Un atelier de fabrication de pièces mécaniques,
- Un atelier de menuiserie,
- Une usine de production de semi-conducteurs,
- Un laboratoire d'analyses médicales,
- Un hôpital qui nécessite la planification de l'utilisation de ces ressources matérielles et immatérielles,
- La cuisine d'un restaurant,
- Un atelier de pâtisserie,
- Etc.

2.2.4 Hypothèses

Pour le cas d'un Job shop classique, on retient certaines hypothèses. En s'inspirant de la classification de Rinnooy Kan [1976], nous pouvons les classer dans les groupes suivants :

- **Hypothèses concernant les jobs :**
  - **H1** : Le nombre de jobs est connu et fixé.
  - **H2** : Tous les jobs ont la même importance.
  - **H3** : Tous les jobs sont disponibles dès le début de l'ordonnancement et prêts à être exécutés à tout moment.
  - **H4** : Chaque job peut être dans l'un des trois états suivants : en attente de la prochaine machine, en exécution sur une machine ou ayant déjà fini une opération sur la précédente machine.
  - **H5** : Il y a une relation de précédence entre deux opérations d'un même job mais pas entre deux opérations de jobs différents.
  - **H6** : Le routage d'un job est défini par sa gamme opératoire et par la pré-affectation des machines à chacune de ses opérations. Le routage est fixe et connu pour chaque job mais peut être différent d'un job à un autre.
  - **H7** : Il n'y a pas de date d'échéance (due date) imposée aux jobs.

- **Hypothèses concernant les machines :**
  - **H8** : Le nombre de machines est connu et fixé.
  - **H9** : Toutes les machines ont la même importance.

- **H10** : Toutes les machines sont toujours disponibles durant la période de l'ordonnancement (pas de pannes et pas de maintenance).
- **H11** : Chaque machine peut être dans l'un des trois états suivants : en attente d'un prochain job, en exécution d'un job ou ayant déjà fini l'exécution de l'opération d'un job.
- **H12** : Pas de machines alternatives.

- **Hypothèses concernant l'exécution des jobs (les opérations) :**
  - **H13** : Une machine ne peut exécuter qu'une seule opération à la fois.
  - **H14** : Une opération ne peut être exécutée que par une seule machine.
  - **H15** : Pas de préemption, pas de chevauchement. Chaque opération, une fois entamée. doit être terminée avant de pouvoir lancer une autre opération sur cette machine.
  - **H16** : Pas de recirculation. Chaque job comporte m_i opérations distinctes. Nous n'autorisons pas qu'un job fasse plus qu'une opération sur la même machine $(m_i \leq m)$.
  - **H17** : Une opération sur un job ne peut être effectuée tant que toutes les précédentes opérations de ce job ne sont pas terminées.
  - **H18 :** L'exécution d'une opération se fait sur un temps opératoire déterministe, positive, continu et connu à l'avance. Ce temps opératoire comprend aussi le temps de transport des jobs et le temps de réglage de la machine (s'ils existent).
  - **H19 :** Tous les temps opératoires sont fixes et indépendants de l'ordonnancement (sequence-independent).
  - **H20 :** Le temps nécessaire pour basculer entre deux opérations successives sur une machine est nul.
  - **H21 :** Le temps nécessaire pour le passage entre deux opérations successives sur une machine est nul.
  - **H22 :** Pas d'annulation. Chaque job doit être traité jusqu'à la fin.

En pratique, certaines de ces hypothèses, posées pour un Job shop classique, ne reflètent pas la réalité. Elles doivent être éliminées afin d'avoir un modèle de Job shop qui s'approche le plus possible de la réalité pratique. Un changement dans les hypothèses émises nous permet d'avoir des Job shop avec plusieurs variantes et donc le Job shop classique pourrait subir de nouvelles extensions.

2.2.5 Les extensions possibles du Job shop

Pour rendre le modèle de Job shop classique applicable aux problèmes réels, plusieurs extensions peuvent être considérées, chacune d'elles tente de couvrir un aspect pratique et donnera par conséquent d'autres variantes aux Job shops [Lee, 1997]. Ainsi, on peut avoir un Job shop avec :

- sequence-dependent ou setup times,
- machines parallèles (Multiprocessors),
- date de fin au plus tard (Job release times),
- capacité de buffers d'entrée/sortie limités et des contraintes de blocage (Limited buffers and blocking constraints),
- contraintes sans attente généralisées (Generalized no-wait constraints),
- flexibilité des machines (Processor flexibility),
- Inclusion des temps transport et de transfert (Transport and transfert times dependent).

2.3 Formulation d'un problème d'ordonnancement d'un Job shop

Dans le cas d'un atelier Job shop, les jobs empruntent différents routages (cheminements), mais puisqu'ils partagent des machines communes, des conflits sont susceptibles de survenir. Afin d'éviter ce genre de conflits, il est très important d'établir pour cet atelier, un ordonnancement qui va déterminer les dates de passage des jobs sur les différentes machines, tout en respectant les contraintes imposées, et en optimisant les objectifs poursuivis.

Le problème d'ordonnancement de l'atelier Job shop (JSP : *Job shop Scheduling Problem*) représente un cas particulier du problème d'ordonnancement général. Il est le plus étudié dans la littérature. En effet, il appartient à la catégorie des problèmes d'ordonnancement dans les ateliers à ressources multiples et à cheminement multiples.

La résolution des problèmes d'ordonnancements en général et en particulier ceux de l'atelier de type Job shop sont considérés parmi les problèmes les plus difficiles et les plus complexes. Leur difficulté relative se manifeste en fonction des données telles que leurs taille (nombre de machines et nombre de jobs) ainsi que la multitude de leurs contraintes (conjonctives, disjonctives, disponibilité etc.). En effet, ces données et ces contraintes constituent les éléments de bases servant à la bonne modélisation d'un problème d'ordonnancement dans l'atelier de type Job shop.

2.3.1 Notations

Les notations suivantes sont utilisées pour la formulation du JSP :

- *Indices et ensembles :*

  i Indices pour les jobs

  j Indices pour les opérations

  k Indices pour les machines

  O_{ij} j-ième opération du job J_i

  J Ensemble des n jobs à réaliser, $J = \{J_1, \dots, J_i, \dots, J_n\}$

  M Ensemble des m machines disponibles, $M = \{M_1, \dots, M_k, \dots M_m\}$

  O_i Ensemble ordonné de toutes les opérations du job J_i, $O_i = \{O_{i1}, \dots, O_{ij}, \dots, O_{im_i}\}$

  O_k Ensemble de toutes les opérations exécutées sur la machine M_k

- *Paramètres*

  p_{ij} Durée d'exécution de l'opération O_{ij}, (operating time ou processing time)

  L Un grand nombre

- *Variables :*

  $\alpha_{ijk} = \begin{cases} 1 & \text{Si } O_{ij} \text{ est excutée sur la machine } M_k \\ 0 & \text{Sinon} \end{cases}$

  $\beta_{iji'j'} = \begin{cases} 1 & \text{Si } O_{ij} \text{ précède } O_{i'j'}, \quad \text{pour } O_{ij}, O_{i'j'} \in O_k, i \neq i' \text{ et } j \neq j' \\ 0 & \text{Sinon} \end{cases}$

  r_i Date de disponibilité du job j_i, (release date)

  r_{ij} Date où l'opération O_{ij} est disponible pour être exécutée

  w_{ij} Temps d'attente pour l'opération O_{ij}, (waiting time)

  s_{ij} Date de début de l'opération O_{ij}, (starting time), $S_{ij} = r_{ij} + w_{ij}$

  C_{ij} Date de fin de l'opération O_{ij} (completion time of operation O_{ij}). $C_{ij} = s_{ij} + p_{ij}$

  C_i Date de fin d'exécution du job j_i, (completion time of job j_i)

  C_{max} Date de fin de tous les jobs, (Makespan), $C_{max} = max\{C_1, C_2, \dots, C_n\}$

2.3.2 L'objectif du JSP

De manière générale, le coût global de production est lié au temps nécessaire à la fabrication des différents produits (jobs). En conséquence, l'objectif principal du problème d'ordonnancement de Job shop (JSP) consiste à réduire la durée globale de fabrication, appelée « Makespan » et notée C_{max}. Le C_{max} représente la date de fin de tous les jobs.

$$C_{max} = \max \{C_1, C_2, \dots, C_n\}$$

$$C_{max} = \max_{\substack{1 \le i \le n \\ 1 \le j \le m_i}} (s_{ij} + p_{ij})$$

Pour résoudre le JSP, nous devons déterminer un ordonnancement admissible (réalisable) pour les opérations de tous les jobs de manière à respecter l'ordre de traitement de chaque opération et la capacité de chaque machine. Cependant, des ordonnancements différents donnent généralement des Makespan différents. Par conséquent, l'objectif est de trouver un ordonnancement admissible, qui donne le plus petit Makespan, C^*_{max}.

$$C^*_{max} = \min_{\substack{\text{ordonnancements} \\ \text{admissibles}}} (C_{max})$$

La résolution d'un problème d'ordonnancement de Job shop revient alors à résoudre un problème d'optimisation dans lequel on doit déterminer les variables de décision qui sont dans ce cas, les dates de début s_{ij} pour chaque opération O_{ij} et ceci afin de satisfaire le critère qui minimise le C_{max} sous respect des contraintes de précédence.

Mathématiquement parlant, le critère d'optimalité le plus souvent étudié dans le JSP est la minimisation du C_{max}. En effet, un délai de fabrication minimum implique généralement une bonne utilisation des machines et une augmentation de la productivité ; car un nombre donné de jobs va être achevé dans les plus courts délais.

Le JSP avec la minimisation du Makespan comme critère, peut être noté $n|m|J|C_{max}$, selon la notation de Conway et al. [1967]. Ou bien, $J||C_{max}$, Selon la classification des 3-champs de Graham et al. [1979].

La dimension du problème est égale à *D* qui représente le nombre de toutes les opérations à ordonnancer, avec :

$$D = \sum_{i=1}^{n} m_i \qquad (2.1)$$

On attribue aux différentes instances des Job shop la notation $(n \times m)$ avec n le nombre de jobs et m le nombre de machines. Puisqu'on suppose que chaque job passe une seule fois sur

toutes les m machines, le nombre des opérations m_i est égal au nombre de machines $(m_i = m)$. Dans ce cas :

$$D = nm \tag{2.2}$$

La difficulté de trouver de bonnes solutions pour le JSP est due au nombre total de toutes les solutions possibles, un nombre étant égal à $(n!)^m$. Pour les problèmes de taille élevée, le nombre élevé de solutions possibles, rendrait l'exploration de tout l'espace des solutions presque impossible.

2.3.3 Le modèle mathématique

Dans la littérature de Pan [1997], on trouve que le Job shop peut être représenté sous différentes formulations, à savoir :

- La formulation de Manne [Manne,1960].
- La formulation Liao-You, [Liao et You,1992].
- La formulation adaptative de Manne, [Pan, 1997].
- La formulation de Wagner, [Wagner, 1959]
- La formulation de Wilson, [Pan, 1997].

D'après les travaux de Pham [2008], qui a fait une évaluation des formulations linéaires du Job shop, il trouve que la formulation de Manne reste la plus appropriée pour l'étude d'un JSP.

Inspirés de la formulation de Manne [1960], nous avons formulé la problématique de l'ordonnancement de Job shop à l'aide du modèle de programmation en nombres entiers suivant :

$$C^*_{max} = \underset{\substack{\text{ordonnancements}\\ \text{admissibles}}}{\text{Min}} \left\{ C_{max} = \max_{i=1,\dots,n}\{C_i\} \right\} \tag{2.3}$$

Équivalent à :

$$\underset{\substack{\text{Dâtes de début}\\ \text{admissibles}}}{\text{Min}} \left\{ \max_{i=1,\dots,n} \left\{ (s_{ij} + p_{ij}) \quad ; \quad \text{for } j = 1, \dots, m_i \right\} \right\} \tag{2.4}$$

Sous les contraintes suivantes :

$$s_{i(j+1)} - s_{ij} \geq p_{ij} \quad ; \quad \text{for } i = 1, \dots, n \ \& \ j = 1, \dots, (m_i - 1) \qquad (2.5)$$

$$\sum_{O_{ij} \in OM_k} \alpha_{ijk} = 1 \quad ; \quad \text{for } k = 1, \dots, m \qquad (2.6)$$

$$s_{i'j'} + L\left(1 - \beta_{iji'j'}\right) - s_{ij} \geq p_{ij} \quad ; \quad \text{for } k = 1, \dots, m \ \& \ O_{ij}, O_{i'j'} \in OM_k \qquad (2.7)$$

$$s_{ij} + L\beta_{iji'j'} - s_{i'j'} \geq p_{i'j'} \quad ; \quad \text{for } k = 1, \dots, m \ \& \ O_{ij}, O_{i'j'} \in OM_k \qquad (2.8)$$

$$\sum_{k=1}^{m} \alpha_{ijk} = 1 \quad ; \quad \text{for } i = 1, \dots, n \ \& \ j \ 1, \dots, m_i \qquad (2.9)$$

$$\sum_{j=1}^{m_i} \alpha_{ijk} \leq 1 \quad ; \quad \text{for } i = 1, \dots, n \ \& k = 1, \dots., m \qquad (2.10)$$

$$\sum_{j=1}^{m_i} p_{ij} \leq C_i \quad ; \quad \text{for } i = 1, \dots, n \qquad (2.11)$$

$$s_{ij} \geq r_{ij} \quad ; \quad \text{for } i = 1, \dots, n \ \& \ j = 1, \dots, m_i \qquad (2.12)$$

$$\beta_{iji'j'} + \beta_{i'j'ij} \leq 1 \quad ; \quad \text{for } k = 1, \dots, m \ \& \ O_{ij}, O_{i'j'} \in OM_k \qquad (2.13)$$

$$s_{ij} \geq 0 \quad ; \quad \text{for } i = 1, \dots, n \ \& \ j = 1, \dots, m_i \qquad (2.14)$$

$$\alpha_{ijk}, \beta_{iji'j'} \in \{0,1\} \quad ; \quad \text{for } k = 1, \dots, m \ \& \ O_{ij}, O_{i'j'} \in OM_k \qquad (2.15)$$

On peut expliciter ce modèle comme suit :

- La fonction objectif est donnée par (2.3).
- Le premier ensemble de contraintes (2.5) garantit à la fois les contraintes de précédence et les contraintes de non-préemption. En effet, l'opération $O_{i(j+1)}$ ne peut pas démarrer avant la fin de l'opération O_{ij}. (Hypothèses : H5, H6, H15 et H17).
- Le second ensemble de contraintes (2.6) impose les contraintes de capacité pour lesquelles une machine ne peut traiter qu'une seule opération à la fois (hypothèse H13).
- Les contraintes disjonctives (2.7) et (2.8) renforcent la contrainte (2.6). Elles garantissent un certain ordre dans l'exécution des opérations de différents jobs qui nécessitent d'être traitées sur la même machine. Pour une machine M_k, un ordonnancement réalisable doit satisfaire soit $s_{i'j'} - s_{ij} \geq p_{ij}$ ou $s_{ij} - s_{i'j'} \geq p_{i'j'}$. Pour toutes $O_{ij}, O_{i'j'} \in O_k$. Effectivement, si O_{ij} précède $O_{i'j'}$ sur la machine M_k,

alors $\beta_{iji'j'} = 1$ et (2.7) devient $s_{i'j'} - s_{ij} \geq p_{ij}$ tandis que (2.8) devient redondante en raison de la grande valeur positive de L. D'autre part, si $O_{i'j'}$ précède O_{ij} sur la machine M_k, alors $\beta_{iji'j'} = 0$ et (2.8) devient $s_{ij} - s_{i'j'} \geq p_{i'j'}$ et (2.7) devient redondante. La valeur de L doit être assez grande pour satisfaire $L \geq s_{i'j'} + p_{i'j'} - s_{ij}$ pour toutes $M_k \in M$ et $O_{ij}, O_{i'j'} \in O_k$. Pour cela, $L = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{ij}$ est suffisant.

- Les contraintes (2.9) garantissent qu'une opération ne peut être traitée que par une seule machine (hypothèse H14).
- L'hypothèse de la non-recirculation (H16) est assurée par les contraintes (2.10)
- L'hypothèse de la non-annulation (H22) est donnée par les contraintes (2.11).
- Les contraintes (2.12) garantissent que chaque opération ne peut être exécutée qu'à partir de sa date de disponibilité.
- Les deux derniers ensembles de contraintes délimitent le domaine de définition de chaque variable.

2.4 La complexité du JSP

Pour les problèmes d'ordonnancement, qui sont généralement considérés comme des problèmes d'optimisation combinatoire, la question de la complexité est très importante. Avant d'essayer de développer un algorithme qui résout un problème particulier, il est tout d'abord important d'avoir une idée sur la complexité du problème afin de savoir s'il s'agit d'un problème *facile* ou *difficile* à résoudre.

Pour le problème d'ordonnancement d'un Job shop de *n*-job, *m*-machine, il y a certains cas restreints, pour lesquels, il existe des algorithmes polynomiaux permettant de trouver rapidement l'ordonnancement optimal. Garey et Johnson [1979] ont montré que les seuls cas pouvant être résolus efficacement sont les suivants:

- Le JSP avec deux jobs, $n = 2$.
- Le JSP à deux machines, où chaque job est composé de deux opérations au plus $m = 2$ et $m_i \leq 2,$ pour $i = 1, \ldots, n.$
- Le JSP à deux machines avec un temps opératoire égale à une unité de temps pour toutes les opérations. $m = 2$ et $p_{ij} = 1, \ \forall \ i, j.$
- Le JSP à deux machines avec un nombre fixe de jobs (les jobs subissent des traitements répétitifs sur ces machines).

Lenstra et Rinnooy Kan [1979] ont démontré que le JSP général devient un problème NP-difficile au sens fort, à partir de trois machines (pour $m \geq 3$). En effet, pour l'instance $(n \times m)$, le nombre de d'ordonnancements semi-actifs possibles est estimé à $(n!)^m$ et ainsi, il pourrait être excessif. Par exemple, pour une instance de Job shop (6×6), le nombre maximum de solutions possibles est : $(6!)^6 = 1.3931 \times 10^{17}$. Dans ce cas, pour trouver un ordonnancement optimal, il faudrait consommer un temps de calcul excessif, et même parfois il serait impossible d'en trouver un.

Pour avoir plus de détails sur la complexité de quelques classes de Job shop, le lecteur pourrait trouver dans [Yahouni 2017] un résumé bien concis et complet.

Comme c'est déjà mentionné ci-dessus, le problème de JSP ne peut être résolu par des méthodes exactes que pour de très petites instances pour n et m. Autrement, pour les plus grandes instances, où le problème devient fortement NP-difficile, nous devons utiliser des méthodes heuristiques ou métaheuristiques. Avec ces méthodes, la garantie de trouver des solutions optimales est sacrifiée dans l'espoir d'obtenir des solutions acceptables avec un temps de calcul considérablement réduit.

2.5 Représentation de la solution d'un JSP

Le JSP est un typique problème d'optimisation combinatoire où les solutions sont codées avec des variables discrètes. Ainsi, la solution de cette catégorie de problèmes peut se définir par $X = \{x_1, x_2, \dots, x_D\}$. D est la dimension du vecteur X et donc la dimension du problème à résoudre. $x_i \in Dx_i$, pour $i = 1, \dots, D$. Dx_i est le domaine de définition de la variable x_i.

Généralement, une solution réalisable est la solution qui satisfait toutes les contraintes du problème posé. Le but est de trouver parmi ces solutions réalisables la solution optimale qui minimise ou maximise (en fonction du problème) la fonction objectif considérée.

Pour trouver une solution optimale ou quasi optimale pour le JSP, des méthodes et des algorithmes spécifiques peuvent être utilisés. Mais, il est clair qu'avant d'appliquer une méthode, une représentation (ou un codage) appropriée de la solution doit être introduite.

2.5.1 Taxonomie des représentations de la solution d'un JSP

Dans leur article, Cheng et al. [1996] ont fait une présentation détaillée de toutes les méthodes de codage de solution destinées pour la résolution du JSP par les algorithmes génétiques (AG).

Ils classent les représentations de solutions en deux approches de codage de base: l'approche directe et l'approche indirecte. La distinction entre les approches directes et indirectes dépend du fait qu'une solution (un ordonnancement) est directement codée dans le vecteur de la solution X (chromosome pour les AG) ou non. De ce fait, toutes les représentations proposées pour le codage des solutions, au cours de ces dernières années, sont résumées pour les deux approches, comme suit:

- Approche directe :
  - représentation basée sur les opérations
  - représentation basée sur les jobs
  - représentation basée sur la relation de précédence entre une paire de jobs sur les machines correspondantes.
  - représentation basée sur la date d'achèvement (date de fin)
  - représentation basée sur des clés aléatoires.

- Approche indirecte:
  - représentation basée sur une liste de préférences
  - représentation basée sur des règles de priorité
  - représentation basée sur le graphe disjonctif
  - représentation basée sur les machines

Bien que ces représentations de solution aient été données pour les algorithmes génétiques utilisés pour le JSP, on peut noter qu'elles peuvent être également valides pour tout autre algorithme itératif. Pour notre étude, on opte pour la représentation basée sur les opérations.

2.5.2 Représentation basée sur les opérations

La représentation basée sur les opérations encode chaque solution possible (ordonnancement) avec un vecteur correspondant à la séquence d'opérations. Pour un problème de Job shop à n-job et m-machine, si chaque job consiste en m opérations, le vecteur de solution contient $(n \times m)$ variables et chaque variable correspond à une opération. Dans ce cas, l'espace de codage est constitué de toutes les permutations possibles de ces opérations.

La représentation basée sur les opérations peut être utilisée avec plusieurs variantes [Werner, 2011]. Pour les illustrer, considérons un exemple de JSP composé de 3 jobs et de 3

machines. Le Tableau 2-2 affiche les données de cet atelier. La représentation de la solution code un ordonnancement sous la forme d'une séquence de 9 opérations, sachant que l'énumération de ces opérations peut être donnée sous deux formes. Comme c'est indiqué dans le Tableau 2-3, les opérations sous la forme (a), sont numérotées de 1 à 9. Sous la forme (b) ces numéros correspondent à la notation (i, j), qui indique qu'il s'agit de l'opération du job i sur la machine j.

La Figure 2-2, illustre un exemple de solution à notre problème. La solution est codée avec quatre variantes de représentation basée sur les opérations :

- Dans la première variante, le vecteur de solution (V_1) est une *séquence des opérations* (i, j). À partir d'un tel vecteur, nous obtenons immédiatement l'ordre des jobs et des machines en fonction de la séquence d'opérations appartenant au même job ou à la même machine.
- Dans la deuxième variante qui se base sur *la répétition des jobs*, la k-ième occurrence du job j_i dans (V_2) fait référence à la k-ième opération de ce job, selon sa gamme opératoire indiquée dans le Tableau 2-2.

D'autre part, la troisième et la quatrième variante utilisent la numérotation des opérations sous les formes données dans le Tableau 2-3.

- Dans la variante de *séquence d'opérations unidimensionnelle*, le k-ième élément de (V_3) contient le numéro de l'opération à exécuter à la position k, par exemple, le numéro 9 dans l'élément 2 signifie que la 9-ième opération (3,3) est séquencée à la position 2.
- Par contre, dans la variante décrivant *les positions des opérations*, le k-ième élément de (V_4) contient la position de la k-ième opération dans le vecteur (V_1), par exemple, le nombre 7 dans l'élément 3 signifie que la 3-ième opération (1,3) est à la position 7 dans (V_1).

Job	Opérations 1	2	3
	Le séquencement des Machines		
J_1	M_2	M_1	M_3
J_2	M_1	M_2	M_3
J_3	M_3	M_1	M_2
	Temps opératoires		
J_1	1	4	4
J_2	4	3	2
J_3	3	2	1

Tableau 2-2 : Les données d'un Job shop, 3-job×3-machine.

	Enumération des opérations								
Forme (a)	1	2	3	4	5	6	7	8	9
Forme (b)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)

Tableau 2-3 : les deux formes d'énumérations pour toutes les opérations.

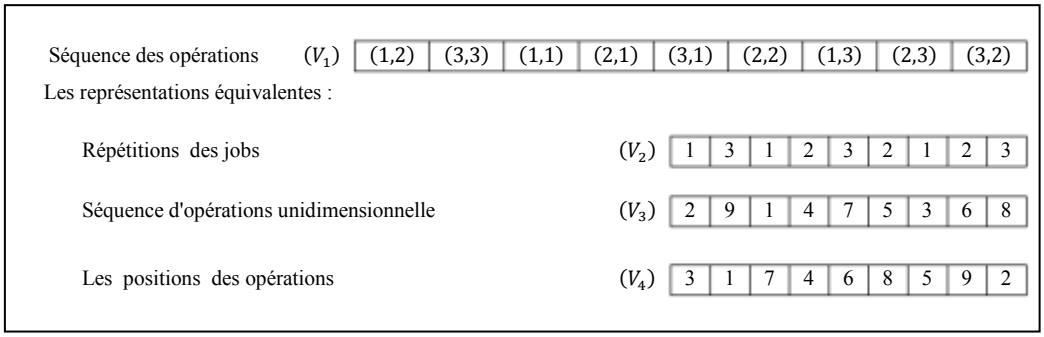

Figure 2-2 : Les quatre variantes de la représentation basée sur les opérations.

Idéalement, la représentation d'un problème devrait associer une seule solution codée (dans l'espace de codage) à une seule solution réalisable (dans l'espace de solutions). De plus, toutes les solutions possibles et réalisables devraient pouvoir être codées (couverture complète de l'espace de solutions). Cependant, dans certains cas, divers schémas de codage s'adressent non seulement à des solutions infaisables, mais également à des solutions illégales (c'est-à-dire des solutions en dehors de l'espace des solutions). Par conséquent, la représentation de chaque problème doit être choisie avec soin, et toujours en liaison avec la méthodologie utilisée pour le développement de la solution.

2.6 Diagramme de Gantt

Il y a plus de 100 ans, Henry Laurence Gantt a développé le fameux « diagramme de Gantt ». Jusqu'à nos jours, ce diagramme est considéré comme l'outil le plus simple et le plus utilisé pour visualiser graphiquement l'exécution des tâches d'un problème d'ordonnancement.

Le diagramme de Gantt est utilisé aussi bien dans la littérature que dans les entreprises. Dans le cas de problèmes d'atelier, ce diagramme se compose de plusieurs lignes horizontales, chacune d'entre elles désignant une machine.

Les opérations exécutées sur une machine donnée sont représentées sous forme de rectangles (barres) ayant des longueurs proportionnelles à leurs temps opératoires. L'axe des abscisses représente le temps. Les rectangles sont positionnés aux dates de début d'exécution des opérations. De cette manière, le diagramme de Gantt permet de montrer : les périodes d'occupation et d'inoccupation des machines dans le temps, les séquences de traitement sur chaque machine ainsi que les dates de fin des jobs (Makespan).

Si l'on reprend le problème considéré dans la section précédente avec les données du Tableau 2-2, on trouve sur la Figure 2-4, le diagramme de Gantt qui illustre la solution présenté en Figure 2-3.

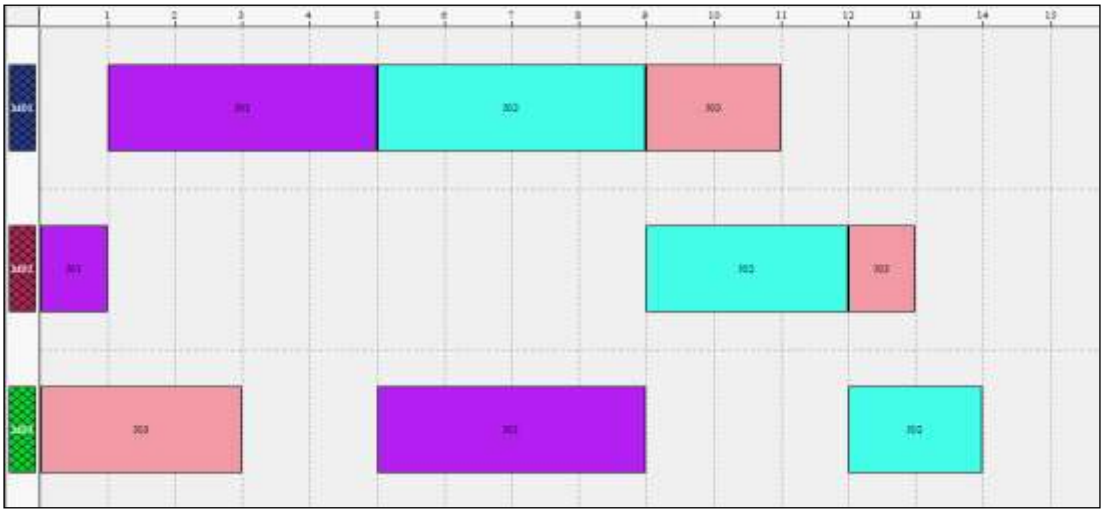

Figure 2-3 : Diagramme de Gantt pour l'ordonnancement réalisable construit à partir de la solution présentée à la Figure 2-3.

2.7 L'histoire du Job shop

Si l'on va chercher dans l'historique du problème de Job shop, on ne pourra pas se prononcer sur le premier qui a proposé le problème sous sa forme actuelle.

En 1964, Roy et Sussman [1964] ont été les premiers à proposer la représentation par graphe disjonctif et en 1969, Balas [1969] a été le premier à appliquer une approche énumérative basée sur le graphe disjonctif. Cependant, il se trouve qu'il y avait auparavant, des travaux déjà fait sur l'ordonnancement de Job shop. On a par exemple : les travaux de Giffler et Thompson [1960] qui ont proposé en 1960 un modèle d'acheminement de jobs suivant des règles de priorités ; on a Jackson [1956] qui en 1956, a généralisé pour le problème de Job shop, l'algorithme de Johnson déjà proposé en 1954 pour le problème de Flow shop [Johnson, 1954]. Akers et Friedman [1955] ont appliqué en 1955 l'approche de l'algèbre booléenne pour représenter les séquences des exécutions.

Ces travaux traitaient un problème qui consistait à exécuter *n* jobs sur *m* machines avec des contraintes de précédence. Ainsi, chaque job dans ce problème est traité par des machines dans un ordre différent. L'objectif dans tous les travaux déjà cités était d'exécuter tous les jobs le plus rapidement possible. Sans être surpris, on trouve que même ces travaux citaient des références qui existaient au préalable. Par exemple, Akers et Friedman [1955] citent en référence un document que Salvesen et Anderson [1951] ont préparé en 1951 sur la production industrielle. Bien qu'on ne soit pas sûr de la première personne qui a introduit le problème du Job shop, il est largement reconnu que le livre "*Industrial Scheduling*" édité en 1963, par Muth et Thompson [1963] et dans lequel les auteurs ont rassemblé tous les résultats des recherches de l'époque, est la base de la plupart des recherches menées par la suite.

On peut clairement supposer que les travaux sur le problème d'ordonnancement de Job shop (JSP) ont été lancés en 1954 par les travaux de Johnson [1954] qui sont considérés comme étant les premières dans la théorie de l'ordonnancement. Un algorithme optimal pour un Flow shop à deux machines a été développé pour ces recherches. Ensuite d'autres méthodes efficaces ont été proposées pour résoudre le JSP en un temps polynomial : en 1956 Akers [1956] a résolu le problème de Job shop de taille (2 × m) et Jackson [1956] celui de taille (n × 2) (où il n'y a pas plus de 2 opérations par job). Le JSP de taille n × 2 (où toutes les opérations ont un temps de traitement égale à l'unité) a été résolu en 1982 par la méthode de Hefetz et Adiri [1982].

Les années 50 furent l'une des périodes les plus productives dans les domaines des sciences de gestion et de la recherche opérationnelle. Pendant ce temps, de nombreux problèmes ont été résolus par l'application de méthodes heuristiques simples mais efficaces et qui formaient la base du développement de la théorie de l'ordonnancement classique.

Durant les années 60, l'intérêt s'est porté sur la recherche des solutions exactes par application d'algorithmes énumératifs adoptant des concepts mathématiques plus élaborés et sophistiqués. D'ailleurs, Rinnooy Kan [1976] a indiqué dans ses écrits que "*a natural way to attack scheduling problems is to formulate them as mathematical programming models*". Bien que ces stratégies soient de valeur théorique considérable, fournissant d'importants accomplissements en recherches, les résultats de plusieurs de ces travaux sont extrêmement décevants. La majorité de ces techniques sont incapables de parvenir à des solutions réalisables pour de nombreux problèmes et sont donc d'une utilité pratique limitée. Par conséquent, elles ne sont appliquées que dans le calcul de bornes inférieures.

Durant les années 70 et jusqu'au milieu des années 80, l'accent a été mis sur la justification de la Complexité du JSP. De nombreux travaux, à commencer par ceux de Cook en 1971 jusqu'à ceux de Lawler et al. en 1982. Ils ont clairement mis en évidence l'impossibilité de traiter le JSP en démontrant que seul quelques cas particuliers peuvent être résolus en un temps polynomial. Pour ces raisons, en 1995, Parker fait référence à l'ère d'avant 1970 comme étant l'ère de « *l'avant complexité* » notée « *BC* » (*Before compexity*). Par conséquent, l'ère qui s'en est suivi, est désignée par l'ère « *des difficultés avancées* » désignée par « *AD* (*Advanced Difficulty*) ». Les recherches sur la AD ont rapidement montré que les problèmes JSP résolus dans les années 50 pour des instances particulières, aussi bien que d'autres problèmes JSP, sont NP-difficiles dès qu'ils sont généralisés. Il n'est pas étonnant alors que Garey et Johnson citent en 1979, 320 cas de problèmes NP-difficiles.

En raison des limites auxquelles on est confronté avec les techniques énumératives exactes, les méthodes approchées sont devenues une bonne alternative. Bien que ces méthodes renoncent à la garantie d'avoir une solution optimale pour gagner en contre partie en temps de résolution, elles peuvent être utilisées pour résoudre plusieurs problèmes. La première méthode approchée fut l'heuristique qui considère les règles de priorité (PDR : Priority Dispatch Rules).

À la fin des années 80, il a été réalisé que l'heuristique basée sur les règles de priorité présentées certains défauts. Cependant, Il y avait un besoin croissant de trouver d'autres

techniques plus appropriées qui promettent des perspectives meilleures. Les travaux sur ces méthodes ont été initiés par Fisher et Rinnooy Kan [1988], où ils ont mis l'accent sur les propriétés fondamentales permettant de générer de bonnes techniques heuristiques, à savoir : la conception, l'analyse et l'implémentation. La pertinence de ces techniques a été reconnue en 1988 par le Comité CONDOR (Committee On the Next Decade of Operations Research) où il a indiqué qu'elles sont extrêmement prometteuses.

La courte période de 1988 à 1991, appelé « *the boom period* », a été une période de foisonnement pour les méthodes approchées. Certains des algorithmes les plus innovants ont été formulés lors de cette période.

Dans le prochain chapitre nous présenterons plus de détails sur les méthodes les plus utilisées jusqu'à nos jours pour la résolution du JSP.

Afin de pouvoir juger et comparer les différents algorithmes et techniques de résolution, il était préférable de les tester sur les mêmes problèmes. D'où il y a eu l'émergence des « *Benchmarks* » qui sont des « *problèmes de référence* » très intéressants et très utiles pour valoriser les recherches sur le JSP.

2.8 Les Benchmarks du Job shop

Afin de valider les modèles développés et surtout les méthodes et les algorithmes dédiés à la résolution d'un certain type de problèmes, il est souvent utile de les tester en utilisant des modèles de problèmes difficiles, dits benchmarks. Des benchmarks qui correspondent bien au type des problèmes en question.

Un benchmark, appelé aussi instance, est un modèle de référence pour des problèmes-types construits par plusieurs auteurs. Il permet de mesurer les performances d'une approche de résolution pour la comparer aux performances obtenues par d'autres approches utilisées pour le même benchmark.

Dans la littérature plusieurs benchmarks existent pour les problèmes de recherche opérationnelle. Ils sont regroupés sur le site de la « *Operations Research Library* » (OR-Library) où il est possible de les Télécharger [Beasley, 1990]. On y trouve aussi des résultats et des références des auteurs de ces benchmarks.

Concernant le Job shop, on peut télécharger un fichier contenant 82 exemples d'instances d'atelier Job shop de différentes tailles et ayant comme objectif la minimisation du Makespan. Les instances portent des noms composés des lettres qui représentent souvent les initiales des

noms de leurs auteurs et des chiffres pour les différencier entre elles. Elles sont composées de *n* jobs et de *m* machines et leur taille est donnée par (*n*×*m*).

Dans cette thèse notre travail consiste à mettre au point des méthodes de résolution pour le problème d'ordonnancement de Job shop. Ainsi, pour tester et valider ces méthodes nous allons considérer les benchmarks (instances) suivants (es) [Beasley, 2014]:

- 3 instances introduites en 1963 par **Fisher et Thompson** [1963], notées : {Ft06 (6x6), Ft10 (10x10), Ft20 (20x5)}.
- 5 instances introduites en 1988 par **Adams** et al. [1988], notées : {Abz5 à Abz9}
- 40 instances de différentes tailles (10x5, 15x5, 20x5, 10x10, 15x10, 20x10, 30x10 et 15x15). Introduites en 1985 par de **Lawrence** [1985],
- 10 instances introduites en 1991 par **Applegate et Cook** [1991], notées : {Orb01 à Orb10}.
- 20 instances introduites en 1992 par **Storer** et al.[1992], notées : {Swv01-Swv20}.
- 4 instances introduites en 1991 par **Yamada et Nakano** [1991], notées : {Yn1-Yn4}.

On trouve dans l'article de Jain et Meeran [1999] les résultats de recherches sur 242 benchmarks. Il y a certains pour lesquels la meilleure valeur du makespan, jugée optimale est déjà trouvée et d'autres sont encore des problèmes ouverts et donc seulement leurs bornes inférieure et supérieure sont fournies.

Récemment van Hoorn [2018] a publié un article dans lequel il a mis à jour les valeurs des meilleures solutions trouvées jusqu'à présent (BKS : *Best Known Solution*) pour tous ces benchmarks. Cette mise à jour a été faite à partir des résultats des nouveaux travaux qui ont pu améliorer les bornes inférieures et supérieures de certaines instances de Job shop. Il a même conçu un site web dans lequel il a ressemblé pour tous les benchmarks cités ci-haut : les données sur les temps opératoires, la matrice de séquencement, la meilleure valeur trouvée jusqu'à présent pour le makespan et même les solutions correspondantes [van Hoorn 2015].

2.9 Conclusion

Dans ce chapitre nous avons accordé une attention particulière au problème d'ordonnancement de Job shop (JSP : Job shop scheduling problem). On a commencé par définir l'atelier Job shop avec toutes ses caractéristiques. Quelques exemples de systèmes pouvant être considérés comme des ateliers Job shop ont été donnés aussi.

Pour considérer un Job shop classique, certaines hypothèses sont émises. Un changement dans ces hypothèses nous permet d'avoir des Job shop avec plusieurs variantes, ainsi, le Job shop classique pourrait subir de nouvelles extensions qui le rapprocheraient de la réalité pratique.

Par la suite, nous avons présenté les notations et le modèle qui correspond au problème d'ordonnancement de Job shop classique. Après avoir présenté certaines représentations de solutions destinées au JSP, plus de détails ont été donnés pour la représentation de solution basée sur les opérations avec répétitions de jobs. C'est la représentation qui sera utilisée par la suite car elle génère toujours des solutions qui respectent la contrainte de précédence et de cette manière toutes les solutions générées seront admissibles.

On a défini la classe de complexité du JSP. Il a bien été démontré par Lenstra et Rinnooy Kan [1979] qu'à partir de trois machines, le JSP devient un problème NP-difficile au sens fort. De ce fait, le JSP est connu pour être un des problèmes d'ordonnancement combinatoire les plus difficiles à résoudre.

Par conséquent, la résolution du JSP nécessite la mise au point de méthodes bien adaptées à son degré de complexité (difficulté). Dans ce chapitre, on s'est limité à donner juste un petit aperçu historique sur l'évolution des travaux faits pour résoudre le JSP. Dans le prochain chapitre nous allons définir, d'une manière générale, les différentes méthodes de résolution d'un problème d'optimisation combinatoire, et on va donner en particulier un état-de-l'art sur le JSP.

Chapitre 3

Méthodes de résolution des problèmes d'optimisation combinatoire

Il vaut mieux viser la perfection et la manquer

que viser l'imperfection et l'atteindre.

Bertrand Russell

3.1 Introduction

Dans la vie courante il y a de nombreuses situations où l'on souhaite optimiser une certaine quantité, telle que : minimiser une distance à parcourir, minimiser un retard, minimiser un coût énergétique, maximiser une production, minimiser un risque, etc. Pour traiter de tels problèmes d'optimisation, on a recourt à des méthodes mathématiques qui se révèlent très fructueuses en domaine d'optimisation. Cependant le choix de la méthode doit être en adéquation avec le problème en question.

Le but de notre travail est de résoudre un problème d'ordonnancement qui mathématiquement parlant est un problème d'optimisation combinatoire. Dans ce cas, on va commencer ce chapitre par quelques définitions de l'optimisation, en l'occurrence l'optimisation combinatoire. Ensuite, une étude bibliographique et un état de l'art sur les différentes méthodes de résolutions du problème d'ordonnancement de job shop seront abordés. Des méthodes qui intègrent aussi bien les méthodes exactes que les méthodes approchées.

3.2 L'optimisation

3.2.1 Pourquoi optimiser ?

Est-ce pour chercher le parfait ou plutôt pour améliorer l'existant ?

D'après Beighter et al [1979] :

"*Le désir humain de perfection trouve son expression dans la théorie de l'optimisation. Elle étudie comment décrire et atteindre ce qui est meilleur, une fois que l'on connaît comment mesurer et modifier ce qui est bon et mauvais ...* ".

D'autre part, Oscar Wilde disait dans une de ses citations :

« *Il faut toujours viser la lune, car même en cas d'échec, on atterrit dans les étoiles* ».

A priori, l'idéal serait d'atteindre le parfait (*la lune*). Mais à défaut de celui-ci, on doit viser ce qui serait le mieux pour nous (*les étoiles*). Il serait donc, beaucoup plus réaliste d'essayer de chercher une situation (solution) relativement meilleure, que de chercher en vaine la situation parfaite.

De nos jours, la vie courante est pleine de défis qui nous incitent à améliorer nos performances. Donc, du point de vue pratique, on peut considérer que l'objectif le plus important de l'optimisation est l'*amélioration*. Mais du point de vus théorique, on peut se permettre d'être plus exigeants et d'imposer le *parfait* comme objectif à atteindre. De ce fait, l'optimisation peut être présentée par des définitions plus strictes.

3.2.2 Problème d'optimisation

Un problème d'optimisation consiste à chercher, parmi un ensemble de solutions possibles S (appelé aussi espace de décision ou espace de recherche), la (ou les) solution(s) X^* qui rend (ent) minimale (ou maximale) une fonction $f(X)$ mesurant la qualité de cette solution. Cette fonction est appelée fonction objectif ou fonction coût.

Mathématiquement parlant, soit les définitions suivantes :

Définition 3-1 :

*Soit $f: S \mapsto \mathcal{R}$ une fonction définie sur une partie $S \subset \mathcal{R}^D$. D étant la dimension du problème à optimiser. On dit que f admet un **minimum** (resp. maximum) **global** sur S au point X^* , si :*

$$\forall X \in S \quad f(X^*) \leq f(X) \qquad (resp.\ f(X^*) \geq f(X))$$

Remarques :

- Dans ce cas le minimum de f sur S est $f(X^*)$, c'est le plus petit élément de $f(S)$ et on le note : $\mathrm{Min}_{X \in S} f(X)$. On parle encore de minimum global *absolu* de f sur S.
- S'il existe qu'un seul point de S qui réalise le minimum de f sur S, on parle de minimum *strict,* (les inégalités des définitions deviennent strictes).
- Les deux remarques précédentes sont valables aussi pour le contexte du maximum.

Définition 3-2 :

Soit $f : S \longmapsto \mathcal{R}$ une fonction définie sur une partie S et $X^* \in S$

*On dit que f admet un **extremum** en X^*, si et seulement si f admet un **maximum** ou un **minimum** en X^*.*

Lorsque l'on veut résoudre un problème d'optimisation, on recherche la meilleure solution possible à ce problème, c'est-à-dire **l'optimum global** (extremum global). Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : on parle alors **d'optimums locaux**. Cette notion est illustrée par la Figure 3-1.

Définition 3-3 :

Soit :

- *$f : S \longmapsto \mathcal{R}$ une fonction définie sur une partie $S \subset \mathcal{R}^D$. D étant la dimension du problème à optimiser.*
- *le voisinage $V(X^*) \subset S$, un ensemble contenant X^* et toutes les solutions voisines de X^*.*

*On dit que f admet un **minimum** (resp. maximum) **local** sur V au point X^* , si :*

$$\forall\, X \in V \quad f(X^*) \leq f(X)$$

(*resp.* $f(X^*) \geq f(X)$)

Remarques :

- Une fonction f peut admettre plusieurs minimums (maximums) locaux.
- Un extremum global est un extremum local.

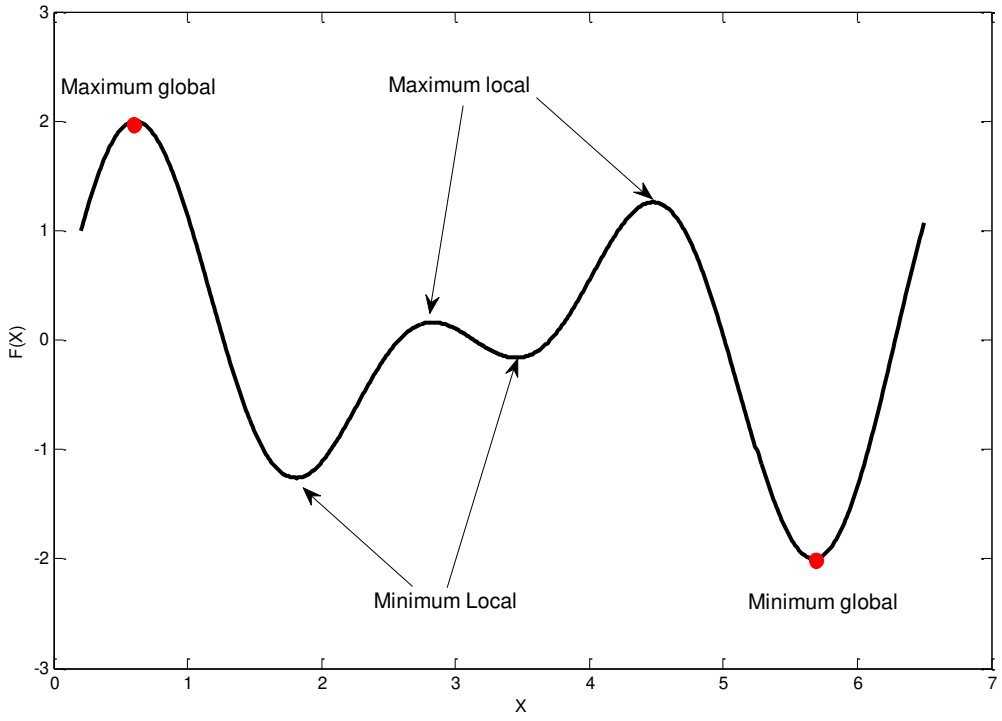

Figure 3-1 : Différence entre un optimum global et des optima locaux.

3.2.3 L'optimisation combinatoire

L'optimisation combinatoire, appelée aussi optimisation discrète, est une branche de l'optimisation en mathématiques appliquées et en informatique. Elle est également liée à la recherche opérationnelle, l'algorithmique et la théorie de la complexité.

Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation [Papadimitriou et Steiglitz, 1982] et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire [Ribeiro et Maculan, 1994].

L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même.

L'optimisation combinatoire consiste à trouver la meilleure solution dans un ensemble discret, dit ensemble des solutions réalisables. En général, cet ensemble est fini mais compte un très grand nombre d'éléments, et il est décrit de manière implicite, c'est-à-dire par une liste, relativement courte, de contraintes que doivent satisfaire les solutions pour qu'elles soient réalisables.

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. À chaque instance du problème est associé :

- un ensemble discret de solutions S,
- un sous-ensemble Ω de S représentant les solutions admissibles (réalisables),
- et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $X \in \Omega$ le nombre réel (ou entier) $f(X)$.

Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $X^* \in \Omega$, optimisant (minimisant ou maximisant) la valeur de la fonction de coût f. X^* Représente la *meilleure* solution et on l'appelle *solution optimale*.

Formellement on définit un problème d'optimisation combinatoire comme suit [Papadimitriou et Steiglitz, 1982] :

Définition 3-4 :

Soit P un problème auquel est associée l'instance I définie par le couple (Ω, f). Ω l'ensemble discret et fini de solutions admissibles du problème P. f une fonction univoque appelée « fonction coût (ou objectif) ».

Trouver $X^* \in \Omega$ tel que $f(X^*) = \mathrm{Min}_{X\in\Omega} f(X)$ ou $X^* \in \Omega$ tel que $f(X^*) = \mathrm{Max}_{X\in\Omega} f(X)$, sont deux problèmes d'optimisation combinatoire.

Trouver une *solution optimale* dans un ensemble discret et fini est un problème facile en théorie : il suffit d'essayer toutes les solutions, et de comparer leurs qualités pour voir la meilleure. Cependant, en pratique, l'énumération de toutes les solutions peut prendre trop de temps ; or, le temps de recherche de la solution optimale est un facteur très important et c'est à cause de ça que les problèmes d'optimisation combinatoire sont réputés si difficiles. La théorie de la complexité donne des outils pour mesurer ce temps de recherche. De plus, comme l'ensemble des solutions réalisables est défini de manière implicite, il est aussi parfois très difficile de trouver ne serait-ce qu'une solution réalisable.

Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. Déjà la plus simple des conditions nécessaires, qui est la continuité des fonctions n'est pas vérifiée, puisque la solution à chercher, évolue dans un ensemble d'éléments discrets, d'où commencent les difficultés à résoudre de tels problèmes d'optimisation. De plus , la plupart de ces problèmes appartiennent à la classe des

problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données [Garey et Johnson,1979b].

Le problème d'optimisation combinatoire qui nous intéresse particulièrement dans cette étude est le problème d'ordonnancement d'ateliers, en l'occurrence le problème d'ordonnancement d'atelier Job shop. Il a bien été démontré par Lenstra et Rinnooy Kan [1979] qu'à partir de trois machines, le JSP devient un problème NP-difficile au sens fort.

Pour bien se rendre compte de la complexité du problème d'ordonnancement de Job shop (JSP), nous allons illustrer sa nature combinatoire par un petit exemple.

Supposons que l'on veuille ordonnancer un atelier Job shop (10×1). Dans ce cas, on a 10 opérations différentes à ordonnancer sur une machine unique, la combinatoire sous-jacente est donc de 10! (Un choix parmi 10 pour la première opération, puis un choix parmi 9 pour la suivante, etc.). Si l'on dispose d'un calculateur capable de calculer un million de solutions par seconde, il nous faut 3,62 secondes pour calculer toutes les solutions de ce problème (10! solutions). Par contre s'il faut ordonnancer 20 opérations au lieu de 10 et toujours sur une seule machine, il nous faut plus de 771 siècles !

De plus pour le cas d'un Job shop d'instance $(n \times m)$, on va avoir $(n!)^m$ combinaisons possibles pour avoir toutes les solutions possibles. Pour l'exemple d'un Job shop (6×6), on peut énumérer 13931×10^{13} solutions. Dans ce cas, avec une capacité de calculer 1 millions de solutions/seconde, il nous faut plus de 44 siècles pour évaluer toutes les solutions possibles !

Comme nous pouvons le constater à travers ces petits exemples, tant que la taille du problème combinatoire est petite on peut l'aborder en utilisant des méthodes exactes qui permettent d'explorer l'espace des solutions en un temps raisonnable (polynomial). Cependant, lorsque l'on aborde des problèmes de grande taille, il est difficile d'éviter le phénomène d'explosion combinatoire caractérisant les problèmes NP-difficiles dans l'optimisation combinatoire. Le cas échéant, on a judicieusement recourt à l'utilisation des méthodes approchées dictées par des heuristiques ou en encore des métaheuristiques.

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Cependant, malgré les progrès réalisés et l'amélioration de la vitesse de calcul des ordinateurs, le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème. Les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante. Ainsi, les méthodes

approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale.

La section suivante donne une typologie des méthodes d'optimisation incluant les méthodes exactes et approchées. Des méthodes souvent utilisées pour la résolution des problèmes d'ordonnancement.

3.3 Méthodes de résolution du JSP

Etant donné l'importance du problème d'ordonnancement de Job shop qui est considéré comme étant un problème d'optimisation combinatoire, de nombreuses méthodes pour sa résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories :

- les méthodes exactes (complètes) qui garantissent la complétude de la résolution et par conséquent, l'optimalité de la solution car elles explorent d'une manière complète toutes les solutions de l'espace de recherche.
- les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité. Ces approches sont incomplètes dans le sens où elles n'explorent délibérément (mais intelligemment) qu'une partie de l'espace de recherche. Par conséquent, elles peuvent ne pas trouver la solution optimale dans certains cas, et encore moins prouver l'optimalité des solutions trouvées ; en contrepartie, le temps de résolution est généralement faiblement polynomial.

Dans la littérature, de considérables travaux de recherches ont été consacrés au problème d'ordonnancement de Job shop (JSP). Nous allons présenter dans les prochaines sections un petit état de l'art sur certaines méthodes dédiées au JSP. Toutefois, nous orientons le lecteur vers quelques références qui présentent des revues de littérature très intéressantes sur le JSP.

Nous commençons par l'article de Jain et Meeran [1999] dans lequel les auteurs ont présenté une excellente revue de littérature sur toutes les techniques utilisées pour le traitement d'un JSP. On peut y trouver des définitions, des aperçus historiques, des classifications, et mêmes les résultats rapportés par l'application de ces techniques sur des Benchmarks de Job shop.

L'article de Çalis et Bulkan [2015] donne un bon état de l'art sur les stratégies de résolution d'un JSP, mais il est dédié surtout aux approches de l'intelligence artificielle (IA). De plus, récemment, Zhang et al. [2017], Bien qu'ils aient cherché à présenter de nouvelles

perspectives de recherche sur l'ordonnancement de Job shop sous Industrie 4.0, ils ont réservé une partie importante de leur article à une revue de la littérature tirée de plus de 120 papiers sur les stratégies de résolution d'un JSP classique.

3.3.1 Méthodes exactes

On peut définir une méthode exacte comme étant une méthode qui fournit une solution optimale pour un problème d'optimisation. Dans ce type de méthodes, on distingue les techniques dites efficaces et les techniques d'énumération. Le premier type de techniques est généralement adapté à certains problèmes spécifiques qui peuvent être résolus par un algorithme polynomial. Les méthodes d'énumération complètes sont basées sur l'énumération exhaustive de toutes les solutions possibles ; elles examinent d'une manière implicite, la totalité de l'espace de recherche pour trouver4 ; la solution optimale. L'utilisation de ces méthodes s'avère particulièrement intéressante dans les cas des problèmes de petites tailles.

Les méthodes exactes sont multiples et variées telles que : la programmation linéaire, la programmation dynamique, les méthodes par séparation et évaluation progressive, etc.

3.3.1.1 La programmation linéaire

C'est l'une des techniques classiques de recherche opérationnelle. Elle est composée de :

- variables de décision qui sont clairement définies,
- d'une fonction objectif associée à un coût ou profit lié avec les variables de décision qui ont un sens d'optimisation (soit la minimisation, soit la maximisation)
- et des contraintes qui limitent et représentent la structure du problème.

A partir de ces concepts, nous disons qu'une solution est réalisable si elle respecte les conditions exprimées par les contraintes et qui a un coût (profit) calculé par la fonction objectif.

Différents types de programmations linéaires peuvent être employés selon la nature des variables de décision à utiliser :

- programmation binaire (variables binaires),
- programmation linéaire en nombres entiers, PLNE, (variables entières)
- ou programmation linéaire mixte (variables entières et réelles ou MIP "Mixed Integer Problem").

La résolution d'un modèle de programmation linéaire se fait en utilisant différentes méthodes, la plus connue est l'algorithme du simplexe, qui a été utilisé dans de nombreux domaines. Il y a aussi des solveurs dédiés comme : CPLEX, XPRESS, GAMS, EXCEL pour les modèles linéaires et LINGO pour les modèles non-linéaires.

- Les premières modélisations du problème de Job shop ont été des programmes linéaires en nombre entiers ou à variables mixtes. Les premiers travaux datent de 1959 avec les travaux de Wagner [1959], Bowman [1959] et Manne [1960]. De plus, dans leur survey, Jones et Rabelo [1998], indiquent que le problème Job shop a été encore formulé par Balas [1967] en utilisant la programmation linéaire en nombre entier et par Balas [1969,1970] en utilisant la programmation mixte en nombre entier.

3.3.1.2 La programmation dynamique

Cette technique repose essentiellement sur le principe d'optimalité de Bellman [Bellman, 1986]. Elle est applicable à n'importe quel problème sachant qu'il est décomposable en une séquence de sous-problèmes. La procédure de résolution se fait itérativement d'une manière ascendante par détermination d'une solution optimale à chaque sous-problème et ce, en tenant compte des informations des sous-problèmes précédents. Cette résolution s'arrête jusqu'à ce qu'elle trouve une solution pour le problème initial, sachant que le critère à minimiser par exemple puisse être mis sous forme d'une relation de récurrence entre n'importe quelles paires de niveaux successifs.

La programmation dynamique est l'une des méthodes exactes les plus performantes et les plus robustes pour les problèmes d'énumération. Elle est destinée à résoudre des problèmes d'optimisation à vocation plus générale que la méthode de séparation et d'évaluation (branch and bound) sans permettre pour autant d'aborder des problèmes de tailles importantes [Brusco et Stahl, 2005].

- Bien que cette méthode ait été utilisée pour résoudre des problèmes d'ordonnancement depuis les années soixante, elle n'est pas très répandue pour résoudre le problème Job shop, à part quelques travaux, dont on peut citer ceux de Srinivasan [1971] et de Chen et al. [1998], et ceci est due au fait qu'elle nécessite un temps de calcul de plusieurs heures [Pinedo, 2005].

3.3.1.3 Séparation et évaluation progressive (SEP) ou « branch and bound (B&B) »

Cette technique de recherche est apparue vers les années 60 [Land et Doig, 1960]. Elle consiste à décomposer le problème en sous-problèmes de plus en plus petits et faire une

exploration intelligente de l'espace de solutions. Par exemple en ordonnancement, l'algorithme Branch and Bound consiste à placer progressivement les opérations sur les ressources en explorant un arbre de recherche décrivant toutes les combinaisons possibles. Il s'agit de trouver la meilleure configuration donnée de manière à élaguer les branches de l'arbre qui conduisent à de mauvaises solutions. La démarche de l'algorithme Branch and Bound [Collette et Siarry, 2002] consiste à :

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

Une description détaillée des principes de cette méthode peut être trouvée dans [Clausen, 1999].

- La méthode de séparation et d'évaluation (Branch & Bound) est l'une des méthodes d'énumération la plus connue dans la littérature des premiers travaux pour la résolution du problème de type Job shop. En effet, le problème d'ordonnancement Job shop (10×10) (de 10 jobs traités sur 10 machines), formulé depuis 1963 par Muth et Thompson [1963] est resté un problème ouvert pendant 25 ans, et a été finalement résolu par la méthode de branch and bound par Carlier et Pinson [1989]. On peut citer quelques travaux d'ordonnancement de Job shop utilisant cette méthode, par exemple ceux de : [Brucker et al. 1994], [Caseau et Laburthe, 1996], [Blazewicz et al., 1998], [Clausen, 1999] et [Artigues et Feillet, 2008].

3.3.2 Méthodes approchées

La majorité des problèmes d'ordonnancement de Job shop sont classées NP-difficiles et leurs résolutions ne s'avèrent pas possibles dans un contexte de temps de calcul et de mémoire limités. En effet, pour des problèmes Job shop ayant une taille industrielle assez grande, il n'est pas possible de trouver avec les méthodes exactes un ordonnancement optimal en un temps raisonnable. Dans ce cas, pour obtenir malgré tout des solutions, des méthodes approchées ont été développées. Ces méthodes donnent des solutions certes sous-optimales, mais en un temps de calcul acceptable. La performance de telles méthodes est estimée par le pourcentage d'erreur entre la solution fournie et la valeur de la solution optimale si elle est

calculable. Dans les cas où la solution optimale est non calculable, on peut évaluer ces méthodes en comparant les solutions fournies à des bornes inférieures.

Parfois les méthodes approchées sont désignées par l'appellation « *heuristiques* ». D'après Reeves [1993] :

« Une heuristique est une technique de recherche de bonne solutions (i.e. quasi-optimales) obtenues par un effort de calcul raisonnable sans que l'on soit capable de garantir la faisabilité ni l'optimalité ; et même dans beaucoup de cas, sans être capable de quantifier l'écart d'une solution réalisable à une solution optimale ».

Mais à vrai dire, les méthodes approchées peuvent être classées en deux principales catégories : les « *heuristiques* » et les « *métaheuristiques* ».

- Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité.
- Les méthodes dites métaheuristiques sont des méthodes génériques, des heuristiques polyvalentes applicables sur une grande gamme de problèmes.

3.3.2.2 Les heuristiques (approches constructives)

Une heuristique est une procédure qui exploite au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible [Nicholson, 1971]. C'est une méthode, souvent simple, guidée généralement par l'intuition et qui fournit relativement et rapidement, une bonne solution à des problèmes difficiles.

Une heuristique est approximative dans le sens où elle fournit une bonne solution avec relativement peu d'efforts de calcul, mais elle ne garantit pas l'optimalité [Maniezzo et al., 2009].

Les heuristiques suivent des approches constructives qui consistent à construire une solution pour le problème à partir des données initiales. L'approche constructive démarre d'une solution vide et insère à chaque étape une composante du problème considéré dans la solution partielle jusqu'à l'obtention d'une solution complète admissible.

Pour l'ordonnancement de Job shop, les premières heuristiques de construction sont apparues à la fin des années 50 et début des années 60. Elles ont été proposées par Giffler et Thompson [1960] et Fisher et Thompson [1963].

Pour ces heuristiques on peut distinguer deux types d'approches constructives :

A. L'approche constructive par « opérations »

Cette approche constructive repose principalement sur les règles de priorité qui sont probablement les heuristiques les plus utilisées pour la résolution des problèmes Job shop, et ce pour deux raisons, d'une part vue la facilité (simplicité) de leur implémentation et d'autres part la taille réduite du temps alloué au calcul des problèmes correspondants [Blazewicz et al., 2007]. Dans une étude réalisée par Haupt [1989] sur l'état de l'art des règles de priorité utilisées pour l'ordonnancement, une règle de priorité est définie comme étant une politique décrivant une décision de séquencement spécifique à chaque fois qu'une machine est libre. Ainsi une règle de priorité permet à une machine libre de sélectionner de la prochaine opération à exécuter parmi celles disponibles ou en attente.

- Pour l'ordonnancent de Job shop utilisant l'approche constructive par opérations, on trouve en premier, les travaux de [Panwalkar et Iskander,1977] qui ont proposé une centaine de règles de priorité ainsi qu'une classification de ces règles. Vancheeswaran et Townsend ont proposé en 1993 d'autres règles de priorité afin de trouver une valeur minimale du Makespan. Une méthode basée sur la technique d'insertion des opérations a été proposée par [Werner et Winkler 1991]. On trouve aussi les travaux de Chiang et Fu [2007] et Tay et Ho [2008].

B. L'approche constructive par « machines »

C'est l'heuristique du déplacement de la machine goulot, appelée aussi « *Shifting bottleneck»*. Elle a été proposée par Adams et al. [1988] pour la minimisation du Makespan des Job shops. Elle est considérée comme l'une des méthodes les plus efficaces et sert de méthode de référence pour l'ordonnancement des Job shops.

Cette méthode calcule l'ordonnancement par construction progressive. Elle résout successivement des problèmes d'ordonnancement à une machine. Chaque problème à une machine détermine l'ordre de passage des opérations qui nécessitent la machine considérée, tout en respectant l'ordonnancement des machines déjà ordonnancées. Ce problème peut être prouvé équivalent à la minimisation du Makespan de *n* tâches, sur une machine avec dates de disponibilité et de latence des tâches. Le temps de latence d'une opération correspond au temps séparant la fin de l'opération et la fin du job correspondant. Ce problème peut être résolu, de manière efficace, à l'aide d'un algorithme mis au point par Carlier [1982].

❖ Pour cette approche, nous pouvons citer aussi les travaux de : [Demirkol et al., 1997] ; [Zhang et Wu 2008] ; [Chen et Chen, 2009] et [Tan et al., 2016].

3.3.2.3 Les métaheuristiques

Les métaheuristiques sont des *algorithmes* génériques pouvant être utilisés pour une large gamme de problèmes. Ceux sont des méthodes de recherches générales, dédiées aux problèmes d'optimisation difficile. Elles peuvent construire une alternative aux méthodes heuristiques lorsqu'on ne connaît pas l'heuristique spécifique à un problème donné.

En fait, *Métaheuristique* est un mot composé de **méta** et d'**heuristique** qui viennent du grec (*au-delà*) pour *méta* qui signifie à un plus haut niveau, et *heuristique* (*heuriskein*), qui signifie trouvé.

Les métaheuristiques sont des procédures utilisées pour guider l'exploration de l'espace des solutions d'un problème complexe, de façon à trouver une solution réalisable, qui offre un bon compromis entre effort de calcul et qualité de la solution. La performance d'une métaheuristique dépend de la combinaison et l'équilibre entre deux critères opposés : l'*intensification* et la *diversificatio*n :

- **L'intensification** correspond à l'exploitation d'une région de l'espace de recherche jugée prometteuse. C'est une étape de recherche locale, dans laquelle on cherche à améliorer la qualité de la solution courante.
- **La diversification** fait référence à l'exploration de différentes branches de l'espace des solutions. Elle a pour objectif de diriger la procédure de recherche vers des régions inexplorées de l'espace de recherche. Elle permet donc, de sortir de zones très explorées, pour en visiter d'autres, en évitant de rester sur la région d'un optimum local. La stratégie de diversification la plus simple consiste à redémarrer périodiquement le processus de recherche à partir d'une solution, générée aléatoirement ou choisie judicieusement, dans une région non encore visitée de l'ensemble des solutions admissibles.

Une métaheuristique peut être classée suivant de nombreux critères [Talbi, 2009], selon qu'elle soit :

- **Inspirée ou non-inspirée de la nature**: De nombreuses métaheuristiques sont inspirées de processus naturels, tels que : de la biologie (les algorithmes évolutionnaires, les systèmes immunitaires artificiels,…) ; de l'éthologie et de

l'intelligence des essaims de différentes espèces (optimisation par : colonies de fourmis, par essaims particulaires, par colonies d'abeilles,…) ; de la physique (le recuit simulé).

- **Avec mémoire ou sans mémoire** : Certains algorithmes de métaheuristiques sont sans mémoire; c'est-à-dire qu'aucune information extraite dynamiquement n'est utilisée pendant la recherche. Certains représentants de cette classe sont : la recherche locale, le GRASP et le recuit simulé. Alors que d'autres métaheuristiques utilisent une mémoire qui contient des informations extraites directement lors de la recherche telles que : les mémoires à court-terme et à long-terme dans la recherche tabou.
- **Déterministe ou stochastique** : une métaheuristique déterministe résout un problème d'optimisation en prenant des décisions déterministes (par exemple, la recherche locale, la recherche tabou). En métaheuristique stochastique, certaines règles aléatoires sont appliquées pendant la recherche (par exemple, recuit simulé, algorithmes évolutionnaires). Dans les algorithmes déterministes, l'utilisation de la même solution initiale conduira à la même solution finale, alors qu'en métaheuristique stochastique, différentes solutions finales peuvent être obtenues à partir de la même solution initiale. Cette caractéristique doit être prise en compte dans l'évaluation des performances des algorithmes des métaheuristiques.
- **basée sur la recherche sur une solution unique ou sur une population de solutions** : les algorithmes basés sur une solution unique (par exemple, la recherche locale, le recuit simulé, la recherche tabou,…) manipulent et transforment une seule solution pendant la recherche, tandis que dans les algorithmes basés sur la population (par exemple, les essaims particulaires, les algorithmes évolutionnaires) toute la population des solutions évolue. Ces deux familles ont des caractéristiques complémentaires: les métaheuristiques basées sur une solution unique sont orientées vers l'exploitation; elles ont le pouvoir d'intensifier la recherche dans les régions locales. Les métaheuristiques basées sur la population de solutions sont orientées vers l'exploration; elles permettent une meilleure diversification dans tout l'espace de recherche.
- **Itérative ou constructive** : dans les algorithmes itératifs, nous partons d'une solution complète (ou population de solutions) et la transformons à chaque itération à l'aide de quelques opérateurs de recherche. Les algorithmes constructifs

(gloutons) partent d'une solution vide, et à chaque étape une variable de décision du problème est affectée jusqu'à l'obtention d'une solution complète. La plupart des métaheuristiques sont des algorithmes itératifs.

3.4 Quelques métaheuristiques

Depuis quelques années, on constate que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle. Dans cette section, nous allons présenter quelques métaheuristiques qui semblent très prometteuses pour la recherche en optimisation combinatoire en général, et pour la résolution du problème d'ordonnancement de Job shop en particulier.

3.4.1 Méthodes de recherche locale (métaheuristiques à solution unique)

Les méthodes de recherche locale ou métaheuristiques à base de voisinages, s'appuient toutes sur un même principe. A partir d'une solution unique X_0, considérée comme point de départ (ou calculée par exemple par une heuristique constructive), la recherche consiste à passer d'une solution à une solution voisine par déplacements successifs. L'ensemble des solutions que l'on peut atteindre à partir d'une solution X est appelé voisinage $V(X)$ de cette solution. Déterminer une solution voisine de X dépend bien entendu du problème traité.

De manière générale, la méthode de recherche locale standard s'arrête lorsqu'il n'est plus possible d'améliorer la solution après un certain nombre d'itérations, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Dans ce cas la solution trouvée correspond à un optimum local. Mais accepter uniquement ce type de solution n'est bien sûr pas satisfaisant. C'est le cas de la plus simple des méthodes de recherche locale, appelée « *Recherche locale* ou *méthode de descente* » présentée ci-dessous (voir section 3.4.1.1).

Dans un cadre plus général, il serait plus intéressant de pouvoir s'échapper de ces minima locaux. Pour se faire, des méthodes de recherche locale plus sophistiquées ont été développées au cours de ces vingt dernières années. Ces méthodes acceptent des solutions voisines moins bonnes que la solution courante afin d'échapper aux minima locaux. Il faut alors permettre à l'opérateur de recherche locale de faire des mouvements pour lesquels la nouvelle solution retenue sera de qualité moindre que la précédente. C'est le cas immédiat des méthodes de *recuit simulé* (section 3.4.1.3), de *la recherche tabou* (section 3.4.1.4) et de *L'Iterated Local Search* (ILS), (section 3.4.1.2).

3.4.1.1 La recherche locale « LS : Local Search »

La méthode de recherche locale, appelée aussi méthode de descente ou méthode de voisinage, est un processus itératif fondé sur deux éléments essentiels : le voisinage et une procédure exploitant ce voisinage. Nous allons d'abord définir la notion fondamentale du voisinage :

Définition 3-5 :

Soit S l'ensemble des solutions admissibles d'un problème, on appelle voisinage toute application $V: S \longmapsto 2^S$. On appelle mécanisme d'exploration du voisinage toute procédure qui précise comment la recherche passe d'une solution $X \in S$ à une solution $X' \in V(X)$. Une solution X^* est un optimum (minimum) local par rapport au voisinage $V(X)$, si $f(X^*) \leq f(X')$ pour toute solution $X' \in V(X)$.

La méthode de recherche locale s'articule autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage, et accepter cette dernière si elle améliore la solution courante. L'Algorithme 3-1 décrit le principe de cette méthode.

Algorithme 3-1. La ***Recherche locale***

Nom de procédure : $Recherche\_Locale$
Entrée : X (solution initiale)
Sortie : $X^* = Recherche\_Locale(X)$
Début
    Initialise $X^* = X$
    Tant que nécessaire **faire**
        Chercher dans le voisinage : trouver une solution $X' \in V(X)$
        Si $f(X') \leq f(X)$
            $X^* = best(X^*, X')$
        Fin Si
    Fin Tant que
Fin

La recherche locale débute avec une solution initiale X, et réalise ensuite un processus itératif qui consiste à remplacer la solution courante X par l'un de ses voisins X' en tenant compte de la valeur de la fonction de coût $f(X)$. Autrement dit, cette procédure fait intervenir à chaque itération le choix d'un voisin qui améliore la solution courante. Cependant, plusieurs possibilités peuvent être envisagées pour décider du nombre à effectuer pour le choix des voisins et donc de décider du critère d'arrêt.

Ce processus s'arrête et retourne la meilleure solution trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt concerne généralement une limite pour le nombre d'itérations ou un objectif à réaliser.

Il est possible aussi d'adopter pour le critère d'arrêt les deux stratégies d'arrêts suivantes :

- Stratégie de la première amélioration (First improvement strategy) : pour laquelle il est possible d'énumérer les voisins jusqu'à ce que l'on découvre un candidat qui améliore strictement la solution courante.
- Stratégie de la meilleure amélioration (Best improvement strategy) : pour laquelle on va énumérer tous les voisins possibles, et en choisir le meilleur (au sens de la fonction coût). Cette stratégie d'arrêt semble plus coûteuse en temps de calcul, mais le voisin découvert sera en général une solution de meilleure qualité.

L'avantage principal de cette méthode de recherche locale réside dans sa grande simplicité et sa rapidité. Cependant, elle a l'inconvénient majeur de nous donner seulement l'optimum local car elle cherche à faire une amélioration progressive de la solution, mais seulement dans un espace de recherche limité. Il faut alors noter, qu'à moins d'être extrêmement chanceux, l'optimum local est souvent de qualité assez médiocre et de coût très supérieur au coût optimal.

Si par exemple, pour un problème de minimisation cette méthode reste bloquée dans un minimum local dès qu'elle en rencontrera un, il est clair qu'il y a absence de diversification et que l'équilibre souhaité entre intensification et diversification n'existe pas. Pour remédier à ce problème, des techniques différentes pour échapper des minima locaux, sont considérées comme variantes de la méthode de recherche locale (LS). On trouve par exemple :

- "*Multi-start descent*" qui consiste à re-exécuter l'algorithme de recherche locale en prenant un autre point de départ. Comme l'exécution de cette méthode est souvent très rapide, on peut alors inclure cette répétition au sein d'une boucle générale.
- La « *recherche locale guidée* » ou la « *Guided Local Search* » (GLS) consiste à modifier la fonction à optimiser en ajoutant des pénalités. La recherche locale est appliquée alors sur cette fonction modifiée. La solution trouvée (qui se trouve être un optimum local) sert à calculer les nouvelles pénalités. Pour cela, on calcule l'utilité de chacun des attributs de la solution et on augmente les pénalités associées aux attributs de valeur maximale. Ces étapes successives sont répétées jusqu'à ce qu'un critère d'arrêt soit validé. Cette méthode a été présentée pour la première fois

dans un rapport de recherche de Voudouris et Tsang [1995]. Pour une présentation très approfondie, la thèse de Voudouris [1997] est le document le plus complet.

- « *La recherche à voisinages variables* » ou la «*Variable Neighbourhood Search* » (VNS), est une méthode récente basée sur la performance des méthodes de descente. Introduite par Mladenovi´c et Hansen [1997], la méthode repose sur l'utilisation de plusieurs voisinages afin d'explorer au mieux l'espace des solutions. Chaque voisinage est exploré l'un après l'autre afin de créer un voisin de la solution actuelle. Une recherche locale est généralement appliquée à la suite de l'obtention d'un voisin et si la solution est meilleure, la solution est actualisée. La VNS est décrite en détail par Hansen et Mladenovi´c [1999].
- Etc.

Pour encore améliorer ses performances, la méthode de recherche locale a encore été rehaussée par de nouvelles options, d'où l'émergence d'autres nouvelles méthodes de recherche locale. Certaines d'entre elles seront définies par la suite.

❖ Des exemples d'utilisation de la recherche locale pour des problèmes de d'ordonnancement de Job shop peuvent être consultés dans [Vaessens et al. 1997].

3.4.1.2 La recherche locale itérée « ILS : Iterated Local Search »

La méthode de recherche locale itérée est une variante très simple des méthodes de descente qui pallient au problème de l'arrêt de ces dernières dans des optima locaux. On peut citer dans l'ordre chronologique quelques-unes des références présentant des méthodes très proches : [Baxter 1981], [Baum 1986], [Martin et al. 1991]. Par contre, le nom de la méthode pourrait revenir sans conteste à Lourenço et al. [2000], (voir aussi [Lourenço et al. 2002]).

Dans cette méthode, on génère une solution initiale qui servira de point de départ. Ensuite, on va répéter deux phases : une phase de perturbation aléatoire dans laquelle la solution courante va être perturbée (parfois en tenant compte d'un historique maintenu à jour) et une seconde phase de recherche locale qui va améliorer cette solution jusqu'à buter sur un optimum local. L'Algorithme 3-2 présente la structure générale de la méthode Iterated Local Search (ILS).

Il est clair que dans la recherche locale itérée, la diversification est obtenue par la perturbation aléatoire. Même si cette dernière peut dans certaines implémentations tenir compte d'un historique des perturbations, il est important de conserver ici le caractère aléatoire. Encore une fois, c'est une recherche locale qui intensifie la recherche. La phase

d'acceptation de la solution au sein de la boucle générale est importante. On peut par exemple accepter une solution si elle améliore strictement la solution courante (Y) . Mais on peut aussi accepter seulement si elle améliore la meilleure solution trouvée (Y^*), ou pourquoi pas, si elle ne dégrade pas la meilleure solution trouvée de plus d'un certain pourcentage. Comme on le voit, on peut adapter cette méthode facilement et ajouter des facteurs d'intensification ou de diversification à ce niveau.

Algorithme 3-2. La recherche locale itérée

1: **Nom de procédure** : $Iterated\_Local\_Search$
2: **Entrée** : X (solution initiale)
3: **Sortie** : $Y^* = Iterated\_Local\_Search\ (X)$
4: **Début**
5: **Initialise** $Y = Recherche\_locale(X)$
6: **Tant que** nécessaire **faire**
7: $X' = perturbation\ (Y)$: trouver une solution X' voisine de Y
8: $Y' = Recherche\_locale(X')$
9: $Y = best(Y, Y')$
10: $Y^* = best(Y^*, Y)$
11: **Fin Tant que**
12: **Fin**

3.4.1.3 Le recuit simulé « Simulated Annealing (SA) »

La méthode du recuit simulé a été introduite en 1983 par Kirkpatrick et al [1983]. Cependant sa version originale est basée sur les travaux bien antérieurs de Metropolis et al. [1953]. Cette méthode que l'on pourrait considérer comme la première métaheuristique "grand public" a reçu l'attention de nombreux travaux et principalement de nombreuses applications.

Cette métaheuristique s'appuie sur un principe physique, qui est celui du recuit utilisé par les forgerons pour donner de la résistance et améliorer la structure du fer en alternant des périodes de chauffe (recuit) et de refroidissement. En effet, l'amélioration de la qualité d'un métal solide, afin qu'il retrouve une structure proche du cristal parfait, nécessite la recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement et de retrouver ainsi le meilleur équilibre thermodynamique. Pour cela, Chaque niveau de température est maintenu jusqu'à obtention

d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires du métal non satisfaisants, mais conduisant à la longue à des états meilleurs.

L'analogie avec une méthode d'optimisation est trouvée en associant une solution (X) à un état du métal, son équilibre thermodynamique est la valeur de la fonction objectif $H(X)$ de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution (X) à une solution voisine (Y). $H(X)$ et $H(Y)$ représentent l'énergie respective de (X) et (Y).

Pour passer à une solution voisine, il faut respecter l'une des deux conditions suivantes :

- Soit le mouvement améliore la qualité de la solution précédente, i.e. en minimisation la variation de coût est négative ($\Delta C = H(Y) - H(X)\ ;\ \Delta C < 0$),
- Soit le mouvement détériore la qualité de la solution précédente. Dans le cas échéant, bien que la situation soit défavorable à cause de l'augmentation d'énergie $\big(H(Y) > H(X)\big)$, la transition est acceptée avec une probabilité $P \leq e^{-\frac{\Delta C}{T}}$.

On peut constater que la température T est un paramètre très important pour cette métaheuristique. En effet, la probabilité P permet à l'algorithme d'admettre des transitions défavorables et donc de sortir d'un éventuel minimum local. Cette probabilité P dépend de la température T et elle est d'autant plus grande que cette dernière est élevée. Nous constatons donc qu'un maximum de transitions défavorables sera accepté au début. Cela va nous permettre de mieux explorer l'espace des solutions possibles et ainsi, d'accroître les chances d'approcher le minimum global. En règle générale, la température est diminuée par paliers, à chaque fois qu'un certain nombre d'itérations est effectué. La meilleure solution trouvée est mémorisée. L'algorithme est interrompu lorsqu'aucune solution voisine n'a été acceptée pendant un cycle complet d'itérations à température constante.

La popularité du recuit simulé a été incontestable pendant des années. D'abord cette méthode est facile à implémenter et elle a permis de résoudre de nombreux problèmes NP-difficiles [Vidal, 1993]. Parmi les bibliographies et articles de synthèse intéressants, on peut citer Koulamas et al. [1994] et Collins et al. [1988]. On peut trouver aussi dans [Pirlot et Vidal, 1996] un excellent tutorial pour guider les chercheurs dans leurs premiers pas avec cette méthode.

❖ Pour l'ordonnancement de Job shop on trouve les travaux de : Matsuo et al. [1988], Van Laarhoven et al. [1988], Yamada et al. [1994], Sadeh et Nakakuki [1996], Yamada et Nakano [1996], Aydin et Forgarty [2004], Suresh et Mohanasundaram [2006] et

récemment ceux de Tamssaouet et al. [2018]. Cependant, d'après Jain et Meeran [1999] le recuit simulé utilisé en tant que technique générique pour la résolution du problème d'ordonnancement de Job shop, est incapable de converger rapidement aux bonnes solutions. Ainsi, pour les recherches actuelles, l'utilisation du recuit simulé est dirigée vers les méthodes hybrides c'est-à-dire la combinaison de cet algorithme avec d'autres méthodes tel que l'algorithme génétique, la méthode tabou,… et ce afin d'améliorer les résultats et réduire le temps de calcul. Comme exemple de ces méthodes, on trouve les travaux de Hernández-Ramírez et el. [2019].

3.4.1.4 La recherche tabou « Tabu (ou taboo) Search (TS) »

En 1986, Glover [1986] a présenté un article dans lequel on voit apparaître pour la première fois, le terme « *tabu search* » et le terme « *métaheuristique* ». A la même époque, Hansen [1986] présente une méthode similaire, mais dont le nom n'a pas marqué autant que tabou. En fait, les prémices de la méthode ont été présentées initialement à la fin des années 1970 par Glover [1977]. Pourtant ce sont les deux articles de référence de Glover [1989, 1990] qui vont contribuer de manière importante à la popularité de cette méthode. Cette méthode propose de surmonter le problème des optima locaux par l'utilisation d'une mémoire. Pour certains chercheurs, elle apparait même plus satisfaisante sur le plan scientifique que le recuit simulé, car la partie "aléatoire" de la méthode a disparu.

La méthode tabou est une procédure de recherche locale et itérative qui, partant d'une solution initiale X, tente de converger vers la solution optimale X^* en exécutant, à chaque pas, un mouvement dans l'espace de recherche. Contrairement au recuit simulé qui ne génère qu'une seule solution X' "aléatoirement" dans le voisinage $V(X)$ de la solution courante X, la méthode tabou, dans sa forme la plus simple, examine le voisinage $V(X)$ de la solution courante X. La nouvelle solution X' est la meilleure solution de ce voisinage, mais ça lui arrive parfois d'être moins bonne que X elle-même. En acceptant de détériorer la valeur de la solution courante, le minimum local peut être évité mais, en contrepartie, des parcours répétitifs sont explorés. Pour éviter de cycler sur les mêmes solutions, on mémorise les k dernières solutions visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces solutions. Cette mémoire est appelée la *liste tabou* (d'où le nom à la méthode). La liste tabou est une des composantes essentielles de cette méthode. Elle est tenue à jour et interdit de revenir à des solutions déjà explorées.

Il existe aujourd'hui un très grand nombre de références sur cette méthode. Un des résultats les plus attendus et pourtant ayant un impact limité en pratique est la preuve de

convergence de la méthode proposée par Glover et Hanafi [2002]. Parmi les articles introductifs intéressants, on peut citer celui de De Werra et Hertz [1989] et de Glover [1990]. Une autre référence particulièrement adaptée se doit de figurer ici, c'est celle de Hertz et al. [1997]. De plus, Glover et Laguna [1999] ont publié un livre très complet qui présente plus de détails sur la recherche tabou ainsi que sur d'autres méthodes de recherche.

- Parmi les travaux basés sur la méthode tabou destinée pour la résolution du problème d'ordonnancement de Job shop, on peut citer ceux de : Dell'Amico et Trubian [1993]; Nowicki et Smutnicki [1996] ; Pezzella et Merelli [2000]; Chen et al. [2007] ; Velmurugan et Selladurai [2007] ; Eswarmurthy et Tmilarasi [2009] ; etc.

3.4.2 Métaheuristiques à population de solutions

3.4.2.1 L'algorithme génétique « Genetic Algorithm (GA) »

Les algorithmes génétiques sont une métaheuristique qui s'appuie sur le principe de la génétique et des mécanismes de sélection naturelle, de l'évolution et de l'hérédité. Ils ont été développés dès 1950 par les biologistes qui utilisaient des ordinateurs pour simuler l'évolution naturelle des espèces. Cependant, c'est à partir de 1975, que ces algorithmes ont commencé à être adaptés pour la recherche de solutions à des problèmes d'optimisation par les travaux de Rechenberg [1973], Holland [1975], Schwefel [1977] et ceux de Goldberg [1987]. Au fait, les algorithmes génétiques doivent leur popularité au livre de Goldberg [1989] qui est une des références les plus citées dans le domaine de l'informatique. De nos jours, on trouve plusieurs milliers de références sur les techniques des AG et de leurs applications. Toutefois, nous renvoyons le lecteur vers deux livres importants, [Haupt et Haupt 1998, 2004], [Mitchel 1998] et vers un bon article introductif qui a le mérite d'être en français, c'est celui de Fleurent et Ferland [1996] et qui explique avec suffisamment de détails une implémentation parmi d'autres. De plus, malgré que l'intérêt pratique de la preuve de convergence soit limité, il est toujours plus satisfaisant de savoir que la méthode converge sous certaines conditions. Cette preuve est apportée par les travaux de Cerf [1994].

Les AGs font partie des algorithmes évolutionnaires qui sont basés sur le principe du processus d'évolution naturelle des espèces vivantes, ([De Jong et Spears, 1993] ; [Bäck et Schwefel, 1993] ; [Schoenauer et Michalewicz 1997]). Un algorithme évolutionnaire typique est composé de trois éléments essentiels :

1) une *population* constituée de plusieurs individus représentant des solutions potentielles du problème donné;

2) un *mécanisme d'évaluation* de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur;
3) un *mécanisme d'évolution* composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

De manière générale, les algorithmes génétiques suivent ce même principe. Ils commencent avec une population d'individus (correspondant à un ensemble de solutions) qui évoluent en même temps comme dans l'évolution naturelle en biologie. Chaque individu X de la population qui est codé sous forme d'une représentation génétique du problème (chromosome), possède une fonction coût $f(X)$. Cette fonction, appelée aussi fitness, est utilisée pour mesurer la faculté d'adaptation de l'individu X à l'environnement (ou à l'objectif visé). On dit qu'un individu est d'autant mieux adapté à son environnement ou plus performant que le coût de la solution qu'il représente est plus faible. Les algorithmes génétiques s'appuient sur trois fonctionnalités :

- **la sélection** qui permet de favoriser les individus qui ont une meilleure fitness (le plus souvent la fitness est la valeur de la fonction objectif (coût) de la solution associée à l'individu).
- **le croisement** qui combine deux solutions parents pour former un ou deux enfants (offspring) en essayant de conserver les "bonnes" caractéristiques des solutions parents.
- **la mutation** qui permet d'ajouter de la diversité à la population en mutant avec une certaine probabilité, certaines caractéristiques (gènes) d'une solution.

L'Algorithme 3-3 présente la structure la plus simplifiée de l'algorithme génétique :

Algorithme 3-3. L'algorithme génétique

1: **Nom de la procédure** : *Genetic_Algorithm*
2: **Début**
3: **Initialise** P : une population de solutions ($|P| = n$ individus)
4: **Répéter**
5: $Sélection$: choisir deux solutions X et X' (expl. Par la technique de la roulette wheel)
6: $Croisement$: combiner les deux solutions parents X et X' pour former la solution enfant Y (il existe plusieurs types de croisements)
7: $Mutation$ de la solution enfant Y
8: Choisir la solution (l'individu) Y' qui va être remplacé dans la population P
9: Remplacer Y' par Y dans la population de solutions.
10: **Jusqu'à** satisfaction du critère d'arrêt
11: **Fin**

L'AG est une technique de recherche stochastique. A partir d'une population de solutions initiales et en répétant le cycle « sélection-croisement-mutation », il génère de nouveaux individus pour faire évoluer progressivement la population par générations successives, de telle sorte que ces individus soient plus performants que leurs prédécesseurs et en maintenant la taille de la population constante. L'arrêt de ce cycle est conditionné par un test qui peut être soit par un nombre d'itérations maximal ou bien l'absence de l'amélioration d'un critère à partir d'un nombre d'itérations fixées à l'avance.

- Concernant les travaux traitant le problème d'ordonnancement de Job shop par la méthode des algorithmes génétiques, un nombre important de références peut être trouvé. L'AG a été adapté la première fois pour le problème de Job shop par Lawrence [1985], puis par Goldberg [1989], ainsi que par Nakano et Yamada [1991]. Par la suite, et jusqu'à nos jours, un grand intérêt est attribué à cette méthode. Comme études faites sur l'état de l'art sur l'utilisation des AGs pour le JSP on peut citer les travaux de : [Cheng et al. 1996,1999] ; [Werner 2011] ; [Gen et Lin 2013] ; [Çaliş et Bulkan 2015] ; [Milošević et al. 2015] ; [Zhang et al. 2017] ; etc.

3.4.2.2 Optimisation par colonies de fourmis « Ant Colony Optimization (ACO) »

La colonie de fourmis est une métaheuristique qui a été proposée dans les années 1990 par Dorigo [1992], Dorigo et al. [1996] et Dorigo et Di Caro [1999]. Son idée de base est de résoudre des problèmes d'optimisation en imitant le comportement collectifs des fourmis réelles lorsqu'elles sont à la recherche d'une source de nourriture.

La Figure 3-2, illustre en trois étapes le comportement des fourmis lorsqu'elles vont se déplacer entre leur nid (colonie *C*) et une source de nourriture (*S*).

En réalité, Les fourmis sortent de leur nid pour chercher de la nourriture et déposent tout au long de leur parcours, une substance volatile chimique appelée « phéromone ». Elles perçoivent cette matière à laquelle elles sont très sensibles grâce à des récepteurs situés dans leurs antennes. Grâce à ce moyen de communication, les fourmis arrivent à résoudre collectivement des problèmes très complexes, notamment les problèmes de choix du plus court chemin entre une source de nourriture et leur nid.

En effet, au début, les fourmis explorent différents chemins en effectuant des déplacements aléatoires. En se déplaçant du nid à la source de nourriture et vice-versa, toutes les fourmis déposent leurs phéromones sur leur trajectoire. Au fait, les premières fourmis ayant trouvé la nourriture reviennent au nid et marquent deux fois leurs parcours. À cet effet, le chemin le

plus court finira par avoir la densité de phéromones la plus élevée ce qui permet d'augmenter le nombre de fourmis qui vont suivre cette trajectoire. En plus, puisque, la phéromone s'évapore au fil du temps, le chemin le plus long est de moins en moins emprunté et sa trace finit par disparaître presque complètement (Figure 3-2). Ainsi le choix des trajets que vont effectuer les autres fourmis est guidé par deux mécanismes qui sont le renforcement et l'évaporation de la phéromone. Pour avoir plus de détails sur l'algorithme qui donne le schéma d'optimisation de la méthode de colonies de fourmis, on renvoie le lecteur à consulter le livre de [Dréo et al. 2003].

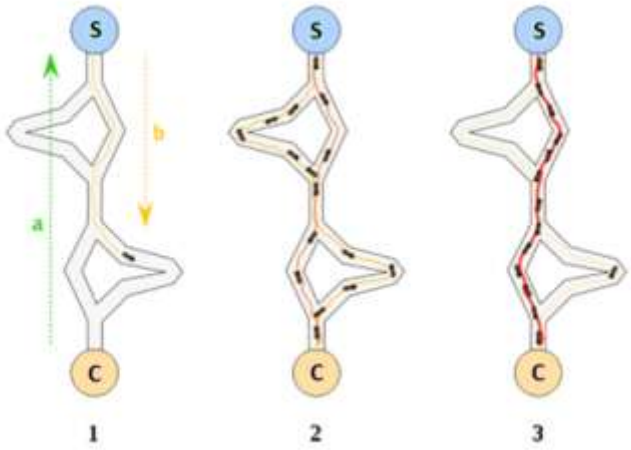

Figure 3-2 : Diagramme du comportement des fourmis [Wikipedia 2019]

- Une des premières applications de la métaheuristique des colonies de fourmis est le problème du voyageur de commerce et depuis elle a considérablement évolué et a eu plusieurs applications dans les problèmes d'ordonnancement. Pour le JSP, les premiers travaux fut proposés par Colorni et al. [1994]. Par la suite, plusieurs travaux sont apparus. On trouve qu'un algorithme à colonie de fourmis pour minimiser le makespan d'un problème de job shop a été développé par Udomsakdigool et al. [2008]. Chang et al. [2008] ont présenté un ACO pour obtenir la solution d'un problème de job shop multi-objectif qui prend en compte les critères quantitatifs et qualitatifs, ainsi que, Zhou et al. [2009] qui ont exposé la performance d'un ACO dans un problème de Job shop dynamique en testant différents environnements d'expérimentation. Eswaramurthy et al. [2009] ont présenté une recherche tabou hybridée avec un algorithme à colonie de fourmis pour résoudre un problème de Job shop qui considère l'influence de la phéromone au moment de sélectionner les voisinages pour améliorer la solution. Korytkowski et al. [2013] ont proposé un ACO pour minimiser le retard et le temps de

séjour d'un problème Job shop en utilisant des règles multi-attributs de lancement de production. On peut trouver plus de références dans la revue de littérature de Neto et Godinho Filho [2013]. Même récemment, Huang et Yu [2017], ont utilisé un ACO amélioré pour un problème d'ordonnancement de Job shop multi-objectif.

3.4.2.3 Optimisation par essaims particulaires « Particular Swarm Optimization (PSO)»

L'optimisation par essaims de particules (EOP) est une métaheuristique qui utilise une population de solutions candidates pour développer une solution optimale au problème donné. Son algorithme qui s'inscrit dans la famille des algorithmes évolutionnaires a été mise au point en 1995, par l'ingénieur électricien Russell C. Eberhart et le socio-psychologue James Kennedy [Eberhart et Kennedy 1995]. Cette méthode trouve son origine dans les observations faites lors des simulations informatiques des vols groupés d'oiseaux et de bancs de poissons de Reynold [1987] et Heppner et Grenander [1990].

En effet, on peut observer chez les animaux en essaims, des dynamiques de déplacement relativement complexes. Cette dynamique obéit à des règles et des facteurs bien spécifiques, à savoir :

- Chaque individu dispose d'une certaine « intelligence limitée » (qui lui permet de prendre une décision).
- Chaque individu doit connaitre sa position locale et disposer d'information locale de chaque individu se trouvant dans son voisinage.
- Obéir à ces trois règles simples, « rester proche des autres individus », « aller dans une même direction » ou « aller à la même vitesse ».

Tous ces facteurs et règles sont indispensables pour le maintien de la cohésion dans l'essaim, et permettent la mise en œuvre de comportements collectifs complexes et adaptatifs.

L'essaim de particules correspond à une population d'agents simples, appelés *particules*. Chaque particule est considérée comme une solution du problème, où elle possède une position (le vecteur solution) et une vitesse. De plus, chaque particule possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines » (informatrices) : chaque particule dispose en effet d'un groupe d'informatrices, historiquement appelé son *voisinage*.

Un essaim de particules, qui sont des solutions potentielles au problème d'optimisation, « survole » l'espace de recherche, à la recherche de l'optimum global. Le déplacement d'une particule est influencé par les trois composantes suivantes :

1. **Une composante d'inertie** : la particule tend à suivre sa direction courante de déplacement ;
2. **Une composante cognitive** : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
3. **Une composante sociale** : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

La Figure 3-3 illustre la stratégie de déplacement d'une particule, [Cooren 2008] :

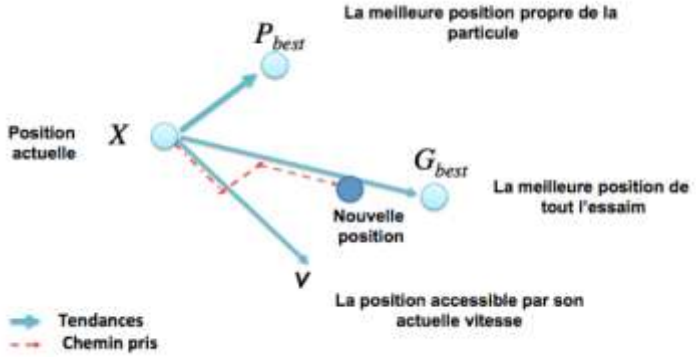

Figure 3-3 : La stratégie de déplacement d'une particule.

Au début de l'algorithme de l'OEP, les particules de l'essaim sont réparties au hasard dans l'espace de recherche. Par la suite, pendant le déplacement, chaque particule ajuste sa position selon sa propre expérience, et selon l'expérience des particules voisines, se servant de sa meilleure position produite et de celle de ses voisines. Ce comportement est semblable à celui du comportement humain consistant à prendre des décisions où les individus considèrent leur expérience antérieure et celle des personnes qui les entourent. L'OEP peut ainsi combiner des méthodes de recherche locale avec des méthodes de recherche globale.

Même si beaucoup de similitudes existent entre les méthodes évolutionnaires et l'OEP, cette dernière se distingue par le fait qu'elle n'utilise pas l'opérateur de sélection qui choisit les individus à garder dans la prochaine génération, en éliminant les individus les moins performants. L'optimisation par essaims particulaires se base sur la coopération entre les individus afin de faire évoluer chacun. Pour plus d'informations et de détails sur l'algorithme

OEP, le lecteur peut s'orienter aux références suivantes : [Clerc 2005] ; [Souier 2012] et [El Dor 2012].

- L'optimisation par essaims particulaires (EOP-PSO) a été appliquée pour le JSP en 2006, par Lian et al. Cependant, cette méthode n'a été efficace que pour des problèmes de petite taille. Plus tard, Lin et al [2010] ont testé et approuvé une PSO modifiée pour plus d'instances. Pour minimiser le Makespan pour un JSP, une version discrète de l'algorithme PSO a également été proposée par Rameshkumar et Rajendran [2018].

3.4.2.4 Optimisation Par Colonies D'abeilles : (Artificial Bee Colony (ABC))

Le comportement intelligent des abeilles pour la recherche d'une source de nourriture de bonne qualité dans la nature a été une bonne source d'inspiration pour Karaboga [2005] qui a introduit une nouvelle métaheuristique formulée par l'algorithme ABC (Artificial Bee Colony) [Karaboga et Basturk 2007]. Un algorithme à base de colonie d'abeilles artificielles.

- Depuis son apparition, la métaheuristique des colonies d'abeilles a attiré un grand nombre de chercheurs. Ils se sont intéressés à son étude et à son application dans la résolution de différents problèmes du monde réel. Elle a été appliquée avec succès sur certains problèmes liés aux réseaux de télécommunication, à l'optimisation des réseaux sans fil, au traitement d'images et de vidéos et même à certains problèmes d'ordonnancement [Karaboga 2014], [Agarwal et Yadav 2019].

Dans cette thèse, nos investigations sont orientées vers la métaheuristique des colonies d'abeilles. Pour cette raison, l'algorithme de colonie d'abeilles artificielles, sous sa forme fondamentale, et sous la nouvelle forme adaptée au problème de JSP, seront présentées avec plus de détails dans le chapitre 4.

3.5 Conclusion

Le but de nos investigations dans cette thèse est de pouvoir résoudre le problème d'ordonnancement de Job shop qui est considéré comme étant un problème d'optimisation combinatoire. Dans ce cas, avant de présenter les éventuelles méthodes de résolution, nous avons préféré commencer le chapitre par quelques notions de base concernant l'optimisation, et plus particulièrement ceux de l'optimisation combinatoire.

Etant donné l'importance du problème d'ordonnancement de Job shop, de nombreuses méthodes pour sa résolution ont été développées en recherche opérationnelle (RO) et en

intelligence artificielle (IA). Nous avons alors présenté quelques-unes des méthodes les plus utilisées dans la littérature et que l'on peut classer sommairement en deux grandes catégories, à savoir : les méthodes exactes et les méthodes approchées.

Comme il a été établi, la majorité des problèmes d'ordonnancement de Job shop sont NP-difficiles et leurs résolutions par des méthodes exactes ne sont, alors, pas possibles dans un contexte de temps de calcul et de mémoire limités. En effet, pour des problèmes ayant une taille industrielle, on ne peut pas trouver avec les méthodes exactes un ordonnancement optimal en un temps raisonnable. Heureusement il existe une deuxième catégorie de méthodes dites approchées. A défaut d'obtenir l'optimum, il faudrait se contenter d'une solution approchée. L'efficacité de ces méthodes dépend de leur capacité à s'approcher le plus possible -et en un temps raisonnable- de la valeur optimale du critère à optimiser. Parmi ces méthodes approchées, nous avons cité quelques méthodes heuristiques et quelques métaheuristiques.

La plus part des méthodes de résolution existantes, telles que les métaheuristiques, ont l'avantage de nous fournir des performances acceptables. Cependant, elles pourraient présenter certaines défaillances, que ce soit par exemple au niveau de la diversification ou de l'intensification. Pour pallier à ce défaut, la tendance actuelle est l'émergence des méthodes hybrides, qui s'efforcent de tirer parti des avantages spécifiques de deux ou plusieurs approches différentes en les combinant. Notre contribution dans ce sens sera présentée au chapitre 5 dans lequel on va commencer par définir les méthodes hybrides.

Toutefois, il est vraiment évident de reconnaitre que le domaine d'étude du JSP est très riche en travaux. Dans ce cas, il est très difficile d'établir un état de l'art exhaustif et on a choisi alors de citer dans ce chapitre, juste les travaux les plus pertinents. Bien qu'on trouve de plus en plus de travaux sur les extensions du Job shop, on a constaté que le Job shop sous sa forme classique continu à intéresser les scientifiques car il demeure encore un problème ouvert.

La revue de littérature indique que pour résoudre les problèmes d'ordonnancement de Job shop, il y a eu un intérêt significatif dans l'application des métaheuristiques. Cependant, on remarque que la métaheuristique des colonies d'abeilles artificielles (ABC), qui est une nouvelle technique reconnue très efficaces pour résoudre divers problèmes d'optimisation [Karaboga et al. 2014], est très peu développée pour le traitement d'un JSP.

De ce fait, nous nous sommes intéressés dans cette thèse à appliquer la métaheuristique des colonies d'abeilles artificielles pour la résolution du JSP. Il faut juste noter que cette

métaheuristique formulée par l'algorithme ABC a été formulée initialement pour les problèmes d'optimisation continue. Notre objectif maintenant est d'établir la version discrète de l'ABC, afin de l'adapter au JSP qui est un problème de nature combinatoire (chapitre 4).

Chapitre 4

Algorithme CABC
(Combinatorial Artificial Bee Colony)
pour l'ordonnancement
d'un Job shop

La république des abeilles, c'est la Tour de Babel renversée…

Personne ne parle et chacun s'entend....

République vraiment exemplaire, où l'on agit au lieu de discuter,

où l'on ne prend les armes que pour défendre

la frontière et protéger le travail….

République admirable, où l'industrie est un honneur,

la fraternité un principe,

l'égalité un fait,

l'entente une coutume,

le travail une loi,

le respect de la mère une religion,

l'éducation de la jeunesse un besoin du cœur et une affaire d'Etat....

Auteur inconnu.

4.1 Introduction

Lorsque l'on parle d'abeille, l'image qui nous vient immédiatement est celle d'une ruche abritant des milliers de ces insectes mythiques capables de s'auto-organiser pour assurer la pérennité de la ruche et de produire ce produit si précieux qu'est le miel. Les abeilles vivent en colonie et y travaillent sans relâche suivant une organisation sociale très élaborée. Cette exemplaire organisation a donné à la colonie d'abeilles le mérite d'être étudiée, suivie de près et d'être un bon modèle à suivre.

En entreprise, l'équipe de travail assimilée à une colonie d'abeilles est formée d'un groupe d'individus qui doivent s'unifier afin de partager un objectif commun et de réussir un projet d'entreprise. Ces individus, tout comme les abeilles, partagent les problèmes à affronter, mais aussi les réussites. D'autre part, de nos jours, on constate que les surprenantes facultés cognitives des abeilles inspirent de plus en plus l'intelligence artificielle, où certaines méthodes ont été développées pour résoudre des problèmes très complexes.

Ainsi, le comportement intelligent des abeilles pour la recherche d'une source de nourriture de bonne qualité dans la nature a été une bonne source d'inspiration pour plusieurs chercheurs, entre autres, Karaboga qui a introduit en 2005, une nouvelle métaheuristique formulée par l'algorithme ABC (Artificial Bee Colony) [Karaboga 2005], [Karaboga et Basturk 2007].

Dans notre travaille, on s'intéresse particulièrement à l'algorithme ABC qui est un algorithme d'optimisation à base de colonie d'abeilles *artificielles*. Nous allons donc commencer dans ce chapitre par détailler les différentes étapes de la version fondamentale de cet algorithme. Cependant, il serait plus opportun de présenter en premier lieu, le comportement d'une colonie d'abeilles *naturelles*.

D'autre part, il se trouve que l'algorithme ABC fondamental, a été conçu à l'origine pour résoudre des problèmes d'optimisation de nature continue et ne peut pas être utilisé directement pour le cas combinatoire. Dans cette thèse, afin de résoudre le problème de JSP (Job shop Scheduling Problem) qui est de nature combinatoire, certaines modifications doivent être apportées à l'algorithme ABC fondamental. Nous avons donc proposé une nouvelle version appelée : « *Combinatorial Artificial Bee Colony*» (CABC) algorithm.

Nous allons présenter dans ce chapitre, le CABC algorithme, qui est une adaptation de l'algorithme de base des colonies d'abeilles au problème combinatoire de l'ordonnancement

d'un Job shop (JSP). Notre objectif dans la résolution du JSP, est de trouver l'ordonnancement des opérations des jobs qui minimise la valeur du Makespan.

Le premier point à prendre avec soin lors d'une adaptation d'un algorithme à un problème donné, c'est le choix de la représentation de la solution. Pour cette raison, nous avons réservé une section juste avant de passer à la présentation des différentes étapes du CABC proposé. Par la suite, afin d'établir le réglage des paramètres de cette métaheuristique et pour démontrer son efficacité, des simulations ont été effectuées sur plusieurs Benchmarks de Job shop.

4.2 Les abeilles naturelles

Les abeilles (*Apis mellifica*) sont des insectes sociaux qui vivent en groupe appelé *essaim d'abeilles* ou plus couramment *colonie d'abeilles*. Les recherches récentes ont révélé les capacités cognitives mises en jeu par les abeilles lors de la collecte de nourriture [Avarguès-Weber, 2013]. Elles sont d'autant plus étonnantes que l'abeille a un cerveau minuscule qui mesure moins de $1mm^3$, pèse autour d'un gramme et contient environ 960.000 neurones. Ce n'est pas bien lourd comparé au dm^3 du cerveau humain, et surtout à ses 100 milliards de neurones !

L'abeille prise isolement comme un individu, se démarque par des **comportements individuels surprenants**. Elle est capable d'agir et de travailler en solitaire tout en supportant le poids social. Cependant, les performances de l'abeille à miel ne résident pas seulement dans son individualité, mais dans le fait qu'elle vit en société au sein d'une colonie très organisée et collaborative.

La colonie d'abeilles vit dans une ruche que les abeilles construisent elles-mêmes sous forme de rayons d'alvéoles en cire (Nid) ou éventuellement dans des ruches que les hommes ont construites pour y récolter plus facilement le miel produit par ces abeilles.

Les abeilles d'une colonie appartiennent à trois classes différentes. On trouve habituellement dans une ruche : une seule reine, quelques centaines de faux-bourdons (les mâles), et près de 40000 à 60000 ouvrières (Figure 4-1). Ils vivent tous dans la ruche dont une partie est composée du couvain qui abrite les individus immatures tels que : les œufs, les larves et les nymphes (Figure 4-2).

Reine Ouvrière Faux-bourdon

Figure 4-1 : Les habitants de la ruche.

Figure 4-2 : Evolution : œuf-larve-nymphe-abeille.

4.2.1 Qui fait quoi ?

Chaque abeille a un travail précis à faire. Que ce soit la reine, le faux-bourdon, ou l'ouvrière, chacune d'elle contribue à construire la ruche et veiller à la survie de l'espèce par la collaboration au sein de la colonie.

- **La reine :**

Il n'y a qu'une reine par colonie. Elle est plus grande en taille que les faux bourdons et fait presque le double de la taille des ouvrières. Son rôle principal est la procréation, car avec un abdomen bien plus développé que celui des autres, c'est la seule abeille capable d'assurer la ponte des œufs. Elle se nourrit de gelée royale et a une espérance de vie d'environ 3 à 5 ans. Elle peut pondre des œufs (fertiles ou stériles) à un bon rythme (200 000 œufs par an) pendant ses 2-3 premières années. Les œufs stériles (non fécondés) deviennent des faux bourdons, tandis que les œufs fertiles (fécondés) deviennent des ouvrières ou de nouvelles reines. Une larve nourrie à la gelée royale donne naissance à une nouvelle reine.

Bien que la reine puisse avoir un second objectif tel que, d'organiser et de motiver (grâce aux phéromones) les ouvrières pour qu'elles travaillent pour la ruche, elle n'a pas vraiment de

rôle hiérarchique. En effet, il est impossible par exemple, que 50 000 abeilles attendent les ordres d'une seule d'entre elle pour agir : ces dernières ont la capacité de s'auto-organiser.

- **Les faux-bourdons :**

Ils sont en quelques centaines, mais juste certains d'entre eux sont les pères de la colonie. La durée de vie d'un faux-bourdon est de 21 à 32 jours, depuis le printemps jusqu'au milieu de l'été. Cependant, à la fin de l'été et en automne, il peut vivre jusqu'à 90 jours. Bien que les faux-bourdons leur arrive de contribuer à entretenir de la chaleur ou de la fraîcheur dans la ruche, ils restent des éléments paresseux dans une ruche. Ils ne butinent pas, ne nettoient pas, ne peuvent pas défendre la ruche contre les envahisseurs car ils ne possèdent pas de dard et ils mangent du miel. Pour toutes ces raisons, ils finissent par être expulsés de la ruche une fois la période des vols nuptiaux terminée et dès que les réserves diminuent.

- **Les ouvrières :**

L'espérance de vie des ouvrières varie selon les saisons : de 30 à 45 jours pour les abeilles nées au printemps et en été, à plusieurs mois pour celles qui naissent à l'automne et passe leur hiver en hibernation. Au fur et à mesure de leur existence et de leur maturation physiologique, elles changent de rôles. Ainsi, au cours de sa vie, en passant d'une activité à l'autre, chaque abeille ouvrière aura participé à tous les travaux de la ruche. Etant donné que toute la vie des abeilles est déterminée par les glandes, suivant que telle ou telle glande se développe, l'abeille ouvrière entreprend une fonction donnée. Elle exerce dans la ruche ce que l'on appelle : « les 7 métiers de l'abeille », à savoir :

1. **Nettoyeuse**, (1^{er} au $3^{ième}$ jour) : Lorsqu'elle émerge, la première tâche de la jeune abeille est de grignoter l'opercule qui recouvrait sa cellule. Puis, Elle nettoie et prépare les cellules (alvéoles) pour qu'elles puissent ensuite accueillir la nourriture ou recevoir les œufs. À cet âge elle a pour mission de garder la ruche propre et en bonne santé.
2. **Nourrice**, ($3^{ième}$ au $6^{ième}$ jour) : Ensuite, l'abeille nourrice s'occupe de nourrir les œufs et les larves qui grandissent dans les alvéoles. Elle leur donne un mélange de miel, de nectar et de gelée royale. Attentionnée, la nourrice peut rendre visite à une seule larve jusqu'à 1 300 fois par jour !
3. **Architecte** (Cirière), ($6^{ième}$ au $12^{ième}$ jour) : L'abeille cirière ou maçonne produit des pastilles de cire sur ses glandes cirières, situées sur la face interne de son abdomen. En malaxant ces minuscules plaquettes avec ses mandibules, elle parvient à construire des rayons d'alvéoles.

4. **La manutentionnaire** (13ième au 16ième jour) : L'ouvrière joue les manutentionnaires. Postée à l'entrée de la ruche, elle accueille les butineuses rentrantes d'expédition, le jabot chargé de nectar. Le précieux liquide est réceptionné par l'abeille, dans son propre jabot. Elle le régurgitera à plusieurs fois, pour transformer progressivement le nectar en miel, grâce à ses enzymes. Une fois le miel obtenu, il sera déversé dans une alvéole qui sera fermée par de la cire pour bien préserver le miel stocké. Vue cette dernière fonction, elles sont appelées aussi les chimistes.
5. **Ventileuse** (17ième au 19ième jour) : Lorsqu'elle devient plus expérimentée, l'abeille occupe des rôles plus exigeants. La ventilation par battement des ailes exige beaucoup d'énergie; l'abeille fait circuler l'air dans la ruche pour contrôler la température et le taux d'humidité.
6. **Gardienne** (19ième au 21ième jour) : La gardienne veille à l'entrée de la ruche : seules ses congénères pourront réintégrer la colonie. Pas question de se faire piller! C'est aussi elle qui sonne l'alarme lors d'une attaque d'un prédateur, comme un ours, une mouffette ou des guêpes.
7. **Butineuse** (21ième à son dernier jour) : Enfin, la butineuse expérimentée fait constamment l'aller-retour entre les fleurs et la colonie pour rapporter du nectar, du pollen et de la propolis. Elle peut voyager dans un rayon de 5 km autour de sa ruche. Un travail exigeant qui, ultimement, la fera mourir d'épuisement.

Au fait, La succession de ces tâches et la durée allouée à leur exécution ne se font pas toujours de façon stricte, et même dans cet ordre parfait, des fluctuations peuvent se produire si le besoin s'en fait sentir. Les ouvrières sont capables de changer de fonction et de s'adapter aux besoins immédiats de leur colonie.

4.2.2 Le butinage

4.2.2.1 Définition

Le butinage revient à faire des provisions en : Nectar, pollen, propolis et en eau. Ceux sont les quatre matières de base dans une ruche. Le nectar (un liquide sucré contenu dans les fleurs) deviendra miel, aliment énergétique des abeilles. Le pollen est l'élément mâle des fleurs il est riche en protéines nécessaires au développement des larves. La propolis, très gluante, est récoltée sur les bourgeons des arbres. Les abeilles s'en servent dans la ruche comme un mastic, pour colmater les trous ou pour aseptiser l'atmosphère. Enfin, l'eau est indispensable, car les larves en pleine croissance en consomment beaucoup.

Vers le milieu de leur existence, à partir du 21ième jour, les abeilles ouvrières adultes perçoivent une prime d'ancienneté et deviennent des butineuses. Butiner signifie pour elle, voler de fleur en fleur à la recherche de nourriture pour l'approvisionnement de la ruche. En butinant l'abeille assure également une action très importante qui est la pollinisation, c'est-à-dire le transport du pollen permettant la reproduction des plantes. La Figure 4-3 montre une abeille en train de recueillir du pollen sur la fleur et de fabriquer avec une pelote sur ses pattes arrières.

Figure 4-3 : Une butineuse chargée d'une pelote de pollen recueilli.

4.2.2.2 La communication chez les abeilles :

La communication chez les abeilles est très élaborée. Elle permet la cohésion de la ruche, la reconnaissance entre individus, la diffusion des alertes, mais également le repérage des sources de nourriture, d'eau, ou de résine. Elle est basée sur : des échanges tactiles à l'aide des *antennes*, des messages chimiques appelés *phéromones*, et sur un langage bien typique à elles « *la danse des abeilles* ».

En 1946, Un chercheur allemand, d'origine autrichienne, nommé Karl Von Frisch, dont ses recherches lui ont apporté un prix Nobel, a décodé le prodigieux « *langage des abeilles* » [Von Frisch et Lindauer 2003]. Il a observé que c'est à travers la danse, qu'une abeille communique à ses semblables à son retour dans la ruche, la direction, la distance et la qualité de la source de nourriture qu'elle vient de trouver. En effet, lorsqu'une butineuse revient à la ruche au terme d'une exploration d'une source de nourriture, elle exécute dans l'obscurité de la ruche une sorte de danse. Les autres abeilles perçoivent sont agitation et tente de décoder les informations contenues dans ses mouvements. Le type de danse est différent selon la distance à laquelle se trouve la nourriture, deux cas se présentent : la nourriture se trouve à moins de 100 mètres, ou à une distance supérieure à 100 mètres :

- Lorsque la source est à moins de 100 mètres, une danse en rond signale simplement la proximité du repas, sans indication de direction. L'abeille fait un tour afin de décrire l'endroit et une fois revenue à son point de départ, elle fait demi-tour et reprends sa figure de danse en sens inverse.
- Lorsque la source est supérieure à 100 mètres, l'abeille effectue une danse en huit, appelée : « *danse frétillante* ». Contrairement à l'autre danse, celle-ci indique la distance, la direction et la qualité de la source. Grâce à des oscillations abdominales et des vibrations émises; la distance est indiquée par le nombre et la vitesse de tours effectués par l'abeille sur elle-même. Quant à la direction, elle est exprimée par la position de la source de nourriture par rapport au soleil. En effet, l'abeille dirige sa danse dans une direction qui fait un certain angle par rapport à la verticale ; et cet angle indique la direction de la nourriture par rapport au soleil (Figure 4-4, [Ferme St Antoine 2020]). Cette danse pourrait être mieux perçue par une animation vidéo, comme par exemple sur le lien [Elytra 2019], [Maud Miran 2014] ou [Les ruchers de la découverte 2014].

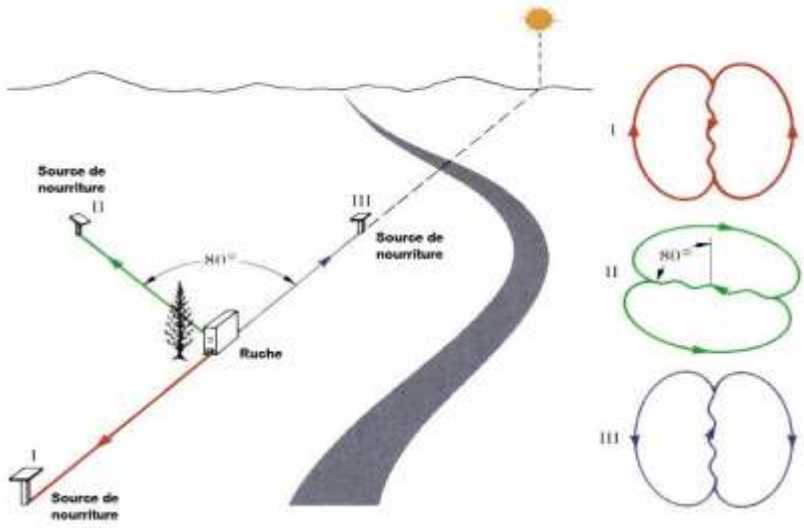

Figure 4-4 : La danse frétillante des abeilles.

4.2.2.3 La procédure de butinage

Selon la situation dans laquelle la butineuse peut se retrouver, elle peut jouer l'un des trois rôles suivants :

- Butineuse active (Employed bee), dite aussi : abeille active ou abeille travailleuse.
- Butineuse inactive (Onlooker bee), dite aussi : abeille inactive ou abeille spectatrice.
- Butineuse éclaireuse (Scout bee), dite aussi : abeille scoute.

Dans ce document, nous avons opté pour les appellations : abeille *active*, abeille *spectatrice* et abeille *scoute*.

En attribuant ces différents rôles aux abeilles butineuses, la procédure de butinage se déroule selon les étapes suivantes, [Seeley, 1995] :

- **Étape 1 :** Le processus de recherche de nourriture commence lorsque les abeilles butineuses **actives** quittent la ruche et volent jusqu'à une source de nourriture *prédéterminée*, explorent les sources de nourriture voisines, recueillent la nourriture et reviennent à la ruche. Les abeilles scoutes (éclaireuses) explorent aussi -mais *aléatoirement*- la zone autour de la ruche à la recherche de nouvelles sources de nourriture attirantes. Il n'est pas rare que cette zone fasse jusqu'à 13 km². Dans une ruche, environ 10 % des butineuses sont des abeilles scoutes.

  Pendant le butinage, l'abeille remplit son estomac en 30 à 120 minutes environ et le processus de fabrication du miel commence par la sécrétion d'une enzyme qu'elle ajoute au nectar stocké momentanément dans son estomac. Aussitôt de retour à la ruche, elle va le décharger dans les endroits appropriés. Cependant, la mission de l'abeille active ne s'arrête pas là ! Elle va aller dans une zone commune de la ruche, appelée piste de danse, pour exécuter la fameuse danse des abeilles. Cette danse est essentielle pour la communication de la colonie. Par cette danse, elle va s'adresser aux abeilles spectatrices qui attendaient dans la ruche. Elle va partager avec elles des informations sur la position, la distance et la qualité de la source de nourriture qu'elle vient d'exploiter.

- **Étape 2:** C'est une étape de recrutement des abeilles **spectatrices**. En effet, au début de cette étape, les abeilles spectatrices sont inactives et ne font qu'attendre à l'intérieur de la ruche, les informations qui vont être fournies par de nombreuses abeilles actives dansantes. Ainsi, elles vont avoir une idée sur toutes les sources d'alimentation proposées. Néanmoins, elles ne sont attirées que par les sources de nourriture les plus intéressantes (les plus riches en pollen ou en nectar et les plus proches). Par conséquent, selon une probabilité proportionnelle à la qualité d'une source de nourriture, une abeille spectatrice est recrutée pour l'exploitation de cette source de nourriture particulière et deviendra ainsi une nouvelle abeille active.

Il est claire que cet ingénieux langage de communication permet aux abeilles de la colonie de rapidement repérer et monopoliser les sources de nourriture les plus rentables, tout en ignorant celles qui sont de mauvaise qualité.

- **Étape 3:** Qu'une abeille soit scoute ou spectatrice, chaque fois qu'elle trouve une source de nourriture à exploiter, elle devient une abeille active. D'autre part, chaque fois qu'une source de nourriture est pleinement exploitée (expl : tout le nectar est épuisé), toutes les abeilles actives qui lui sont associées l'abandonnent et peuvent redevenir éclaireuses ou spectatrices.

Les caractéristiques d'auto-organisation et de division du travail définies par Bonabeau et al. [1999] et les principes de satisfaction énoncés par Millonas [1994] pour l'intelligence des essaims sont clairement et fortement observés dans les colonies d'abeilles. Ayant cette capacité instinctive connue sous le nom d'intelligence d'essaim, la colonie d'abeilles à miel est qualifiée pour être un essaim très intéressant dans la nature et attire l'intérêt de nombreux chercheurs. Teorodorvic et al. [2015] et Karaboga et al. [2014], ont donné un excellent aperçu des algorithmes inspirés du comportement des abeilles dans la nature, ainsi que plusieurs exemples de leurs applications.

4.3 L'algorithme de colonie d'abeilles artificielles (ABC Algorithm)

Pour découvrir de bonnes solutions à un problème d'optimisation, Karaboga et Basturk [2007] ont proposé l'algorithme « *Artificial Bee Colony*» (ABC). C'est un algorithme d'optimisation qui repose sur des concepts issus de la recherche de nourriture chez les abeilles butineuses. Les composantes essentielles de l'ABC sont donc établies comme suit :

- **Une source de nourriture** : représente une solution possible pour un problème d'optimisation.
- **La valeur de la Fitness** représente la rentabilité d'une source de nourriture. Pour simplifier, c'est la seule valeur que prend la fonction objectif pour une solution possible.
- **Les agents abeilles** sont l'ensemble des agents de calcul.

Les agents de l'algorithme ABC sont classés en trois groupes : les abeilles actives, les abeilles spectatrices et les abeilles scoutes. La colonie est également partagée en abeilles actives (50%) et en abeilles inactives (spectatrices + scoutes).

La métaheuristique des colonies d'abeilles est une méthode stochastique, itérative et à population de solutions. Chaque solution dans l'espace de recherche consiste en un ensemble

de paramètres d'optimisation, qui représente une "localisation" de la source de nourriture. Le nombre d'abeilles actives est égal au nombre de sources de nourriture, c'est à dire qu'il y aurait une abeille active pour chaque source de nourriture. Puisque les abeilles actives représentent la moitié de la colonie, la taille de la population de solutions est égale à la moitié de la taille de la colonie.

Les abeilles actives seront chargées d'explorer leurs sources de nourriture et de partager des informations concernant ces sources pour recruter des abeilles spectatrices. À la base de ces informations, les abeilles spectatrices prendront une décision pour choisir une source de nourriture. La source de nourriture qui a la meilleure qualité aura la plus grande probabilité d'être sélectionnée par les abeilles spectatrices. Une abeille active dont la source de nourriture est de faible qualité est rejetée par les abeilles actives et spectatrices et se transforme en abeille scoute à la recherche aléatoire de nouvelles sources de nourriture. Le pseudo-code de L'algorithme ABC est donné comme suit :

Étape 1 : phase d'initialisation

Répéter :

Étape 2 : mise à jour des solutions possibles par les abeilles actives

Étape 3 : sélection des solutions possibles par les abeilles spectatrices

Étape 4 : mise à jour des solutions possibles par les abeilles spectatrices

Étape 5 : prévention des solutions sous-optimales par les abeilles scoutes

Mémorisation de la meilleure solution obtenue jusque là

Jusqu'à : satisfaction du critère d'arrêt (nombre d'itérations maximal).

Les étapes de l'algorithme seront détaillées par la suite :

- **Étape 1 : phase d'initialisation**

La position d'une source de nourriture est représentée par la solution X_i avec $i \in \{1,2,3,\dots,SN\}$. SN est le nombre de sources de nourriture (Source Number). SN représente donc la taille de la population de solutions. $SN = CS/2$, avec CS étant la taille de la colonie (Colony Size).

La solution est donnée par un vecteur $X_i \in \mathbb{R}^D$, avec $X_i = [x_{i1}, x_{i2}, x_{i3}, \dots, x_{iD}]$. D est la dimension de la solution, il représente le nombre de paramètres à optimiser.

En premier lieu, par les abeilles scoutes et suivant l'équation (4.1), un ensemble de solutions initiales est généré aléatoirement dans le domaine de définition des paramètres à optimiser :

$$x_{ij} = x_j^{min} + \propto (x_j^{max} - x_j^{min}) \quad (4.1)$$

Où $i = 1, \dots, SN$; $j = 1, \dots, D$ et $\propto$ un nombre réel aléatoire choisi dans l'intervalle $[0,1]$.

x_j^{min} est la limite inférieure de la position de la source de nourriture à la dimension j et x_j^{max} est la borne supérieure de la position de la source nourriture à la dimension j.

- **Étape2 : Mise à jour des solutions possibles par les abeilles actives**

La position de la nouvelle source possible de nourriture découverte par une abeille active est calculée à partir de l'équation (4.2) :

$$u_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \quad (4.2)$$

U_i est la nouvelle solution possible qui est modifiée à partir de la valeur de la solution précédente (x_{ij}) sur la base d'une comparaison avec la position de sa solution voisine (x_{kj}) choisie au hasard.

ϕ_{ij} est un nombre aléatoire choisi entre $[-1,1]$, utilisé pour ajuster l'ancienne solution pour devenir une nouvelle solution dans l'itération suivante.

$k \in \{1,2,3, ..., SN\} \wedge k \neq i$ et $j \in \{1,2,3, ..., D\}$ sont des indices choisis au hasard. La différence entre x_{ij} et x_{kj} est une différence de position dans une dimension particulière (le j-ième indice d'une position dans X_i et X_k).

La fonction objectif détermine la qualité d'une solution. Elle peut être représentée par :

$$F(X_i)$$

Si une nouvelle source de nourriture (U_i) est meilleure que l'ancienne source (X_i), cette ancienne source est alors remplacée par la nouvelle source de nourriture.

- **Étape 3 : Sélection des solutions possibles par les abeilles spectatrices**

Lorsque les abeilles actives retournent à leur ruche, elles partagent avec les abeilles spectatrices des informations sur les solutions qu'elles viennent de trouver. Les abeilles spectatrices sélectionnent ces solutions sur la base d'une *probabilité*. Les solutions ayant la plus grande valeur de fitness ont une plus grande chance d'être sélectionnées par les abeilles

spectatrices que celles ayant de faibles fitness. La probabilité pour qu'une source de nourriture soit sélectionnée peut être obtenue à partir de l'équation (4.3) :

$$P_i = \frac{fit_i}{\sum_{l=1}^{SN} fit_l} \tag{4.3}$$

Où fit_i est la valeur de la fitness de la i-ième source de nourriture. Cette valeur est liée à la valeur de la fonction objectif $F(X_i)$ de la source de nourriture i.

- **Étape 4 : Mise à jour des solutions possibles par les abeilles spectatrices**

D'après les informations obtenues des abeilles actives, les abeilles spectatrices sélectionnent les sources de nourriture les plus intéressantes. Comme à l'étape 2, Les sources de nourriture sélectionnées sont ensuite mises à jour en utilisant l'équation (4.2) et l'ancienne source de nourriture est remplacée par une nouvelle source si celle-ci est de meilleure qualité.

- **Étape 5 : Prévention des solutions sous-optimales par les abeilles scoutes :**

Cette étape est effectuée pour la réaffectation des abeilles actives dont les contributions sont jugées de faible qualité et donc rejetées. Ces abeilles vont devenir des abeilles scoutes (éclaireuses) qui vont, au hasard, rechercher de nouvelles sources de nourriture (solutions). La nouvelle position aléatoire choisie par l'abeille scoute sera calculée à partir de l'équation (4.1).

4.4 Optimisation combinatoire à base de colonies d'abeilles

L'algorithme proposé est appelé « *Combinatorial Artificial Bee Colony*» (CABC) algorithme. Car il est le fruit de l'adaptation de la version continue de l'algorithme ABC au problème combinatoire de l'ordonnancement du Job shop.

Soit à résoudre un problème d'ordonnancement de Job shop de *n* jobs et *m* machines; où chaque job nécessite l'exécution de *m* opérations ordonnées (soumises à des contraintes de précédence).

La première étape dans la résolution d'un problème d'ordonnancement, c'est le choix de la représentation de la solution. Il doit être fait en parfaite adéquation avec l'environnement du problème à traiter. Avant de présenter l'algorithme proposé, nous allons aborder ce point dans la section suivante.

4.4.1 Représentation de la solution du JSP

Dans l'algorithme CABC, chaque solution correspond à une source de nourriture 'X' exploitée par une abeille. On appeler 'X' : «solution», «source de nourriture», « Food source » ou « ordonnancement », cela revient au même.

Pour représenter notre solution, nous avons opté pour la "*représentation basée sur les opérations*" avec "*répétition de jobs*" ("*operation-based representation"* with "*job repetition*"). (déjà définie en chapitre 2, section 2.5.2).

Ainsi, la solution de ce JSP est une liste d'opérations ordonnancées et est représentée dans notre algorithme CABC en tant que source de nourriture 'X'. Cette source de nourriture 'X' est un vecteur de (n × m) dimensions, chaque dimension représente une opération d'un job. Selon la représentation de la solution considérée, chaque job apparaît exactement *m* fois dans la solution 'X'. La Figure 4-5 illustre un exemple de représentation de solution pour un problème d'ordonnancement de Job shop.

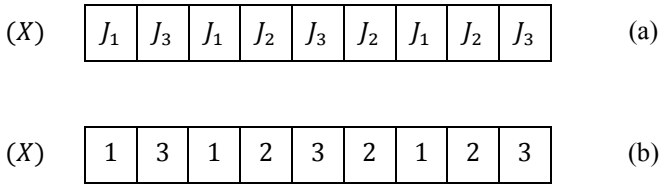

Figure 4-5: Exemple d'une solution pour JSP (3-job×3-machine).

Dans la solution (X) de la Figure 4-5(a), J_i représente l'opération du job i. Puisque chaque job nécessite trois opérations, il apparaît trois fois dans cette liste d'opérations ordonnancées. Le premier J_1 correspond à la première opération du job J_1. Le premier J_3 correspond à la première opération du job J_3. Le deuxième J_1 correspond à la deuxième opération du job J_1, etc.

En fait, dans les programmes de calcul, nous utilisons pour (X) la forme donnée par la Figure 4-5 (b), dans laquelle seulement une répétition des numéros des jobs est utilisée.

4.4.2 L'algorithme CABC proposé

Le pseudo-code de l'algorithme CABC proposé pour résoudre le JSP est donné par l'algorithme 4.1 et ses différentes étapes sont détaillées dans les prochaines sections.

Algorithme 4.1 : **Combinatorial Artificial Bee Colony (CABC) algorithm**

Input : $CS, \beta, Limit, NI\_Max$, les paramètres du JSP
Output : La meilleur solution globale X_{Gbest}, ${C^*}_{max}$
$SN = CS/2$
For $i = 1 \ to \ SN$
    Générer aléatoirement la source de nourriture X_i
    Initialiser le compteur des essais invalides, $trial_i = 0$
End For
$it = 1$
repeat
    phase des abeilles actives
    phase des abeilles spectatrices
    phase des abeilles scoutes
    Mémoriser la meilleure solution trouvée jusqu'à présent.
    $it = it + 1$
Until $it = NI\_Max + 1$
Return X_{Gbest}

4.4.2.1 Phase d'initialisation

Dans cette phase, nous commençons par définir les paramètres de l'algorithme, tels que:

- CS : la taille de la colonie,
- $SN = CS/2$: le nombre de sources de nourriture qui est égal au nombre d'abeilles actives (employed bees), et égal au nombre d'abeilles spectatrices (onlooker bees),
- *N_ch* : Le nombre d'opérations sélectionnées pour être modifiées lors de la mise à jour.
- β : un paramètre de seuil pour contrôler le niveau de qualité des solutions sélectionnées durant la phase des abeilles spectatrices.
- $Limit$: Ce paramètre représente la valeur limite du nombre de fois qu'une solution soit mise à jour sans subir d'amélioration. C'est-à-dire, si par des mises à jour, on va essayer plusieurs fois d'améliorer la solution X_i et on n'y arrive pas au bout de '$Limit$' tentatives, on l'abandonne et on la remplace dans la population de solutions par une nouvelle solution choisie au hasard pour créer de la diversification (c'est la phase de l'abeille Scoute).
- NI_max: Le nombre d'itérations maximal.
- Les paramètres du Job shop, tels que : le nombre de jobs, le nombre de machines, le temps de traitement des opérations sur chaque machine et les séquences des machines suivies par chaque job.

Ensuite, des solutions initiales sont générées aléatoirement. Bien que ces solutions soient construites en générant une suite aléatoire de nombres de jobs (selon la taille du JSP), elles seront toujours des solutions admissibles (ordonnancement réalisables). En effet, lorsque la «représentation basée sur les opérations» avec «répétition de job» est utilisée pour la représentation de solution, les contraintes de précédence sont toujours respectées.

Sous sa version originale, l'algorithme ABC, cherche à trouver la solution qui maximise la fitness. Cependant, notre objectif est de résoudre le JSP avec une minimisation du Makespan (le C_{max}). Par conséquent, dans le CABC algorithme, la fitness sera calculée comme suit:

$$fit_i = \frac{1}{C_{max_i}} = \frac{1}{F(X_i)} \tag{4.4}$$

Où fit_i représente la valeur de la fitness de la source de nourriture X_i et $F(X_i)$ est la fonction objectif de la source de nourriture X_i ; (notée $C_{max}(X_i)$ ou C_{max_i}).

Dans l'algorithme CABC, une autre variable $trial_i$ est affectée à chaque source de nourriture (X_i). $trial_i$ est un compteur d'essais infructueux pour lesquels la source de nourriture (X_i) n'est pas améliorée après sa mise à jour. C'est un indicateur pour trouver des sources de nourriture (solutions) à abandonner dans les prochaines itérations après un nombre d'essais infructueux égale à $Limit$. Au début, tous les compteurs $trial_i$ $(i = 1,2, \dots, SN)$, sont initialement mis à zéro.

4.4.2.2 Phase des abeilles actives (Employed bees phase)

Dans cette phase, l'abeille active génère une nouvelle source de nourriture (V_i), *en mettant à jour* (updating) l'ancienne source de nourriture déjà exploitée (X_i), avec une source de nourriture voisine (X_k) prise aléatoirement parmi les autres sources de nourriture dans la population de solutions. Le pseudo-code de cette phase est donné par l'algorithme 4.2.

(V_i) est le résultat d'un croisement entre (X_i) et (X_k). Afin d'obtenir une nouvelle solution valable (V_i), le mécanisme du *Position Base Crossover* (PBX) est utilisé [Syswerda, 1991]. Un exemple de ce type de croisement est illustré par la Figure 4-6. La procédure PBX est décrite par les étapes suivantes :

- **Étape A:**

Initialement, la nouvelle source de nourriture est créée en copiant dans un nouveau vecteur (solution) (V_i) toutes les opérations de l'ancienne source de nourriture (X_i), à l'exception des opérations qui seront sélectionnées de manière aléatoire pour être modifiées. Le nombre

d'opérations sélectionnées pour être modifiées est noté *N_ch*. Pour notre JSP avec *n* jobs et *m* machines, la solution est donnée par la «représentation basée sur les opérations» avec «répétition de jobs». Ainsi, dans la solution (X), chaque job apparaît exactement *m* fois. Dans ce cas, pour éviter les modifications redondantes, le nombre de changements *N_ch* sera en rapport avec le nombre d'opérations requises pour chaque job. Donc le choix de la valeur de *N_ch* sera en relation avec le nombre de machines *m*. Les *N_ch* positions vides dans (V_i) seront remplies à partir de (X_k) comme ça va être indiqué dans l'étape suivante.

- **Étape B:**

Les éléments de la source de nourriture voisine (X_k) seront pris de gauche à droite et placés dans les emplacements vides de la nouvelle source de nourriture (V_i), à partir de gauche à droite aussi. Il faut s'assurer que chaque job sera inclus exactement *m* fois dans la nouvelle source de nourriture (V_i). Pour cela, si on prend un job de (X_k) pour éventuellement le placer dans (V_i), et on trouve qu'il se répète déjà *m* fois dans ce (V_i) (soit en provenance de (X_i) ou de (X_k)) , ce job sera ignoré et on passera au job suivant dans (X_k).

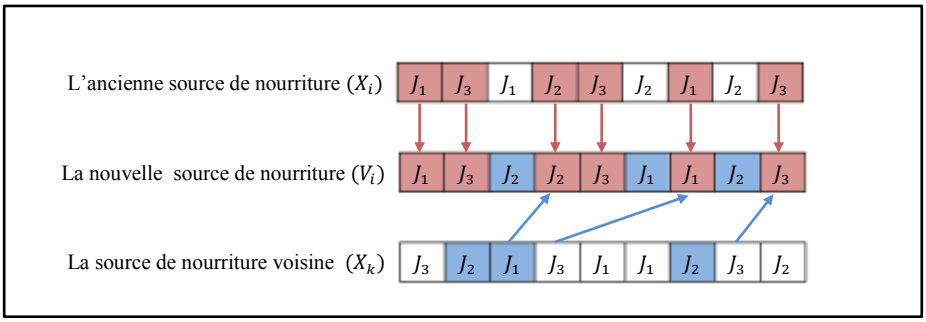

Figure 4-6 : la mise à jour de la solution par le croisement PBX.

Une fois que la nouvelle source de nourriture (V_i) est obtenue après mise à jour de l'ancienne source de nourriture (X_i), une sélection de la meilleure solution est appliquée. L'ancienne source de nourriture (X_i) dans la mémoire des abeilles actives sera remplacée par la nouvelle source de nourriture candidate (V_i) si cette dernière a une meilleure fitness. Dans le cas où la mise à jour aurait amélioré (X_i), son compteur $trial_i$ est remis à '0'. Sinon, si l'abeille active ne change pas de source de nourriture (X_i), $trial_i$ est incrémenté de '1'.

L'abeille active va faire des essais de changement de position pour améliorer sa source de nourriture (X_i). Si à chaque fois elle n'arrive pas à l'améliorer et que le nombre de ces essais dépasse la valeur définie par '$Limit$', $(trial_i \geq limit)$, cette source de nourriture va être abandonnée et retirée de la population de solutions. L'abeille active qui exploitait la source

(X_i) va devenir une abeille scoute qui va faire une recherche aléatoire et va trouver une nouvelle source de nourriture pour redevenir ainsi une abeille active à nouveau.

Algorithme 4.2 : **La procédure de la phase des abeilles actives**

For all X_i ; $i = 1\ to\ SN$ (pour toutes les abeilles actives)
    Choisir aléatoirement dans la colonie, une solution voisine X_k
    Produire une nouvelle solution V_i par mise à jour de X_i avec X_k (PBX crossover)
    If $\boldsymbol{C_{max}}(V_i) < \boldsymbol{C_{max}}(X_i)$ **then**
        $X_i \leftarrow V_i$
        $trial_i = 0$
        $X_{Gbest} = update(X_{Gbest}, V_i)$
    Else
        $trial_i = trial_i + 1$
    End if
End For

4.4.2.3 Phase des abeilles spectatrices (Onlooker bees phase)

Cette phase commence par une évaluation de la qualité de toutes les sources de nourriture exploitées par les abeilles actives. Après cela, SN abeilles spectatrices seront recrutées (SN = nombre d'abeilles actives). En effet, les SN abeilles spectatrices vont être recrutées pour exploiter de nouvelles sources de nourriture en sélectionnant et en mettant à jour les sources de nourriture les plus intéressantes parmi celles des abeilles actives. Le pseudo-code de cette phase est donné par l'Algorithme 4.3 où les détails sont les suivants:

a) La Sélection :

Le principe de sélection que nous utilisons dans le CABC est différent de celui de l'ABC. Dans l'algorithme ABC fondamental, la sélection est donnée en utilisant la méthode de "*roulette wheel selection*" [Goldberg, 1989]. Pour cette méthode, la source de nourriture est sélectionnée en fonction de sa valeur de probabilité p_i, calculée par expression (4.5).

$$p_i = \frac{fit_i}{\sum_{k=1}^{SN} fit_k} \tag{4.5}$$

p_i est comparée à une valeur de $\propto$ générée aléatoirement entre [0,1]. Si $p_i > \propto$, la source de nourriture correspondante (X_i) est sélectionnée. Cependant, pour le JSP, la fitness fit_i, exprimée par l'inverse de $C_{max}(X_i)$, prend des valeurs relativement très petites. De cela, il en résulte une probabilité p_i également très faible. p_i Pourrait être 1000 fois plus petit que $\propto$. Par conséquent, dans le cas du JSP, on constate que la "*roulette wheel selection*" pourrait ne pas être une méthode de sélection appropriée pour les abeilles spectatrices.

Dans le CABC, nous gardons le principe qui stipule que pour avoir une meilleure exploitation, les abeilles spectatrices ne devraient sélectionner que les sources de nourriture les plus intéressantes parmi celles proposées par les abeilles actives. Cependant, la sélection ne sera pas basée sur la probabilité mais plutôt sur la valeur du C_{max}. Puisque notre objectif est de minimiser le C_{max}, on considère que la solution est meilleure qu'une autre, lorsque la valeur correspondante de son C_{max} est plus faible. D'un autre côté, la possibilité de trouver de meilleures solutions est plus élevée au voisinage d'une bonne solution. De ce fait, la plupart des abeilles spectatrices choisiront de sélectionner et de suivre les bonnes solutions. C'est-à-dire les solutions ayant les plus faibles C_{max}.

Notre procédure de sélection par classement (Rank-based selection) peut être décrite comme suit :

- Étape 1: Comme le montre la Figure 4-7, les sources de nourriture des abeilles actives sont classées par ordre croissant des valeurs de C_{max}.
- Étape 2: Seul un quota des premières sources de nourriture bien classées pourrait éventuellement être sélectionné. L'estimation de ce quota est donnée par un pourcentage β. Par exemple, pour β = 25%, le C_{max} le plus élevé parmi les 25% des sources de nourriture bien classées est noté C_{max}_β.
- Étape 3: Les abeilles spectatrices ne peuvent sélectionner que les sources de nourriture (X_i) ayant un $C_{max}(X_i) \leq C_{max}\_\beta$.

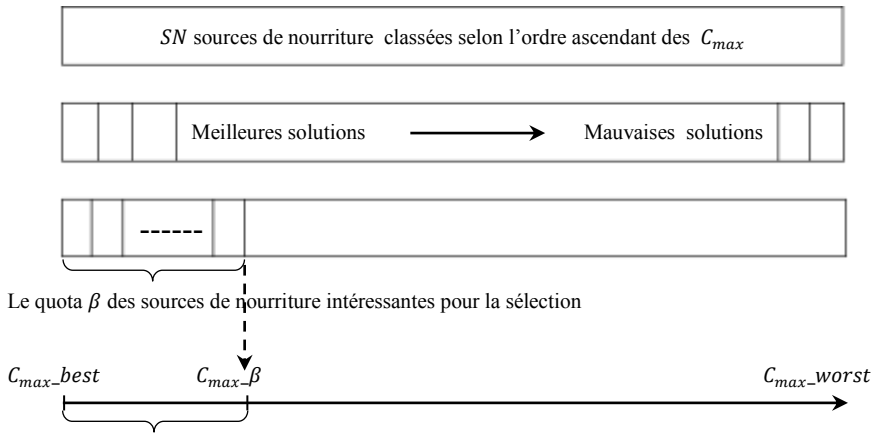

Figure 4-7 : les candidats classés pour la sélection des sources de nourriture.

b) La mise à jour (Updating) :

La mise à jour de la source de nourriture sélectionnée (X_i) se fera de la même manière que celle de la phase des abeilles actives. Une sélection par élitisme est également appliquée entre la source de nourriture sélectionnée (X_i) et la nouvelle source de nourriture générée (V_i) afin de conserver la meilleure solution. Les compteurs $trial_i$ sont également mis à jour.

Dans le CABC, la phase des abeilles spectatrices fournit l'intensification de la recherche locale sur des solutions choisies relativement prometteuses. Cela signifie que seules les meilleures sources de nourriture proposées par les abeilles actives seront candidates à une mise à jour dans la phase des abeilles spectatrices. Il serait très intéressant d'améliorer encore le côté exploitation de cette phase. Cet objectif sera abordé en chapitre 5.

Algorithme 4.3 : **La procédure de la phase des abeilles spectatrices**

For all X_i ; $i = 1\ to\ SN$ (pour toutes les abeilles actives)
    Evaluer $\boldsymbol{C_{max}}(X_i)$
End For
Classer tous les $\boldsymbol{C_{max}}(X_i)$; $= 1\ to\ SN$, dans l'ordre ascendant.
Calculer $\boldsymbol{C_{max}}\_\beta$ correspondant au quota β des bonnes solutions.
$t = 0$, $i = 1$
While $t < SN$
    If $\boldsymbol{C_{max}}(X_i) < \boldsymbol{C_{max}}\_\beta$ **then**
        $t = t + 1$
        Choisir aléatoirement dans la colonie, une solution voisine X_k
        Produire une nouvelle solution V_i par mise à jour de X_i avec X_k (PBX crossover)
        If $\boldsymbol{C_{max}}(V_i) < \boldsymbol{C_{max}}(X_i)$ **then**
            $X_i \leftarrow V_i$
            $trial_i = 0$
            $X_{Gbest} = update(X_{Gbest}, V_i)$
        Else
            $trial_i = trial_i + 1$
        End if
    End if
    $i = i + 1$
    If $i == SN$ **then**
        $i = 1$
    End if
End while

4.4.2.4 Phase des abeilles scoutes (Scout bees phase)

Après avoir effectué les phases d'abeilles actives et spectatrices, certaines solutions qui n'ont pas été améliorées après de nombreux essais sont soit des solutions de très mauvaise qualité ou des solutions qui sont prises au piège des optima locaux. Dans les deux cas il faut les éviter. La phase des abeilles scoutes est faite pour remédier à cette situation. En effet, si une source de nourriture (solution) ne peut pas être améliorée après un certain nombre d'essais, noté $Limit$, cette solution dois être abandonnée. L'abeille active associée à cette source de nourriture devient une abeille scoute. Cette abeille scoute va chercher aléatoirement une nouvelle source de nourriture et redevient encore une abeille active.

Le paramètre $Limit$ joue un rôle très important dans l'algorithme CABC. Il permet d'assurer un équilibre entre l'exploration et l'exploitation. Une petite valeur du paramètre $Limit$ favorise l'exploration par rapport à l'exploitation et l'inverse est vrai pour le cas où il prend une grande valeur. En effet, en remplaçant de temps en temps les mauvaises solutions dans une colonie (population), par de nouvelles, on assure une bonne diversification.

Dans l'ABC fondamental, une seule abeille scoute est utilisée. Dans l'algorithme CABC, le nombre d'abeilles scoutes n'est pas un nombre fixe. À chaque itération, *toutes* les abeilles actives, pour lesquelles la source de nourriture correspondante n'a pas été améliorée après '$Limit$' essais deviendront des abeilles scoutes (éclaireuses) et leurs compteurs d'essais ($trial_i$) seront remis à zéro. Cette étape est détaillée par l'algorithme 4.4.

Algorithme 4.4 : La **procédure de la phase des abeilles Scoutes**

For all X_i ; $i = 1$ to SN (pour toutes les abeilles)
    If $trial_i \geq Limit$ then
        Générer aléatoirement une nouvelle solution X_i
        Evaluer la nouvelle solution X_i
        $trial_i = 0$
    End if
End For

4.5 Résultats expérimentaux

Afin d'examiner l'efficacité de la version Combinatoire de l'algorithme des colonies d'abeilles (CABC), des simulations numériques sont effectuées sur de nombreuses instances de benchmarks de Job shop. Ces benchmarks sont disponibles en ligne sur la bibliothèque de recherche opérationnelle (OR-Library) [Beasley, 1990, 2014]. En chapitre 2, la section 2.8 a

été spécialement réservée à la présentation de ces benchmarks. On tient juste à rappeler que tous ces Benchmarks sont des problèmes NP-difficiles (Le nombre de machines m ≥ 3). En effet, la taille de leurs instances varie de (6 à 50) jobs et de (5 à 20) machines.

Le CABC algorithme est une méthode itérative et stochastique, donc pour pouvoir juger sa qualité et sa robustesse, nous allons l'exécuter 10 fois sur chaque exemple d'instances et nous allons considérer les paramètres et les indicateurs de performances suivants :

- **Instance** : Représente le nom de l'instance testée. Exemple : la première instance de Lawrence c'est la « La01 ».
- **Size** : la taille du problème. Exemple : l'instance de taille $(n \times m)$ correspond à un Job shop de n jobs m machines.
- **BKS** : *Best Known Solution,* la meilleure solution connue pour cette instance, elle peut être une valeur optimale ou une borne inférieure. Les BKS sont disponibles dans la revue de littérature de Jain et Meeran [1999] ou dans les travaux de Van Hoorn [2015,2018] qui contiennent aussi des mises à jour des valeurs des bornes inférieures récemment améliorées.
- $\boldsymbol{Best\_C_{max}}$: La meilleure valeur de C_{max} relevée sur les 10 exécutions de l'algorithme.
- $\boldsymbol{Avrg\_C_{max}}$: La moyenne calculée sur tous les C_{max} obtenus par les 10 exécutions.
- $\boldsymbol{RPE_{Best}}$: *Relative Percent Error*, l'écart relatif entre la meilleure solution trouvée et la meilleure solution connue. C'est un pourcentage d'erreur calculé par l'équation (4.6).
- $\boldsymbol{RPE_{Avrg}}$: *Relative Percent Error* aussi, mais cette déviation représente l'écart relatif entre la moyenne des solutions trouvées et la meilleure solution connue. C'est un autre type de pourcentage d'erreur calculé par l'équation (4.7). c'est un indicateur de robustesse.
- **GAP** : Il est exprimé par le pourcentage de l'écart relatif entre la solution obtenue à chaque itération générée par l'algorithme et la solution optimale connue. L'expression du GAP est donnée par l'équation (4.8).
- $\boldsymbol{NIROS_{min}}$: Minimal Number of Iteration to Reach Optimal Solution, le nombre minimal d'itérations pour atteindre la solution optimale. Ce paramètre nous donnera une idée sur la vitesse de convergence de l'algorithme

$$RPE_{Best} = \frac{(Best\_C_{max} - BKS)}{BKS} \times 100 \quad (4.6)$$

$$RPE_{Avrg} = \frac{(Avrg\_C_{max} - BKS)}{BKS} \times 100 \quad (4.7)$$

$$GAP = \frac{(C_{max_{iter}} - BKS)}{BKS} \times 100 \;\; ; \;\; iter = 1 \;\; to \;\; NI\_Max \tag{4.8}$$

4.5.1 Paramètres de simulation

La métaheuristique des colonies d'abeilles a l'avantage d'avoir très peu de paramètres à régler. Dans l'algorithme ABC, en plus de la taille de la population de solutions, et le nombre maximal d'itérations qu'il faut fixer, nous devons régler seulement le paramètre $Limit$ car il ajuste l'équilibre entre l'exploration et l'exploitation.

Par ailleurs, lors de l'adaptation de cette méthode au cas du JSP, nous avons introduit deux nouveaux paramètres dans l'algorithme CABC, à savoir :

- *N_ch* : car lors de la mise à jour d'une solution, *N_ch* représente le nombre d'éléments à changer lorsqu'on lui fait un PBX crossover avec une solution voisine.
- β : un paramètre de seuil pour contrôler le niveau de qualité des solutions sélectionnées durant la phase des abeilles spectatrices.

Sur la base d'une analyse de sensibilité portant sur plusieurs combinaisons des valeurs de paramètres, les paramètres de l'algorithme sont fixés comme suit:

- Taille de la colonie $CS = 1000$, alors la taille de la population de solutions est donnée par $SN = \frac{CS}{2} = 500$ sources de nourriture.
- $N\_ch = m$, m étant le nombre de machines.
- $\beta = 0.25$ car on suppose que 25% des solutions classées peuvent être sélectionnées pour la mise à jour.
- $Limit = 20$ essais.
- Pour le critère d'arrêt, c'est soit atteindre $NI\_max = 1000$ itérations ou $C_{max} = BKS$.

4.5.2 Résultats de simulation

Bien que nous ayons testé l'algorithme combinatoire de colonies d'abeilles artificielles (CABC) sur plusieurs instances de Job shop, nous allons nous limiter ici à présenter seulement les résultats des instances pour lesquelles nous avons obtenu la meilleure solution (optimale) connue dans la littérature (BKS).

Le Tableau 4-1 présente les résultats relevés sur les simulations faites sur le benchmark de Fisher et Thompson [1963], noté « Ft06 », et sur certains de ceux de Lawrence [1984], notés « La01-La15 ».

Pour chaque instance nous avons lancé 10 fois les simulations et relevé les indicateurs de performances déjà définis dans la section précédente.

Lorsque $RPE_{Best} = 0$ pour une instance, cela signifie que sa solution optimale est obtenue et si de plus, si $RPE_{Avrg} = 0$, cette solution optimale est obtenue avec les 10 répétitions de simulation. Le tiret «-» dans la colonne $NIROS_{min}$ signifie que la solution optimale n'est pas encore atteinte pour cet algorithme.

Instance	**Size**	**BKS**	$Best\_C_{max}$	$Avrg\_C_{max}$	RPE_{Best} (%)	RPE_{Avrg} (%)	$NIROS_{min}$
Ft06	6x6	55	55	55	0	0	1
La01	10x5	666	666	666	0	0	10
La02	10x5	655	655	661,3	0	0.9618	349
La03	10x5	597	604	609,4	1.1725	2.0771	-
La04	10x5	590	590	596,3	0	1.0678	839
La05	10x5	593	593	593	0	0	1
La06	15x5	926	926	926	0	0	3
La07	15x5	890	890	890	0	0	75
La08	15x5	863	863	863	0	0	1
La09	15x5	951	951	951	0	0	2
La10	15x5	958	958	958	0	0	1
La11	20x5	1222	1222	1222	0	0	23
La12	20x5	1039	1039	1039	0	0	8
La13	20x5	1150	1150	1150	0	0	13
La14	20x5	1292	1292	1292	0	0	1
La15	20x5	1207	1207	1212.4	0	0.4474	701

Tableau 4-1 : les résultats de simulation de l'algorithme CABC.

A partir des résultats contenus dans le Tableau 4-1, on peut conclure que l'algorithme CABC s'avère efficace pour la résolution du problème NP-difficile du JSP. En effet, il permet d'avoir la solution connue optimale pour toutes les instances considérées, à l'exception de

l'instance La03. Pour La03, le $Best\_C_{max} = 604$ au lieu de la meilleure valeur de C_{max} donnée par *BKS*=597, donc l'erreur relative $RPE_{Best} = 1.1725$ %. Dans ce cas, on estime que même si on n'atteint pas la solution optimale, la solution obtenue avec un tel pourcentage d'erreur reste assez satisfaisante.

Pour la présentation de nos résultats, nous avons fixé la taille de la population *SN*=500 sources de nourriture et le nombre d'itérations maximal *NI_Max*=1000 itérations. Cependant, dans nos tests de simulation, il est possible d'obtenir les meilleures solutions connues pour certaines instances avec des valeurs de paramètres nettement inférieures à celles fixées ici. Par exemple, pour l'instance La05, l'algorithme CABC nous donne la solution optimale (BKS=593) à la première itération ($NIROS_{min} = 1$), avec seulement $SN = 5$ et une durée de simulation inférieure à 1 seconde.

4.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'adaptation d'une métaheuristique des colonies d'abeilles artificielles pour résoudre le problème d'ordonnancement de Job shop (le JSP).

Avant de présenter les différentes étapes de l'algorithme de cette métaheuristique, nous avons préféré présenter le comportement naturel des abeilles butineuses au sein d'une colonie. Cette partie, bien que non exhaustive, nous a donné un aperçu sur l'extraordinaire organisation des abeilles. Une organisation qui a été une bonne source d'inspiration pour mettre au point des méthodes d'optimisation très efficaces pour résoudre des problèmes très complexes.

Par la suite, après avoir présenté la version originale de l'ABC algorithme conçue pour les problèmes de nature continue, nous avons détaillé les différentes étapes de la version combinatoire (CABC) proposée pour résoudre le problème NP-difficile du JSP. L'algorithme proposé a fait preuve d'une grande efficacité pour résoudre plusieurs instances de Job shop.

Il faut bien noter que pour l'algorithme CABC, dans lequel les abeilles représentent les agents de calcul, l'exploitation est assurée par les abeilles actives et spectatrices alors que l'exploration est faite par les abeilles scoutes. Ce mécanisme lui permet d'avoir la possibilité d'équilibrer entre l'exploitation locale et l'exploration globale. Cependant, cet algorithme demeure encore faible en exploitation. Pour cela, nous avons pensé à remédier à ce problème en hybridant cet algorithme avec une procédure de recherche local. Cela va faire l'objet du prochain chapitre.

Chapitre 5

Algorithme combinatoire et hybride à base de colonies d'abeilles

Tout seul on va plus vite…

… ensemble on va plus loin !

Proverbe africain

5.1 Introduction

L'efficacité d'un processus de recherche dans tous les algorithmes à population de solutions, inspirés d'un phénomène naturel, dépend de deux composantes : l'exploration et l'exploitation [Črepinšek et al., 2013]. Pour l'algorithme CABC, l'exploration est bien assurée par la phase des abeilles scoutes. Cependant, l'exploitation telle qu'elle est faite aux niveaux des phases des abeilles actives et des abeilles spectatrices, semble encore faible.

En fait, à chaque itération, chaque solution est mise à jour la première fois lors de la phase des abeilles actives et si cette solution est suffisamment bonne, elle peut être sélectionnée pour une autre mise à jour lors de la phase des abeilles spectatrices. Néanmoins, cela ne suffit pas pour une bonne exploitation.

Afin d'améliorer l'exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale est effectuée dans le présent chapitre. Nous avons appelé la procédure proposée: "Simple Iterated Local Search (SILS)". On notera « CABC_SILS », la nouvelle version combinatoire et hybride des colonies d'abeilles artificielles. La méthode va être testée sur plusieurs Benchmark de Job shop et les résultats comparés à ceux obtenus par la méthode CABC sans hybridation.

5.2 Les méthodes hybrides

Le développement et l'application des métaheuristiques hybrides suscitent de plus en plus l'intérêt académique. Les méthodes hybrides combinent différents concepts et composants de diverses métaheuristiques [Talbi, 2009]. À cette fin, elles tentent de fusionner les points forts et éliminent les points faibles des différents concepts de métaheuristiques. Par conséquent, l'efficacité de la recherche dans l'espace de solutions peut être encore améliorée et de nouvelles opportunités pourront émerger, ce qui peut conduire à des méthodes de recherche encore plus puissantes et plus flexibles.

Les techniques d'hybridation, telles qu'elles sont représentées sur la Figure 5-1, sont classées selon leur architecture, en trois catégories, [Duvivier, 2000] :

- Hybridation séquentielle,
- Hybridation parallèle synchrone,
- Hybridation parallèle asynchrone.

- **Hybridation séquentielle**

Cette technique d'hybridation est la plus simple et la plus populaire. Elle consiste à exécuter séquentiellement différentes méthodes de recherche de telle manière que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante. Cette hybridation ne nécessite pas de modification des méthodes de recherche utilisées : il suffit de pouvoir initialiser chaque méthode de recherche à partir de solutions pré-calculées.

- **Hybridation parallèle synchrone**

Cette hybridation est réalisée par incorporation d'une méthode de recherche particulière dans un opérateur. Elle est plus complexe à mettre en œuvre que la précédente. L'objectif est de combiner une recherche locale avec une recherche globale dans le but d'améliorer la convergence.

- **Hybridation parallèle asynchrone (coopérative)**

Elle consiste à faire évoluer en parallèle différentes méthodes de recherche. Cette coévolution permet une bonne coopération des méthodes de recherche au travers d'un coordinateur : chargé d'assurer les échanges d'informations entre les méthodes de recherche constituant l'hybride, son rôle est essentiel pour une bonne coopération de l'ensemble. Cette technique d'hybridation nécessite des modifications mineurs des méthodes de recherche

utilisées afin d'assurer les communications avec le coordinateur. Elle offre l'avantage de faire fonctionner « de pair » les différentes méthodes de recherche et de laisser ainsi à tout moment la possibilité à chaque méthode d'exploiter les résultats des autres méthodes en vue d'une utilisation optimale de ses de ses potentialités d'exploration/exploitation. En contrepartie, elle nécessite la réalisation d'un coordinateur.

Il est possible aussi de combiner plusieurs techniques d'hybridation au sein d'une méthode de recherche hybride. Pour plus de détails, on peut se référer au livre de Talbi [2009] dans lequel il propose une bonne taxonomie des métaheuristiques hybrides et présente aussi une grammaire qui permet de décrire précisément les différents schémas d'hybridation.

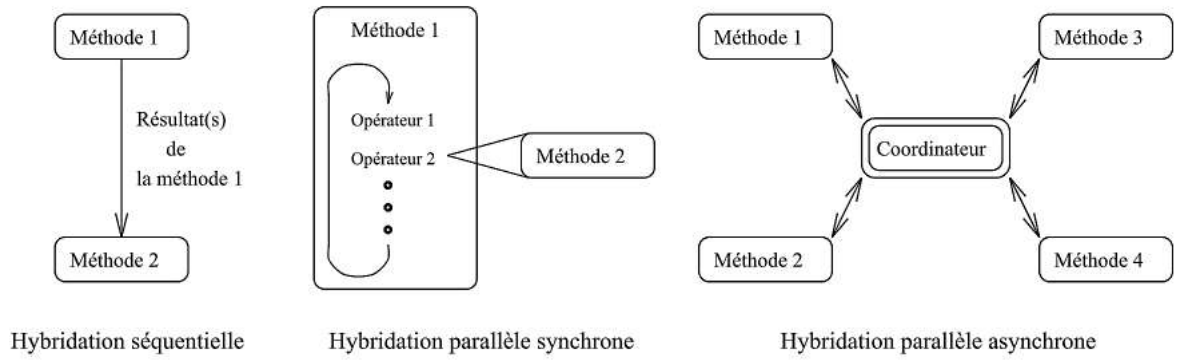

Figure 5-1 : Techniques d'hybridation [Duvivier, 2000].

Dans notre étude, nous avons conçu notre méthode hybride en adoptant la technique de l'hybridation séquentielle.

5.3 La nouvelle version combinatoire et hybride des colonies d'abeilles artificielles

Dans la recherche d'une solution à un problème donné, la métaheuristique des colonies d'abeilles, avec ses différentes étapes, a bien l'avantage d'assurer aussi bien l'exploitation que l'exploration. Pour l'algorithme CABC, l'exploration est très bien assurée par la phase des abeilles scoutes. Cependant, on a constaté que l'exploitation telle qu'elle est faite aux niveaux des phases des abeilles actives et des abeilles spectatrices, demeure encore faible.

Dans le but d'améliorer l'exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale est proposée. Cette procédure de recherche local que nous avons appelé "Simple Iterated Local Search (SILS)" est une version simplifiée de la métaheuristique de la recherche locale itérée (ILS : Iterated Local Search).

L'optimisation combinatoire par l'approche des colonies d'abeilles est une méthode itérative et à population de solutions. Donc, à chaque itération 'it' ($it = 1{:}\, NI\_max$), l'algorithme CABC va traiter toute la population des solutions et va fournir la meilleure solution que l'on va noter $X(it)_{best}$. Après les deux phases des abeilles actives et des abeilles spectatrices, la procédure SILS intervient pour intensifier la recherche au voisinage de la solution trouvée : $X(it)_{best}$. Cependant, nous avons décidé que cette intervention ne soit pas systématique pour toutes les itérations. En effet, si à l'itération 'it', on trouve une solution $X(it)_{best}$, qui a déjà été trouvée puis exploitée par SILS dans l'une des itérations précédentes ($< it$), la procédure SILS ne sera pas appliquée dans le cas de la présente itération. Pour mieux organiser l'intervention de la procédure SILS, on établit une liste tabou (LX_{best}) qui contiendra toutes les solutions $X(it)_{best}$ exploitées au préalable. Ainsi si à une itération 'it' on trouve que la solution $X(it)_{best}$ appartient à la liste tabou, la procédure SILS sera ignorée à ce niveau. Cela évitera de faire des recherches locales redondantes.

Les différentes étapes du nouvel algorithme CABC_SILS sont illustrées par L'organigramme représenté sur la Figure 5-2. Plus de détails concernant la procédure SILS seront donnés dans la prochaine section.

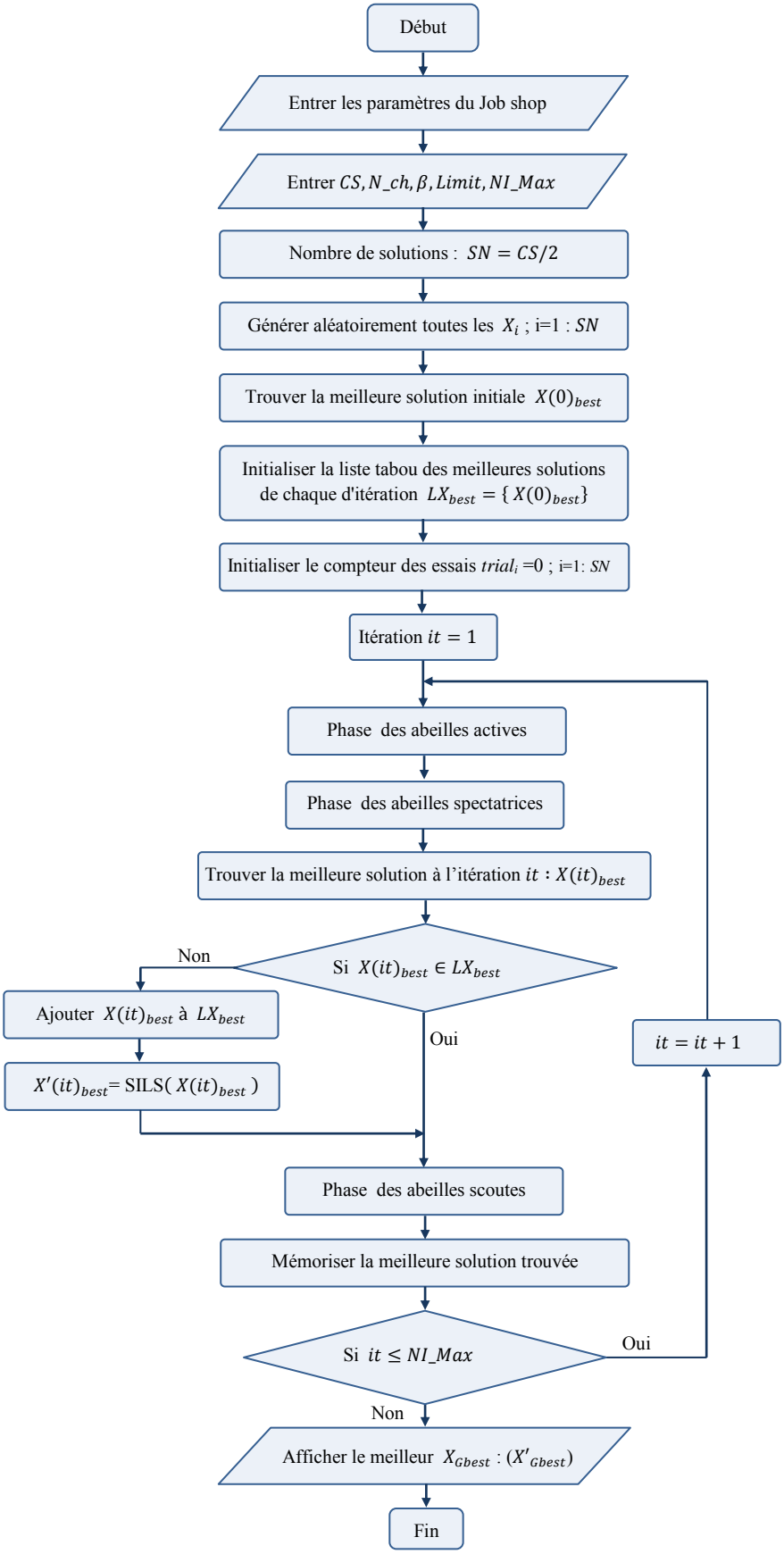

Figure 5-2 : Organigramme de l'algorithme hybride CABC_SILS pour le JSP.

5.3.1 Recherche locale itérée et simple

La recherche locale itérée et simple (SILS) est une méthode qui consiste à appliquer itérativement une recherche locale sur la meilleure solution actuelle $X(it)_{best}$ pour en arriver à l'optimum local $X'(it)_{best}$. La procédure SILS est donnée par l'algorithme 5.1 et fonctionne comme suit :

- Considérons que $X(it)_{best}$ qui est la meilleure solution trouvée à l'itération 'it', va être la solution en entrée de la procédure SILS. Après quelques améliorations itérées, données par certains cycles de recherche locale, la solution de sortie sera $X'(it)_{best}$. Donc on a, $X'(it)_{best}$= $SILS\_procedure(X(it)_{best})$.
- Chaque cycle de recherche locale commence par la solution d'entrée, notée s_{in} (c'est la graine de la recherche locale).
- s^* représente la solution en sortie, $s^* = local\_search(s_{in})$. C'est la meilleure solution trouvée par une recherche locale autour de s_{in}(dans le voisinage de s_{in}).
- Le premier cycle commence par $s_{in} = X(it)_{best}$. La première solution trouvée s^*, sera la solution d'entrée pour le deuxième cycle de recherche locale.
- De même pour les prochains cycles. On prend la solution s^* d'un cycle et on la réinjecte comme solution d'entrée s_{in} dans le prochain cycle de recherche locale.
- Ces cycles sont arrêtés lorsque la recherche locale ne peut plus améliorer la solution s_{in}. En effet, quand on trouve pour le même cycle $s^* = s_{in}$, on considère que le minimum local est atteint.
- Par conséquent, le s^* du dernier cycle, sera la solution finale de la procédure SILS, $X'(it)_{best}$=s^*.

SILS semble similaire à la métaheuristique de Recherche Locale Itérée (ILS : Iterated Local Search), mais au fait, elle diffère de cette dernier dans certains détails. Effectivement, pour l'ILS, quatre composantes doivent être spécifiées [Lourenço et al., 2003] :

- La génération de *la solution initiale*,
- La *modification* à faire,
- La *recherche locale*,
- Et le *critère d'acceptation*.

Mais en ce qui concerne la procédure SILS, que nous avons pris comme variante de la ILS, seulement la partie *recherche locale* doit être spécifiée. En effet, nous avons établi la procédure SILS, en tenant compte de ces trois points :

a) la *solution initiale* est déjà fournie par la solution de l'algorithme CABC, ($s_0 = X(it)_{best}$).

b) Afin de préserver la qualité de la solution proposée par CABC, SILS vise à exploiter la meilleure solution actuelle ($X(it)_{best}$) uniquement dans son bassin d'attraction. Dans ce cas, nous allons réinjecter la solution résultante (s^*) dans la recherche locale suivante sans qu'on lui fasse subir des *modifications* préalables $(s_{in} = {s^*}_{précédante})$.

c) De plus, comme il n'y a pas de modifications à faire sur la solution à exploiter $(s_{in} = {s^*}_{précédante})$, le *critère d'acceptation* n'est pas nécessaire ; parce que la solution s^* obtenue par la *recherche locale* sera sûrement meilleure que la solution s_{in} , ou au pire la même ($s^* = s_{in}$). Dans le cas où l'on trouve que ($s^* = s_{in}$) cela signifie qu'on est arrivé au minimum local et donc la procédure SILS sera stoppée.

La SILS diffère également de l'ILS par le nombre de cycles à exécuter. ILS fonctionne pour un nombre fixe de cycles, mais SILS continue de générer une suite de solutions candidates pour d'éventuelles nouvelles recherches locales, jusqu'à atteindre le minimum local, ($s^* = s_{in}$).

L'algorithme SILS, tel qu'il est présenté par l'Algorithme 5.1, semble très simpliste. Cependant, dans le cas de notre étude, nous avons constaté qu'il est très efficace comme approche de recherche locale, et même plus efficace que l'approche ILS.

Algorithme 5.1 : **La procédure SILS**

1: Input : $s_0 = X(it)_{best}$
2: $s^* = s_0$
3: **Repeat**
4: $s_{in} = s^*$
5: $s^* = Local\_Search(s_{in})$
6: **Until** $s^* = s_{in}$
7: Output : $X'(it)_{best} = s^*$

5.3.2 La procédure de recherche locale

On définit dans cette section la procédure de *recherche locale* qui est considérée comme composante principale dans l'algorithme SILS. Les étapes de cette procédure sont résumées dans l'Algorithme 5.2. C'est une recherche locale qui s'appuie sur la notion de voisinage généré par un opérateur d'insertion.

Le processus d'insertion est illustré par la Figure 5-3. Il est représenté dans l'Algorithme 5.2 par la procédure $Insert\_process(p_2, p_1, s)$. Cette procédure d'insertion consiste à extraire l'opération du job qui se trouve à la position p_2 de la solution (s) et à l'insérer dans la position p_1 de la même solution. Toutes les opérations qui vont de la position p_1 à $p_2 - 1$, seront décalées vers le devant pour combler le vide qui a été créé en p_2 (forward shift). Tous les voisins seront pris en compte. Ainsi, chaque opération retirée de sa position initiale p_2 est insérée dans toutes les positions p_1 possibles, $p_1 = 1{:}D - 1$. Où D représente la dimension de la solution, et donc la dimension du problème à traiter. Elle est exprimée par $D = n \times m$; le nombre de toutes les opérations à exécuter dans un Job shop $(n \times m)$.

Pour choisir le meilleur voisin, on considère la stratégie du best-improvement. Par conséquent, tous les voisins possibles (s') seront explorés de manière exhaustive, mais seule la meilleure solution (s^*) ayant le plus petit C_{max} est retournée comme résultat de la composante $Local\_Search$ dans la procédure SILS.

Algorithme 5.2 : **La procédure *Local_Search***

Input: $s = s_{in}$
$s^* = s$
For $p_1 = 1$ **to** $D - 1$
    For $p_2 = p_1 + 1$ **to** D
        $s' = Insert\_process(p_2, p_1, s)$
        If $C_{max}(s') < C_{max}(s^*)$**then**
            $s^* = s'$
        End if
    End For
End For
Return s^*

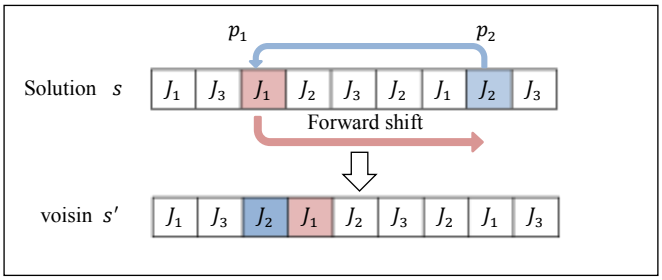

Figure 5-3 : Processus d'insertion pour la procédure $Local\_Search$ de l'algorithme SILS.

5.4 Résultats de simulation

Le Tableau 5-1 présente les résultats expérimentaux de l'algorithme combinatoire de colonies d'abeilles artificielles (CABC) et de sa version hybride CABC_SILS. L'algorithme est exécuté 10 fois sur chaque instance de Job shop et avec le même jeu de paramètres utilisé dans le chapitre 4.

D'après les résultats du Tableau 5-1, on constate que l'algorithme CABC_SILS conduit à la meilleure solution connue pour toutes les instances de La01 à La15. De plus, on voit bien que la solution optimale est obtenue sur les 10 exécutions pour les instances Ft06 et La01 ainsi que pour La05-La15.

De plus, les tests de simulation ont montré que les instances pouvaient être classées en problèmes difficiles et encore plus difficiles. En effet, en comparant les tailles de l'instance La03 (10x5) et l'instance La14 (20x5), La14 est considérée comme une instance de taille plus grande que celle de La03. Cependant, la simulation montre qu'il est relativement plus facile de résoudre La14 que La03. La résolution de La14 prend moins de temps et peut être réalisée qu'avec une petite taille de population (SN). La même remarque peut être faite pour certains cas ayant la même taille. Exemple : la comparaison de La05 avec La03, ou La14 avec La15.

Dans le Tableau 5-1, le paramètre $NIROS_{min}$ met en évidence la possibilité de classer les instances de Benchmark en problèmes très très difficiles et relativement moins difficiles. Avec l'algorithme CABC_SILS : pour La03, $NIROS_{min} = 780$; pour La04, $NIROS_{min} = 94$ et pour La05, $NIROS_{min} = 1$. Cela signifie que la meilleure solution connue pourrait être obtenue pour La03 après 780 itérations, et après 94 itérations pour La04. Cependant, pour La05 la meilleure solution est obtenue à la première itération. Par conséquent, il est relativement plus facile de résoudre La05 que La03. Par conséquent, La05 peut être classé comme un difficile Benchmark de JSP et La03 comme étant encore plus difficile.

Afin d'avoir une idée sur les caractéristiques de convergence du CABC_SILS proposé, nous avons relevé le GAP sur les 5 instances La01-La05. Le GAP calculé selon l'équation (4.8) est illustré par la Figure 5-4, confirme qu'il existe des cas d'instances difficiles et plus difficiles à résoudre. En effet, les GAP de La01 et La05 sont donnés par un seul point car il arrive que l'algorithme converge très rapidement vers la solution optimale avec une seule itération (et même avec une taille de population SN = 5).

En comparant les deux algorithmes CABC et CABC_SILS, il est clair que la version hybride apporte des améliorations intéressantes en termes de qualité de la solution et de

vitesse de convergence. En effet, on peut noter une réduction significative du paramètre $NIROS_{min}$du CABC_SILS par rapport au CABC.

Concernant les résultats de simulations faites sur les autres benchmarks de Job shop qui figurent dans la (OR-Library) [Beasley, 1990, 2014] et qui ne sont pas présentés dans le Tableau 5-1, on a trouvé une valeur de RPE_{Best} qui varie dans l'intervalle [0-5%]. Ainsi, on conclut que les algorithmes proposés sont efficaces même pour les grandes instances.

Instance	Size	BKS	$Best\_C_{max}$		$Avrg\_C_{max}$		RPE_{Best} (%)		RPE_{Avrg} (%)		$NIROS_{min}$	
			CABC	CABC SILS	CABC	CABC SILS	CABC	CABC SILS	CABC	CABC SILS	CABC	CABC SILS
Ft06	6x6	55	55	55	55	55	0	0	0	0	1	1
La01	10x5	666	666	666	666	666	0	0	0	0	10	1
La02	10x5	655	655	655	661,3	657,3	0	0	0.9618	0.3511	349	154
La03	10x5	597	604	597	609,4	606,3	1.1725	0	2.0771	1.5578	-	780
La04	10x5	590	590	590	596,3	591,3	0	0	1.0678	0.2203	839	94
La05	10x5	593	593	593	593	593	0	0	0	0	1	1
La06	15x5	926	926	926	926	926	0	0	0	0	3	1
La07	15x5	890	890	890	890	890	0	0	0	0	75	4
La08	15x5	863	863	863	863	863	0	0	0	0	1	1
La09	15x5	951	951	951	951	951	0	0	0	0	2	1
La10	15x5	958	958	958	958	958	0	0	0	0	1	1
La11	20x5	1222	1222	1222	1222	1222	0	0	0	0	23	1
La12	20x5	1039	1039	1039	1039	1039	0	0	0	0	8	1
La13	20x5	1150	1150	1150	1150	1150	0	0	0	0	13	1
La14	20x5	1292	1292	1292	1292	1292	0	0	0	0	1	1
La15	20x5	1207	1207	1207	1212.4	1207	0	0	0.4474	0	701	4

Tableau 5-1 : Comparaison entre les résultats de CABC et de CABC_SILS.

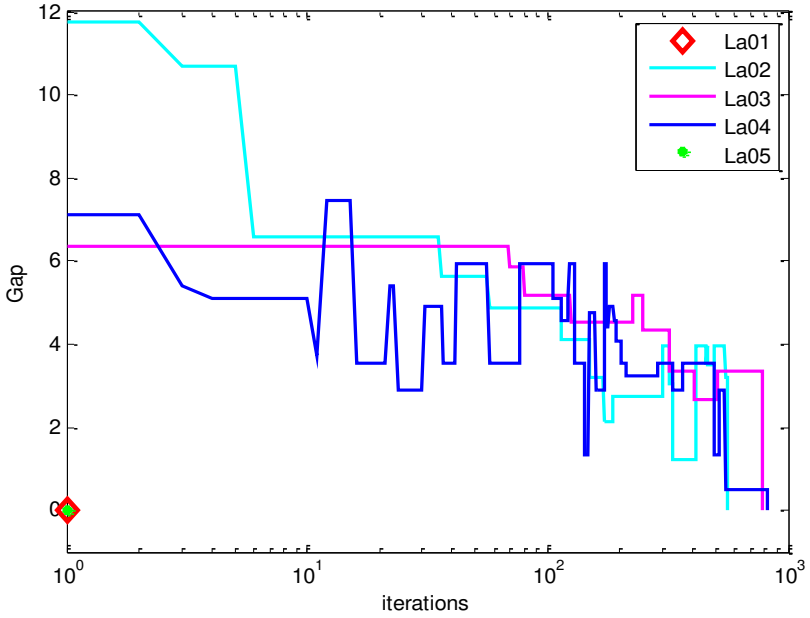

Figure 5-4 : Caractéristiques de convergence pour les instances: La01-La05.

5.5 Conclusion

Afin d'améliorer l'exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale a été effectuée.

Malgré que la nouvelle procédure de recherche locale proposée (SILS) est une version simplifiée de la recherche locale itérée (ILS), on a constaté qu'elle arrive à réaliser de résultats meilleurs que ceux donnés par la ILS.

Les résultats expérimentaux réalisés sur les benchmarks de Job shop, ont démontré que la version hybride CABC_SILS améliore efficacement l'exploitation de l'algorithme combinatoire proposé. En effet, elle offre de meilleurs résultats en termes de qualité de la solution évaluée par le Makespan et en termes de vitesse de convergence.

Conclusion générale et perspectives

Les travaux effectués dans cette thèse contribuent aux études menées sur les problèmes d'ordonnancement des ateliers de type Job shop. Nous avons abordé le problème avec comme critère, la minimisation de Makespan et ceci afin d'améliorer la productivité des ateliers Job shop.

L'importance pratique et théorique du Job shop lui confère un perpétuel intérêt chez les spécialistes de l'ordonnancement. En effet, en raison de sa grande applicabilité et de son inhérente difficulté, il est considéré comme l'un des problèmes d'ordonnancent les plus courants mais aussi les plus complexes.

Afin de résoudre le JSP qui est considéré comme étant un problème d'optimisation combinatoire, de nombreuses méthodes ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Lorsque le JSP est de petite taille, on peut le résoudre en un temps raisonnable et d'une manière optimale par des méthodes exactes. Par contre, ces dernières s'avèrent impossibles à appliquer sur des problèmes Job shop ayant une taille industrielle assez grande. Effectivement, il a été démontré qu'à partir de trois machines le JSP devient un problème NP-difficile au sens fort. Dans ce cas, pour obtenir malgré tout des solutions, des méthodes approchées, en l'occurrence les métaheuristiques ont été développées.

De notre côté nous nous sommes intéressés à la métaheuristique des colonies d'abeilles et plus particulièrement à l'algorithme des colonies d'abeilles artificielles (ABC). Cette méthode a été introduite en 2005, par Karaboga pour résoudre des problèmes d'optimisation de nature continue [karabora 2007]. De plus, des versions étendues de cette méthode ont été utilisées avec succès, comme outil pour résoudre des problèmes d'optimisation très complexes et dans différents domaines tels que : les télécommunications, l'informatique, la robotique, etc.

Notre première contribution dans cette thèse fut d'adapter la version continue de l'algorithme ABC au problème combinatoire du Job shop. La métaheuristique proposée, formulée par l'algorithme combinatoire des colonies d'abeilles artificielles (CABC), est basée sur le comportement intelligent des abeilles butineuses dans la recherche d'une source d'alimentation de bonne qualité. Elle s'articule sur trois phases: la phase des abeilles actives et la phase des abeilles spectatrices qui assurent l'exploitation et la phase des abeilles scoutes pour assurer une bonne exploration et pour permettre d'échapper aux optima locaux.

À chaque itération de l'algorithme CABC, chaque solution est d'abord mise à jour à la phase des abeilles actives. De plus, elle pourrait être sélectionnée pour une deuxième mise à jour lors de la phase des abeilles spectatrices si cette solution est assez intéressante du point de vu fitness. La « mise à jour » et la « sélection » sont deux points clés de cette méthode, alors nous avons pris soin de les choisir en adéquation avec le problème en question. Pour la mise à jour d'une solution, nous avons opté de lui faire un croisement PBX (Position based Crossover) avec une solution voisine. Et concernant le deuxième point, nous avons procédé par la sélection après classement des solutions selon la valeur de leur fonction objectif (ranked based selection).

Pour tester la validité et les performances du CABC, nous l'avons appliqué sur plusieurs benchmarks de Job shop tirés de la littérature. Les paramètres de l'algorithme ont été fixés sur la base d'une analyse de sensibilité portant sur plusieurs combinaisons de valeurs de paramètres. La métaheuristique des colonies d'abeilles a l'avantage d'avoir très peu de paramètres à régler. (à l'origine un seul paramètre « Limit » et nous en avons ajouté deux autres). L'algorithme CABC a permis d'obtenir la meilleure solution connue de plusieurs instances de benchmarks, sinon la solution est obtenue avec une erreur de 0 à 5 % par rapport à la solution optimale. Avec de tel résultats expérimentaux, on estime que l'algorithme proposé est capable de bien résoudre le problème d'ordonnancent de Job shop. Par ailleurs, nous avons jugé qu'il est encore possible d'apporter des améliorations à cette première version combinatoire.

En effet, comme nous l'avons mentionné ci-haut, le concept du CABC qui est proposé avec seulement deux processus de mise à jour semble encore faible en exploitation et qu'il faudrait l'améliorer encore. Notre deuxième contribution fut alors dans ce sens.

Ainsi, pour améliorer le côté exploitation de l'algorithme CABC, une hybridation séquentielle avec une nouvelle procédure de recherche locale a été introduite. La procédure proposée pour être hybridée avec le CABC est appelée «Recherche locale itérative simple (SILS)». C'est une métaheuristique simple qui applique la recherche locale de manière itérative pour affiner la meilleure solution de l'itération actuelle de l'algorithme CABC, mais uniquement dans le cas où cette solution n'appartient pas à la liste tabou des solutions déjà soumises à la procédure SILS dans les précédentes itérations. La procédure de recherche locale s'appuie sur la notion de voisinage généré par un opérateur d'insertion.

Au terme de cette hybridation, nous avons effectué des simulations numériques qui ont confirmé l'efficacité de la version hybride CABC_SILS. En effet, les résultats expérimentaux réalisés sur les benchmarks de Job shop, ont démontré que la version hybride CABC_SILS améliore efficacement l'exploitation de l'algorithme combinatoire proposé. Elle offre de meilleurs résultats en termes de qualité de la solution évaluée par le Makespan et en termes de vitesse de convergence.

Durant ce travail de thèse, entre autres, nous avons eu l'occasion de faire une étude approfondie de la métaheuristique des colonies d'abeilles. Cette étude nous a permis de prendre connaissance de ses points les plus pertinents pour pouvoir l'améliorer et encore mieux l'exploiter.

Effectivement, l'efficacité et la subtilité de la métaheuristique des colonies d'abeilles, nous donne encore plus de goût à l'exploiter dans nos futures travaux. Par conséquent, notre travail peut être enrichi en continuant sur plusieurs axes de recherche:

1. Une amélioration peut être apportée aux étapes de mise à jour et de sélection des algorithmes proposés afin de résoudre plus rapidement le JSP pour d'autres instances plus complexes.
2. Il serait utile d'étendre les algorithmes proposés aux problèmes d'ordonnancement multi-objectifs.
3. Il est également intéressant d'appliquer les algorithmes proposés à d'autres types de problèmes d'optimisation combinatoire.

Les travaux présentés dans cette thèse ont donné lieu à une publication dans une revue internationale [Ouis Khedim et al. 2020], et à des communications dans des conférences internationales [Khedim et al., 2014, 2016].

Bibliographie

Aarts E. H. L., Van Laarhoven P. J. M., Lenstra J. K., and **Ulder N. L. J.** (1994). "A computational study of local search algorithms for job-shop scheduling", *ORSA Journal on Computing* 6/2, Spring, 118-125.

Abdelmaguid R. (2010). Representations in genetic algorithm for the job shop scheduling problem: A computational study. J Software Engineering & Applications, 3, 1155 - 1162.

Adams J., Balas E. and **Zawack D.** (1988). The shifting bottleneck procedure for job-shop scheduling. Management Science; 34 (3): 391-401.

Agarwal S.K., Yadav S. (2019). A Comprehensive Survey on Artificial Bee Colony Algorithm as a Frontier in Swarm Intelligence. In: Hu YC., Tiwari S., Mishra K., Trivedi M. (eds) Ambient Communications and Computer Systems. Advances in Intelligent Systems and Computing, vol 904. Springer, Singapore.

Akers S. B. Jr, (1956). "A graphical approach to production scheduling problems", *Operations Research* 4, 244-245.

Akers S. B. Jr., and **Friedman J.** (1955). "A Non Numerical Approach to Scheduling Problems", *Operations Research* 3, 429-442.

Anthony R.N. (1965). Planning and control systems: a framework for analysis. Studies in management control. Division of Research, Graduate School of Business Administration, Harvard university.

Applegate D., and **Cook W.** (1991). A computational study of the job-shop scheduling problem. ORSA Journal on Computing; 3 (2): 149-156.

Artigues C. (2004). Optimisation et robustesse en ordonnancement sous contraintes de ressources. URL http://www.laas.fr/~artigues/hdrChristianArtigues.pdf . Habilitation à diriger des recherches, Université d'Avignon.

Artigues C. and **Feillet D.** (2008). "A branch and bound method for the job-shop problem with sequence-dependent setup times," Annals of Operations Research, Vol. 159, pp. 135-159.

Avarguès-Weber A. (2013). L'intelligence des abeilles. Pour La Science N° 429.

Aydin M. E. and **Fogarty T. C.** (2004). A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application. Journal of intelligent manufacturing, Vol. 15, No. 6, pp.805-814.

Bäck T. and **Schwefel H-P.** (1993). An overview of evolutionary algorithms for parameter optimization. Evolutionay Computation 1(1) : 1-24.

Balas E. (1967). "Discrete programming by the filter method," Operations Research, Vol. 15, pp. 915-957.

Balas E. (1969). "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," Operation research, Vol. 17, pp.941-957.

Balas E. (1970). "Machine sequencing: disjunctive graphs and degree-constrained sub graphs," Naval Research Logistics Quarterly, Vol. 17, pp. 941-957.

Baum E.B. (1986). Iterated descent : A better algorithm for local search in combinatorial optimization problems. Technical report, Caltech, Pasadena, CA.

Baxter J. (1981). Local optima avoidance in depot location. Journal of the Operational Research Society, 32 :815–819.

Beasley J. E. (1990). OR-Library: distributing test problems by electronic mail. Journal of the operational research society, Vol. 41, No.11, pp.1069-1072. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

Beasley, J. E. (2014). OR-library. http://people.brunel.ac.uk /~mastjjb/jeb /orlib/files /jobshop1.txt

Beighter C. S., Philip D. T. and **wilde D. J.** (1979). Foundation of optimization, 2nd Ed. Prentice Hall, Englewood.

Bellman R.E. (1986). « The Bellman continuum ». Editions Robert S. Roth.

Blazewicz , J., K.H. , E., E. , P., G. , S. et J. , W. (2007). Scheduling computer and manufacturing processes. Springer, 2rd édition.

Blazewicz J., Ecker K. H., Pesch E., Schmidt G. and **Weglarz J.** (2007). "Handbook on scheduling from theory to application," a book chapter :10 Scheduling in Job Shops, International Handbooks Information System, pp. 345-396.

Blazewicz J., Ecker K.H., Schmidt G. and **Weglarz J.** (1996). Scheduling in computer and manufacturing systems. Springer-Verlag.

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

Blazewicz J., Sterna M. et **Persch E.** (1998). A Branch and Bound algorithm for the Job Shop scheduling problem. Springer Verlag; :219-254.

Bonabeau E., Dorigo M. and **Theraulaz G.** (1999). Swarm intelligence: from natural to artificial systems. Oxford University Press, Inc, New York.

Bowman E. H., (1959). "The schedule-sequencing problem," Operation research, Vol. 7, n°.5, pp. 621-624, September-October.

Brucker P. and **Sigrid Knust** (2009). Complexity results for scheduling problems.

Brucker P., Jurisch B. and **Sievers B.** (1994). A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics, 49 :107 127.

Brusco, M.J. et **Stahl, S.**(2005). Branch-and-Bound Applications in Combinatorial Data Analysis, volume 221 de Statistics and Computing. Springer New York, xii édition.

Çaliş B. and **Bulkan S**. (2015). A research survey: review of AI solution strategies of job shop scheduling problem". Journal of Intelligent Manufacturing, Vol. 26, No. 5, pp.961-973.

Carlier J., (1982). "The one-machine sequencing problem". European Journal of Operational Research.11. pp 42-41.

Carlier J., Pinson E. (february 1989). "A Branch and Bound Method for Solving the Job Shop Problem", Management Science, Vol. 35, No. 2, pp. 164-176.

Caseau, Y. and **Laburthe F.** (1996). "Improving branch and bound for Job shop scheduling with constraint propagation," Combinatorics and Computer Science, Lecture Notes in Computer Science, Vol. 1120, pp. 129-149,.

Cerf R. (1994). Une théorie asymptotique des algorithmes génétiques. PhD thesis, Université de Montpellier II, France.

Chang P.T., Lin K.-P., Pai P.-F., and **C.-Z. Zhong** (2008). Ant colony optimization system for a multi-quantitative and qualitative objective job-shop parallel-machine-scheduling problem. International Journal of Production Research, 46 : 5719-5759.

Chen, C. L., Chen, C. L. (2009). Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. *Computers and Industrial Engineering, 56*(4), 1393–1401.

Chen, L., Bostel, N., Dejax, P., Cai, J., & xi, L. (2007). A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research, 181*(1), 40–58.

Cheng R., Gen M., and Tsujimura Y. (1996). "A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation," International Journal of Computers and Industrial Engineering, Vol.30, n° 4, pp. 983-997.

Cheng R. , Gen M., and **Tsujimura Y.** (1999). "A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation," Computers and Industrial Engineering, Vol. 36, pp. 343-364.

Chiang, T. C., Fu, L. C. (2007). Using dispatching rules for job shop scheduling with due-date based objectives. *International Journal of Production Research, 45*(14), 3245–3262.

Clausen J. (March 12, 1999). "Branch and Bound Algorithms - Principles and Examples," department of compsc., university of Copenhagen, Universitetsparken 1, DK-2100 Corpenhagen, Denmark, http://www.imm.dtu.dk/_jha/

Clerc M. (2005). L'optimisation par essaims particulaires, Lavoisier ISBN 2-7462-0991-8, France.

Collette Y. et **Siarry P.** (2002) « Optimisation Multiobjectif ». Editions Eyrolles, Paris.

Collins N.E., Eglese R.W. et **Golden B.L.** (1988). Simulated annealing : An annotated bibliography. American Journal of Mathematical and Management Sciences, 8 :209–307.

Colorni A., Dorigo M., Maniezzo V. and **Trubian M**. (1994). "Ant-system for job-shop scheduling", *Belgian Journal of Operations Research, Statistics and Computer Science* 34/1, 39-54.

Conway, RW., Maxwell W.L. and **Miller L.W.** (1967). Theory of Scheduling, Addison-Wesley, Reading, MA.

Cook S. A. (1971). "The complexity of theorem proving procedures", *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, Association of Computing Machinery, New York, 151-158.

Cooren Y. (2008). Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire. Applications en génie médical et en électronique. Thèse de Doctorat, Université de Paris 12 Val de Marne, France.

Črepinšek M., Liu S. H. and **Mernik M**. (2013) 'Exploration and exploitation in evolutionary algorithms: A survey'. *ACM Computing Surveys (CSUR)*, Vol. 45, No. 3, pp.35.

De Jong, K.A. and **Spears W.** (1993). On the state of evolutionary computation. Proc. of International Conference on Genetic Algorithms (ICGA'93), p. 618-623.

De Werra D. et **Hertz A.** (1989). Tabu search techniques : A tutorial and an application to neural networks. OR Spektrum, 11 :131–141,.

Dell'Amico M. et **Trubian M.** (1993). Applying tabu searchto the job-shop scheduling problem. *Annals of Operations Research, 41*, 231–252.

Demirkol E., Mehta S. & Uzsoy R. (1997). A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics, 3*(2), 111–137.

Dorigo M. (1992). Optimization, Learning and Natural Algorithms. PhD thesis, Dipartimento di electronica, Politecnico di Milano , Italy.

Dorigo M., Di Caro G. et **Gambardella L.M.** (1999). Ant algorithms for discrete optimization. Artificial Life, 5 :137–172.

Dorigo M., et **Di Caro G. (1999)**. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, et Glover F., editors, New Ideas in Optimization, pages 11–32, London. McGraw Hill.

Dorigo M., Maniezzo V. and **Colomi A.** (1996). The ant system: Optimization by a colony of cooperating agents, IEEE Transactions On Systems, Man, And Cybernetics. Part B, 26(1): pp.29-41.

Dréo J., Petrowski A., Siarry P. et **Taillard E.** (2003). Métaheuristique pour l'optimisation difficile. Edition Eyrolles.

Du D.-Z., and **Ko K.-I** (2000). Theory of Computational Complexity. John Wiley & Sons, Inc.

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

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.

El Dor A. (2012). Perfectionnement des algorithmes d'Optimisation par Essaim Particulaire. Applications en segmentation d'images et en électronique. Thèse De Doctorat En Informatique. Université Paris-Est École Doctorale Mathematiques Et Stic (Mstic, E.D. 532)

Elytra, (2019), Navigation et communication chez l'abeille (animation), https://www.youtube.com/watch?v=imRHXyHLMAQ. (Dernière vue en avril 2020).

Esquirol P. and **Lopez P**. (2001). Concepts et méthodes de base en ordonnancement de la production. Ordonnancement de la production.

Esquirol, P. et **Lopez, P.** (1999). L'ordonnancement. Economica.

Eswaramurthy V. P. and **Tamilarasi A**. (2009). Hybridizing tabu seach with ant colony optimization for solving job shop scheduling problems. International Journal of Advanced Manufacturing Technology, 40 :1004-1015.

Eswarmurthy V. & Tmilarasi A. (2009). Hybridizing tabu search with ant colony optimization for solving job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 40*, 1004–1015.

Ferme Saint Antoine De Grasse, (2020). Site web : *https://www.ferme-saint-antoine-de-grasse.com*

Fisher H., and Thompson G. L. (1963), "Probabilistic learning combinations of local job shop scheduling rules", in: J. F. Muth and G. L. Thompson, (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, 225-251.

Fisher M. L., and Rinnooy Kan A. H. G. (1988). "The design, analysis and implementation of heuristics", *Management Science* 34/3, Special Issue, March, 263-401.

Fleurent C. et **Ferland J.A.** (1996). Algorithmes génétiques hybrides pour l'optimisation combinatoire. RAIRO Recherche Opérationnelle, 30(4): 373–398.

French S. (1982). Sequencing and scheduling : an introduction to the mathematics ofthejobshop, John wiley and sons ed. Ellis horwood limited, 243 p.

Garey M. R., Johnson D. and **Sethi R.** (1976). The complexity of flowshop and jobshop scheduling. Mathematics of operations research, 1:117–129.

Garey M., et **Johnson D.S.** (1979a), A guide to the Theory of the NP-completeness. Computers and intractability, Freeman New York.

Garey M.R. and **Johnson D.S.** (1979b). Computers and intractability, a guide to the theory of "NP-completeness. New Jersey: Murray hill.

Gen M. and **Lin L.** (2013). Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey. Journal of Intelligent Manufacturing, 25(5), 849–866.

Giffler B. and **Thompson G. L.** (1960). "Algorithms for solving production scheduling problems", *Operations Research*, **8**(4), 487-503.

Glover F. (1977). Heuristics for integer programming using surrogate constraints. Decision Sciences, 8 :156–166.

Glover F. (1986). Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13 :533–549,.

Glover F. (1989). Tabu search – part I. ORSA Journal on Computing, 1 :190–206.

Glover F. (1990a). Tabu search – part II. ORSA Journal on Computing, 2 : 4–32.

Glover F. (1990b). Tabu search : A tutorial. Interface, 20(1) :74–94, Special issue on the practice of mathematical programming.

Glover F. et **Laguna M.** (1999). Tabu Search. Kluwer, Boston.

Glover F.et **Hanafi S.** (2002). Tabu search and finite convergence. Discrete Applied Mathematics, 119 :3–36.

Goldberg D. (1987). Genetic algorithms with sharing for multimodal function optimization. In Proceedings of the Second International Conference on Genetic Algorithms, pages 41–49,.

Goldberg D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley.

Graham R. L., Lawler E. L., Lenstra J K. and **Rinnooy Kan AHG**. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics, 5 :287–326.

Hansen P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. In Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Hansen P. et **N. Mladenovi´c**. (1999). An introduction to variable neighbourhood search. In S. Voß, S. Martello, I.H. Osman, et C. Roucairol, editors, *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458, Kluwer, Boston.

Haupt R.L. et **Haupt S.E.** (1998). Practical genetic algorithm. John Wiley & Sons, New York.

Haupt, R. L. and **Haupt S. E.** (2004). Practical Genetic Algorithms, second edition, Wiley/ Interscience, New York.

Haupt, R. (1989). "A survey of priority rule-based Scheduling," OR Spectrum, Springer-Verlag, Vol. 11, pp.3-16.

Hefetz N. and **Adiri I.** (1982). "An efficient optimal algorithm for the two-machines unit time job-shop schedule-length problem", *Mathematics of Operations Research* 7, 354-360.

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.

Hernández-Ramírez L., Solis J. F., Castilla-Valdez G., González-Barbosa J. J., Terán-Villanueva D. & Morales-Rodríguez M. L. (2019). A hybrid simulated annealing for job shop scheduling problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *10*(1), 6-15.

Hertz A., Taillard E. et **de Werra D.** (1997). Local search in combinatorial optimization, chapter A tutorial on tabu search, pages 121–136. J. Wiley & Sons Ltd, New York.

Holland J.H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.

Holland J.H. (1992). Adaptation in natural and artificial systems : An introductory analysis with applications to biology, control, and artificial intelligence. MIT Press/Bradford books, Cambridge, MA,. 2nd edition.

Huang R. Yu T. (2017). An Effective Ant Colony Optimization Algorithm for Multi-Objective Job shop Scheduling with Equal-Size Lot-Splitting, Applied Soft Computing Journal, http://dx.doi.org/10.1016/j.asoc.2017.04.062

Jackson, J. R. (1956). "An extension of Johnson's result on job lot scheduling", *Nav. Res. Logist. Quart*. 3/3, 201-203.

Jain A.S. and **Meeran S.** (1999): Deterministic job-shop scheduling: Past, present and future, European Journal of Operational Research, 113(2): pp.390-434.

Johnson S. M. (1954). "Optimal two- and three-stage production schedules with set-up times included", *Nav. Res. Logist. Quart*. 1, 61-68.

Jones A. and **Rabelo L.C.** (1998). "Survey of job shop scheduling techniques," Technical Paper, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD. (Téléchargeable du site Web http://www.mel.nist.gov/msidlibrary/doc/luis.pdf).

Karaboga D. (2005). An idea based on honey bee swarm for numerical optimization. Technical report-TR06, Engineering faculty, Computer Engineering Departement, Erciyes University, Kayseri, Turkey.

Karaboga D., Gorkemli B., Ozturk C. and **Karaboga N.** (2014). A comprehensive survey: artificial bee colony (ABC) algorithm and applications". Artificial Intelligence Review, Vol. 42, No. 1, pp.21-57.

Karaboga D. and **Akay B.** (2009). 'A comparative study of artificial bee colony algorithm'. Applied mathematics and computation, Vol. 214, No. 1, pp.108-132.

Karaboga D. and **Basturk B.** (2007). 'A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm'. Journal of global optimization, Vol. 39, No. 3, pp.459-471.

Karaboga D. and **Basturk B.** (2008) 'On the performance of artificial bee colony (ABC) algorithm'. Applied soft computing, Vol. 8, No. 1, pp.687-697.

Karaboga D., Gorkemli B., Ozturk C. and **Karaboga N.** (2014). 'A comprehensive survey: artificial bee colony (ABC) algorithm and applications'. Artificial Intelligence Review, Vol. 42, No. 1, pp.21-57.

Karp R.M. (1972). Reducibility among combinatorial problems, in Complexity of Computer Computations, M.e. Thatcher, Editor. Plenum Press, p.85-103.

Khedim A. et **Souier M.** (2016). « adaptation d'un algorithme de colonie d'abeilles pour l'ordonnancement d'un job-shop ». La 3ème édition de la conférence IEEE internationale. Gestion Opérationnelle De La Logistique (GOL'16). Du 23 - 25 mai 2016. Fès, Maroc.

Khedim A., Sari Z. et **Souier M.** (2014). «*Artificial Bee Colony (ABC) algorithm for the job-shop scheduling problem* »**.** META'2014 International conference on Metaheuristics and Nature Inspired Computing, organized in Marrakech (Morocco) in October 2014.

Kirkpatrick S., Jr C.D. et **Vecchi M.P.** (1983). « Optimization by simulated annealing ». Science, vol.220, pp. 671-680.

Korytkowski P., Rymaszewski S. and **Wisniewski T**. (2013). Ant colony optimization for job shop scheduling using multi-attribute dispatching rules. International Journal of Advanced Manufacturing Technology, 67 :231-241.

Koulamas C., Antony S.R., et **Jaen R**.(1994). A survey of simulated annealing applications to operations research problems. Omega, 22 :41–56,.

Land A. H. et **Doig A. G.** (1960). "An automatic method for solving discrete programming problems", Econometrica, Vol. 28, pp. 497-520.

Laribi I. (2018). Résolution de problèmes d'ordonnancement de type Flow-Shop de permutation en présence de contraintesde ressources non-renouvelables. Thèse de doctorat, Université de Tlemcen, Algérie.

Lawler E. L., Lenstra J. K. and **Rinnooy Kan A. H. G.** (1982). "Recent developments in deterministic sequencing and scheduling: A survey", in: M. A. H. Dempster, *et al*. (eds.), *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 35-73.

Lawrence D. (1985). Job-Scheduling with genetic algorithms. In: First International conference on genetic algorithms, Mahwah, New Jersey:136-140.

Lee C.Y., de Matta R. and **Hsu V.N.** (1997). Current trends in deterministic scheduling, Annals of Operations Research 70, 1- 41.

Lenstra J.K. and **Rinnooy Kan A.H.G.** (1979). "Computational complexity of discrete optimization problems". Annals of Discrete Mathematics; Vol. 4, pp.121-140.

Les ruchers de la découverte, 2014, La danse des abeilles, https://www.youtube.com/watch?time_continue=19&v=ABGqdQfqOog&feature=emb_logo, (Dernière vue en avril 2020).

Lian Z., Jiao B. et **Gu X.** (2006). A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Applied Mathematics and Computation*, *183*(2), 1008–1017. doi:10.1016/j.amc.2006.05.168.

Liao C.L. and **You C.T.** (1992). An improvement formulation for the job shop scheduling problem, Journal of Operational Research Society, 43, 1047-1054.

Lin T. L., Horng S. J., Kao T. W., Chen Y. H. and **Run R. S.** (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications, 37*(3), 2629–2636.

Lourenço H.R., Martin O. et Stützle T. (2000). Iterated local search. Technical report, Technical University of Darmstadt, Germany. Preprint.

Lourenço H.R., Martin O. et **Stützle T.** (2002). Iterated local search. In F. Glover et G. Kochenberger, editors, Handbook of Metaheuristics, pages 321–353. Kluwer Academic Publishers, Nowell, MA.

M. Pirlot et **Vidal R.V.** (1996). Simulated annealing : A tutorial. Control & Cybernetics, 25 :9–31.

Maniezzo V., Stutzle T. et **Vob S.** (2009). Metaheuristics: Hybridizing Metaheuristics and Mathematical Programming. Springer.

Manne, A.S. (1960). On the job shop scheduling problem, Operations Research. 8, 219-223.

Martin O., Otto S.W. et Felten E.W. (1991). Large-step Markov chains for the traveling salesman problem. Complex Systems, 5 :299–326.

Matsuo H., Suh C. J. et **Sullivan R. S.** (1988), "A controlled search simulated annealing method for the general job-shop scheduling problem", Working Paper #03-04-88, Graduate School of Business, The University of Texas at Austin, Austin, Texas, USA.

Maud Miran, (2014). La dance du huit, https://www.youtube.com/watch?v=AjMgF2swRB0 , (Dernière vue en avril 2020).

Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H. and **Teller, E.** (1953). Equations of state calculations by fast computing machines. Journal of Chemical Physics 21(6), 1087-1092.

Millonas M.M. (1994). Swarms, phase transitions, and collective intelligence. In: Artificial life III. Addison- Wesley, Reading, pp 417–445.

Milošević M., Lukić D., Đurđev M., Antić A. and **Borojević S.** (2015). An overview of genetic algorithms for job shop scheduling problems. *J Prod Eng*, *18*, pp.11-15.

Mitchel M. (1998). An introduction to genetic algorithms. MIT Press, Cambridge, MA.

Mladenovi´c N. et **Hansen P.** (1997). Variable neighbourhood decomposition search. Computers and Operations Research, 24 :1097–1100.

Muth, J. F. and **Thompson G. L.** (1963), (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, 225-251.

Nakano R. and **Yamada T. (**1991). "Conventional genetic algorithm for job shop problems," in: R.K. Belew, L.B. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 474-479.

Neto R. F. T., & Godinho Filho M. (2013). Literature review regarding ant colony optimization applied to scheduling problems: Guidelines for implementation and directions for future research. Engineering Applications of Artificial Intelligence, 26(1), 150–161.

Nicholson T. (1971), Optimization in industry, in Optimization Techniques. Longmann Press: London, Chapitre 10.

Nowicki E., & Smutnicki C. (1996). A fast taboo search algorithm for the job-shop problem. *Management Science, 42*(6), 797–813.

One bee, https://www.one-bee.fr/index.php? option=com_content&view=article&id=61 &Itemid= 162, (Dernière vue en avril 2020).

Ouis Khedim, A., Souier, M. and Sari, Z. (2020). 'Combinatorial artificial bee colony algorithm hybridised with a new release of iterated local search for job shop scheduling problem', forthcoming in Int. J. Operational Research. https://www. inderscience.com /info/ingeneral/forthcoming.php?jcode=ijor.

Pan C.H. (1997). A study of integer proramming formulations for scheduling problems, International Journal of Systems Science 28, no. 1, 33-41.

Panwalkar S. S. & Iskander W. (1977). A survey of scheduling rules. *Operations Research, 25*(1), 45–61.

Papadimitriou C. (1994). Computational Complexity. Addison Wesley.

Papadimitriou, C.H. and **Steiglitz K. (**1982). Combinatorial optimization - algorithms and complexity. Prentice Hall.

Parker R. G. (1995). *Deterministic Scheduling*, Chapman and Hall, London.

Pezzella F. & Merelli E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operation Research, 120*, 297–310.

Pham D. N. (2008). Phd thesis, Complex Job Shop Scheduling: Formulation, Algorithms and Healthcare Application. Faculty of Economics and Social Sciences, Fribourg, Switzerland.

Pillet M., Courtois A., Bonnefous C., (2011). Gestion de production, 5ième édition , Éditions d'Organisation.

Pinedo M. L. (2005). Planning and Scheduling in Manufacturing and Services, ISBN 0-387-22198-0, Springer, New York.

Pinedo M. L. (2016). Scheduling. Springer, 5th édition.

Pirard, F. (2005). Une démarche hybride d'aide à la décision pour la reconfiguration et la planification stratégique des réseaux logistiques des entreprises multi-sites. Thèse de doctorat. Université de Mons (Belgique).

Rameshkumar K. and **Rajendran C.** (2018). "A novel discrete PSO algorithm for solving job shop scheduling problem to minimize makespan", IOP Conf. Ser.: Master. Sci. Eng. 310012143.

Rechenberg I. (1973). Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, PhD thesis, Reprinted by Fromman-Holzboog.

Reeves C.R. (1993). Modern heuristic techniques for combinatorial poblems. J.Wiley and Sons, New York, 320.

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", 21(4): 25–34.

Ribeiro C.C. and **Maculan N.** (1994). (Eds.), Applications of combinatorial optimization. Annals of Operations Research 50.

Rinnooy Kan A.H.G. (1976). Machine Scheduling Problem: Classification, Complexity and Computations. Nyhoff.

Roy B. et **Portmann M. C.** (1979), Les problèmes d'ordonnancement, applications et méthodes, Cahier du LAMSADE, Université Paris-9.

Roy B., Sussmann B. (1964). Les problèmes d'ordonnancement avec contraintes disjonctives. Note D.S. no. 9 bis, SEMA, Paris, France.

Sadeh N. and **Nakakuki Y.** (1996), "Focused simulated annealing search - an application to job-shop scheduling," *Annals of Operations Research* 63, 77-103.

Sakarovitch M. (1986). De l'efficacité des algorithmes à la complexité des problèmes, en Analyse combinatoire, Dunod, Chapitre 2.

Salvesen M. E., and **Anderson S. L.** (1951), " Some notes on the problem of programming industrial production", George Washington University Logistics papers.

Satake T., Morikawa K., Takahashi K. & **Nakamura N.** (1999). Simulated annealing approach for minimizing the make-span of the general job shop. *International Journal of Production Economics, 60*(61), 515–522.

Schoenauer M. Michalewicz Z.(1997). Evolutionary computation: an introduction. Control and Cybernetics 26(3) : 307-338.

Schwefel H. P. (1977). Numerische Optimierung Von Computer-Modellen Mittels Der Evolutions- Strategie, Birkhauser Verlag.

Schwefel H.P. (1984). Evolution strategies : A family of non-linear optimization techniques based on imitating some principles of organic evolution. Annals of Operations Research, 1 :165–167.

Seeley T. D., Visscher P.K. (1988). "Assessing the benefits of cooperation in honeybee foraging: search costs, forage quality, and competitive ability", Behav. Ecol. Sociobiol., 22: 229-237.

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.

Srinivasan V. (1979). "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness," Naval Research Logistics Quarterly, Vol. 18, pp. 317-327, 1971.

Storer R.H., Wu S.D., Vaccari R. (1992). New search spaces for sequencing problems with applications to job-shop scheduling. Management Science; 38 (10) : 1495-1509.

Suresh R. K. and Mohanasundaram K. M. (2006). „Pareto archived simulated annealing for job shop scheduling with multiple objectives". The International Journal of Advanced Manufacturing Technology, Vol. 29, No. 1-2, pp.184-196.

Syswerda G. (1991). 'Scheduling optimization using genetic algorithms', in: Davis, L. (Eds.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, pp.332-349 (Chapter 21).

T'kindt V. and **Billaut J.C.** (2006). Multicriteria scheduling : theory, models and algorithms. Springer Science & Business Media.

Talbi E. (2009). Metaheuristics: From Design to Implementation. John Wiley and Sons, Inc.

Tamssaouet K., Dauzère-Pérès S. and **Yugma, C.** (2018). Metaheuristics for the job-shop scheduling problem with machine availability constraints. Computers & Industrial Engineering, vol. 125, pp.1-8.

Tan Y., Hildebrandt T. and **Scholz-Reiter B.** (2016). Configuration and the advantages of the shifting bottleneck procedure for optimizing the job shop total weighted tardiness scheduling problem". Journal of Scheduling, vol. 19, No. 4, pp.429-452.

Tay J. C., Ho N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible jobshop problems. *Computers&Industrial Engineering, 54*(3), 453–473.

Teodorović D., Šelmić M. and Davidović T. (2015). 'Bee colony optimization part II: the application survey'. Yugoslav Journal of Operations Research, Vol. 25, No. 2, pp.185–219.

Udomsakdigool A. and **Kachitvichyanukul V.** (2008). Multiple colony ant algorihtm for jobshop scheduling problem. International Journal of Production Research, 46 :4155-4175.

Vaessens R. J. M., Aarts E. H. L. and **Lenstra J. K.** (1996). Job Shop Scheduling by Local Search. INFORMS Journal on Computing 8, 3, 302–317.

Van Hoorn J. J. (2015). Job shop instances and solutions. http://jobshop.jjvh.nl

Van Hoorn J.J. (2018). The Current state of bounds on benchmark instances of the job-shop scheduling problem. *J Sched* **21,** 127–128. https://doi.org/10.1007/s10951-017-0547-8

Van Laarhoven P. J. M., Aarts E. H. L. and **Lenstra J. K**. (1988), "Job shop scheduling by simulated annealing", Report OS-R8809, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

Van Looveren A., Gelders L., Van Wassenhove L. (1986). A review of FMS planning models, in: Modeling and Design of Flexible Manufacturing Systems. Elsevier Science, Amsterdam.

Vancheeswaran R. & **Townsend M.A.**(1993). Two-stage heuristic procedure for scheduling job shops. *Journal of Manufacturing Systems*, 12, 4.

Velmurugan P. S., Selladurai Z. V. (2007). "A Tabu Search Algorithm for Job Shop Scheduling Problem with Industrial Scheduling Case Study," International Journal of Soft Computing, Medwell © Medwell Journals, Vol. 2, n°4, pp. 531-537.

Vidal R.V., editor. Applied simulated annealing, volume 396 of LNEMS. Springer-Verlag, Berlin, 1993.

Von Frisch K. & **Lindauer M.** (2003). The "Language" and Orientation of the Honey Bee. Annual Review of Entomology. 1. 45-58. 10.1146/annurev.en.01.010156.000401.

Voudouris C. (1997). Guided local search for combinatorial optimization problems. PhD thesis, Dept. of Computer Science, University of Essex, Colcherster, UK.

Voudouris C. et **E. Tsang. (**1995). Guided local search. Technical Report TR CSM-247, University of Essex, UK.

Wagner H.M., (1959) An integer linear programming model for machine scheduling, Naval Research Logistics Quarterly, 6, 131-140.

Werner F. (2011). Genetic Algorithms for Shop Scheduling Problems: A Survey. Chapter1. Otto-von-Guericke-Universität, Fakultät für Mathematik, 39106 Magdeburg, Germany.

Werner F. (2011). Genetic algorithms for shop scheduling problems: a survey. Preprint, 11, 31.

Werner F. Winkler A. (1991). Insertion techniques for the heuristic solution of the job shop problem. *Communication Personnelle*.

Widmer M. (2001). « Les Métaheuristiques : Des outils performants pour les problèmes industriels ». 3ème Conférence Francophone de MOdélisation et SIMulation MOSIM'01, 25-27 avril 2001, Troyes.

Wikipedia. Image "Algorithme à colonie de fourmis", 2019. Online ; accédé le : 2019-11-02. wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis#/media/Fichier:Aco_branches.svg.

Yahouni Z. (2017). Le meilleur des cas pour l'ordonnancement de groupes : Un nouvel indicateur proactif-réactif pour l'ordonnancement sous incertitudes. Thèse de doctorat, Université de Tlemcen, Algérie.

Yamada T. and **Nakano R.** (1992). A genetic algorithm applicable to large-scale jobshop problems. In: Manner, R., Manderick, B. (Eds.), Proceedings of the Second International Workshop on Parallel Problem Solving from Nature (PPSN'2), Brussels, Belgium; 281-290.

Yamada T. and **Nakano R.** (1996). "Job-shop scheduling by simulated annealing combined with deterministic local search", *Meta-heuristics: Theory and Applications*, Kluwer Academic Publishers, MA, USA, 237-248.

Yamada T., Rosen B. E. and **Nakano R.** (1994). "A simulated annealing approach to jobshop scheduling using critical block transition operators", *IEEE ICNN'94 International Conference on Neural Networks*, Orlando, Florida, USA, 4687-4692.

Zhang J., Ding G., Zou Y., Qin S. and **Fu J.** (2017). "Review of job shop scheduling research and its new perspectives under Industry 4.0". Journal of Intelligent Manufacturing, pp.1-22.

Zhang R., Wu C. (2008). A hybrid approach to large-scale job shop scheduling. *Applied Intelligence, 32*(1), 47–59.

Zhou R., Nee A.Y.C. and **Lee H.P.**(2009). Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problem. International Journal of Production Research, 47 : 2903- 2920.