الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي و البحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة أبي بكر بلقايد– تلمسان

Université Aboubakr Belkaïd-Tlemcen

كلية التكنلوجيا

Faculté de Technologie

Département de Génie Electrique et Electronique (GEE)

Filière : Génie Industriel

MASTER GENIE INDUSTRIEL

PROJET DE FIN D'ETUDES

Présenté par :

EMBOUAZZA Khaled&BETTAHAR Mohamed Riad

Intitulé du Sujet

Résolution d'un problème d'ordonnancement dans un atelier flow-shop avec l'Algorithme du chauve-souris

Soutenu le 08 novembre 2020, devant le jury composé de :

SOUIER Mehdi	**MCA**	Univ. Tlemcen	Président
BESSENOUCI Hakim Nadhir	**MAA**	Univ.Tlemcen	Examinateur
HOUBAD Yamina	**MAA**	Univ. Tlemcen	Encadreur
BOUMEDIENE Fatima Zohra	**Docteur**	Univ. Tlemcen	Co-encadreur

Année Universitaire : 2019/2020

Dédicace

À ma très chère mère Leila

Qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

À la mémoire de mon père Noureddine

Disparu trop tôt. J'espère que, du monde qui est sien maintenant, il apprécie cet humble geste comme preuve de reconnaissance de la part d'un fils qui a toujours prié pour le salut de son âme. Puisse Dieu, le tout puissant, l'avoir en sa sainte miséricorde !

À ma sœur et mes frères.Ismahane, Redouane, Faiçal et Lotfi

En témoignage de mon affection fraternelle, de ma profondetendresse et reconnaissance, je vous souhaite une vie pleine de bonheur et desuccès et que Dieu, le tout puissant, vous protège et vous garde.

Àma chère grand-mère

Qui m'a accompagné par ses prières, sa douceur, puisse Dieu lui prêter longuevie et bcp de santé et de bonheur dans les deux vies.

À mes chères tantes FATIMA et WARDA

Vous avez toujours été présents pour les bons conseils.
Votre affection et votre soutien m'ont été d'un grand secours au long de ma vie professionnelle et personnelle.

À mes amis de toujours : Abdelkader, Habib

En souvenir de notre sincère et profonde amitié et des momentsagréables que nous avons passés ensemble.
Veuillez trouver dans ce travail l'expression de mon respect le plusprofond et mon affection la plus sincère.

-EMBOUAZZA Khaled-

Dédicace

À :

La mémoire de mon défunt grand père

Mohamed MOKHTARI,

La personne qui m'a donnée tout le courage, la tendresse et la patience

Ma chère mère,

Ma chère grand-mère maternelle,

Mon cher oncle **ABDELKADER**

Mes chères tantes **DJOHER** et **HANANE** et leurs petites familles

Surtout, **Hamza, Youcef, Hamid, Zaki, toufik, ...**

Et, Ceux qui, avec je partage de bons souvenirs,

Enfin Toute la promo du Génie industriel 2015/2016

De la part de :

-BETTAHAR Mohamed Riad-

Remerciement

En préambule à ce mémoire nous remercions ALLAH le tout puissant et miséricordieux, qui nous a donné le courage, la force et la foi de mener à terme ce modeste travail.

À notre encadreur Madame **Houbad Yamina**et Co-encadreur**BOUMADIENE Fatima Zohra**

Nous tenons d'abord à vous remercier très chaleureusement d'avoir accepté de suivre notre projet et pour votre attention particulière que vous nous avez donnée au courant de l'année, pour vos conseils indispensables, votre disponibilité, votre aide, vos orientations, le tempsconsacré et votre enthousiasme envers ce mémoire, qui ont constitués un apport considérable grâce auquel ce travail a pu être mené à bon port.

Nos vifs remerciements vont également aux membres du jury, qui ont bien voulu nous honorer par leur précieuse présences parmi nous, afin d'examiner d'évaluer ce modeste travail.

À notre Président Mr **SOUIER Mehdi**

Qui nous a donné le privilège d'avoir accepté de présider le jury de la soutenance de notre PFE. Vos qualités, vos valeurs, votre sérieux, votre compétence et votre sens du devoir nous ont énormément marqués. Veuillez trouver ici l'expression de notre respectueuse considération et notre profonde admiration pour toutes vos qualités scientifiques et humaines. Ce travail est pour nous l'occasion de vous témoigner notre profonde gratitude.

À notreexaminateur

Monsieur BESSENOUCI Hakim Nadhir

Vous nous faites l'honneur d'accepter avec une très grande amabilité de siéger parmi notre jury. Nous vous remercions pour votre estimable participation dans l'élaboration de ce travail. Permettez-nous de vous exprimer notre admiration pour vos qualités humaines et professionnelles ainsi que, votre amabilité et votre gentillesse qui méritent toute reconnaissance.

Nous saisissons cette occasion pour vous exprimer notre profonde gratitudetout en vous témoignant notre respect.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

Introduction générale

Dans un environnement de production complexe, On rencontre de grands problèmes pour l'ordonnancement avec une grande complexité donc pour les résoudre on fait appel aux méthodes dites approchées pour trouver des solutions généralement, de bonne qualité avec un temps raisonnable.

Les métaheuristiques sont des algorithmes de type stochastique visant à résoudre une large gamme de problème d'optimisation difficile, pour lesquels on ne connaît pas de méthodes classiques plus efficace. Souvent inspirées d'analogies avec la réalité comme la physique (recuit simulé, diffusion simulée...) la biologie (les algorithmes évolutionnaires, la recherche taboue...) et l'éthologie (les colonies de fourmis, les essaims particulaires...). Elles sont généralement conçues au départ pour résoudre des problèmes discrets, mais peuvent s'adapter aux autres types de problèmes.

Notre travail s'inscrit dans le cadre d'utilisation d'une métaheuristiques pour la résolution de problèmes flow shop avec minimisation de makespan. Nous nous sommes intéressés à l'adaptation, la programmation de l'algorithme chauve-souris (Bat Algorithm) pour pouvoir comparer les résultats obtenus avec l'optimum. Ce document s'articule autour de trois chapitres :

Le premier chapitre de ce mémoire donne des notions et définitions pour les problèmes d'ordonnancement dans les systèmes de production.

Dans le deuxième chapitre nous nous intéressons de manière approfondie aux métaheuristiques tout en décrivant certaines méthodes à solution unique et à base de population. Ensuite nous présentons la métaheuristique que nous avons adapté pour la résolution de notre problème d'ordonnancement : l'algorithme de chauve-souris en incluant sa définition, son origine et son algorithme de base.

Dans le troisième chapitre, nous présentons l'adaptation de notre algorithme pour la résolution de problème d'ordonnancement d'atelier de type flow-shop dont l'objectif est la minimisation Makespan et les résultats de simulations obtenus qui ont montré l'efficacité de l'algorithme chauve souri (Bat Algorithm).

Enfin, on termine par une conclusion générale qui fera la synthèse de notre travail en ouvrant de nouvelles perspectives.

Chapitre 1

Les problèmes d'ordonnancement dans les systèmes flexibles de production

1. Introduction

Le système de production rassemble l'ensemble des moyens qui permettent la transformation de ressources en produits ou en services.

Dans un système de production un problème d'ordonnancement peut être considéré comme un sous problème de planification. Il est formulé en problème d'optimisation combinatoire, leur résolution nécessite des algorithmes plus conformant tel que les méthodes approchées, et plus particulièrement les métaheuristiques.

Ce chapitre est consacré aux problèmes d'ordonnancement dans les systèmes flexibles de production. La première partie du chapitre présente les notions de base sur les problèmes d'ordonnancement, à savoir les contraintes, les objectifs, et les problèmes d'ordonnancement liés aux ateliers multi machines. La seconde partie présente les méthodes de résolution des problèmes d'ordonnancement.

1.1. Les problèmes d'ordonnancement

Les problèmes d'ordonnancement apparaissent dans tous les domaines de l'économie, l'informatique (les tâches sont les programmes ; les ressources sont les processeurs), la construction (suivi de projet), l'industrie (activités des ateliers en gestion de production et problèmes de logistique), l'administration (emplois du temps). [23]

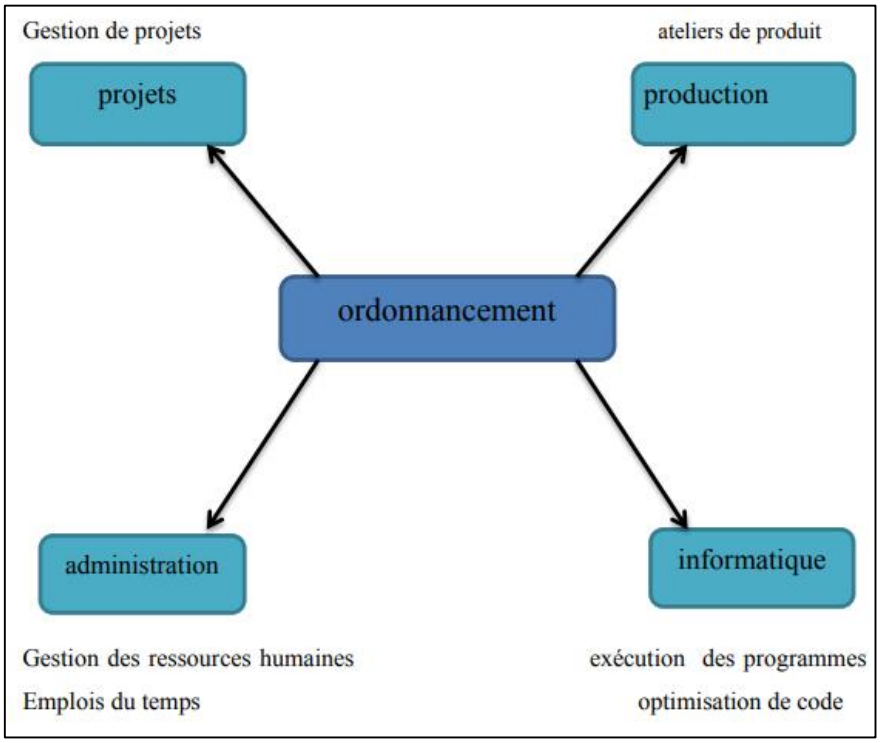

Figure 1-1 : les domaines de l'ordonnancement

1.1.1. Définitions de l'ordonnancement

Ordonnancer un ensemble de tâches c'est programmer leur exécution en leur allouant les ressources requises et en fixant leurs dates de début. La théorie de l'ordonnancement traite de modèles mathématiques mais analyse également des situations réelles fortes complexes ; aussi le développement de méthodes utiles ne peut-il être que le fruit de contacts entre la théorie et la pratique [1].

1.1.2. Paramètres de l'ordonnancement

Dans un problème d'ordonnancement interviennent quatre notions fondamentales : les tâches, les ressources, les contraintes et les objectifs (ou critères) [2].

1.1.2.1. Les tâches

Une tâche est une entité élémentaire localisée dans le temps par une date de début et/ou de fin, dont la réalisation nécessite une durée, et qui consomme un moyen selon une certaine intensité. Certains modèles intègrent la notion de date due, une date à laquelle la tâche doit être finie ; dans ces cas, le retard induit une pénalité. [24]

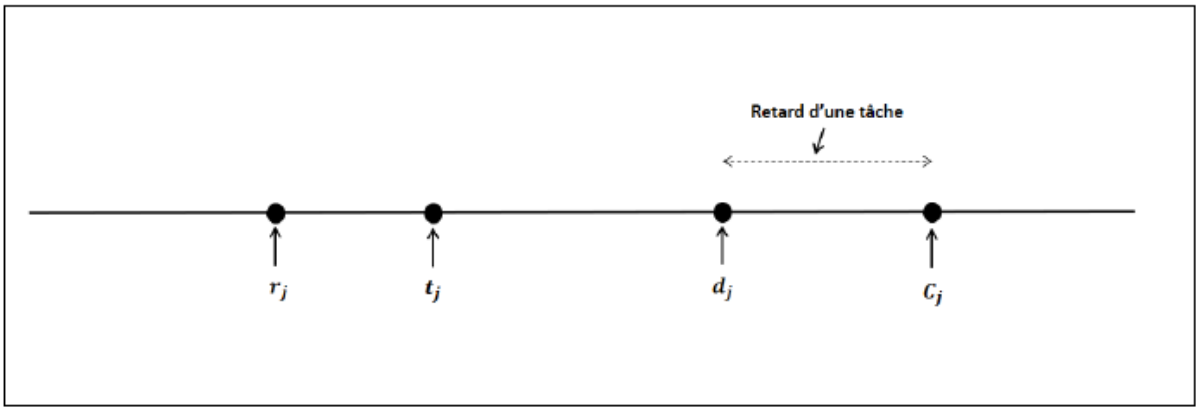

Figure 1-2 : Représentation du retard d'une tâche.

Les données spécifiant chaque tâche J_j peuvent être représentées comme suit :

p_{ij} Le temps de traitement de la tâche J_j par la ressource M_i (cas où la tâche est composée de plusieurs opérations).

p_jLe temps de traitement de la tâche J_j (la tâche est équivalente à une seule opération).

r_j La date de disponibilité (release date) de la tâcheJ_j (sa date de début au plus tôt).

d_j La date de fin au plus tard (due date) de la tâche J_j.

t_j La date de début d'exécution de la tâche J_j.

w_j Le poids de la tâche J_j.

C_j La date de fin de la tâche J_j (completion time).

F_j La durée de séjour (ou de présence) de la tâche J_j (flow time) : $F_j = C_j - r_j$.

T_j Le retard vrai de la tâche J_j (tardiness) :$T_j = max\ (0, C_j - d_j)$.

E_j L'avance de la tâche J_j (earliness) : $E_j = max\ (0, d_j - Cj)$.

L_j L'écart par rapport à la fin souhaitée de la tâche J_j (lateness) : $L_j = C_j - d_j$.

U_jIndicateur de retard de la tâche J_j: $U_j = 1$ si $T_j > 0$, $U_j = 0$ [25]

1.1.2.2. Ressources

Une ressource est un moyen, technique ou humain, dont la disponibilité limitée ou non est connu a priori.

On distingue plusieurs types de ressources :

- **Ressources consommables :** Une ressource est consommable si après avoir été utilisée par une ou plusieurs tâches, elle n'est plus disponible, donc elle ne peut être utilisée plus d'une fois (LOPEZ, 1991). Les matières premières et le budget peuvent être considérés comme ressources consommables
- **Ressources renouvelables :** Une ressource renouvelable (réutilisable) est disponible à nouveau, après avoir été utilisée par une ou plusieurs tâches (les hommes, les machines, l'équipement en général…). Ce type de ressource peut être à son tour décomposé en deux types (LOPEZ, 1991) : Les ressources disjonctives (non partageables) qui ne peuvent n'exécuter qu'une tâche à la fois (machine-outil, robot manipulateur) et Les ressources cumulatives (partageables) qui peuvent être utilisées par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail). [25]

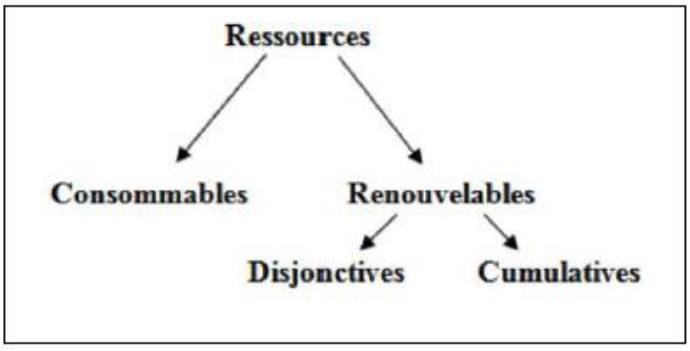

Figure 1-3 : Classification des ressources

1.1.2.3. Contraintes

Une contrainte est une condition que doit respecter le plan de travail. Elle limite le degré de liberté d'exécution de processus (LOPEZ, 1991). Cette condition est liée aux opérations entre elles, aux ressources, etc.

Les contraintes potentielles : Ce sont des contraintes spécifiant les limites relatives à la localisation temporelle des tâches et à la réalisation de successions entre elles.

Les contraintes de localisation temporelles : Elles représentent les bornes de l'intervalle de temps durant lequel une tâche doit être traitée. L'intervalle est représenté par la date de début d'exécution au plus tôt et la date de fin d'exécution au plus tard.

$r_i \leq t_i$ et $c_i \leq d_i$

Les contraintes de succession ou de précédence : Elles prennent en compte la succession des opérations de la gamme opératoire. Les contraintes indiquent que le début d'exécution d'une tâche j doit être après la date de fin d'exécution d'une autre opération i qui la précède, d'où :

$T_i \geq t_i + p_i$

Les contraintes de ressources : Ces contraintes de limitation de ressources renouvelables expriment la nature de la quantité des moyens utilisés par les tâches. Il y a deux types de contraintes suivant la nature des ressources :

- **Contraintes disjonctives :** Elles obligent à réaliser toute paire de tâches sur des intervalles de temps disjoints. Soit une tâche i, elle s'exécute soit avant j, soit après j.
- **Contraintes cumulatives :** Elles interdisent la réalisation d'un nombre trop important de tâches, compte tenu de la disponibilité maximale de la ressource à chaque instant et des quantités requises individuellement par les tâches. [25]

1.1.2.4. Les objectifs (critères)

Les critères représentent des conditions qualitatives et quantitatives à satisfaire lors de la résolution d'un problème d'ordonnancement.

Ils existent plusieurs formes de critères tel que :

- La minimisation du temps d'exécution de la dernière tâche (makespan),
- La minimisation de la moyenne des dates d'achèvement des tâches,
- La minimisation des retards sur les dates d'achèvement des tâches,
- La minimisation du maximum des retards sur les dates d'achèvement des tâches,
- La minimisation des encours,
- La minimisation du coût de stockage des matières premières,
- L'équilibrage des charges des machines,
- L'optimisation des changements d'outils.

1.2. Les problèmes d'ordonnancement de type ateliers

Dans un problème d'atelier, une pièce doit être usinée ou assemblée sur différentes machines. Les tâches sont regroupées en n entités appelées travaux ou lots. Chaque lot doit

être exécuté sur m machines distinctes. L'atelier est constitué de m machines et n travaux (jobs), il est caractérisé par le nombre de machines qu'il contient et par son type.

Selon les caractéristiques des machines on peut distinguer plusieurs types des problèmes représentés par la figure 1.4: [23]

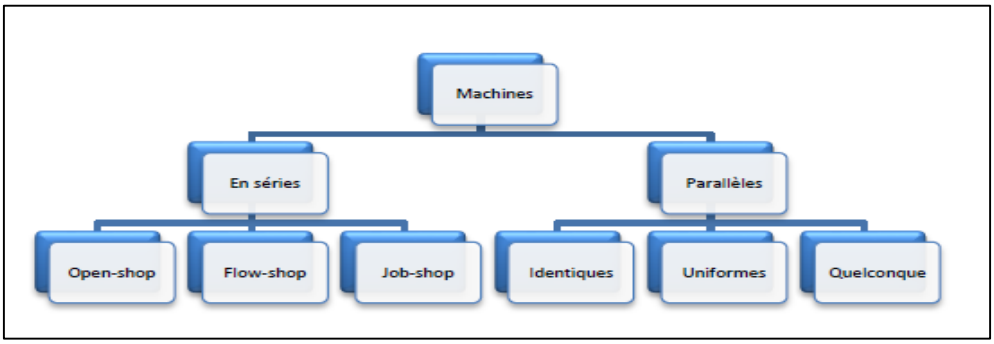

Figure 1-4 : types de machines.

1.2.1. Schémas de classification

Classification à trois champs **α**, **β**, **γ** :

- **Le champ α** : l'environnement machine

Représente le type d'atelier. Il peut prendre les valeurs suivantes :

_ 1 : problèmes à une seule machine,

_ P : problèmes à machines parallèles identiques (machines à vitesses identiques),

_ Q : problèmes à machines uniformes (machines à vitesses différentes),

_ R: problèmes à machines parallèles non liées (machines différentes),

_ F: problèmes de type flow shop,

_ FFc : problèmes de flow shop flexible

_ J : problèmes de type job shop,

_ FJc : problèmes de job shop flexible,

_ O : problèmes de type open shop.

- **Le champ β** : les caractéristiques des tâches

On trouve par exemple : Pmtn : préemption, Prec : précédence, brkdwn : Pannes, prmu: permutation, Block : Blocage, nwt : no-wait, recrc : recirculation…etc.

- **Le champ γ** : représente le critère (ou les) critère(s) à optimiser. [23]

1.2.2. Les types de problème d'ordonnancement d'ateliers

1.2.2.1. Machine unique

Pour ce type de machine, l'ensemble des tâches à réaliser est fait par une seule machine figure 1.5. Les tâches alors sont composées d'une seule opération. Cette opération nécessite la même machine. On peut rencontrer ce genre de configurations dans des situations intéressantes comme dans le cas où on est devant un système de production qui comprend une machine goulot et cette dernière a une influence sur l'ensemble du processus [4].

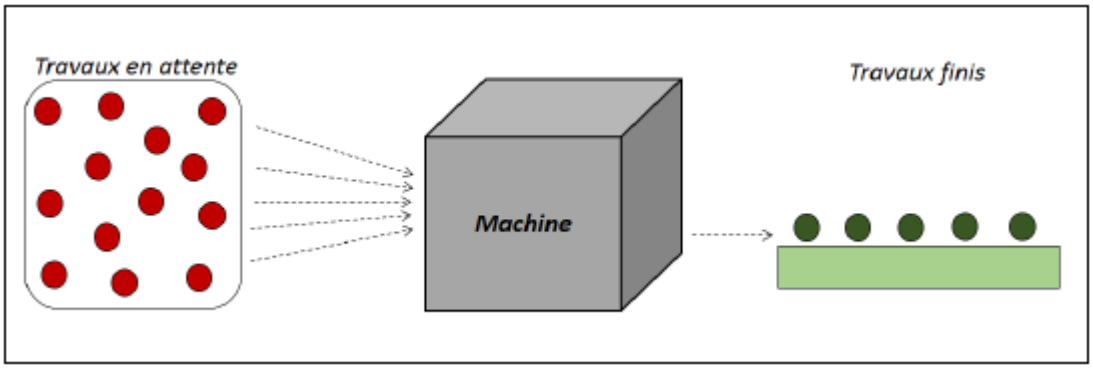

Figure 1-5 : la représentation de la machine unique.

1.2.2.2. Machines parallèles

Pour ce type on dispose d'un ensemble de machines identiques pour réaliser les travaux. Les travaux sont composés d'une seule opération. Cette opération exige une seule machine. L'ordonnancement est effectué en deux phases : la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine [4]. La figure 1.6 représente un exemple des machines parallèles. [4]

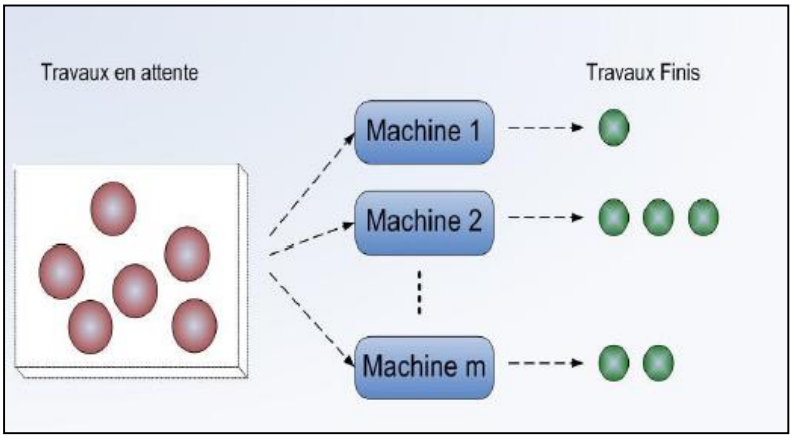

Figure 1-6 : Représentation des machines parallèles.

1.2.2.3. L'atelier flow shop

Un atelier à cheminement unique est un atelier où le processus d'élaboration de produits est dit « linéaire », c'est-à-dire lorsque les étapes de transformation sont identiques pour tous les produits fabriqués. Selon les types de produits élaborés, on distingue la production continue et la production discrète. La production continue est caractérisée par la fluidité de son processus et l'élimination du stockage. C'est le cas notamment dans les raffineries, les

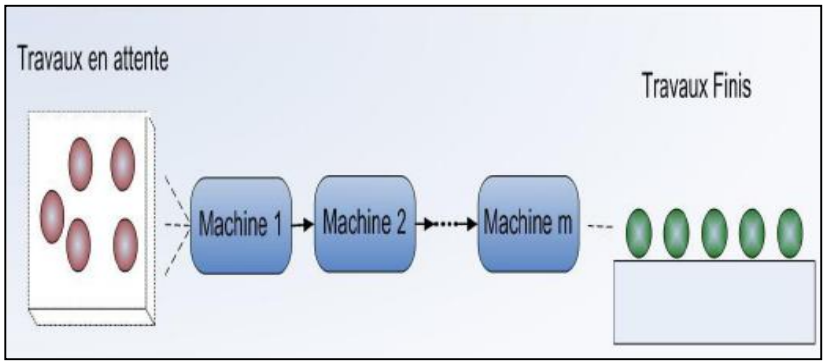

Figure 1-7 : Représentation ateliers à cheminement unique (flow-shop).

cimenteries, les papeteries... Un exemple de l'atelier est illustré par la figure 1.7

1.2.2.4. L'atelier job-shop

Les ateliers à cheminements multiples (ACM) sont des unités manufacturières traitant une variété de produits individuels dont la production requiert divers types de machines dans des séquences variées. L'une des caractéristiques d'un atelier à cheminement multiple est que la demande pour un produit particulier est généralement d'un volume petit ou moyen. Une autre caractéristique est la variabilité dans les opérations et un mix produit constamment changeant.

L'objectif le plus considéré dans le cas d'un atelier à cheminements multiples est le même que celui considéré pour un atelier à cheminement unique, à savoir trouver une séquence de tâches sur les machines qui minimise le temps total de production.

La figure 1.8 montre un exemple d'un atelier à cheminements multiples [4].

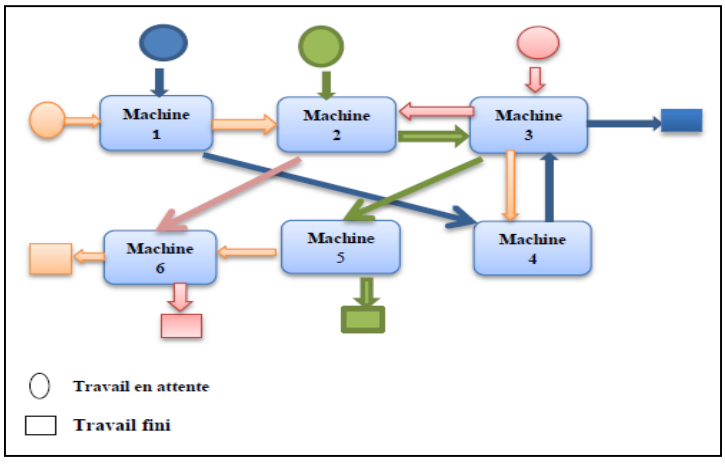

Figure 1-8 : Représentation d'ateliers à cheminement multiple (job-shop).

1.2.2.5. L'atelier open-shop

C'est un modèle d'atelier moins contraint que le Flow shop et le job shop, car l'ordre des opérations n'est pas fixe à priori. Le problème d'ordonnancement comprend d'une part à déterminer le cheminement de chaque travail et d'autre part à ordonnancer les travaux en tenant compte des gammes trouvées (Un problème de type open shop est un job shop dans lequel les contraintes de précédence sont relâchées. C'est-à dire, les opérations peuvent être effectuées dans n'importe quel ordre). [4]

1.2.3. La notion de la flexibilité

La flexibilité est une mode de gestion de la main d'œuvre qui permet aux entreprises d'ajuster rapidement la production et l'emploi (l'offre) aux fluctuations rapides des commandes des clients (demande), et il y a cinq types de flexibilité du travail :

- **La flexibilité externe quantitative** qui permet de faire fluctuer les effectifs de l'entreprise en fonction des besoins en ayant recours aux licenciements et aux contrats de travail de courte durée.
- **La flexibilité externe qualitative (ou externalisation)** qui « consiste à déplacer sur une autre entreprise le lien contractuel avec le travailleur » en aillant recours par exemple aux travailleurs intérimaires ou à l'externalisation d'un certain nombre d'activités annexes à la production (gardiennage, restauration, nettoyage…).
- **La flexibilité salariale** qui permet de faire varier à travers la rémunération des salariés, le coût de la masse salariale de l'entreprise. Elle « est conçue comme un moyen de répercuter sur les salaires les évolutions du chiffre d'affaire et de coûts de revient de l'entreprise en fonction des mouvements conjoncturels ».
- **La flexibilité interne quantitative** qui consiste à faire varier la quantité d'heures travaillées pour un effectif donné. Elle peut être réalisée par des modulations saisonnières à partir d'un contrat portant sur une durée annuelle, des temps partiels, des travails intermittents, des heures supplémentaires...
- **La flexibilité interne qualitative (ou flexibilité fonctionnelle)** qui « consiste, à quantité de travail donnée, à employer les travailleurs à des fonctions variables en fonction des besoins de la chaîne de production ou des fluctuations de la production ». [23]

Pour être plus réactives et productives, les entreprises ont essayé de trouver des solutions pour augmenter la flexibilité de leurs systèmes de production. Donc, ils ont trouvé qu'il est possible de multiplier le nombre des machines qui peuvent réaliser la même opération. Ces machines, sont regroupées en étage ou cellule. Alors on peut avoir d'autres types d'ateliers :

1.2.3.1. Flow shop hybride

Le Flow Shop hybride est une généralisation du Flow Shop classique au cas où nombreuses machines sont disponibles sur un ou plusieurs étages pour exécuter les différentes tâches du Flow Shop. Ces problèmes présentent alors une difficulté supplémentaire par rapport aux problèmes sans flexibilité des ressources. En effet, la machine qui sera utilisée pour exécuter une opération n'est pas connue d'avance, mais doit être sélectionnée parmi un ensemble donné pour construire une solution au problème. La figure 1.9 illustre un exemple d'un atelier flow show hybride à « k » étages. [23]

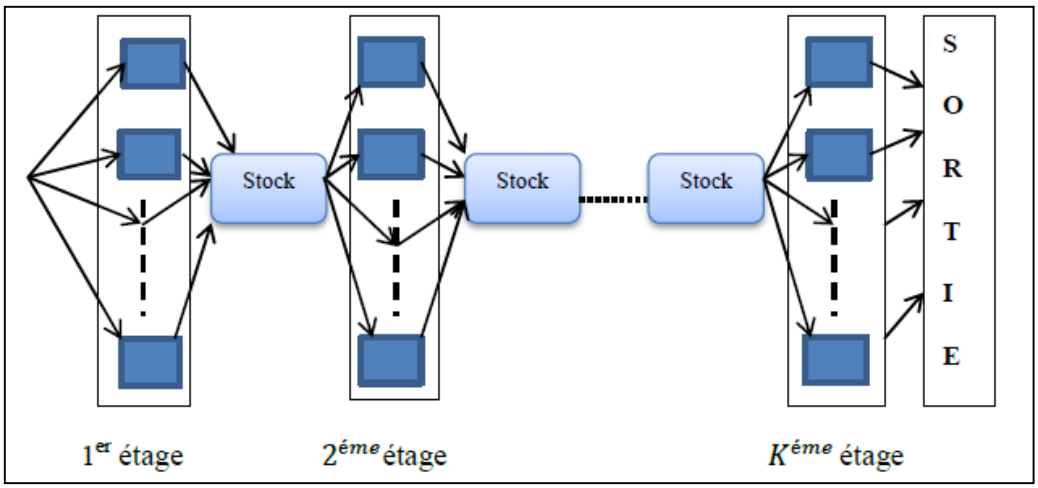

Figure 1-9 : Représentation d'un flow show hybride à « k » étages. [23]

1.2.3.2. Job shop flexible

L'atelier Job Shop Flexible est une extension du problème classique décrit précédemment. La flexibilité est due aux ressources, c'est-à-dire, l'attribution d'un sous-ensemble de ressources (machines candidates) pour le traitement de chaque opération de telle sorte que le temps opératoire de chaque opération dépend de la machine candidate sélectionnée.

En effet, il existe plusieurs degrés de flexibilité : la flexibilité faible dans laquelle quelques opérations qui sont traitable dans quelques machines. Ensuite, dans la flexibilité moyenne et forte le nombre d'opérations traitables dans plusieurs machines devient de plus en plus important. En arrivant à la flexibilité externe (totale), n'importe quelle opération est traitable sur n'importe quelle machine. La figure 1.10 montre un exemple d'un atelier Job shop simple et Job shop hybride. [3]

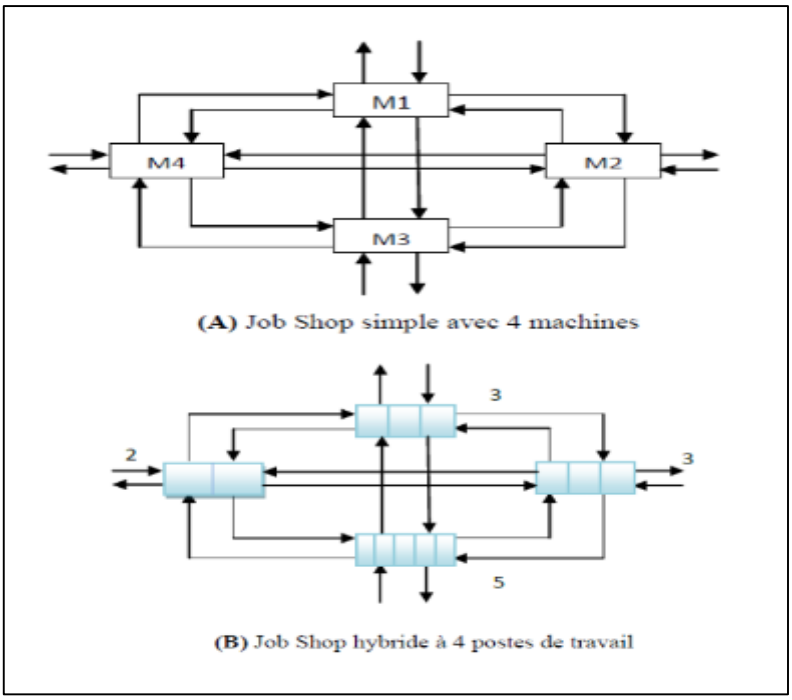

Figure 1-10 : Job shop simple & hybride [3]

1.3. Les méthodes de résolution des problèmes d'ordonnancement

Les problèmes d'ordonnancement d'ateliers sont des problèmes combinatoires extrêmement difficiles et il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les cas. Beaucoup d'entre eux peuvent prendre un temps considérable pour être résolus, la théorie de complexité des algorithmes a donné un sens précis au terme d'algorithme efficace et de problème difficile.

Déterminer la complexité d'un algorithme donné n'est pas toujours si simple. Il s'agit de déterminer quelle est la plus faible complexité d'un algorithme de résolution, parmi tous les algorithmes que nous pouvons imaginer.

La complexité d'un problème est la complexité de son meilleur algorithme connu. On peut ainsi classer les problèmes suivant leur complexité.

Cependant, on distingue deux familles de problèmes : les problèmes qui ont une complexité polynomiale, et les autres, ceux dont la complexité est exponentielle (problèmes NP-complets ou NP-difficiles) [5].

1.3.1. Méthodes exactes

Ces méthodes sont basées soit sur une résolution algorithmique ou analytique, soit sur une énumération exhaustive de toutes les solutions possibles.
Elles s'appliquent donc aux problèmes qui peuvent être résolus de façon optimale et rapidement. Les méthodes exactes ont l'avantage d'obtenir des solutions dont l'optimalité est garantie. Ces méthodes sont génériques et demandent une particularité vis-à-vis d'un problème spécifique. Parmi les méthodes exactes, on peut citer : [5]

– La méthode Branch and Bound

– Programmation dynamique

– Programmation linéaire

1.3.2. Méthode approchée

Une méthode approchée ou heuristique est un algorithme d'optimisation qui a pour but de trouver une solution réalisable de la fonction objective, mais sans garantir d'optimalité. Le principal avantage de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, bien ou mal formulés, avec ou sans contraintes. En particulier, elles ne nécessitent pas une modélisation mathématique du problème. [5]

1.4. Conclusion

Dans ce chapitre, nous avons donné des définitions générales de l'ordonnancement, ainsi que quelques notions de base, par exemple la notion de tâche, les notions des ressources, et les problèmes d'ordonnancement, les différentes classifications, les contraintes qu'on peut faire face. Et les variétés des types de problèmes d'ordonnancement d'atelier.

Les problèmes d'ordonnancement d'ateliers sont des problèmes combinatoires extrêmement difficiles et il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les cas.

Chapitre 2

Les métaheuristiques

2. Introduction

Dans la vie courante, nous sommes souvent, confrontés à des problèmes qui peuvent être décrits sous forme d'un problème d'optimisation, comme le fait de minimiser les coûts de production. L'optimisation combinatoire permet de minimiser ou maximiser des fonctions dans des systèmes dans lesquels peut intervenir un grand nombre de paramètres. Cependant, pour les problèmes dits difficiles, on ne connaît pas d'algorithmes exacts rapides permettant de les résoudre donc on fait appel aux méthodes approchées qui ne cherchent pas forcément l'optimum absolu mais donnent une solution très -proche de l'optimum absolu montrant l'inexistence d'une solution sensiblement meilleure et largement acceptée. [22]

Ce chapitre est réservé aux méthodes approchées, nous allons présenter celles les plus connues, leurs origines, leurs principes de base, et leurs algorithmes.

2.1. Généralités sur les méthodes approchées

Les méthodes exactes nécessitent des temps de traitement prohibitifs pour résoudre les problèmes de grande taille. Pour obtenir malgré tous des solutions, des méthodes approchées ont été développées. Ces méthodes donnent des solutions certes sous-optimales, mais en un temps de calcul raisonnable**.** Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale, c'est-à-dire, trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Certains problèmes demeurent hors de portée des méthodes exactes. Les méthodes approchées, dites aussi d'approximation, constituent une alternative intéressante pour traiter les problèmes d'ordonnancement de grande taille. Elles sont définies comme étant des procédures exploitant au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible. [26]

2.2. Les heuristiques

Depuis toujours, les chercheurs ont tenté de résoudre les problèmes NP-difficiles le plus efficacement possible. Pendant longtemps, la recherche s'est orientée vers la proposition d'algorithmes exacts pour des cas particuliers polynomiaux [8]. Certains problèmes demeurent hors de portée des méthodes exactes. Les méthodes approchées, dites aussi d'approximation, constituent une alternative intéressante pour traiter les problèmes d'ordonnancement de grande taille. Elles sont définies comme étant des procédures exploitant au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible [9]. Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle heuristique.

L'apparition des heuristiques a permis de trouver des solutions en général de bonne qualité pour résoudre les problèmes [8]. Lorsque celles-ci sont générales, adaptables et

applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de méta-heuristique [6]. Lorsque les premières métaheuristiques apparaissent, beaucoup de chercheurs se sont lancés dans l'utilisation de ces méthodes. Cela a conduit à une avancée importante pour la résolution pratique de nombreux problèmes.

Cela a aussi créé un engouement pour le développement même de ces méthodes. Il existe des équipes entières qui ne travaillent qu'au développement des métaheuristiques [8].

2.3. Les métaheuristiques

Apparues dans les années 1980. Les métaheuristiques forment un ensemble d'algorithmes permettant de trouver la solution la plus rapide et la plus efficace pour une large garnie de problèmes d'optimisation difficile et pour lesquels on ne connaît pas de méthode classique plus efficace. Les métaheuristiques fonctionnent selon un comportement itératif c'est-à-dire que le même schéma se reproduit un certain nombre de fois au cours de l'optimisation, généralement, elles s'articulent autour des trois notions suivantes :

- La diversification (exploration)
- L'intensification (exploitation)
- La mémorisation (apprentissage)

La diversification ou exploration désigne le processus qui dirige la procédure pour récolter de l'information sur le problème â optimiser. La stratégie de diversification la plus simple consiste â redémarrer périodiquement le processus de recherche â partir d'une solution générée aléatoirement ou choisie judicieusement dans une région non encore visitée de l'ensemble des solutions admissibles. Pour sa part, l'intensification ou exploitation utilise l'information déjà_ récoltée pour explorer, en détail, les zones jugées prometteuses dans l'espace de recherche. Sa mise en œuvre réside. Le plus souvent. Dans l'élargissement temporaire du voisinage de la solution courante. Quand â la mémorisation, elle est le support de l'apprentissage qui permet à l'algorithme de ne tenir compte que des zones où optimum global est susceptible de se trouver. Évitant ainsi, les optima locaux qui sont de bonnes solutions, mais qui ne sont pas les meilleures des solutions possibles. Ainsi, en alternant l'intensification, la diversification et la mémorisation

Le fonctionnement des métaheuristiques est progressif et itératif. L'étape initiale est souvent choisie de façon aléatoire et l'étape d'arrêt est souvent fixée t l'aide d'un critère d'arrêt. Toutes les métaheuristiques s'appuient sur l'équilibre entre l'intensification et la diversification de la recherche. Sinon, on assistera t une convergence trop rapide vers des minima locaux par manque de diversification ou â une exploration trop longue par manque d'intensification.

Le fonctionnement des métaheuristiques est généralement inspiré à partir de systèmes physiques comme le recuit simulé, de systèmes biologiques comme les algorithmes évolutionnaires. De systèmes ethnologiques comme les algorithmes de colonies de fourmis ou de l'optimisation par essaims particulaires etc.[22]

2.3.1. Définition des métaheuristiques

Le mot métaheuristique est dérivé de la composition de deux mots grecs : Heuristique qui vient du verbe heuriskein (euriskein) et qui signifie ‘trouver’, META qui est un suffixe signifiant ‘au-delà’, ‘dans un niveau supérieur’.

Selon la définition proposée par Osman « Une Métaheuristique est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l’espace de recherche. Des stratégies d’apprentissage sont utilisées pour structurer l’information afin de trouver efficacement des solutions optimales, ou presque-optimales » [10].

Pour résumer ces définitions, on peut dire que les propriétés fondamentales des Métaheuristiques sont les suivantes :

_ Les Métaheuristiques sont des stratégies qui permettent de guider la recherche d’une solution optimale ;

_ Le but visé par les métaheuristiques est d’explorer l’espace de recherche efficacement afin de déterminer des solutions (presque) optimales ;

_ Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure de recherche locale à des processus d’apprentissage complexes ;

_ Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d’optimalité ;

_ Les métaheuristiques peuvent contenir des mécanismes qui permettent d’éviter d’être bloqué dans des régions de l’espace de recherche ;

_ Les concepts de base des métaheuristiques peuvent être décrits de manière abstraite, sans faire appel à un problème spécifique ;

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

_ Les Métaheuristiques peuvent faire usage de l’expérience accumulée durant la recherche de l’optimum, pour mieux guider la suite du processus de recherche [11].

L’ensemble des métaheuristiques proposées dans la littérature sont partagées en deux catégories : des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions. Nous présentons dans ce qui suit quelques métaheuristiques des deux catégories [12].

2.3.2. Métaheuristiques à solution unique (méthodes de recherche locales)

Dans cette section, nous présentons les métaheuristiques à base de solution unique, aussi appelées méthodes de trajectoire. Contrairement aux métaheuristiques à base de population, les métaheuristiques à solution unique commencent avec une seule solution initiale et s’en éloignent progressivement, en construisant une trajectoire dans l’espace de recherche. Les

méthodes de trajectoire englobent essentiellement la méthode de descente, le recuit simulé, la recherche taboue, la recherche à voisinage variable, la méthode GRASP, la recherche locale itérée, la recherche locale guidée et leurs variantes.

Ces méthodes s'articulent toutes autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante.

2.3.2.1. Les méthodes de descente (DM : DescentMethod)

L'algorithme 2.1 présente le squelette d'une méthode de descente simple (Simpledescent). A partir d'une solution initiale, on choisit une solution dans le voisinage () de x. Si cette solution est meilleure que, (′) < ()) alors on accepte cette solution comme nouvelle solution et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage. [17]

1.*initialise* : find an initial solution x

2. **repeat**

3. *neighbourhoodsearch* : find a solution $x' \in N(x)$

4. **if** $f(x') < f(x)$ **then**

5. $x' \longleftarrow x$

6. **end if**

7. **until** $f(y) \geq f(x), \forall y \in N(x)$

Algorithme 2-1 :L'algorithme de la méthode de descente

Une version plus agressive de la méthode de descente est la méthode deplus grande descente (Deepestdescent). Au lieu de choisir une solution ′dans le voisinage de, on choisit toujours la meilleure solution du voisinage de. L'algorithme 2.2donne une description de cette méthode. [17]

1. *initialise* : find an initial solution x

2. **repeat**

3. *neighbourhoodsearch* : find a solution $x' \in N(x)$ /$f(x') \leq f(x'')$, $\forall x'' \in N(x)$

4. **if** $f(x') < f(x)$ **then**

5. $x' \longleftarrow x$

6. end if

7. **until** $f(x') \geq f(x), \forall x' \in N(x)$

Algorithme 2-2 : L'algorithme de la méthode de plus grande descente

Ces deux méthodes sont évidemment sujettes a de nombreuses critiques. Elles se basent toutes les deux sur une amélioration progressive de la solution et donc resteront bloquées dans un minimum local dès qu'elles en rencontreront un. Il existe de manière évidente une absence de diversification. L'équilibre souhaité entre intensification et diversification n'existe donc plus et l'utilisateur de ces deux méthodes doit en être conscient.

Un moyen très simple de diversifier la recherche peut consister à re-exécuter un des algorithmes en prenant un autre point de départ. Comme l'exécution de ces méthodes est souvent très rapide, on peut alors inclure cette répétition au sein d'une boucle générale. On obtient alors un algorithme de type "Multi-start descent" décrit par l'algorithme.

1. *initialise* : find an initial solution $x,k \longleftarrow 1$, $f(B) \longleftarrow +\infty$
2. **repeat**
3. *Starting point* : choose an initial solution x_0 atrandom
4. x result of simple Descent or Deepestdescent
5. **if** $f(x) < f(B)$ **then**
6. $B \longleftarrow x$
7. **end if**
8. $k \longleftarrow 1$
9. **until** stoppingcriterionsatisfied

Algorithme 2-3 : L'algorithme de la méthode de descente avec relance

De manière évidente, la diversification est totalement absente des algorithmes 2.1 et 2.2. En ne conservant que l'aspect intensification, la convergence est souvent trop rapide et on se trouve très rapidement bloqué dans un optimum local. Les résultats communément admis indiquent que ces techniques conduisent en général à des solutions en moyenne à 20% de l'optimum. Dans le cas de l'algorithme2.3, la diversification est simplement insérée par le choix aléatoire d'une solution de départ.[17]

2.3.2.2. La recherche tabou (TS : TabuSearch)

La recherche taboue est une métaheuristique originalement développée par Glover, 1986 et indépendamment par Hansen, 1986. Elle est basée sur des idées simples, mais elle est néanmoins très efficace. Cette méthode combine une procédure de recherche locale avec un certain nombre de règles et de mécanismes permettant à celle-ci de surmonter l'obstacle des optima locaux, tout en évitant de cycler. Elle a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire : problèmes de routage de véhicule, problèmes d'ordonnancement, problèmes de coloration de graphes, etc.

Dans une première phase, la méthode de recherche tabou peut être vue comme une généralisation des méthodes d'amélioration locales. En effet, en partant d'une solution quelconque appartenant à l'ensemble de solutions, on se déplace vers une solution ' située dans le voisinage (). Donc l'algorithme explore itérativement l'espace de solutions.

Afin de choisir le meilleur voisin' dans (), l'algorithme évalue la fonction objective en chaque point ', et retient le voisin qui améliore la valeur de la fonction objective, ou au pire celui qui la dégrade le moins.[18]

1. *initialise* : find an initial solution *x*
2. **repeat**
3. *neighbourhoodsearch* : *find a solution x´€ N*ˣ(*x*)
4. *update memory* : tabulist, frequency-based memory, aspiration level, …
5. *movexx´*⟵
6. **until**stoppingcriterionsatisfied

Algorithme 2-4 : l'algorithme de recherche tabou classique

2.3.3. Métaheuristiques à base de population

Contrairement aux algorithmes partant d'une solution singulière, les métaheuristiques à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions .On distingue dans cette catégorie, les algorithmes évolutionnaires, qui sont une famille d'algorithmes issus de la théorie de l'évolution par la sélection naturelle, énoncée par CharlesDarwin 20 et les algorithmes d'intelligence en essaim qui, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels.

2.3.3.1. L'algorithme de la recherche coucou

La recherche coucou a été proposée en 2009 par Yang et Deb sous le nom de CuckooSearch (CS) [13]. Elle s'inspire du comportement de reproduction d'une espèce spéciale d'oiseaux parasites de nids appelés « Coucous ».

Dans l'algorithme de la recherche coucou, une solution possible est appelée « nid » ou « coucou ». En fait, la recherche coucou part du principe que chaque nid comporte un seul coucou. Au cours du processus de la recherche de l'algorithme CS, chaque coucou crée son propre poussin en fonction de sa représentation actuelle (le coucou lui-même) et du vol de Lévy. L'évaluation de la qualité du poussin et de son père permet de sélectionner lequel d'entre eux survivra et subira quelques modifications dans le but d'améliorer sa qualité.[19]

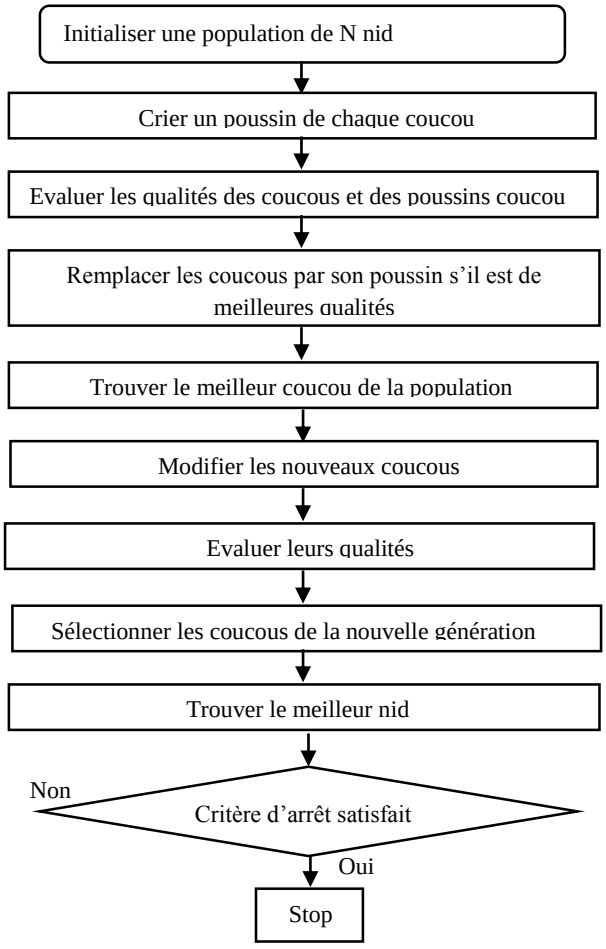

Figure 2-1 : Organigramme décrit l'algorithme de la recherche coucou

2.3.3.2. Les algorithmes Génétiques (GeneticAlgorithms)

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation stochastiques basés sur les mécanismes de la sélection naturelle et de la génétique. Leur fonctionnement est simple, on part d'une population de solutions potentielles (chromosomes) initiales choisies aléatoirement, on évalue ensuite leurs performances (fitness). A travers les résultats obtenus, on crée une nouvelle population de solutions potentielles en se basant sur trois opérations simples :

- La sélection
- Le croisement
- La mutation

Quelques individus se produisent, d'autres disparaissent et seuls les individus les mieux adaptés sont supposés survivre

On recommence ce cycle jusqu'à ce qu'on trouve une solution satisfaisante. L'héritage génétique à travers les générations permet à la population d'être adaptée et don répondre aux critères d'optimisation

La figure suivante montre un organigramme décrivant le fonctionnement d'un algorithme génétique. [20]

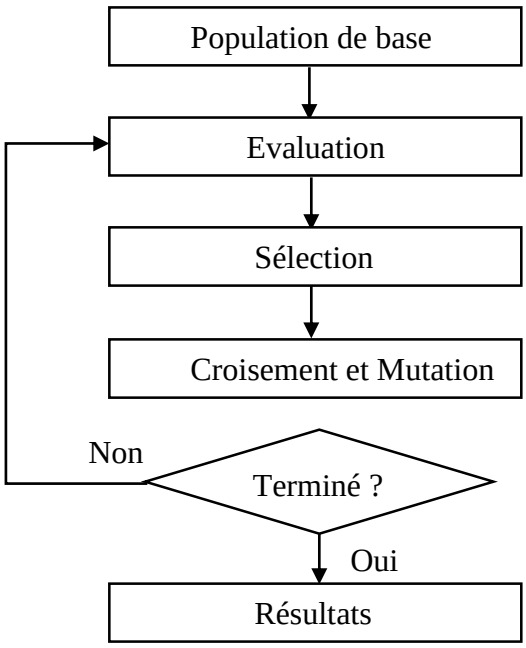

Figure 2-2 : Fonctionnement d'un algorithme génétique

2.4. L'algorithme de chauve-souris (Bat Algorithm)

L'algorithme des chauves-souris (Bat Algorithm: BA) proposé par Xin Yang en 2010, c'est une approche inspirée par le comportement d'écholocation des chauves-souris, avec différents taux d'émission et d'intensité d'impulsions.

L'algorithme standard de chauve-souris présente de nombreux avantages L'un d'entre eux est qu'il peut obtenir une convergence rapide aux étapes initiales en passant de l'exploration à l'exploitation. Cela en fait un algorithme efficace lorsqu'une solution rapide est nécessaire. Afin d'améliorer les performances, de nombreuses modifications ont été ajoutées pour augmenter la diversité de la solution et pour améliorer les performances de l'algorithme de chauve-souris standard.

2.4.1. L'écholocation des chauves-souris

Les chauves-souris sont les seuls mammifères avec des ailes. Il existe deux types de chauves-souris : les Mégachiroptères (méga-chauvesouris) - et les Microchiroptères (micro-chauves-souris) [14].

Les microchiroptères utilisent un sonar biologique, appelé écholocation en raison de leur manque de capacité de vision pour identifier et localiser leurs proies, les obstacles...

Elles peuvent grâce à ce bruit émis détecter la distance et distinguer la différence entre les aliments et les obstacles environnementaux même dans l'obscurité complète, contrairement aux mégachiroptères qui ont une excellente vision.

L'écholocalisation est réalisée grâce à la présence d'une forme mutée de la protéineprestine au sein de l'oreille, permettant ainsi la perception des ultrasons. Lorsqu'une chauve-souris émet des ultrasons par sa gueule ou son nez, ceux-ci sont réfléchis par les différents obstacles puis sont captés par les oreilles (Figure 2.7). Grâce à ces ultrasons, la chauve-souris est capable de reconstituer un modèle 3D fidèle de son environnement.

La plupart des micros chauve-souris modulent les ultrasons lors de ses déplacements en fonction de ses propres mouvements, de la stratégie de chasse suivie, de la distance de ses proies et de l'environnement. Cette modulation consiste à adapter la puissance, la fréquence et le rythme des rafales d'ultrasons qu'elles émettent afin d'obtenir une grande précision lui permettant de s'adapter efficacement. La modélisation du comportement de chasse par écholocalisation de ces micros chauves-souris a conduit à la création de l'algorithme des chauves-souris (Bat Algorithm) par X.-S. Yang.

Cet algorithme s'appuie sur l'hypothèse que seule l'écholocalisation est utilisée pour la chasse des proies, la perception des distances et la perception de l'environnement.

On considère que les chauves-souris se déplacent en volant et que les solutions de l'espace de recherche S sont des positions de l'espace. A chaque instant t, chacune des N chauves-souris possède une position dans l'espace Xi et une vitesse Vi. Au cours de ses déplacements, chaque chauve-souris émet des ultrasons avec une puissance Ai ϵ [Amin ; Amax] à une fréquence fi ϵ [fmin ; fmax] (voir figure II.8). Les émissions d'ultrasons s'effectuent en rafale selon le taux d'impulsions ri ϵ [0 ; 1]. Lorsque la proie est proche de la chauve-souris, elle émet plus fréquemment des ultrasons avec une puissance faible (ri grand et Ai petit).

Inversement, lorsque la proie est éloignée, les émissions sont moins fréquentes mais plus puissante pour permettre d'appréhender des cibles éloignées. Sur la base de la description et les caractéristiques d'écholocation de chauve-souris ci-dessus, Xin-She Yang en 2010 a développé l'algorithme de chauve-souris avec les trois règles idéalisées suivantes :

1. Toutes les chauves-souris utilisent l'écholocation pour détecter la distance, et ils peuvent aussi « avoir » la différence entre alimentaires / proies et les obstacles en quelque sorte magique (voir figure II.3) ;

2. Les chauves-souris volent au hasard avec une vitesse Vi à la position Xi avec un fmin de fréquence, variant la longueur d'onde λ et le volume A0 à la recherche d'une proie.

Ils peuvent ajuster automatiquement la longueur d'onde (ou fréquence) des impulsions émises et ajuster le taux d'émission d'impulsions r ϵ [0, 1], en fonction de la proximité de la cible.

3. Bien que le volume puisse varier de plusieurs manières, nous supposons que le volume varie à partir d'un grand A0 (positive) à une valeur constante minimale Amin.[16]

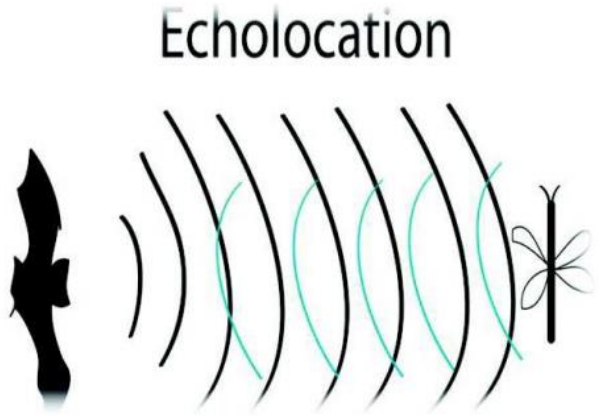

Figure 2-3 : L'écholocation

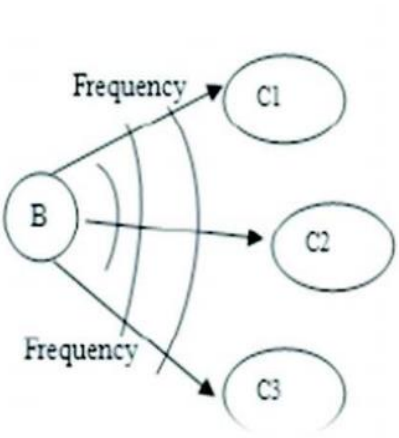

Figure 2-4: Bat envoie un signal à une fréquence f

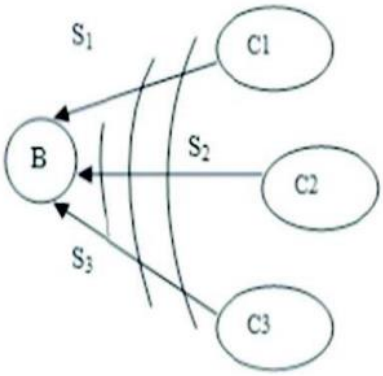

Figure 2-5 : Signal d'écho utilisé pour calculer la distance S

2.4.2. Comportement des chauves-souris

Les chauves-souris qui utilisent l'écholocation ont la capacité de déterminer l'environnement autour d'eux, ils peuvent détecter la distance et l'orientation de la cible (proie) ainsi que l'emplacement des obstacles.

La nature et le rythme, le temps de retard de la réponse, le volume et la différence de temps entre les deux oreilles de la chauve-souris lui permettent de dessiner l'image 3D de l'environnement dans son cerveau.

Les chauves-souris volent au hasard avec une vitesse vi à la position xi avec une fréquence f min fixe, et font varier la longueur d'onde ƛ, la fréquence f et le volume A0 lors de la recherche d'une proie.

- Si aucune proie n'est dans la portée de la chauve-souris, la valeur de la longueur d'onde est augmentée, et la fréquence est diminuée.
- Si la proie est proche de la chauve-souris, la longueur d'onde est réduite, et la fréquence est augmentée pour mesurer précisément emplacement de la proie et la suivre plus rapidement [14].

Les chauves-souris peuvent automatiquement régler la longueur d'onde (ou fréquence) de leurs impulsions émises et ajuster le taux d'émission de l'impulsion r ϵ [0, 1] qui varie en fonction de la proximité de la cible [15].

2.4.3. Algorithme BA

2.4.3.1. Mouvement des chauves-souris

Le déplacement des chauves-souris virtuelles permet de générer de nouvelles solutions, on doit définir les positions Xi et les vitesses Vi dans un espace de recherche.

Les nouvelles solutions X^t_i et les nouvelles vitesses V^t_i au temps t sont données selon les équations suivantes [16] :

$f_i = f_{min} + (f_{max} - f_{min}) \beta$ où: $\beta \epsilon [0,1]$ est un vecteur aléatoire

$V^t_i = V^{t-1}_i + (X^t_i - X_*) f_i$

$X^t_i = X^{t-1}_i - V^t_i$

Tel que,

fi est la fréquence d'émission de pulsation de la chauve-souris Xi, et qui appartient à la gamme [fmin ; fmax] correspondant à la gamme de longueurs d'ondes [λmin; λmax].

Par exemple la gamme de fréquence [20KHz ; 500KHz] correspond à la longueur d'ondes [0.7mm ; 17 mm]. Afin de simplifier l'implémentation, il a été supposé que f ϵ [0 ; fmax]

Sachant que les hautes fréquences correspondent aux courtes longueurs d'ondes. Par conséquent le taux d'émission de pulsations peut être de la gamme [0 ; 1] ou 0 signifie qu'il n'a aucune pulsation, et 1 signifie le taux maximal d'émission de pulsations.

X* est la meilleure location (solution) courante globale, qui sera calculée par comparaison de toutes les solutions obtenues par chacune des n chauves-souris.

Sur la base de ces approximations l'algorithme de chauve-souris est décrit comme Suit :

Fonction objective f(X), X=$(X_1, \dots \dots, X_d)^T$

Initialiser la population des chauves-souris X_i et V_i i= (1, 2, …, n)

Définir la fréquence d'impulsions f_i à la position X_i

Initialiser le taux de pulsations ri et l'intensité A_i

Tant que(t < Nombre max d'itérations)

{Générer de nouvelles solutions en ajustant la fréquence, et mettre à jour les vitesses et emplacements/solutions

Si(rand> ri)

Choisir une solution parmi les meilleures

Générer une solution locale autour de la meilleure solution choisie

Fin si

Générer une nouvelle solution en volant aléatoirement

Si (rand <A_i) && (f (X_i) <f (X^*))

Accepter les nouvelles solutions

Augmenter ri et réduire A_i

Fin si

Classer les chauves-souris et trouver le meilleur X^* courant

}

Fin tant que

Algorithme 2-5 : Algorithme de chauve-souris

2.5. Conclusion

Les méthodes méta-heuristiques permettent la résolution de nombreux problèmes complexes en cherchant des solutions optimales à partir de solutions aléatoires.

Ces méthodes sont en évolution permanente, souvent, de nouvelles améliorations leurs sont apportées, elles deviennent cependant de plus en plus utilisées et maitrisées.

Dans ce qui suit, nous allons adapter notre algorithme (BA) à un problème d'atelier flow-shop, nous allons par la suite faire une étude sur cet algorithme dans différents modes de fonctionnement et nous discuterons les résultats obtenus

Chapitre 3

Adaptation de l'algorithme de chauve-souris (Bat Algorithm) et résultats de simulation

3. Introduction

Les métaheuristiques sont des méthodes qui forment une famille d'algorithmes d'optimisation combinatoire, visant à résoudre les problèmes de type NP-difficiles pour lesquels on ne connait pas de méthodes classiques plus efficaces.

Dans ce chapitre nous nous intéressons à résoudre le problème d'ordonnancement dans un système de production de type flow shop avec l'aide du métaheuristique proposé l'algorithme de chauve-souris, pour faire ça nous commence par la présentation et description du problème considéré, ensuite l'adaptation de l'algorithme de chauve-souris avec les résultats de simulation et comparaison avec l'optimum de ces exemples pour des petite instances.

3.1. Description de Problème étudier d'atelier de type flow shop

Un atelier Flow Shop peut être définit par un ensemble de jobs (produits) (J1,…, Jn) qui doivent être traités sur un ensemble de machines (m1,…, mm). Chaque job Ji (i=1, …, n) doit être traité sur toutes les machines mj (j=1,…, m) durant un temps de traitement noté Pij.

Certaines contraintes sont à considérer :

- Il n'y a aucune contrainte de précédence entre les jobs,
- Chaque machine mj ne peut traiter qu'un seul job Ji à la fois,
- Chaque job Ji ne peut être traité que par une seule machine mj à la fois,
- Le temps de déplacement des jobs entre les machines est négligeable,
- Les temps de traitement sont connus et leurs valeurs sont fixées,

Dans cette étude nous s'intéressons a la minimisation du temps d'exécution maximale makespan Cmax (Cmax =max {Cj ; j=1, ……, n}, où Cj est la date de fin d'exécution de la tache. La formulation mathématique du problème est la suivante :

f=min(Cmax) (1)

Cmax=max{ Cij } pour i=1,...,n; j=1,...,m (2)

Cij = Sij + Pij pour i=1,...,n; j=1,...,m (3)

n: le nombre de jobs.

m: le nombre de machines.

Sij: le temps de début de traitement du job Ji sur la machine mj.

Pij: le temps de traitement du job Ji sur la machine mj.

Cij: la date de fin de traitement du job Ji sur la machine m[21].

3.2. Adaptation de l'algorithme de chauve-souris

Le nombre de proies à une influence directe sur l'algorithme de chauve-souris, dans notre problème une proie représente une solution (une séquence de job) pour cela il est très important de bien choisir ce paramètre pour garantir une meilleure qualité de la solution.

Nous avons considéré les chauves-souris comme étant des procédures de recherche et les proies comme étant des solutions à rechercher.

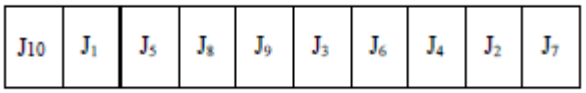

Figure 3-1: exemple d'une solution (une séquence de job)

A l'opposé des autres métaheuristiques qui s'inspirent des phénomènes naturels, l'algorithme de chauve-souris s'inspire par le comportement d'écholocation des chauves-souris. Les étapes du processus de recherche de l'algorithme de chauve-souris sont résumées dans l'algorithme suivant :

1-Générer un ensemble de solution des proies

2-fixer le nombre des chauves-souris

3-Evaluation des xi Cmax

4-Calculer des fréquences pour chaque chauvesouris

Un nombre $\beta \in [0,1]$ tel que $\beta \in [0,0.1,\ldots,1]$ (pas de 0.1)

Tant que le critère d'arrêt n'est pas atteint faire :

Pour chaque chauvesouris

Sélectionner et Examiner le voisinage de xi

Si xi actuelle est la meilleure on actualise f_i selon β

Sélectionner une solution parmi les meilleurs

Générer une solution locale auteur de la solution sélectionné x*

Fin si

Si la solution générée est meilleure alors

Accepter la solution

Mettre à jour ri

Fin si

Fin pour

Fin tant que

5-Enregistrer la meilleure solution

Algorithme 3-1 : l'algorithme de chauve-souris adapté

3.3. Résultats et interprétations

Pour cette section nous avons effectué plusieurs simulations sur un nombre de classes des problèmes flow shop pour confirmer notre adaptation. Pour chaque classe nous considérons des exemples différents (les temps de traitement de pièces diffèrent d'un exemple à l'autre et sont pris dans un intervalle de [1 :31]).

- **L'étude de sensibilité sur la taille de population (nombre de proies)**

Dans cette étude tous les paramètre de l'algorithme sont fixés (taux de pulsation ri=0.5, volume A=0.5, nombre de chauve-souris =20) et nous avons varié la taille de population de 10 à 40.

	Tpop=10	**Tpop=20**	**Tpop=30**	**Tpop=40**
Exemple 1	753	**714**	725	733
Exemple 2	789	776	**756**	762
Exemple 3	767	736	758	**732**
Exemple 4	787	779	784	**741**

Tableau 3-1 : Résultats de simulation de l'algorithme chauve souri pour différente valeur du taille de population avec 10 jobs 30 machines.

Nous remarquons dans les résultats du tableau 3-1, que le Cmax minimum est atteint pour une taille de population égale à 20 pour l'exemple 1, pour une taille de population égale à 30 pour l'exemple 2 et pour une taille de population égale à 40 pour l'exemple 3 et l'exemple 4.

	Tpop=10	**Tpop=20**	**Tpop=30**	**Tpop=40**
Exemple 1	1057	1058	1058	**1054**
Exemple 2	1064	**1050**	1070	1059
Exemple 3	1018	**1009**	1022	1022
Exemple 4	1019	1029	1032	**1017**

Tableau 3-2 : Résultats de simulation de l'algorithme chauve souri pour différente valeur du taille de population avec 20 jobs 30 machines.

Nous remarquons dans les résultats du tableau 3-2, que le Cmax minimum est atteint pour une taille de population égale à 20 pour les exemples 2 et 3 et pour une taille de population égale à 40 pour l'exemple 1 et l'exemple 4.

Les résultats montrent que le Cmax minimum est atteint pour une taille de population égale à 20 (2 fois), pour une taille de population égale à 30 (1 fois) et pour une taille de population égale à 40 (4 fois).

Nous allons garder tous les paramètres précédents de l'algorithme (taux de pulsation ri=0.5, volume A=0.5) et nous avons changé la valeur de nombre de chauve-souris à30 et nous avons varié la taille de population de 10 à 40.

	Tpop=10	**Tpop=20**	**Tpop=30**	**Tpop=40**
Exemple 1	719	745	737	**715**
Exemple 2	778	776	**726**	759
Exemple 3	754	743	763	**720**
Exemple 4	775	785	776	**773**

Tableau 3-3 : Résultats de simulation de l'algorithme pour différente taille de population avec 10 jobs 30 machines.

Nous remarquons dans les résultats du tableau 3-3, que le Cmax minimum est atteint pour une taille de population égale à 30 pour l'exemple 2 et pour une taille de population égale à 40 pour l'exemple 1, l'exemple 3 et l'exemple 4.

	Tpop=10	**Tpop=20**	**Tpop=30**	**Tpop=40**
Exemple 1	1061	1059	1055	**1048**
Exemple 2	1055	1052	1067	**1050**
Exemple 3	**996**	1009	1013	1001
Exemple 4	1137	1052	1030	**1012**

Tableau 3-4 : Résultats de simulation de l'algorithme pour différente taille de population avec 20 jobs 30 machines.

Nous remarquons dans les résultats du tableau 3-4, que le Cmax minimum est atteint pour une taille de population égale à 10 pour l'exemple 3 et pour une taille de population égale à 40 pour les exemples 1,2 et 4.

Les résultats montrent que le Cmax minimum est atteint pour une taille de population égale à 10 (1 fois), pour une taille de population égale à 30 (1 fois) et pour une taille de population égale à 40 (6 fois).

En gros Nous remarquons dans cette étude que l'algorithme de chauve-souris donne des solutions de bonne qualité pour une taille de population égale à 40 (Tpop=40) et un nombre de chauve-souris égale à 30.

3.4. Comparaison des résultats entre l'algorithme de chauve-souris et l'optimum

Dans cette section, nous avons pris les meilleurs résultats de simulation du Cmax pour cette technique et qui sont obtenus avec les paramètres suivantes : taille de population égale à 40 et le nombre de chauve-souris égale à 30 et faire la comparaison avec l'optimum pour l'exemple de simulation de 10 jobs et 30 machines.

		Algorithme BA	Optimum
10 jobs 30 machines Tpop=40 Chauve-souris=30	**Exemple 1**	715	683
	Exemple 2	759	753
	Exemple 3	720	698
	Exemple 4	773	741

Tableau 3-5 : Comparaison entre les résultats de simulation avec l'algorithme de chauve-souris et l'optimum du l'exemple (10jobs, 30 machines).

Nous remarquons dans le tableau 3-5 que les résultats du Cmax obtenus avec l’algorithme chauve-souris sont proches de l’optimum, donc on peut dire que cette méthode est efficace et donne des solutions de bonne qualité.

3.5. Conclusion

Dans ce chapitre, nous nous sommes intéressés à l‘adaptation de l’algorithme de chauve-souris pour la résolution d’un problème d’ordonnancement d’atelier flow shop dont l’objectif la minimisation de makespan. La simulation a été effectué sur des différent classes des problèmes pour étudier l'impact des paramètres sur les performances de l‘algorithme de chauve-souris

Conclusion Générale

Conclusion générale

L'ordonnancement repose essentiellement sur la résolution de problèmes d'optimisation combinatoires. Ces problèmes consistent à rechercher, dans un ensemble de solutions possibles, une solution particulièrement intéressante au regard d'un ou de plusieurs critères. Ces problèmes sont pour la plupart NP-complets.

L'environnement de production Flow-Shop à une grande pertinence en ingénierie, représentant près du quart des systèmes de production, ceci nous a motivés à considérer les contraintes de ressources non-renouvelables dans cet environnement. Dans cette optique, nos travaux sont articulés autour des problèmes d'ordonnancement Flow-Shop pour la minimisation du makespan.

Parmi la multitude de métaheuristiques pouvant être utilisées, nous avons opté pour l'algorithme des chauves-souris. Cet algorithme modélise le comportement d'écholocation chez les chauves-souris.

Dans ce mémoire, nous nous sommes, intéressés à l'adaptation de l'algorithme de chauve souri qui est s'inspiré de comportement d'écholocation des chauves-souris.

Après l'adaptation algorithme est utilisé pour la résolution du problème d'ordonnancement dans un atelier Flow Shop ayant pour objectif la minimisation du makespan.

En premier temps nous avons effectué une étude de sensibilité sue les paramètres de l'algorithme. Ensuite nous avons fait une comparaison entre les résultats obtenu par l'algorithme chauve souri est la valeur optimale. La comparaison montre que l'algorithme chauve souri donné des solutions de bonne qualité.

Les perspectives :

Plusieurs perspectives sont ouvertes pour ce travail :

Au niveau de l'algorithme chauve souri : hybrider l'algorithme API avec une procédure de recherche locale lors de la recherche de nouvelle solution.

Faire une comparaison entre l'algorithme chauve souri et un autre algorithme tel que l'algorithme API (Pachycondyla apicalis)

Annexes

Exemple de simulation 10jobx30machines

[9 9 25 17 18 29 13 10 17 20 18 3 17 21 9 11 20 10 26 13 30 25 26 14 7 6 1 4 13 18 ;

27 18 5 3 10 3 6 19 5 17 16 1 1 25 3 29 29 15 26 29 7 15 24 10 2 26 9 11 16 7 ;

24 24 14 21 31 26 10 13 1 8 7 2 9 16 17 9 8 29 16 6 15 8 30 13 31 20 19 25 16 7 ;

20 6 6 11 9 16 23 2 29 5 22 9 18 12 4 25 15 18 10 31 15 18 21 9 19 22 16 10 21 13 ;

2 29 20 9 21 12 7 27 1 9 25 7 1 17 16 5 1 8 1 22 2 26 24 13 1 22 13 26 19 24 ;

2 18 29 3 1 4 17 28 13 20 2 30 15 14 5 14 16 17 4 14 18 3 16 17 20 4 22 17 13 8 ;

12 6 22 30 15 28 1 9 27 1 13 4 29 31 24 31 7 27 17 8 4 12 4 7 27 21 30 8 2 20 ;

29 7 7 3 30 17 8 21 2 24 18 23 15 27 15 20 17 23 14 29 12 7 22 7 28 26 9 4 20 20 ;

29 14 23 9 9 7 2 9 6 25 4 1 24 4 7 11 20 3 17 30 1 28 7 28 2 3 4 16 17 13 ;

7 5 12 21 6 1 4 7 6 7 24 19 15 18 23 29 24 25 6 8 17 24 22 24 16 21 19 7 19 16]

La séquence optimale est : [10, 6, 4, 5, 3, 2, 7, 8, 9, 1]

Le makespan optimale Cmax =683

Références bibliographiques

[1] Bahmani Y., Optimisation multicritère de l'ordonnancement des activités de la production et de la maintenance intégrées dans un atelier Job Shop, Thèse Doctorat en science, Université de Batna-II, Algérie 2017.

[2] Group Gotha, Les problèmes d'ordonnancement, RAIRO. Recherche opérationnelle, tom 27, n°1(1993).

[3] Meziane M. E. A., Optimisation par phases pour les problèmes d'ordonnancement des ateliers de type job-shop totalement flexibles, Mémoire de magister en des sciences, Université d'Oran, Algérie 2011.

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

[5] Mémoire Magister Houbad-2011- Université Abou Bakr Belkaid –Tlemcen –

Faculté de Technologie.

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

[7] Baptiste M., Méthodes approchées pour la résolution d'un problème d'ordonnancement avec travaux interférant, Rapport final de PFE, Université de Tours, France 2014.

[8] Sevaux, M., (2004). Métaheuristiques Stratégie pour l'optimisation de la production de biens et de services, Habilitation à diriger des recherches, Laboratoire d'Automatique, de Mécanique d'informatique Industrielles et Humaines du CNRS (UMR CNRS 8530) dans l'équipe systèmes de production.

[9] Nicholson, T. A. J., (1971). Optimization in industry, Vol. 1, Optimization Techniques. London : Longman Press.

[10] Metaheuristics Network Websitehttp://www.metaheuristics.net visité en novembre 2004.

[11] I.Driss, K.N.Mouss, A.Laggoun . A New geneticalgorithm for flexible job shop schedulingproblem. Journal of Mechanical Science and Technology. 29 (3) (2015) 1273-1281.

[12] Duvivier, D., Etude de l'hybridation des métaheuristiques, application à un problème d'ordonnancement de type jobshop, Thèse de doctorat, Laboratoire d'Informatique du Littoral, Cote-d'Opale, Calais. (2000).

[13]M. Dorigo et L.M. Gambardella. Ant colony system : à cooperativelearningapproach to the travelling salesmanproblem. IEEE Transactions on Evolutionary Computation. 1(1997). 53-66.

[14] Amr Rekaby, "DirectedArtificial Bat Algorithm (DABA)", A new Bio-InspiredAlgorithm, EgyptianResearch and Scientific Innovation Lab (ERSIL), Cairo, Egypt, 2013.

[15] AbdelliOuardia, "Segmentation d'images par seuillage d'histogrammesbidimensionnels", Mémoire de Magister, Université de TIZI-OUZOU, 2011.

[16] Xin-She Yang, "A new metaheuristic Bat-InspiredAlgorithm", Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK, 2010.

[17]Ordonnancement d'un flow-shop par métaheuristique hybride -2017- Université Abou BekrBelkaid – Tlemcen.

[18] Fred Glover and Manuel Laguna, 1997, Tabusearch, KluwerAcademic Publishers.

[19] Ordonnancement d'un flow-shop par métaheuristique hybride -2017- Université Abou BekrBelkaid – Tlemcen.

[20] Allocation de ressources dans un réseau de radio cognitive en se basant sur les méta-heuristiques : Bat InspiredAlgorithm et Bee ColonyAlgorithm-2014- Université Abou BekrBelkaid – Tlemcen.

[21] Ordonnancement des systèmes flexibles de production basé sur les métaheuristiques hybrides -2019- Université Abou BekrBelkaid – Tlemcen.

[23] http://dspace.univ-msila.dz:8080/xmlui/bitstream/handle/123456789/5028/Adjiri%20Hadjer.pdf?sequence=1&isAllowed=y

[24] https://fr.wikipedia.org/wiki/Th%C3%A9orie_de_l%27ordonnancement

[25] https://docplayer.fr/139977274-faculte-des-sciences-departement-d-informatique.html

[26] J. R. Slagle. Artificial intelligence: The heuristic programming approach. McGraw-Hill.pp. 3. New York, 1971

Résumé :

Les problèmes d'ordonnancement sont souvent classés NP-Difficiles. Leur résolution nécessite des méthodes dédiées à leur degré de complexité ; pour cette raison plusieurs heuristiques et métaheuristiques ont été conçues.

Le problème traité dans ce mémoire concerne l'ordonnancement d'opérations sur des machines dans un atelier de type flow-shop en vue de minimiser le makespan.

Une méthode a été suggérée basée sur l'algorithme de chauve-souris, et des modifications ont été apportées sur ce dernier afin de convenir à la résolution de notre problème d'ordonnancement.

Mots clés : Algorithme de chauve-souris, Ordonnancement, Métaheuristiques, le makespan, Flow shop

Abstract:

Scheduling problems are often NP-Hard combinatorial optimization problems. Their resolution requires methods dedicated to their complexity degree; for this reason, several heuristics and metaheuristics have been designed.

The problem addressed in this thesis concerns the scheduling of operations on machines in à Flow Shop workshop in order to minimize the makespan.

A method was suggested based on the bat algorithm, and modifications have been brought on it in order to suit solving our problem.

Keywords: Bat Algorithm, Scheduling, Metaheuristics, Makespan, Flow shop