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

Mémoire de Magister en Automatique
Option : Modélisation et commande des systèmes

Thème

Planification et Ordonnancement en temps réel d'un Job shop en utilisant l'Intelligence Artificielle

Présenté le : ***Lundi 02Juillet 2012***

par :

HOUARI Habiba

Devant le Jury :

Président :

CHEIKH Abdelmadjid	Professeur	Université de Tlemcen, Algérie.

Examinateurs :

CHIKH Mohamed Amine	Professeur	Université de Tlemcen, Algérie.
HASSAM Ahmed	Maître assistant A	Université de Tlemcen, Algérie.

Encadreur :

SARI Zaki	Professeur	Université de Tlemcen, Algérie.

Co-encadreur :

SOUIER Mehdi	Maître assistant B	EP-ECG, Tlemcen, Algérie.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

LISTE DES FIGURES

Remerciements

En préambule à ce mémoire, je souhaite adresser ici tous mes remerciements aux personnes qui m'ont apporté leur aide et qui ont ainsi contribué à l'élaboration de ce mémoire.

Je tiens à remercier avant tout monsieur SARI Zaki, professeur à l'UABB et Directeur de Laboratoire de Productique de Tlemcen MELT, pour avoir accepté de diriger ce mémoire et pour les conseils et avis pertinents qu'il a pu me donner tout au long de mes deux années de mémoire.

Je remercie vivement mon Co-encadreur :
Monsieur SOUIER Mehdi Maître assistant à l'école préparatoire des sciences économiques et de gestion de Tlemcen pour son aide, sa disponibilité, ses remarques et ses suggestions durant tout mon travail.

Je tiens bien évidemment à exprimer ma reconnaissance aux membres du jury :

Monsieur CHEIKH Abdelmadjid, Professeur à l'Université de Tlemcen, d'avoir bien voulu présider ce jury.

Monsieur CHIKH Mohamed Amine, Professeur à l'Université de Tlemcen, qui m'a fait l'honneur d'examiner ce travail.

Monsieur HASSAM Ahmed, Maître assistant à l'Université de Tlemcen, qui a accepté d'examiner ce travail, et de participer au jury. Je le remercie pour ces conseils.

Je remercie tous les membres de l'équipe productique, particulièrement Mademoiselle HOUBAD Yamin , pour sa collaboration dans la réalisation de ce travail.

Grâce à eux, j'ai pu explorer le monde de la recherche scientifique et rencontrer des chercheurs étrangers, ce qui est un atout et une chance exceptionnelle, et je leur en suis en particulièrement reconnaissante.

Mes derniers mots seront pour ma famille : mon marie, mes enfants, mes parents, mes frères et mes sœurs, qui ont toujours eu confiance en moi.

INTRODUCTION GENERALE

Accroître la productivité en réduisant les coûts est, aujourd'hui, un objectif majeur dans toutes les entreprises. Les systèmes de production sont caractérisés par leur dynamique et leur imprévision, les tâches souvent de caractère complexe sont soumises à des contraintes de temps et d'exigence. Alors, **le système de production** se donne de nouveaux objectifs à atteindre.

Pour cela les entreprises doivent faire face à l'augmentation de la concurrence, à la pression de plus en plus forte de leur environnement (clients et concurrents) pour un renouvellement rapide des produits, mais aussi une augmentation de l'offre vis à vis de la demande. Les systèmes de production et de fabrication actuels offrent une grande **flexibilité.**
Dans le domaine de l'industrie où la concurrence est féroce, les grandes unités de production deviennent de moins en moins compétitives, parce que très peu adaptables aux variations du marché. Ces unités disparaissent progressivement laissant place à de nouvelles structures industrielles aux concepts différents : **les systèmes flexibles de production** (Flexible Manufacturing Systems « FMS »). Ces systèmes sont conçus pour produire en petites et moyennes quantités, à des coûts minimums, une variété de produits avec des temps de préparation des machines et de changement d'outils minimums. Leurs structures leur permettent de produire une très large gamme de produits et donc d'être moins sujet aux crashs économiques.

Un système de production flexible a pour but d'aboutir, non seulement, à une productivité importante, mais aussi à une grande flexibilité de production lui permettant de suivre les variations du marché.

Le problème de sélection du routage optimal consiste à déterminer le meilleur routage tout en minimisant le coût de production en prenant en considération le volume de production, les alternatives associées au processus de mise en production, ainsi que de la disponibilité et la capacité des machines.

L'informatique permet d'avoir un outil d'aide à la décision et aussi aux planifications de la production. Cette planification est d'autant plus efficace que l'algorithme **d'ordonnancement** qu'elle utilise est performant. Le but étant d'atteindre un ordonnancement optimal qui répartit au mieux la charge du travail et tient compte des diverses contraintes de production.

Les problèmes d'ordonnancement sont très variés. On peut les rencontrer dans de très nombreux domaines : les systèmes industriels de production (activités des ateliers en gestion de production et problèmes de logistique), les systèmes informatiques (les tâches sont les programmes et les ressources sont les processeurs, la mémoire...), les systèmes administratifs (gestion du personnel, emplois du temps,…), les systèmes de transport, la construction, … etc. C'est pour cette raison qu'ils ont fait et continuent de faire l'objet de nombreux travaux de recherche.

Dans un système de production, le problème d'ordonnancement consiste à organiser dans le temps l'exécution d'opérations interdépendantes à l'aide de ressources disponibles en quantités limitées pour réaliser un plan de production.

Les approches traditionnelles pour résoudre ce problème consistent à appliquer des techniques d'optimisation à une formulation analytique. Cependant, ces modèles ont montré leurs limites.

Même des formulations idéalistes et simplifiées peuvent s'avérer très difficiles à traiter. Pour trouver un optimum, le seul moyen est souvent de faire une recherche exhaustive sur l'ensemble des solutions réalisables. Le développement de la théorie de la complexité des algorithmes a permis de clarifier la difficulté du problème. La plupart des problèmes d'ordonnancement sont classés NP-difficiles. L'appartenance à la classe des problèmes de ce type signifie qu'on ne pourra probablement jamais résoudre ce problème d'une manière optimale. Il s'ensuit que la plupart des problèmes de taille industrielle (plusieurs milliers de tâches, ressources complexes et contraintes spécifiques) sont impossibles à résoudre de manière exacte.

Les chercheurs se sont alors orientés vers l'utilisation de méthodes approchées appelées **« heuristiques » ou « métaheuristique »**. L'objectif d'une heuristique est de trouver une « bonne » solution en un temps raisonnable.

Parmi les métaheuristiques, les algorithmes évolutionnaires sont des approches particulièrement intéressantes pour résoudre des problèmes d'optimisation. Ces approches sont basées sur la manipulation d'un groupe de solutions admissibles à chacune des étapes du processus de recherche.

Dans ce mémoire nous nous intéressons à l'adaptation des algorithmes de deux métaheuristiques à base de population pour la résolution du problème d'ordonnancement dans un système flexible de production (FMS).

La première métaheuristique est un algorithme mémétique avec gestion de population combinant un algorithme génétique avec une méthode de recherche locale (la méthode de descente) et se basant sur la gestion de la population pour contrôler la diversité d'une petite population de solutions de haute qualité. La deuxième est une métaheuristique basée sur l'évolution de la population, dites la recherche dispersée, l'algorithme est constitué essentiellement par des méthodes de base. Parmi ces méthodes, les méthodes de combinaison et les méthodes d'amélioration, qui sont liées au problème à traiter. Pour valider nos résultats, les deux métaheuristiques ont été programmées en Java et exécutées dans un Core ™ 2 Duo CPU avec 2.2 GHz et 2 Go de RAM.

Organisation du mémoire

Ce mémoire est composé de trois chapitres dont nous présentons une brève description dans les paragraphes suivants :

- Le **Chapitre 1** aborde les notions sur les systèmes flexibles de production, et une introduction aux problèmes d'ordonnancement. Nous rappelons brièvement quelques notions, ainsi que les différentes méthodes de résolution. Il n'apportent rien de nouveau, mais les notions données sont utiles pour la compréhension du sujet pour les non spécialistes.
- Le **Chapitre 2** est consacré aux métaheuristiques évolutionnaires de manière approfondie. Nous présentons d'une manière générale quelques algorithmes (les algorithmes génétiques,

les essaims particulaires, les colonies de fourmis) qui ont déjà été utilisés pour résoudre le problème d'ordonnancement du même système que nous étudions **[Souier 2009]**. Ensuite nous présentons les deux métaheuristiques que nous avons adapté pour la résolution de notre problème d'ordonnancement; l'algorithme mémétique avec gestion de population et l'algorithme de la recherche dispersée. Pour chaque métaheuristique présentée, nous donnons son origine et son algorithme de base.

- Le **Chapitre 3** est le dernier chapitre et il représente notre contribution proprement dite, deux cas de sélection de routages alternatifs en temps réel sont considérés. Nous présentons le FMS étudié pour tester nos algorithmes et l'adaptation de ces techniques pour résoudre le problème de sélection de routages alternatifs en temps réel ainsi que les résultats obtenus après plusieurs simulations avec et sans présence de pannes dans le système, leurs interprétations et la comparaison avec des algorithmes à base population (les algorithmes génétiques, les essaims particulaires et les colonies de fourmis) utilisées pour résoudre le même problème **[Souier 2009].**

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

Chapitre 1

Ordonnancement des systèmes flexibles de production

Ce chapitre présente les notions fondamentales concernant les systèmes de production, et particulièrement les systèmes flexibles de production Pour cela, nous rappelons la définition de flexibilité, nous étudions ensuite les éléments des systèmes flexibles de production et leurs différents types. La prochaine section de ce chapitre est consacrée aux problèmes d'ordonnancement d'ateliers, à savoir la définition d'un problème d'ordonnancement, les objectifs de l'ordonnancement, les problèmes d'ordonnancement, l'ordonnancement de type job shop, et les méthodes de résolution.

Sommaire

Les systèmes de production actuels ne cessent de croître en complexité. Cette complexité résulte pour une large part des exigences du marché, de la concurrence, de la qualité ainsi que de la densité et de la diversité des produits qu'ils traitent. Un exemple typique d'un système complexe, couramment utilisé, dans l'industrie est celui des structures flexibles en job-shop. Ce sont des systèmes capables de s'adapter à une possible évolution de l'environnement. Ils présentent une diversité importante des flux de produits avec des séquences quelconques de

production incluant des phases cycliques (opérations multiples d'un même flux sur une même ressource). L'objectif associé à ce type de systèmes est alors d'assurer un traitement le plus varié possible avec un maximum de productivité au moindre coût **[Tamani, 2008].**

Les systèmes de production actuels offrent une grande flexibilité, tels que les systèmes flexibles de production (FMS) qui fournissent des avantages divers comme l'augmentation d'utilisation des ressources, l'augmentation de la productivité, la réduction des encours…etc.

Les systèmes de production flexibles avec des ressources limités, des pannes de machines aléatoires ou des critères de production multiples ont un aspect qui explique que les problèmes d'ordonnancement dans ces systèmes sont généralement de type NP –difficile. C'est la raison pour laquelle il n'est pas possible de fournir un algorithme exact pour ce genre de systèmes. On s'intéresse alors naturellement à des algorithmes non exacts appelés 'heuristiques'.

Les méthodes approchées, dites aussi heuristiques ou encore métaheuristiques sont des méthodes empiriques qui donnent généralement de bons résultats sans pour autant être démontrables. Elles sont basées sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de ces approches est d'intégrer des stratégies de décision pour construire une solution proche de celle optimale tout en cherchant à avoir un temps de calcul raisonnable.

GÉNÉRALITÉS SUR LES SYSTEMES DE PRODUCTION

1.1 Les systèmes de production

1.1.1 Définitions

Un système de production est généralement vu comme l'association d'un ensemble de ressources en interaction pour réaliser une activité de production. En effet, la production s'effectue par une succession d'opérations dites de transformation, de transfert, d'assemblage et de désassemblage en exploitant les ressources disponibles (machines, moyens de transfert,…) afin de transformer les matières premières (composants entrant dans le système) en produits finis sortant de ce système **[Letouzey, 2001].**

Les systèmes de production ont été classés en trois grandes catégories **[Ammi, 2007]** :

- Les processus **continus** tel que la production électrique.
- Les processus **discrets** tel que l'usinage et toutes les activités d'assemblage,… etc.
- Les processus **discontinus** qui se situe à mi-chemin entre les processus continus et ceux discrets, les deux types de processus sont couplés : la production est continue mais il y a un conditionnement discret des produits**.**

1.1.2 Décomposition du système de production

Les systèmes de production peuvent être des systèmes très complexes et difficiles à gérer au vu de toutes leurs composantes fonctionnelles (fabrication, achat, distribution, maintenance...). Ils sont donc beaucoup étudiés, et ce depuis longtemps. Plusieurs approches ont été envisagées dans le but de mieux comprendre leur fonctionnement et de mieux les appréhender **[Tamani, 2008].**

L'application de la théorie des systèmes aux systèmes de production suggère une décomposition de ces derniers en trois sous-systèmes (Figure 1.1) :
– le système d'information,
– le système de décision,
– le système physique de production.

Le système physique de production est contrôlé par le **système de décision,** qui organise et coordonne les tâches en prenant des décisions basées sur les données transmises par le système d'information. Le rôle de ce dernier est de collecter, stocker, traiter et transmettre des données et des informations.

Le système d'information intervient entre les systèmes de production et de décision et à l'intérieur même de ce dernier, pour la gestion des informations utilisées lors de prise de décision.

Le système de gestion de production se constitue de l'association des parties des systèmes de décision et d'information si cela concerne uniquement la production.

Le **système physique de production** est constitué de ressources humaines et physiques, son rôle est de transformer les matières premières ou composantes en produits finis. C'est le système de gestion de production qui déclenche et vérifie ses activités **[Letouzey, 2001].**

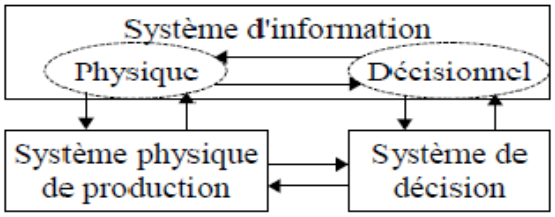


Figure 1.1 : Approche systémique du système de production

1.2 Les systèmes flexibles de production

La compétition entre les entreprises a conduit à chercher à améliorer la production tout en respectant les délais de livraison. Pour satisfaire cet objectif, les systèmes de production doivent être flexibles et c'est ainsi que sont apparus les Systèmes Flexibles de Production FMS (Flexible Manufacturing Systems).

Les FMS sont des systèmes de production fortement automatisés et adaptés à la fabrication en petites et moyennes séries de produits ayant des gammes différentes et ceci à faible coût. La littérature propose différentes définitions des FMS. Nous retenons celle de MacCarthy et Liu (1993) qui a l'avantage de prendre en compte les processus de production et de stockage:

Un système de production capable de produire différents types de pièces, composé de machines à commande numérique ou à contrôle numérique et d'un système automatisé de stockage connectés par un système automatisé de manutention. Le fonctionnement du système entier est sous le contrôle et le pilotage d'un système informatique **[MacCarthy et Liu, 1993].**

Un système flexible de production se caractérise par trois ensembles principaux **[N. Brauner et *al.*, 2004]** :

- les moyens de production et de stockage ;
- les moyens de transport/manutention ;
- la topologie et le type du réseau de guidage du système de transport/manutention.

Les FMS sont des systèmes dynamiques. Ils sont constitués d'un ensemble d'entités dont l'état peut changer en temps réel, par exemple une ressource qui est en état de production d'une pièce peut passer à un état non productif après la fin d'usinage de cette pièce.

La dynamique est aussi réalisée par l'interaction entre ces entités. Pour garantir l'objectif de la production, le système de production doit être doté en plus de **la flexibilité.**

1.2.1 La Flexibilité des systèmes de production

Définitions

Dans sa définition générale, la flexibilité traduit l'aptitude d'un système à changer facilement et à moindre coût pour pouvoir s'adapter aux circonstances.

On peut définir la flexibilité par l'ensemble des propriétés et qualités d'un système manufacturier qui peut supporter des changements dans les types de produits et la capacité de production. Ces changements peuvent être :

- Internes : comme les pannes des équipements, pannes des systèmes informatiques (logiciels de gestion), absentéisme des travailleurs, variations des temps de fabrication…etc.
- Externes : comme le changement dans la conception des produits, la complexité de conception des produits, la variation de la demande ...

Pour absorber les incertitudes causées par les changements de conception des produits, le système de fabrication doit être flexible et capable de fabriquer différents types de produits avec un temps de fabrication et un coût minimum. Donc on peut dire que la flexibilité est la capacité des systèmes de fabrication de répondre à la fois aux changements internes et externes **[Adamou, 1997].**

1.2.3 La classification de systèmes flexibles de production

Il existe de nombreuses classifications des différents types de systèmes flexibles de production. L'étude de la littérature montre que les critères de classification sont variés, citons par exemple :

- La topologie de l'atelier, la méthode de fabrication utilisée (cyclique/acyclique, unitaire/ lot, . . .) **[Rembold, 1993]**.
- La taille des machines et leur complexité **[Kusia, 1985]**.
- Les modèles des flux de pièces à traiter **[Browne et *al.*, 1984]**.
- Les caractéristiques de fonctionnement et de contrôle **[Maccarthy, 1993]**

[Brauner et *al.*, 2004] donne une classification simplifiée en quatre classes, cette classification se fait suivant les caractéristiques de fonctionnement et de contrôle des FMS

- Un **SFM** (*Single Flexible Machine* ou machine unique flexible) est une unité de production pilotée par ordinateur qui contient une unique machine avec capacité de changement d'outil, un élément de manutention et un stock interne ;

- Un **FMC** (*Flexible Manufacturing Cell* ou cellule de production flexible) est un groupe de SFM reliés par un seul élément de manutention, avec ou sans espaces de stockage intermédiaires situés en entrée ou en sortie de chaque SFM ;
- Un **MMFMS** (*Multi-Machine Flexible Manufacturing System* ou système de production flexible à plusieurs machines) est un groupe de SFM reliés par un système de manutention automatisé qui contient deux ou plusieurs chariots. Les systèmes de type MMFMS sont donc capables de servir plusieurs machines simultanément, avec ou sans espaces de stockage intermédiaire qui peuvent être en Entrée/Sortie de chaque SFM.
- Un **MCFMS** (*Multi-Cell Flexible Manufacturing System* ou système de production à plusieurs cellules) constitué de plusieurs FMC et éventuellement de SFM connecté par un système de transport automatique.

1.3 Les problèmes d'optimisation combinatoire

Un problème d'optimisation combinatoire *(POC)* est un problème de minimisation ou de maximisation auquel on associe un ensemble d'instances.
Les éléments qui définissent un problème d'optimisation sont :

(i) une fonction objective f
(ii) un ensemble de solutions S
(iii) un ensemble de contraintes C que doit satisfaire une solution afin d'être admissible

Chaque instance possède un ensemble fini de solutions candidates S, une fonction objectif f et un ensemble fini de contraintes C qui dépendent de la nature du problème. Les solutions du sous-ensemble $X \sqsubseteq S$, dites réalisables, sont celles qui se soumettent aux contraintes de C. La fonction objectif f donne, à chaque solution $s \in X$, la valeur de la fonction objectif f(s), qui peut être réelle ou entière selon la nature du problème. Un problème de minimisation consiste à trouver, pour une instance donnée, une solution $s^* \in X$ telle que $f(s^*) \leq f(s)$, pour toute solution $s \in X$. On dit alors que s^* est une solution optimale (ou optimum global) de cette instance de problème. Pour un problème de maximisation, on veut trouver une solution $s^* \in X$ telle que $f(s^*) \geq f(s)$, pour toute solution $s \in X$ **[Sebastien, 2007].**

Les problèmes d'optimisation combinatoire sont présents dans plusieurs domaines d'applications industrielles, économiques et scientifiques. On les retrouve dans **l'ordonnancement** de tâches dans un système de production. Ce sont, en général, des problèmes faciles à définir, mais difficiles à résoudre.

Avant de s'avancer dans la théorie de complexité des problèmes, il est nécessaire de savoir qu'un *POC* (Problème d'Optimisation Combinatoire) peut être transformé en un problème de décision. Un problème de décision est un problème où le résultat est «oui» ou «non», exclusivement.

Dans le cas d'un problème de minimisation, le problème de décision correspondant serait «existe-t-il une solution s telle que $f(s) \leq a$», où a est un paramètre du problème considéré. Ce problème de décision ne demande pas de trouver la valeur d'une solution inférieure ou égale à a, mais seulement d'indiquer s'il en existe une.

1.4 Notions de complexité

Définition 1: La complexité d'un problème est la complexité de son meilleur algorithme connu. On suppose que, pour chaque problème que l'on veut résoudre, l'on dispose d'une mesure de la taille du problème.

Par exemple, lorsqu'il s'agit d'un problème d'ordonnancement de n tâches, on utilisera comme mesure le nombre de tâches. Le nombre d'opérations élémentaires effectuées par un algorithme est donc une fonction appelée **complexité** qui va dépendre de n **[Rebreynd, 1999].** Les problèmes de la classe P et ceux de la classe NP forment ce qu'on appelle la théorie de complexité **[Mehenni, 2006].**

Définition 2 : On appelle algorithme polynomial, un algorithme dont le nombre d'opérations est un polynôme de n; par exemple $O(n^{\alpha})$ avec α une constante.

Les *POC* appartiennent à la classe **NP- difficile** et il est possible de les transformer en problèmes de décision. Un problème appartenant à la classe des problèmes NP-difficiles est un problème qui ne pourra probablement jamais être résolu façon optimale. On s'intéresse alors à des algorithmes qui fournissent des solutions proches de la solution optimale. Les problèmes de très grande taille sont, en général, classés NP-difficiles.

On qualifie ce type d'algorithmes d' « exponentiels », une caractéristique importante des algorithmes exponentiels est qu'ils sont très sensibles à la taille des problèmes résolus et ils peuvent être utilisés pour des instances suffisamment petites. Pour des applications réelles, il est nécessaire de vérifier que la taille des problèmes est largement inférieure aux capacités de l'ordinateur. Pour les problèmes de taille supérieure, il n'est pas possible de fournir un algorithme exact. On s'intéresse alors naturellement à des algorithmes non exacts appelés « heuristiques ».

La théorie de complexité permet de classer les problèmes de décision en problèmes « faciles » et « difficiles » à traiter.

Deux critères importants sont utilisés : le temps et l'espace requis pour la résolution du problème. Nous ne nous sommes intéressées qu'à la complexité temporelle.

La classe des problèmes difficiles à résoudre NP-difficile, regroupe un ensemble de problèmes très importants comme **l'ordonnancement**. Les problèmes **d'ordonnancement** d'ateliers sont des problèmes combinatoires donc la fonction « ordonnancement » étant l'objet principal de notre étude, elle est abordée plus en détail dans la suite du chapitre.

GÉNÉRALITÉS SUR L'ORDONNANCEMENT

Le plan de charge permet de vérifier si la charge occasionnée dans l'atelier par les commandes n'est pas supérieure à la capacité des ressources de l'atelier. Dans le cas contraire, des réajustements de charge (lissage) ou de capacité (sous-traitance, heures ou équipes supplémentaires) peuvent être faits. Les données sont transmises à la fonction ordonnancement que nous détaillons dans cette partie.

1.5 L'ordonnancement

L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de la production qui vise à améliorer l'efficacité d'une entreprise en termes de coûts de production et de délais de livraison.

On peut rencontrer les problèmes d'ordonnancement dans de très nombreux domaines : les systèmes industriels de production (activités des ateliers en gestion de production et problèmes de logistique), les systèmes informatiques (les tâches sont les programmes et les ressources sont les processeurs, la mémoire...), les systèmes administratifs (gestion du personnel, emplois du temps,…), les systèmes de transport, la construction, … etc.

Les différentes données d'un problème d'ordonnancement sont les tâches, les ressources, et les contraintes.

Dans le système de production, un problème d'ordonnancement se définit comme étant la localisation dans le temps et l'espace la réalisation d'un ensemble de **tâches**, compte tenu des **contraintes** temporelles (une date de début avec une durée ou une date de fin) et de contraintes portant sur l'utilisation et la disponibilité des **ressources** requises par les tâches.

1.5.1 Les tâches

Une tâche est une entité élémentaire organisée dans le temps, par une date de début et/ou de fin, et dont la réalisation nécessite une durée préalablement définie.

Elle est constituée d'un ensemble d'opérations qui requiert, pour son exécution, certaines ressources et qu'il est nécessaire de programmer de façon à optimiser un certain objectif.

On distingue deux types de tâches :

- ***les tâches morcelables*** (préemptives) qui peuvent être exécutées en plusieurs fois, facilitant ainsi la résolution de certains problèmes,
- ***les tâches non morcelables*** (indivisibles) qui doivent être exécutées en une seule fois et ne sont interrompues qu'une fois terminées

1.5.2 Les contraintes d'ordonnancement

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre simultanément les variables représentant les relations reliant les tâches et les ressources. On distingue deux types de contraintes, les contraintes temporelles et les contraintes de ressources.

- **Les contraintes temporelles**

Les contraintes temporelles concernent les délais de fabrication imposés. Ces contraintes peuvent être :

- ***des contraintes de dates butoirs***, certaines tâches doivent être achevées avant une date préalablement fixée,
- ***des contraintes de précédence***, une tâche *i* doit précéder la tâche *j*,
- ***des contraintes de dates au plus tôt***, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches.

- **Les contraintes de ressources**

Ces contraintes concernent la limitation de la quantité de ressources de chaque type. Dans ce cadre, deux types de contraintes de ressources sont distinguées:

- ***Les contraintes disjonctives*** : induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource.
- ***Les contraintes cumulatives*** : impliquant la limitation du nombre de tâches à réaliser en parallèle.

1.5.3 Les ressources

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. On trouve plusieurs types de ressources :

- ***Les ressources renouvelables,*** qui, après avoir été allouées à une tâche, redeviennent disponibles (machines, personnel, … etc),
- ***Les ressources consommables****,* qui lorsqu'après sa libération, elle n'est pas disponible en même quantité (argent, matières premières, … etc).

Dans le cas des ressources renouvelables, on distingue principalement, les ressources disjonctives qui ne peuvent exécuter qu'une tâche à la fois et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité.

1.5.4 Les objectifs d'ordonnancement

Les objectifs des entreprises se sont diversifiés et le processus d'ordonnancement est devenu de plus en plus multicritère. Les critères que doit satisfaire un ordonnancement sont variés. D'une manière générale, on distingue plusieurs classes d'objectifs concernant un ordonnancement **[Esquirol, 1999].**

- ***Les objectifs liés au temps*** **:** On trouve par exemple la minimisation du temps total d'exécution, du temps moyen d'achèvement, des durées totales de réglage ou des retards par rapport aux dates de livraison.
- ***Les objectifs liés aux ressources*** **:** maximiser la charge d'une ressource ou minimiser le nombre de ressources nécessaires pour réaliser un ensemble de tâches sont des objectifs de ce type.
- ***Les objectifs liés au coût*** **:** ces objectifs sont généralement de minimiser les coûts de lancement, de production, de stockage, de transport, etc.

La satisfaction de tous les critères à la fois est souvent délicate, car elle conduit souvent à des situations contradictoires et à la recherche de solutions à des problèmes complexes d'optimisation **[Souier, 2009].**

1.5.5 Les types d'ordonnancement

On distingue deux types d'ordonnancement

• Ordonnancement prévisionnel

Dans ce type d'ordonnancements, les méthodes classiques ont pour but de générer un ordonnancement optimal, c'est-à-dire minimisant un des critères présentés (ou plus rarement une

combinaison de plusieurs critères). Les problèmes d'ordonnancement ainsi définis sont des problèmes d'optimisation combinatoire.

La plupart d'entre eux appartiennent à la classe des problèmes NP-difficiles qui ne peuvent pas être résolus au moyen d'un algorithme polynomial. Ainsi à côté des algorithmes classiques de l'optimisation combinatoire ont été développés un grand nombre d'heuristiques.

• Ordonnancement réactif

C'est un système d'ordonnancement qui inclue une méthode pour réagir en temps réel face aux aléas [**Lamothe, 1996].** Ces aléas peuvent être internes survenant à l'intérieur de l'atelier (pannes de ressources, absence de personnel, …) ou externes provenant de son environnement (retard d'approvisionnement, arrivée imprévue d'un ordre de fabrication).

1.5.6 Les problèmes d'atelier multi-machines

Les problèmes d'ordonnancement peuvent être classifiés selon le nombre de machines et leur ordre d'utilisation pour fabriquer un produit (gamme de fabrication) qui dépend de la **nature de l'atelier**. Un atelier se définit par le nombre de machines qu'il contient et par son type.

Une classification peut exister selon le nombre des machines et l'ordre d'utilisation des machines, pour réaliser un travail (par exemple fabrication d'un produit qui dépend de la nature de l'atelier). Un système (exemple atelier) se définit par le nombre de machines qu'il contient et par son type **[Mehenni, 2006]**.

Les différents types possibles sont les suivants :

- ***Une machine :*** chaque tâche est constituée d'une seule opération.
- ***Machines parallèles* :** elles remplissent toutes les mêmes fonctions. Selon leur vitesse d'exécution, on distingue : les machines identiques où la vitesse d'exécution est la même pour toutes les machines et toutes tâches, et les machines uniformes où chaque machine a une vitesse d'exécution propre et constante et la vitesse d'exécution est la même pour toutes les tâches d'une même machine.
- ***Machines dédiées* :** elles sont spécialisées à l'exécution de certaines opérations. Dans cette catégorie, chaque tâche est constituée de plusieurs opérations. En fonction du mode de passage des opérations sur les différentes machines, trois ateliers spécialisés sont différenciés.

Trois ateliers spécialisés sont différenciés, selon que la gamme de fabrication est commune à tous les travaux, c.à.d. atelier à cheminement unique ou *flow shop*, spécifique à chaque travail c.à.d. atelier à cheminements multiples ou *job shop*, ou que cette gamme n'est pas définie c.à.d. atelier à cheminement libre *open shop*.

- ***Flow Shop* (F)** (figure1.2) : c'est un cas particulier du problème d'ordonnancement d'atelier pour lequel le cheminement des travaux est unique : les *n* travaux utilisent les *m* machines dans l'ordre 1,2, . . ., *m* (lignes de production);

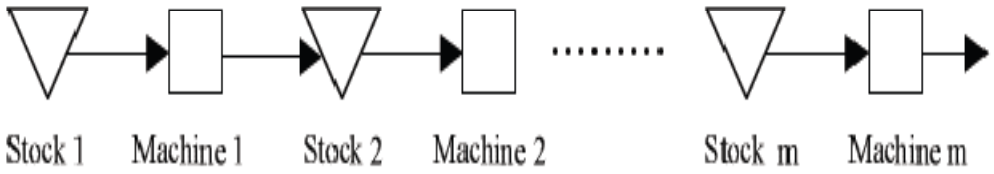


Figure 1.2 : Représentation d'un atelier flow shop

(Le flow shop est une production linéaire, caractérisée par une séquence d'opérations identiques pour tous les produits. Chaque produit passe Successivement sur toutes les machines).

- ***Job Shop (J)*** (figure 1.3) : les *n* travaux doivent être exécutes sur les *m* machines, sous des hypothèses identiques à celles du flow shop, la seule différence est que les séquences opératoires relatives aux différents travaux peuvent être distinctes et sont propres à chaque travail;

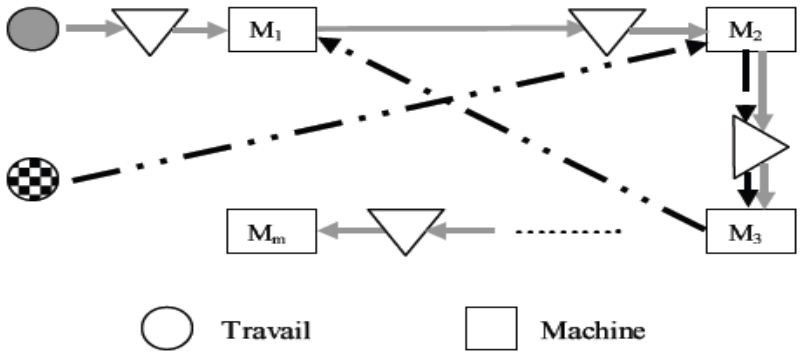


Figure 1.3 : Présentation d'un atelier job shop

(Dans un problème de type jobshop les produits ne passent pas systématiquement sur toutes les machines selon un ordre commun : dans l'atelier, chaque produit emprunte un routage qui lui est propre).

- ***Open Shop (O)*** : 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 consiste 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 openshop est un jobshop dans lequel les contraintes de précédence sont relâchées. C'est-à dire, les opérations peuvent être effectuées dans n'importe quel ordre).

1.5.6 Les méthodes de résolution des problèmes d'ordonnancement

Les *POC* sont des problèmes NP-difficiles; il existe une multitude de méthodes pour les résoudre. La section suivante présente les différentes méthodes de résolution d'un *POC* **« l'ordonnancement ».**

1.5.6.1 Les méthodes exactes

Les méthodes exactes garantissent de trouver la solution optimale et de prouver son optimalité pour toutes les instances d'un *POC*. Le temps nécessaire pour trouver la solution optimale d'un *POC* augmente en général fortement avec la taille du problème. De façon pratique, seuls les problèmes de petites et moyennes tailles peuvent être résolus de façon optimale par des algorithmes exacts. 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. Leur inconvénient majeur est le temps exponentiel de résolution qui croît en fonction de la taille du problème.

Comme méthodes exactes appliquées aux problèmes d'ordonnancement de production, nous citons la programmation mathématique, linéaire et dynamique ainsi que la recherche arborescente ou Procédure par Séparation.

1.5.6.2 Les méthodes heuristiques

Malgré l'évolution permanente des calculateurs et les progrès permanents de l'informatique, il existe pour plusieurs problèmes d'optimisation combinatoire avec une taille critique de l'espace de solutions admissibles. La méthode permettant d'obtenir une solution optimale est bien évidemment celle de l'énumération complète de l'espace de recherche. Cette dernière est dans la plupart des cas prohibitive.

Compte tenu de ces difficultés, la plupart des spécialistes de l'optimisation combinatoire ont orienté leur recherche vers le développement de méthodes heuristiques. Une méthode heuristique est souvent définie comme une procédure exploitant au mieux la structure d'un problème, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible **[Lopez et al. 2001].**

D'autres méthodes, nommées métaheuristiques, sont plus générales et peuvent être appliquées à plusieurs catégories de *POC*. Les métaheuristiques les plus connues sont la recherche locale *(RL),* le recuit simulé *(RS),* la recherche avec tabous *(RT),* les algorithmes génétiques *(AG)* et l'optimisation par colonie de fourmis *(OCF)*.....

Parmi les méthodes à base de population, nous citons les algorithmes des colonies de fourmis **[Dorigo et al. 1999]** qui ont été à l'origine des méthodes basées sur le comportement animal (essaims particulaires, … etc.). Le paradigme des colonies de fourmis repose sur une analogie entre le problème d'optimisation et celui de la recherche d'un plus court chemin. Les fourmis en quête de nourriture déposent en effet des phéromones permettant à leurs consœurs de les suivre. Ces phéromones s'évaporant avec le temps, et une fourmi ayant tendance à suivre la trace la plus forte, les chemins les plus courts, moins sujets à évaporation et renforcés par le passage de nombreuses fourmis, auront tendance à être privilégiés avec le temps

- **La recherche locale**

La recherche locale *(RL)* est un algorithme fort simple qui fouille l'espace de solutions d'un problème en visitant pas à pas chaque solution. Une opération élémentaire est effectuée sur la

solution *s* pour donner une solution voisine *s′*. Le voisinage *N(s)* comprend toutes les solutions pouvant être obtenues en effectuant une opération élémentaire sur *s*.

Il existe différentes opérations élémentaires, chacune d'elles donnant un voisinage différent. Pour résoudre un problème de minimisation, un algorithme de descente accepte une nouvelle solution (donnée par l'opération élémentaire définissant le voisinage) seulement si la valeur de sa fonction objectif est inférieure à celle de la solution précédente. Cette méthode a pour inconvénient d'être souvent coincée dans un optimum local.

- **Le recuit simulé**

Le recuit consiste à faire refroidir un métal le plus lentement possible pour qu'il obtienne une structure moléculaire optimale. L'algorithme recuit simulé (RS) est une méthode de recherche locale inspiré de ce phénomène thermodynamique.

Cette méthode est une méthode de recherche locale dont les origines remontent aux expériences de Metropolis en 1953 **[Metropolis et al., 1953]**. Leurs travaux consistaient à étudier la stabilité thermique d'un système physique. Cette méthode de recherche utilise une analogie avec le processus de recuit d'un matériau : par refroidissement lent après une élévation de température, on peut obtenir à la fin du processus un état d'énergie minimal.

L'idée principale est d'accepter des déplacements dans le voisinage d'une solution en dépit du fait qu'ils dégradent le cout de la solution et cela suivant une probabilité calculée à partir d'une température T du système à moment donné.

Le recuit simulé démarre par une solution initiale admissible *s* et continue l'exploration de l'espace d'état en effectuant des perturbations mineures à la solution courante. Si la solution voisine s' obtenue améliore le critère cherché alors elle est retenue. Cette nouvelle solution s' est acceptée comme solution courante si son coût est inférieur à celui de la solution courante *(*$f(s') < f(s)$ dans le cas d'un problème de minimisation). Sinon, l'acceptation de la solution dépend d'une probabilité calculée à partir de la température T et de différence de coût $f(s') - f(s)$. La température est un paramètre fixé au début de l'algorithme et diminue graduellement tout au long de l'exécution de l'algorithme. Plus la valeur T est élevée, la solution considérée a de la chance d'être acceptée. Plus T s'approche de 0, moins une solution de mauvaise qualité a de chance d'être acceptée. À chaque degré de température, un nombre prédéfini de solutions voisines est visité. Une fois cette quantité atteinte, la température est diminuée.

- **La recherche avec tabou**

La recherche avec tabous *(RT),* qui est aussi une méthode de recherche locale, a été introduite par **[Glover, 1986]**. Cette méthode garde temporairement en mémoire les transformations effectuées sur la solution. Cette mémoire s'appelle la liste taboue, elle tente de trouver la meilleure solution dans un voisinage, en évitant d'examiner les mêmes solutions plusieurs fois. La nouvelle solution est trouvée en fouillant l'ensemble du voisinage de la solution courante. Pour éviter que la liste de tabous ne devienne trop restrictive, le critère d'acceptation permet de révoquer le statut de tabou d'une transformation. Un exemple de critère d'acceptation serait de retirer de la liste de tabous une transformation qui permettrait d'obtenir une solution de qualité supérieure à la meilleure solution rencontrée.

La méthode Tabou est une métaheuristique souvent utilisée pour résoudre des problèmes industriels car elle présente une amélioration importante, plusieurs problèmes de types flow shop ou job shop sont traités par cette méthode.

1.5.6.3 Les méthodes basées sur les techniques de l'intelligence artificielle

L'intelligence artificielle a fourni de nouvelles approches et tendances en ordonnancement depuis deux décennies, notamment par l'utilisation de réseaux de neurones **[Sabuncuoglu, 1998]** ou de systèmes experts **[Smith, 1992]**. L'utilisation de la logique floue dans l'ordonnancement n'a pas pour but de proposer une nouvelle méthode, mais d'apporter ponctuellement des techniques qui vont permettre d'être plus proche de la réalité. L'incertitude et l'imprécision caractérisant les données d'une entreprise, bien que fréquemment constatées, sont rarement prises en compte à court terme **[Smith, 1992]**.

La gestion de cette incertitude et de cette imprécision est plus souvent faite par le biais de méthodes statistiques, mais l'utilisation de la logique floue et de la théorie de possibilités a connu un succès certain **[Souier, 2009]**.

Les méthodes d'ordonnancement issues de l'intelligence artificielle sont toutefois peu implantées dans l'industrie car elles ne sont pas encore suffisamment matures ni suffisamment transparentes pour l'utilisateur.

Pour continuer, nous citerons quelques méthodes rattachables au domaine de l'intelligence artificielle.

- **La logique floue**

La logique floue **[Kaufmann, 1992]**, est une partie de l'intelligence artificielle qui repose sur la notion de sous ensembles flous, visant essentiellement à formaliser de manière « informatisable » l'imprécision et l'incertitude considérées comme inhérentes à beaucoup de connaissances humaines. Une proposition comme « Une commande A est urgente », sera affecter à une valeur de vérité comprise entre 0 et 1 après que la notion d'urgence ait été traduite par une « fonction d'appartenance » décrivant l'évolution de cette valeur de vérité sur le référentiel concerné, au lieu d'être traduite par une valeur de vérité « Vrai » ou « Faux ».

L'utilisation de la logique floue à l'ordonnancement peut être divisée en deux grandes catégories :
- Intégrer dans des systèmes experts des connaissances imprécises et incertaines.
- Décrire des contraintes «flexibles» en propagation de contraintes **[Lopez et al. 2001]**, ce qui ne va plus poser le problème de décrire une base de connaissance complexe.

- **Systèmes experts**

Un système expert se compose d'une base de faits qui regroupe les suppositions « vraies » et d'une base de règles qui renferme la connaissance sur le problème à traiter. Un « moteur d'inférence » permet donc de déterminer les conditions des règles qui sont vérifiées et les conséquences qui peuvent en être conclues et ajoutées à la base de faits.

Les problèmes combinatoires d'ordonnancement ont fait que, beaucoup de tentatives ont été faites pour « mettre en boîte » des connaissances sur le domaine ou sur un atelier donné pour orienter l'élaboration d'un ordonnancement. Ces expériences se sont trouvées face à la difficulté du problème, où il y'a peu de connaissances génériques semblent exister et le développement

d'une base de connaissances relative à un atelier donné demande un effort considérable. De plus, les connaissances mises en jeu en ordonnancement semblent peu se plier à un schéma aussi binaire que celui de règles de production «simples» **[Fabre, 2009].**

- **Réseaux de neurones**

Parmi les autres branches de l'intelligence artificielle on peut citer les réseaux de neurones. Cette branche ne permet non seulement d'imiter le raisonnement humain mais la structure du cerveau et en particulier ses capacités d'apprentissage.

Beaucoup de problèmes d'ordonnancement ont utilisé plusieurs types de réseaux de neurones. Dans la thèse d'Abada **[Abada, 1997],** une citation détaillée de ces différents types de réseaux de neurones a été effectuée. Seule une description qualitative est effectuée dans cette thèse. L'application des réseaux de neurones et des algorithmes génétiques aux problèmes d'ordonnancement classiques a été présentée dans les travaux de Lee et *al*. **[Lee In, 2000]**. Sik et *al*. **[Sik, 2003]** proposent pour le choix de règles de décision dans des problèmes d'ordonnancement comme la gestion des conflits, le couplage des réseaux de neurones avec la simulation **[Senties, 2007]**.

- **Les algorithmes évolutionnaires**

De tout temps, les sciences de la vie et les processus naturels ont constitué des bases d'inspiration et d'imitation pour les chercheurs et les ingénieurs. Les mécanismes du monde vivant sont à l'origine des systèmes artificiels utilisables dans des contextes variés. Les méthodes évolutives qui sont présentées dans cette section constituent la base d'un nouveau champ de la programmation informatique en pleine effervescence.

Contrairement aux méthodes de recherche locale qui font intervenir une solution unique, les méthodes évolutives manipulent un groupe de solutions admissibles à chacune des étapes du processus de recherche. L'idée centrale consiste à utiliser régulièrement les propriétés collectives d'un ensemble de solutions distinguables, appelé population, dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche.

Cette classe d'algorithmes (les algorithmes évolutionnaires) regroupe plusieurs paradigmes incluant les algorithmes génétiques **[Goldberg, 1989]**, la stratégie d'évolution **[Hoffmeister and T. Back, 1991], [Schwefel and T. B¨ack,1998],** la programmation évolutionnaire **[Fogel,1993]**, la recherche dispersée **[Glover, 1995]** et les algorithmes mémétiques **[Moscato, 1993]**.

En règle générale, la taille de la population reste constante tout au long du processus. Après avoir généré une population initiale de solutions, une méthode évolutive tente d'améliorer la qualité moyenne de la population courante en ayant recours à des principes d'évolution naturelle **[Widmer, 2001].** Les algorithmes génétiques constituent à ce jour, l'approche la plus utilisée parmi les méthodes évolutives. Cette méthode donne souvent de très bons résultats aux problèmes d'optimisation, mais un grand nombre de solutions sont susceptibles d'être évaluées, ce qui peut demander un temps de traitement important.

L'intelligence artificielle est susceptible d'être d'un grand intérêt pour l'ordonnancement, lui permettant notamment d'être plus proche de la réalité. Les méthodes d'ordonnancement issues de l'intelligence artificielle sont toutefois peu implantées dans l'industrie car elles ne sont pas encore suffisamment matures ni suffisamment transparentes pour l'utilisateur.

1.5.7 Problématique de l'ordonnancement en Job Shop

Nous nous intéressons plus spécifiquement aux méthodes permettant de résoudre des problèmes d'ordonnancement de *n* ordres de fabrication (ou jobs) sur *m* (*m*>2) machines.

Les problèmes d'ordonnancement en job shop sont des problèmes classiques en littérature d'ordonnancement se compose d'un ensemble fini de travaux *n* devant traité sur un ensemble fini de machines *m* et est dénoté comme problème$n \times m$, sachant que chaque machine ne peut travailler que sur un seul job à la fois. Les notations couramment employées dans les problèmes d'ordonnancement sont les suivantes :

- La date de début au plus tôt du job J_i est notée r_i (*release date*). Cette date correspond à un minorant de la date de disponibilité du job J_i.
- d_i et C_i sont respectivement la date de fin au plus tard souhaitée (*due date*) et la date effective de fin de fabrication (*completion date*) du job J_i

On peut décomposer les problèmes d'ordonnancement en deux phases, une première phase qui tente d'attribuer des caractéristiques temporelles aux opérations à effectuer et une deuxième phase d'attribution des ressources aux différentes opérations **[Carlier, 1988]**.

- La première phase est décrite par des contraintes potentielles qui regroupent les contraintes de précédence entre tâches (aucune panne de machine dans tout le processus d'ordonnancement), prenant en compte la succession des opérations de la gamme opératoire (la tâche *i* doit démarrer après la tâche *j*) et les contraintes d'une tâche dans le temps(le temps transport entre les machines est zéro).
- La deuxième phase est décrite par les contraintes disjonctives qui expriment le fait que deux opérations utilisant une même ressource, ne peuvent avoir d'intervalle de temps commun **[Mebarki, 1995].**

1.5.8 L'ordonnancement en temps réel

Les algorithmes les plus utilisés, notamment dans les logiciels du marché, reposent sur des règles de priorité. Lorsqu'une machine devient disponible et qu'il existe plusieurs opérations en attente d'exécution sur cette machine, on doit choisir l'opération à effectuer. Ce choix se fait selon une règle dite de priorité, ces règles constituent une des approches les plus simples et les plus utilisées pour ordonnancer en temps réel les opérations à traiter.

Parmi les règles de priorité les plus courantes en ordonnancement, on trouve les règles basées sur des critères portant sur les délais, les durées opératoires, les dates d'arrivées, la taille des files d'attente, le nombre d'opérations restant,…etc. Une règle appliquée en ordonnancement peut être constante (règle statique) ou évoluée en fonction de l'environnement (règle dynamique) **[Mirdamadi, 2009]**.

L'ordonnancement temps réel est une des réponses les mieux adaptées aux problèmes de gestion rencontrés dans les systèmes de production. On définit l'ordonnancement temps réel comme une démarche permettant, chaque fois qu'une décision d'affectation doit être prise, de proposer une solution prenant en compte l'état réel du système de manière à optimiser une fonction objectif **[Mebarki, 1995]**.

1.6 Conclusion

Les principales définitions ainsi que les problématiques de l'ordonnancement ont été introduites dans ce chapitre. Ces problèmes d'ordonnancement dans les systèmes flexibles de production sont dans le cas général des problèmes de type NP-difficile ou encore de complexité ouverte. De nombreux ouvrages tels que **[Bourdeaud'huy et Korbaa, 2006], [Esquirol et Lopez, 1999]**, abordent l'étude de la complexité et notamment celle des problèmes d'ordonnancement. Néanmoins, d'un point de vue opérationnel ce type de complexité justifie l'utilisation de méthodes de résolution approchées pour traiter des instances de problème de taille importante.

Le chapitre suivant est dédié à une classe particulière de méthodes approchées de type heuristique : les algorithmes évolutionnaires utilisés pour résoudre les problèmes d'ordonnancement dans les systèmes flexibles étudiés dans ce mémoire, à savoir l'algorithme mémétique avec gestion de population, et la recherche dispersée.

Chapitre 2

Les algorithmes évolutionnaires

Dans ce chapitre nous nous intéressons de manière approfondie aux algorithmes évolutionnaires. Le chapitre est dédié à la présentation des notions nécessaires permettant la compréhension de ces approches. Nous insisterons sur le rôle des algorithmes évolutionnaires, qui ont déjà montré leur efficacité dans de nombreux domaines mais il a fallut adapter ces algorithmes aux particularités induites des systèmes flexibles de production.

Sommaire

Le recours à des méthodes de résolution approchées pour résoudre les problèmes d'ordonnancement, notamment les problèmes d'ordonnancement dans les systèmes flexibles de production, a été justifié dans le chapitre précédent. Notre objectif vise donc à exploiter le potentiel des approches dites méta-heuristiques pour résoudre ces problèmes.

De même que pour le chapitre précédent, l'objet n'est pas ici de réaliser une présentation exhaustive du domaine des algorithmes évolutionnaires qui est en plein foisonnement. Un nombre croissant de livres apparaît chaque année sur ce sujet.

Dans ce chapitre, notre démarche vise simplement à situer le cadre général dans lequel s'inscrivent les travaux décrits dans le chapitre suivant. Il comporte donc un aperçu général sur

les algorithmes évolutionnaires puis se focalise sur l'algorithme génétique hybride. Enfin il aborde l'utilisation des méthodes évolutionnaires dans le domaine de l'ordonnancement.

Une métaheuristique peut être définie comme étant un processus itératif, qui permet de subordonner ou cordonner différentes opérations ou procédures pour pouvoir détecter des solutions de bonne qualité. Elles peuvent manipuler une seule solution ou plusieurs solutions à chaque itération. Les algorithmes évolutionnaires, les colonies de fourmis, les essaims particulaires, le recuit simulé, la recherche tabou, et la recherche dispersée, sont parmi les métaheuristiques les plus citées dans la littérature. Il existe bien d'autres méthodes peut être moins connues.

Les métaheuristiques présentent une alternative intéressante pour la résolution des problèmes d'optimisation classés NP-Difficiles, problèmes auxquels on ne connait pas de méthodes plus efficaces pour les résoudre. Malgré que les métaheuristiques ne garantissent pas l'optimalité des solutions obtenues, elles présentent, généralement, des avantages intéressants, tels que la bonne qualité des solutions, le temps de calcul raisonnable, et la possibilité de les adapter à n'importe quel type de problème (qu'il soit facile ou très difficile, avec ou sans contraintes, bien ou mal formulé,…).

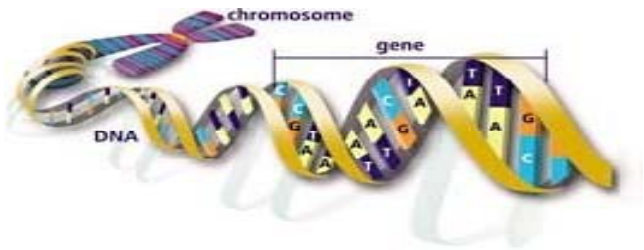
Plusieurs travaux réalisés au cours de ces dernières années ont démontré l'utilité et l'efficacité des métaheuristiques pour la résolution des problèmes d'optimisation combinatoire **[Widmer, 2001] [Cung et al, 2001]**.

Les algorithmes évolutionnaires et en particulier les AG sont parmi les métaheuristiques qui connaissent le plus de succès. Ceci est dû à leur simplicité d'implémentation, et leur nature parallèle fondamentale.

Le développement porte à la fois sur leur diffusion dans de très nombreux domaines d'application ainsi que sur l'étude et l'exploration des mécanismes d'évolution eux-mêmes. De nombreuses applications de ces méthodes concernent le domaine des systèmes de production **[Pierreval, 2003]** et notamment le domaine de l'ordonnancement **[Portmann, 2001].**

2.1 Généralités sur l'approche évolutionnaire

Les algorithmes évolutionnaires forment une classe importante des métaheuristiques à base de population de solutions, elles font partie du champ de l'Inelligence Artificielle (IA). Il s'agit d'IA de bas niveau, inspirée par « l'intelligence » de la nature. Les algorithmes évolutionnaires sont basés sur le concept de la sélection naturelle élaborée par Darwin en 1959. Ces approches sont fondées sur la métaphore de l'évolution et de la sélection naturelle. Un de leurs principes essentiels est que plus une solution est performante (c'est-à-dire adaptée), plus elle a une probabilité élevée de survivre **[Meignan, 2004].**



Les Algorithmes Évolutionnaires commencent par générer des solutions initiales dites individus, chaque individu est codé via un vecteur X^n, appelé chromosome et constitué de gènes. L'ensemble des solutions générées est appelé population de départ ou génération initiale.

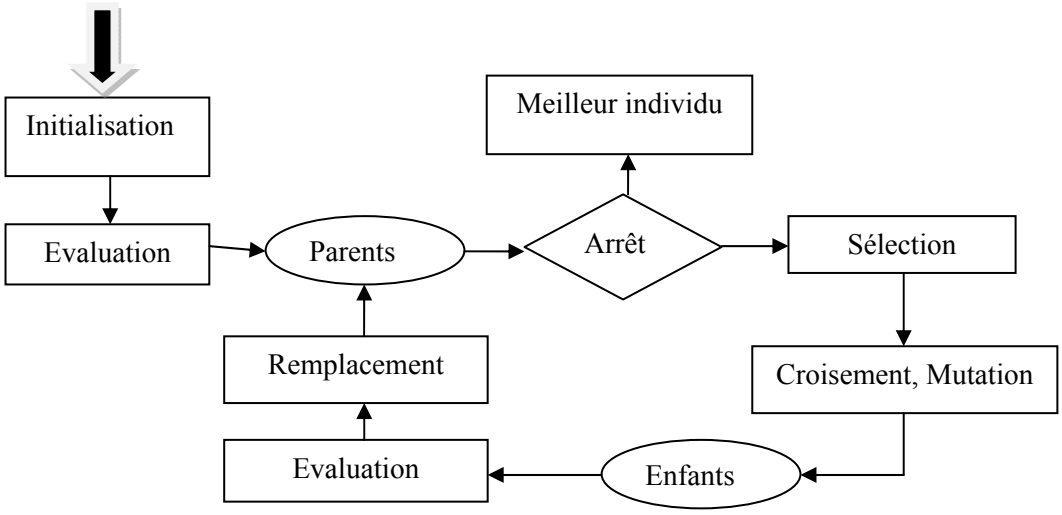


Figure 2.1 : Algorithme évolutionnaire [Schoenauer, 2001].

Une séquence d'opérations est appliquée, par la suite, à cette population. La première étant l'opération de sélection, qui consiste à sélectionner les individus (dits parents), qui vont être soumis à l'opération de "croisement", celle-ci consiste à combiner deux individus parents pour former un autre individu (dit enfant) qui va hériter certaines caractéristiques de ses parents. Les individus résultant du croisement peuvent être légèrement modifiés en appliquant l'opération de mutation. Le rôle de la mutation est maintenir une certaine diversité dans la population.

Une nouvelle population est constituée par "remplacement" de l'ensemble ou d'une partie de la population de départ par les individus résultants du croisement et la mutation

Cette séquence d'opérations forme le cycle de base des algorithmes évolutionnaires présenté dans la Figure 2.1, et chaque nouvelle population est dite génération de solutions. La procédure se répète jusqu'à satisfaire un critère d'arrêt. Le principe des approches évolutionnistes est illustré par l'algorithme 2.1 **[Meignan, 2004].**

Algorithme 2.1 : Principe des approches évolutionnaires

Générer la population initial P
Evaluer les individus de P
Tant que le critère d'arrêt n'est pas atteint **faire**
  Croiser les individus de P pour obtenir P'
  Muter les individus de P'
  Evaluer les individus de P'
  Sélectionner la nouvelle génération P à partir de P et P'
Fin

2.2 Les catégories principales d'algorithmes évolutionnaires

On distingue classiquement quatre catégories principales d'approches évolutionnaires : la programmation évolutionnaire (Evolutionary Programming), les stratégies d'évolution (Evolution Strategies), les algorithmes génétiques et la programmation génétique **[Blum and Roli, 2003]**. Nous donnons ci-dessous quelques éléments de comparaison tirés de **[Bäck et *al.*, 1997]** :

- **Algorithmes génétiques** : l'opérateur de croisement est considéré comme étant le plus important des opérateurs. La mutation est appliquée avec des probabilités très faibles et agit en tant qu'opérateur d'arrière-plan. Le codage des solutions consiste en une représentation généralement binaire des individus.
- **Stratégies d'évolution** : le codage des solutions peut être réalisé par des structures de données plus complexes que dans les algorithmes génétiques. Par ailleurs, les opérateurs de mutation ont une place aussi importante que les opérateurs de croisement.
- **Programmation évolutionnaire** : elle est fondée essentiellement sur l'opérateur de mutation et n'utilise pas d'opérateur de croisement. Comme les Stratégies d'évolution, le codage des solutions peut faire intervenir des structures de données complexes. Cette approche a été développée initialement pour faire évoluer des automates à états finis.
- **Programmation génétique :** le nombre de programmes dans la population peut varier entre les générations, lorsque la sélection et la reproduction sont en cours.

Plusieurs variantes des algorithmes évolutionnaires sont proposées dans la littérature pour améliorer ses performances, en particulier celles basées sur les algorithmes génétiques, le plus connu étant **l'algorithme mémétique** (Memetic Algorithms, MA) et de **la recherche dispersée** (Scatter Search), dit aussi algorithme génétique hybride. Notamment en y ajoutant une procédure de recherche locale, pour améliorer la qualité des solutions examinées ou de la population à explorer.

Les algorithmes mémétiques sont introduits pour la première fois en 1989 par Moscato. L'idée fondamentale de cette technique est de rendre l'algorithme génétique plus performant en lui ajoutant une procédure de recherche locale en plus de la mutation. Un algorithme mémétique est donc une combinaison, ou encore une coopération entre un algorithme génétique et une méthode de recherche locale. On rencontre aussi le nom d'algorithmes génétiques hybrides ou celui de genetic local search. L'origine du nom de cette métaheuristique est le concept de **"Mème".** Un mème est un élément culturel, acquit par apprentissage et pouvant être retransmis de génération en génération **[Moscato, 1989]**.

Concrètement, chaque individu fait évoluer une solution de manière indépendante par recherche locale, puis des opérateurs de croisement, mutation et sélection sont appliqués. De plus, des choix adaptatifs d'heuristiques sont parfois mis en œuvre dans certaines versions. Les algorithmes mémétiques se sont appliqués pour la résolution de plusieurs problèmes d'optimisation, tels que le problème de voyageur de commerce (Traveling Salseman Problem - TSP)…

La recherche dispersée (Scatter search) est une métaheuristique qui fait également partie de la famille des algorithmes évolutionnaires.

Cette méthode proposée par Glover **[Glover, 1998]**, fait combiner par un processus intelligent, un nombre de solutions jusqu'à atteindre un optimum. La principale différence avec les algorithmes génétiques, est que la recherche de l'optimum local est guidée, dans le sens qu'un échantillon référence R est sélectionné parmi l'ensemble de la population. Cet échantillon est intensifié et mis à jour à chaque itération. Une foi les solutions de cet échantillon sont combinées, une recherche locale est lancée pour tenter d'améliorer les résultats obtenus. L'échantillon est mis à jour avec autant les bonnes que les mauvaises solutions.

Parmi les métaheuristique à base de population fondées sur le principe de l'intelligence en essaims, et qui connaissent un grand succès on peut citer l'optimisation par **essaims particulaires** (Particle Swarm Optimization, PSO) et les algorithmes à base de **colonies de fourmis** (Ant Colony Optimization, ACO).

Plusieurs métaheuristiques s'inspirent du comportement social des insectes ou des animaux, les plus connues sont l'algorithme d'essaims particulaires inspirés du comportement des oiseaux lors de recherche de la nourriture. Aucun des oiseaux ne connaît où se trouve exactement la nourriture, mais chacun sait à chaque fois quelle est la distance qui le sépare de la source de nourriture. A chaque essai, (i.e. à chaque itération), la question qui se pose est la suivante : Quelle est la meilleure stratégie pour retrouver la nourriture ? La solution efficace est de suivre l'oiseau le plus proche de la nourriture.

Chaque oiseau est une solution possible dans l'espace de recherche, on l'appelle particule. Chaque particule est caractérisée par sa position courante et un vecteur de changement de position (appelé vélocité). Pour influencer leurs évolutions, ces particules sont dotées d'une mémoire structurée au niveau local (c'est-à-dire entre particules voisines). Chaque particule n'évolue, à chaque itération, qu'en fonction de ses proches voisins, et non pas selon l'état global de la population à l'itération précédente **[Khalouli, 2010]**.

Chaque particule possède une position courante associée à une solution. Elle se déplace en combinant trois composantes **[Meignan, 2004] :**

- La première composante entraîne la particule vers un point accessible en tenant compte de sa vitesse propre.
- La seconde composante l'oriente vers la meilleure solution obtenue lors des déplacements précédents.
- La troisième composante entraîne la particule vers la meilleure solution connue qui a été mémorisée suite aux informations reçues des autres particules.

La prochaine position de la particule est une combinaison de ces trois composantes. Ainsi, le déplacement des particules permet d'explorer l'espace de recherche, mais aussi de concentrer la recherche dans les zones prometteuses.

Les méthodes de colonies de fourmis (Ant Colony Optimization, ACO) sont inspirées du comportement des fourmis recherchant de la nourriture. Quand elles en trouvent, elles marquent le chemin y menant à la source en laissant des quantités de phéromones relatives à la qualité de la source trouvée (éloignement et quantité). Ainsi, les autres fourmis sont averties et attirées dans les directions les plus prometteuses.

Dans le même temps, le chemin le plus court sera davantage parcouru, et donc plus renforcé et plus attractif. En considérant que la phéromone s'évapore, les chemins les moins

renforcés finissent par disparaître, ce qui amène toutes les fourmis à suivre le chemin le plus court.

En fait, les variantes combinatoires apportent un avantage, par rapport aux autres métaheuristiques. Les problèmes de tournées ou de chemins optimaux se prêtent bien à ce type de méthode, telles que les problèmes de tournées de véhicules, et le problème de voyageur de commerce. En effet, les trajets réalisés par les fourmis correspondent alors à des parcours dans le graphe et la qualité de la source est représentée par la fonction-objectif.

L'initialisation du graphe se fait en attribuant des taux de phéromones nuls sur les arcs. Les fourmis sont représentées par les agents construisant la solution. Ils avancent dans le graphe en effectuant des choix soumis à des probabilités sur les arcs à traverser. En effet, la construction de leur chemin est biaisée en favorisant les arcs fortement marqué de phéromone. Ensuite, selon la valeur de la fonction-objectif obtenue, les taux de phéromones sont mis à jour [**Prodhon, 2008].**

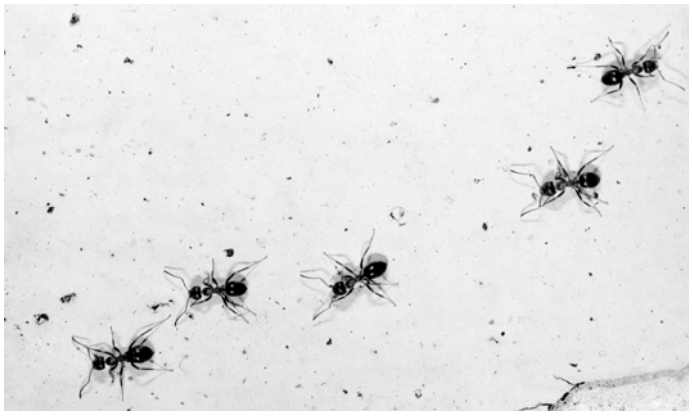


Figure 2.2 : Des fourmis suivant une piste de phéromone [Dréo, 2003]

2.3 L'intensification/diversification

Les métaheuristiques sont analysées au travers de composants d'intensification / diversification**,** l'intensification est une recherche guidée par la fonction objectif alors que la diversification est fondée sur l'aléatoire ou une fonction autre que la fonction objectif.

L'intensification favorise l'exploration des zones de l'espace de recherche où des solutions de bonne qualité ont été trouvées. La diversification permet de déplacer la recherche dans les zones non explorées de l'espace de recherche. L'intensification et la diversification comme étant deux besoins conflictuels dont l'équilibre est essentiel au bon fonctionnement d'une heuristique [**Meignan, 2008].** Ne pas préserver cet équilibre conduit à une convergence trop rapide vers des minima locaux (manque de diversification) ou à une exploration trop longue (manque d'intensification).

Selon Sevaux (2004) l'intensification et la diversification dans les algorithmes évolutionnaires peuvent être décrites selon plusieurs points de vue, les opérateurs de mutation et de croisement permettent d'explorer l'espace de recherche. Ces opérateurs agissent de manière aléatoire sur les individus sans nécessiter l'utilisation de la fonction objectif. Ainsi, la diversification peut être associée à ces deux types d'opérateurs. L'intensification est effectuée par la sélection des individus. Cette sélection est fondée sur la fonction de fitness des individus (évaluation des solutions) et consiste à favoriser la conservation des meilleurs individus.

Le principe d'intensification par la sélection de solutions est connu sous le nom de pression sélective. Certaines études ne partagent pas ce point de vue où l'opérateur de croisement est uniquement associé à l'exploration de l'espace de recherche. L'objectif de cet opérateur est parfois énoncé comme étant la combinaison des meilleures composantes des individus. À ce titre, le croisement peut être considéré comme étant un opérateur d'intensification [**Meignan, 2008].**

Toutes les métaheuristiques que nous venons de présenter utilisent une ou plusieurs solutions pour réaliser la recherche, elles se basent sur un mécanisme d'évaluation de solution qui correspond à la fonction objectif et d'un mécanisme d'évaluation réalisé grâce à un ensemble d'opérateurs. Malgré que les métaheuristiques hybrides soient apparues en même temps que le paradigme lui-même, ce n'est que ces dernières années qu'elles ont connu leur succès, l'algorithme mémétique et la recherche dispersée font partie de ces méthodes. Ces deux méthodes, présentées en détail dans ce chapitre, sont celles qui ont fait l'objet de notre étude.

2.4 Présentation des algorithmes génétiques

Depuis l'idée originale proposée par Holland (1975), le développement des algorithmes génétiques dans les trois dernières décennies a connu un essor considérable. Les algorithmes génétiques sont des techniques de recherche basées sur les mécanismes de la sélection naturelle et de la génétique. Ils se sont avérés très efficaces dans la résolution des problèmes complexes d'optimisation, tels que le problème du voyageur de commerce ou les problèmes d'ordonnancement [**Boukef, 2010].**

Pour mettre en œuvre un algorithme génétique, on doit disposer des cinq éléments suivants :

- Un mécanisme de génération de la population initiale : Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global.
- Un principe de codage de l'élément de population, c'est-à-dire un codage de solutions utilisé sous la forme de chromosomes (La qualité du codage des données conditionne le succès des algorithmes génétiques).
- Une fonction à optimiser : L'application de cette fonction à un élément de la population donne sa fitness, une fonction qui permet d'évaluer l'adaptation d'un chromosome à son environnement, ce qui offre la possibilité de comparer des individus.
- La sélection est basée sur la reproduction et sur le codage génétique, qui stocke les informations décrivant l'individu sous forme de gènes. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'´etats.

- L'algorithme utilise des paramètres de dimensionnement : taille de la population, probabilité de croisement et de mutation, nombre total de générations.

2.4.1 Les étapes de l'algorithme génétique

Le schéma de la Figure 2.3 illustre la structure générale d'un AG. L'algorithme commence par générer une population initiale qui se compose d'un nombre déterminé d'individus. La meilleure solution peut ne pas se trouver dans cette population initiale. À chaque génération, une succession d'opérations de sélection, de croisement, de mutation, d'évaluation et de remplacement est appliquée aux individus de la population, afin de produire la génération suivante. Lorsque le critère d'arrêt est atteint, la meilleure solution trouvée est sélectionnée et l'algorithme s'arrête.

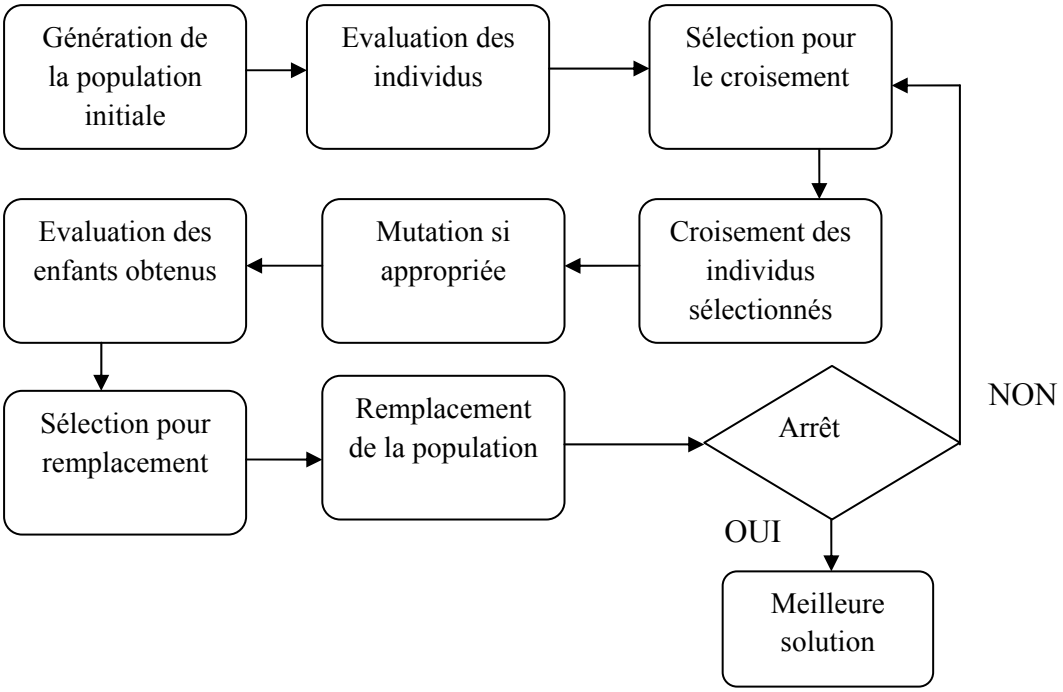


Figure 2.3: Structure générale d'un algorithme génétique **[Lemamou, 2009].**

- **Codage**

Le premier pas dans l'implantation des algorithmes génétiques est de créer une population d'individus initiaux. Chaque individu de la population est codé par un chromosome. Une population est donc un ensemble de chromosomes.

Généralement, un chromosome est une chaine ou un ensemble de chaines, souvent de bit ou d'entiers. Chaque chromosome code un point de l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome.

- **Evaluation** : fitness

Les performances de chaque individu sont évaluées en utilisant une fonction d'évaluation. Cette fonction permet d'évaluer la capacité d'un individu à survivre en lui affectant un poids

appelé fitness. Cette fitness est calculée afin que les plus forts soient retenus dans la phase de sélection, puis modifiés dans la phase de croisement et mutation.

- **La sélection**

Cet opérateur est chargé de favoriser les meilleurs individus. La sélection joue un rôle très important dans les algorithmes génétiques: d'une part, pour diriger les recherches vers les meilleurs individus et d'autre part, pour maintenir la diversité des individus dans la population. Elle est liée au compromis entre la vitesse de convergence élevée et une forte probabilité de trouver un optimum global dans le cas d'un problème d'optimisation. Si la sélection choisit seulement le meilleur individu, la population convergera rapidement vers cet individu.

- **Croisement**

L'opérateur de croisement, appelé aussi recombinaison, est l'instrument majeur des innovations dans l'algorithme génétique. Il a pour objectif d'enrichir la diversité de la population en manipulant la combinaison des chromosomes. Généralement, le croisement se fait en combinat deux parents et génère deux enfants qui vont hériter certaines caractéristiques de leurs parents et qui peuvent se rapprocher de la solution optimale.

Le croisement peut être simple, à un seul point, ou multiple, à deux ou plusieurs points. Dans le premier cas, se croisent en un seul point (figure 2.4). Dans le deuxième cas, les chromosomes se croisent en deux ou plusieurs points, et s'échangent de plusieurs portions.

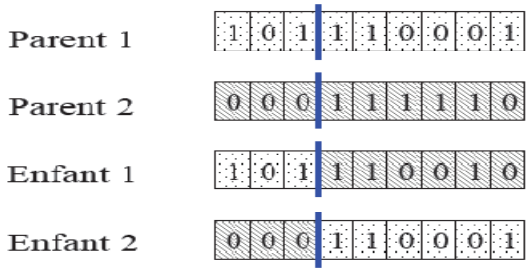


Figure 2.4 : Exemple d'un croisement (*Le croisement génère des individus qui sont le résultat du mélange de plusieurs solutions).*

- **Mutation**

Cet opérateur a pour but de diversifier la population afin d'examiner d'autres parties de l'espace de recherche. Il consiste à remplacer, de façon aléatoire, un gène au sein d'un chromosome par un autre, c'est donc une modification légère dans le chromosome muté. La mutation a pour rôle d'éviter la convergence trop rapide de l'algorithme vers un minimum local. La figure 2.5 montre un exemple de mutation.

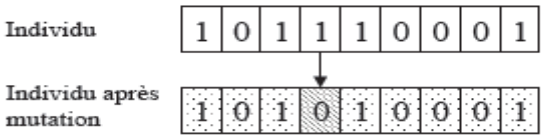


Figure 2.5 : Exemple d'une mutation (*la mutation est une modification aléatoire d'un individu*).

- **Phase de remplacement**

Une fois qu'on a généré de nouveaux individus par croisement et par mutation, il faut décider de la constitution de la nouvelle population, le remplacement décide quels individus conserver et insérer dans la nouvelle population.

- **Critère d'arrêt**

Le test d'arrêt joue un rôle très important dans le jugement de la qualité des individus. En général, l'algorithme s'arrête au bout d'un temps ou d'un nombre de générations donné **[Rebreyen, 1999].**

Les résultats obtenus par les AG sont en principe de bonne qualité mais souvent, pour le rendre plus performant en terme de qualité des solutions obtenues, on lui associe des procédures de recherche locale, on parle alors des algorithmes mémétiques (Memetic Algorithm - MA).

Une nouvelle génération été proposée par Sörensen et Sevaux (2003). Il s'agit d'une forme agrémentée d'une gestion de la population de solutions. Cette version s'appelle MA/PM pour Memetic Algorithm with Population Management. Elle utilise une mesure de distance afin d'apporter une certaine diversité dans les chromosomes-parents.

2.5 Généralité sur l'algorithme mémétique avec gestion de population

2.5.1. Algorithmes mémétiques

Dans les cas où l'obtention d'une solution réalisable n'est pas suffisant, mais le but est d'obtenir une solution de bonne qualité, il est intéressant d'introduire des procédures qui permettent l'amélioration des solutions, c'est le cas des algorithmes mémétiques.

Les algorithmes mémétiques sont introduits pour la première fois par Moscato en 1989. Fondamentalement, ils hybrident les algorithmes génétiques avec des méthodes de recherche locale.

La combinaison qui semble la plus fructueuse est celle qui utilise une méthode de recherche locale dans un algorithme génétique. On remarque que les algorithmes génétiques peuvent être une bonne solution pour résoudre des problèmes d'optimisation combinatoire. Cependant, un inconvénient d'un algorithme génétique est que les opérateurs standards de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche comme le rappellent **[Hoos et Stützle, 2004]**. L'opérateur de mutation apporte une légère modification à l'individu. Son rôle est de favoriser la diversification des individus alors que la sélection se charge de conserver les meilleurs. C'est pourquoi les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale.

Ces deux méthodes sont complémentaires car l'une permet de détecter de bonnes régions dans l'espace de recherche alors que l'autre se concentre de manière intensive à explorer ces zones de l'espace de recherche **[Moscato, 1999]**.

2.5.2 Principe des méthodes de recherche locale

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 (et 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 N(x)* de cette solution. Déterminer une solution voisine de x dépend bien entendu du problème traité **[Sevaux, 2004]**.

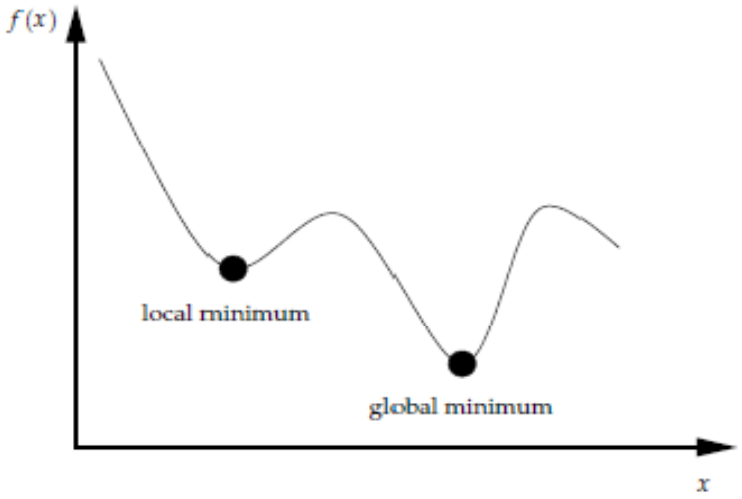


Figure 2.6 : Minimum local et global **[Sevaux , 2004]**

De manière générale, les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Dans un algorithme mémétique la méthode de recherche locale utilisée n'est pas unique, on peut utiliser une méthode de recherche locale simple telle que les méthodes de descente, ou des méthodes plus évoluées telles que le recuit simulé ou la recherche tabou. Dans notre travail, nous avons utilisé une méthode de descente, dans la section suivante, nous allons présenter le principe des méthodes de descente.

2.5.2.1 Méthodes de descente

A partir d'une solution générée initialement ou trouvée par heuristique par exemple, on peut très facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes autour du même principe. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle est meilleure que la solution courante.

L'algorithme 2.2 présente le squelette d'une méthode de descente (simple descente). A partir d'une solution initiale x, on choisit une solution x_0 dans le voisinage *N(x)* de x. Si cette solution est meilleure que x, ($f(x_0) < f(x)$) alors on accepte cette solution comme nouvelle solution x et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de x.

Algorithme 2.2: La méthode de descente

1. initialisation : trouver une solution initiale x
2. Répéter
3. Recherche dans le voisinage : trouver une solution $x' \in N(x)$
4. Si $f(x') < f(x)$ alors
5. $x \leftarrow x'$
6. Fin si
7. Jusqu'à $f(y) \geq f(x), y \in N(x)$

2.5.3 Gestion de population : *(Population management)*

Malgré la performance et l'efficacité apportées par l'introduction de la recherche locale au niveau de l'algorithme génétique, l'algorithme mémétique peut présenter certaines faiblesses telles que la convergence prématurée.

L'un des points importants dans les algorithmes évolutionnaires est le maintien de la diversité dans la population, dans les algorithmes génétiques classiques cette diversité est maintenue par l'opérateur de mutation. Le contrôle de la diversification est géré par une nouvelle classe des algorithmes évolutionnaires proposés par Sorensen et connus sous le nom d' « algorithmes mémétiques avec gestion de la population ».

La gestion de population est une procédure ayant pour but la mesure et le contrôle de la diversité de la population considérée. Trois points sont à noter :

- La population considérée est de taille petite, car plus la population est grande on tendance à avoir plus de ressemblance des solutions,
- Les solutions examinées de la population sont de bonne qualité, car la mesure de la diversité n'est appliquée dans l'algorithme qu'après l'application de la procédure de la recherche locale.
- La manière de mesurer la distance dans ces algorithmes dépend de la représentation des solutions considérées (le codage des solutions).

- **Distance d'une solution à la population** :
  Etant donné une mesure de distance qui peut calculer la distance entre tout couple de solutions, la distance d'un s_k solution donnée à la population peut être calculée comme suit:

$$d_p(s_k) = \min_{s_i \in p}(s_k, s_i) \qquad (1)$$

  Le calcul de la distance d'une solution à la population exige de calculer | P | mesures de distance (où | P | est le cardinal de la population). Cette haute exigence de calcul est l'une des raisons pour lesquelles les techniques de gestion de la population sont plus efficaces lorsqu'elles sont appliquées à de petites populations **[Sörensena, Sevauxb, 2004].**
- **Le paramètre de la diversité**:
  Il est évident qu'une solution qui a une petite distance à l'autre solution déjà dans la population, ne contribue pas beaucoup à la diversité d'une population. Par conséquent, une solution n'est pas ajoutée à la population si sa distance à la population est en dessous d'un certain seuil Δ.

Nous appelons Δ le paramètre de la diversité. En supposant que la qualité de s_k est suffisante, une solution peut être ajoutée à la population si les conditions suivantes sont vérifiées:

$$d_p(s_k) = \min_{s_i \in p}(s_k, s_i) \geq \Delta \qquad (2)$$

En utilisant la distance $d_p(s_k)$ et la valeur da la fitness (la fonction objectif) $f(s_k)$, la fonction d'entrée peut également utiliser une méthode de décision multi-objectifs afin de déterminer si s_k doit être ajoutée à la population ou non. Un moyen très simple de faire ceci (en supposant que *f* doit être minimisée) est de calculer $(s_k + \lambda d_p(s_k))$. Si cette valeur ne dépasse pas un certain seuil, la solution est ajoutée, si non elle est rejetée. λ est un paramètre qui détermine l'importance relative de la diversité par rapport à la qualité de la solution. Bien sûr, plusieurs autres méthodes de décision multiobjectif peuvent être utilisées. Si la procédure de recherche locale est suffisamment efficace pour assurer en permanence la qualité des solutions qu'elle produit, une solution peut être ajoutée si l'équation (2) est vérifiée sans prendre la valeur de la fonction objectif de la solution en compte.

Comme le montre l'algorithme 2.3, une solution qui n'a pas une distance suffisamment grande pour la population, est mutée au hasard jusqu'à ce qu'elle atteint cette distance. Bien sûr, d'autres stratégies sont possibles, comme tout simplement rejeter la solution **[Sörensena, Sevauxb, 2004].**

Algorithme 2.3 : Algorithme mémétique avec gestion de population (MA/PM)

1. **Initialiser** : générer une population initiale P de solutions
2. Fixer le paramètre de la diversité Δ
3. **Répéter**
4. Sélection : choisir deux solutions p_1 et p_2
5. Croisement : combiner deux solutions parents p_1 et p_2 pour former des Solutions c_1 , c_2
6. Recherche locale : appliquer une procédure de recherche locale sur c_1, c_2
7. **Pour** tout enfant *c* faire
8. **tant que** *c* ne satisfait pas la condition d'addition $(d_p(c) < \Delta)$ Faire
9. Mutation(c*)*
10. **Fin tant que**
11. Supprimer la solution de la population
12. Addition: de solution : $\boldsymbol{p} \leftarrow \boldsymbol{p} \cup \boldsymbol{c}$
13. **Fin pour**
14. Mettre à jour le paramètre de diversité Δ
15. **Jusqu'à** satisfaire un critère d'arrêt.

2.5.4 Principe du MA/PM

La principale caractéristique du MA/PM vient de la gestion de la population grâce à une mesure de distance dans l'espace des solutions. Cette opération permet de contrôler la diversité des individus en filtrant l'entrée des enfants dans la population. Ce principe étant appliqué dans la suite au problème d'ordonnancement d'un système flexible de production. Il est basé sur un algorithme génétique mais se différencie des versions classiques par deux principaux éléments :

1. La méthode de recherche locale utilisée pour une amélioration des solutions (Memetic Algorithm - MA),
2. Une technique de gestion de la population par une mesure de distance $d_p(S)$ a une petite population *P* de solutions de bonne qualité (Population Management - PM);

Pour obtenir une métaheuristique plus performante il faut maintenir un équilibre entre l'intensification et la diversification par la gestion de la population PM, cette procédure pouvait contrôler l'équilibre entre ces deux stratégies. Sörensen et Sevaux proposent deux options : soit l'enfant est muté jusqu'à ce que sa distance à la population atteigne au moins le seuil Δ, soit il est tout simplement rejeté. Dans le premier cas, plusieurs mutations successives peuvent être nécessaires et le temps passé pour obtenir un enfant admissible dans la population peut être parfois important.

Le PM signifie qu'une nouvelle solution *s* ne peut intégrer la population courante que si sa distance $d_p(S)$ à la population courante *P* est telle que $d_p(S) \geq \Delta$, avec Δ un seuil donné.
Si Δ=0, l'algorithme se comporte comme un classique MA (sans gestion de la population). Δ≥1 garantit que chaque solution de la population P est distincte des autres (la distance définie est entière). Par contre, si Δ est fixé à une grande valeur, alors la majorité des solutions-enfants est rejetée et l'algorithme passe beaucoup de temps dans des itérations improductives. La valeur de Δ peut être alors ajustée dynamiquement entre ces deux extrêmes afin de contrôler la diversité de la population **[Prodhon, 2008].**

Sörensen et Sevaux (2003) proposent différentes politiques de contrôle pour Δ comme la montre la figure 2.7 :

- stratégie 1: Δ constant : garantit un niveau constant de diversification;
- stratégie 2: Initialiser Δ avec une grande valeur, puis décroître doucement au fur des itérations; favorise une intensification en fin d'algorithme;
- stratégie 3: Initialiser Δ avec faible valeur, puis l'augmenter au fil des itérations, ceci permet une diversification en fin d'algorithme;
- stratégie 4: Initialiser Δ avec une grande valeur au début de l'algorithme, puis décroître doucement tant que des solutions améliorantes sont trouvées, sinon, après un certain nombre d'itérations sans amélioration, ré-augmenter la valeur de Δ afin d'introduire de la diversification dans la population : stratégie adaptative ajustant dynamiquement la valeur de Δ pour contrôler la diversité de population.

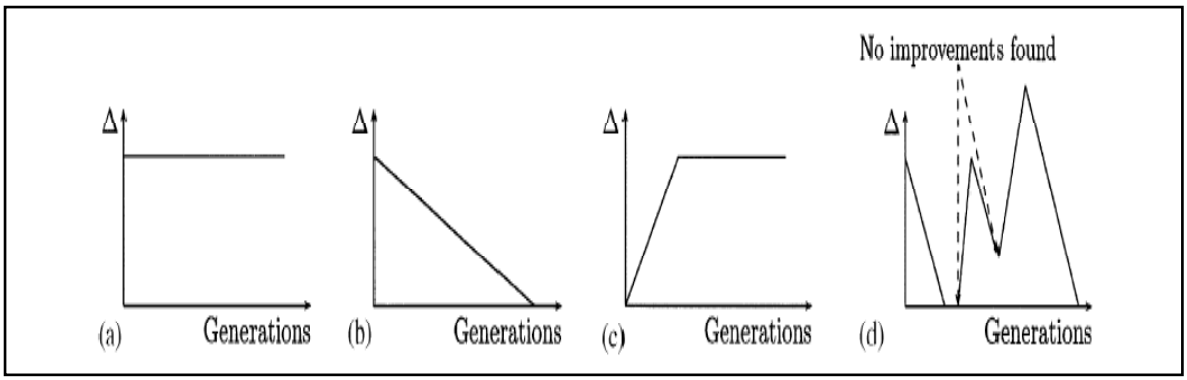


Figure 2.7 Les différentes stratégies de la gestion de population. (a) stratégie 1; (b) stratégie 2; (c) stratégie 3 et (d) stratégie 4.

Dans la version proposée ici pour l'ordonnancement du système flexible de production, c'est la première stratégie qui est retenue.

2.6 La recherche dispersée *(Scatter Search)*

La recherche dispersée (Scatter Search) est une méthode d'évolution qui a été proposée par Glover en 1977, qui s'inspire du principe des algorithmes génétiques, elle est basée sur une population de solutions (vecteurs d'entiers) qui évolue dans le temps à l'aide à la fois d'un opérateur de sélection, de la combinaison linéaire de solutions de la population pour créer une nouvelle solution provisoire (non forcément entière ou admissible), d'un opérateur de projection permettant de rendre la solution provisoire admissible et d'opérateurs d'élimination. Cette métaheuristique a été appliquée avec succès pour les problèmes d'optimisation difficiles. Elle a donné de bons résultats pour les problèmes d'optimisation combinatoire **[Glover, 1998].**

La recherche dispersée du point de vue classification des métaheuristiques est une méthode d'évolution qui est basée sur la population. Elle construit des solutions par la combinaison d'autres solutions en se basant sur des principes qui capturent l'information qui n'existe pas dans les solutions originales. Elle prend l'avantage de quelques méthodes heuristiques pour sélectionner les éléments qui vont être combinés pour générer une nouvelle solution.

Contrairement aux autres méthodes d'évolution telles les algorithmes génétiques **[Jambu, 1999]**, la recherche dispersée est fondée sur des conceptions systématiques et des méthodes pour créer de nouvelles solutions. Elle offre des avantages considérables au-delà de ceux dérivées de recours pour randomisation, comme dans le cas des algorithmes génétiques. La recherche dispersée utilise des stratégies pour diversifier et intensifier la recherche qui ont prouvé leur efficacité dans des variétés des problèmes d'optimisation **[Benalia, 2005].**

2.6.1 La différence entre l'algorithme génétique et la recherche dispersée

Tout comme les algorithmes génétiques, la recherche dispersée travaille sur un ensemble de solutions. La différence, à ce niveau, est la taille de cet ensemble. Pour les algorithmes génétiques, la taille de la population est en moyenne de 100 individus. Dans la recherche dispersée, la taille de la population est plus petite, elle est de l'ordre de 40 solutions. Une autre différence se situe aussi au niveau du nombre de solution à combiner pour former de nouvelles solutions. Cette méthode peut combiner de deux à cinq éléments, ce qui permet la génération d'un nombre important de solutions.

Les algorithmes génétiques utilisent des opérateurs aveugles d'où leur rapidité, alors que la recherche dispersée utilise des opérateurs « intelligents » qui peuvent prendre en considération la nature du problème d'où leur lenteur. Pour éviter une convergence prématurée vers un optimum local, la recherche dispersée gère différemment sa population : Elle consiste à générer un ensemble de solutions dispersées à partir d'un ensemble R de solutions de références.

2.6.2 L'algorithme de la recherche dispersée

La recherche dispersée est une métaheuristique qui est basée sur l'évolution de la population. Durant la recherche, elle capture l'information dans les sous ensembles pour construire de nouvelles solutions, et elle conserve les deux grandeurs (diversité, qualité) de l'ensemble de référence. La recherche dispersée utilise des stratégies pour diversifier et

intensifier la recherche et elle a prouvé son efficacité dans des variétés de problèmes d'optimisation.

La recherche dispersée se décompose en deux phases importantes qui sont: la phase d'initialisation et la phase d'évolution. Les étapes de ces deux phases sont données comme suit **[benatchba, 2005**] :

Phase d'initialisation :

Etape 1 : Générer les solutions de départ : Génère aléatoirement une ou plusieurs solutions d'essai de départ qui seront utilisées pour initialiser le reste de la méthode de recherche.

Etape 2 : Génération des solutions diversifiées : Utiliser la méthode de génération de diversification pour générer des solutions dispersées à partir de la solution de départ.

Etape 3 : Construction de l'ensemble de référence: Pour chaque solution d'essai générée dans l'étape 2, utiliser la méthode d'amélioration pour créer un ou plusieurs solutions plus améliorées. Pendant l'application successive de cette étape, maintenir et faire la mise à jour de l'ensemble de référence qui est constitué par les *b* meilleures solutions trouvées.

Etape 4 : Répéter l'exécution de l'étape 2 et 3 jusqu'à l'obtention d'un nombre voulu de solutions qui constitueront la population initiale de l'ensemble de référence.

Phase d'évolution :

Etape 5 : La génération des sous ensembles : Générer des sous ensembles de la population de l'ensemble de référence à partir desquels seront construites les solutions combinées.

Etape 6 : Combinaison des solutions : pour chaque sous ensemble *X* produit à l'étape 5 on applique une technique de combinaison des solutions pour produire l'ensemble *C(X)* qui consiste en une ou plusieurs solutions combinées. Puis traiter chaque élément de *C(X)* individuellement dans l'étape suivante.

Etape 7 : Amélioration et mise à jour de l'ensemble de référence : pour chaque solution produite à l'étape 6 appliquer la méthode d'amélioration pour avoir une (ou plusieurs) solution(s) de qualité supérieure et faire la mise à jour de l'ensemble de référence.

Etape 8 : Si aucune solution n'est produite à l'étape 6, alors on réinitialise la population avec l'ensemble élit et on reprend à partir de l'étape 2.

Etape 9 : Répéter l'exécution des étapes de 5 à 8 jusqu'à la rencontre d'un critère d'arrêt qui est le plus souvent un nombre maximum d'itérations atteint.

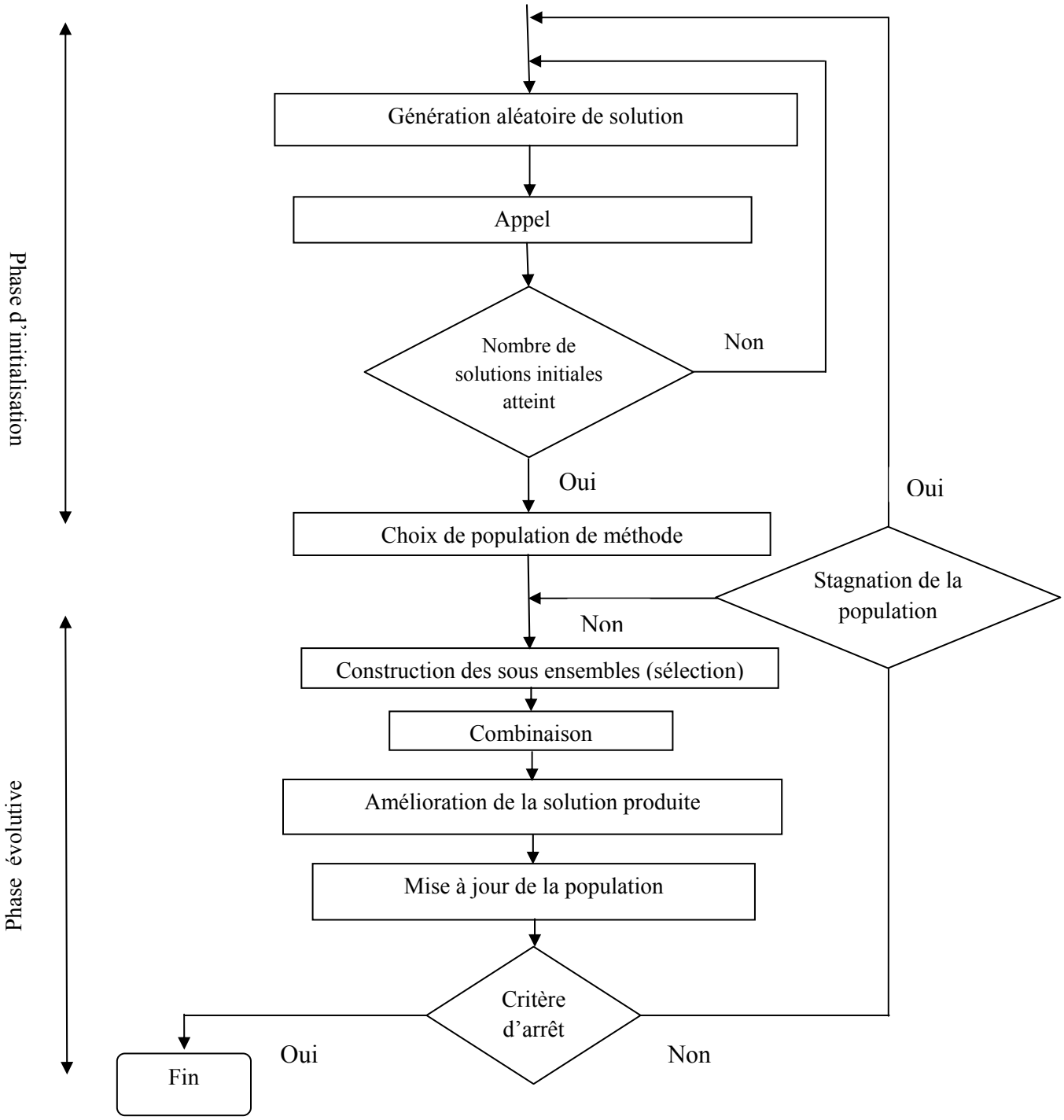


Figure 2.8 : Etapes de la recherche dispersée

2.6.3 Gestion de la population et l'ensemble de référence

La recherche dispersée gère sa population différemment des algorithmes génétiques. Dans la phase d'initialisation, elle démarre d'une population variée P de solutions.

Ensuite, un ensemble de référence *b* de solutions est choisie parmi la population courante pour participer à la phase de reproduction et produire d'autres solutions, pour chaque élément de cette population, le degré d'adaptation (fonction objectif) est calculé, ce qui permet d'extraire deux sous-ensembles.

L'ensemble de référence de la recherche dispersée *b* contient, d'une part, un certain ensemble élite (de taille b_1) de bonnes solutions choisies selon leur fonction objectif. D'autre part, un certain ensemble diversifié (de taille b_2) de solutions sont tirées de la population restante $P-b$ et rajoutées à la collection *b* pour la compléter. Les b_2 solutions sont les solutions les plus éloignées des meilleures solutions de *b*, elles sont appelées : solutions diversifiées. Après avoir choisi un ensemble de solutions bonnes et diversifiées, la phase de reproduction est lancée.

La distance entre un individu et une population peut-être définie comme étant la distance minimale séparant cet individu de chaque individu de la population élite. Ce processus permet de maintenir une certaine diversité dans la population et évite de rester bloqué dans un optimum local. Il permet, par conséquent d'éviter une stagnation rapide de l'algorithme, ce qui constitue un des inconvénients majeurs de l'algorithme génétique **[Benatchba, 2005**].

La recherche dispersée construit l'ensemble de référence à partir de la population initiale. Typiquement, la taille de la population initiale est dix fois plus grande que la taille de l'ensemble de référence. Pour initialiser sa population, la recherche dispersée génère des solutions aléatoires. À partir de chaque solution elle génère d'autres solutions qui lui seront très distantes en faisant appel au **générateur de diversification**, et ce jusqu'à atteindre un nombre maximum de solutions initiales.

2.6.4 Méthode de génération de diversification

Les solutions générées dépendent potentiellement de la méthode de génération de diversification. Au cours des itérations, les solutions très proches donnent des solutions identiques après l'application des méthodes d'amélioration. Dans l'application de la recherche dispersée sur des variétés de problèmes, plusieurs types de générateurs de diversifications ont été implémentés. Ces types se basent sur des méthodes aléatoires, des méthodes déterministes, ou des métaheuristiques. Ces dernières peuvent être comme l'utilisation des recherches à base de mémoire comme recherche local, telle que la recherche taboue.

2.6.5 Mise à jour de l'ensemble de référence

Rappelons que la population initiale, à partir de laquelle sera formé l'ensemble de référence, est générée aléatoirement. Puis on fait appel à un générateur de diversification pour construire des solutions assez diverses qui couvrent une partie de l'espace de recherche. Cette population contient des individus de « bonne qualité » et des individus « différents ».

L'ensemble de référence contient deux sous ensembles de la population initiale, le premier sous ensemble, contient les meilleurs individus de la population initiale dont on note le cardinal par b_1. La population initiale devient $P= P - b_1$ ensemble de référence.

Le deuxième ensemble regroupe les individus de la population les plus différents. Soit b_2 son cardinal. Le minimum des distances euclidiennes (Δ_{min}) entre les éléments de P et de l'ensemble de référence est calculé. Les éléments ayant les distances les plus élevées seront intégrés dans ce sous ensemble. Ces éléments sont éliminés de P au fur et à mesure.

L'ensemble de référence évolue dynamiquement durant la phase de combinaison des solutions. Une solution produite peut faire partie de l'ensemble de référence si :

- Sa fonction objectif est meilleure que celle de la moins bonne solution de deuxième sous ensemble. Cette solution remplace donc la moins bonne.

- Sa fonction objectif est plus « éloignée » que celle des élément de le premier sous ensemble. (Δ_{min} de cette solution est meilleure que la solution ayant le mauvais Δ_{min})

2.6.6 Méthodes de combinaison par l'application des opérateurs génétiques

La mutation et le croisement sont des opérateurs conçus pour les algorithmes génétiques, mais il est possible de les utiliser comme des méthodes de combinaison des solutions pour la recherche dispersée **[Cavique, 2003], [Benalia, 2005]**.

2.6.7 L'algorithme général

L'algorithme suivant présente un squelette du scatter search **[Sevaux, 2004] :**

Algorithme 2.4 : La recherche dispersée

1. **Initialiser** : générer une population initiale P de solutions
2. Mettre les meilleures solutions trouvées dans unes trouvées dans une population c référence **R**
3. **Tant que** le critère d'arrêt non satisfait faire
4. A=R → prendre la population de référence
5. **Tant que** A≠∅ faire
6. Combiner les solutions (croisement)(B←R×A)
7. Améliorer les solutions B
8. Mettre ajoure les solutions R
9. Garder les meilleures solutions à partir R ∪ B
10. A←B−R
11. **Fin tant que**
12. **Enlever** de R la mauvaise solution
13. Ajouter des nouvelles solutions diverses dans R
14. **Fin tant que**

À chaque itération on garde (celles de la boucle externe) un ensemble de référence qui soit à la fois composé de bonnes solutions et de solutions diverses. Les facteurs d'intensifications se retrouvent dans l'application systématique d'une méthode de recherche locale (lignes 1 et 7) ainsi que dans la combinaison répétée des solutions (boucle des lignes 5 à 10) et dans la mise à jour de *R* (ligne 8), et les facteurs de diversification se retrouvent dans le choix des sous ensembles diverse (ligne 2) et le remplacement d'une partie de la population à chaque étape (ligne 12).

2.7 Conclusion

Une introduction au paradigme des méthodes évolutionnaires a été présentée dans ce deuxième chapitre. Ces méthodes inspirées des mécanismes de l'évolution naturelle connaissent un développement très important dû à la source d'inspiration géniale et immense que constitue l'évolution naturelle mais aussi aux bonnes performances qu'elles obtiennent sur des problèmes d'optimisation combinatoire. Cette présentation a été orientée vers leur utilisation pour la résolution de problèmes d'ordonnancement. Nous avons présenté les deux méthodes que nous avons adaptées pour la résolution de notre problème, à savoir l'algorithme mémétique avec gestion de population et l'algorithme de recherche dispersée.

Les problèmes d'ordonnancement sont, dans le cas général, des problèmes de la classe NP-difficile. Dans ce domaine des problèmes dits " combinatoires ", il n'existe pas d'heuristique qui soit meilleure qu'une autre, pour cela on adapte de ces deux algorithmes pour la résolution d'un

problème d'ordonnancement dans un système flexible de production (FMS), et on consacre aux résultats obtenus et leurs interprétations. Les résultats obtenus seront comparés à ceux de l'algorithme génétique et d'essaim particulaire par Souier (2009), pour résoudre le même problème.

Chapitre 3

Apport des algorithmes évolutionnaires pour l'ordonnancement d'un FMS

Ce chapitre présente le modèle FMS étudié, les algorithmes des métaheuristiques adaptées à la résolution de notre problème en temps réel ainsi que les résultats obtenus et leurs interprétations après plusieurs simulations du modèle.

Sommaire

L'Automatique a été introduite en production au début du 20ème siècle et elle a contribué à accroître la productivité et la qualité, tout en diminuant le coût. Dans les années 1970

et 1980, la flexibilité est devenue essentielle dans les systèmes de productions pour répondre à la variété de produits, des lots des productions plus petites et des changements de modèles.

Plusieurs travaux ont été réalisés dans le cadre de l'ordonnancement d'un FMS depuis les années 80. De nombreux travaux dans la littérature se sont intéressés à l'influence de la flexibilité de routage sur les performances d'un FMS avec et sans pannes de machines. La flexibilité de routages offre au système les moyens d'un aiguillage plus souple, de façon à servir les différents segments de procédés libres ou sous engagés.

Le problème de sélection du routage optimal consiste à déterminer le meilleur routage tout en minimisant le coût de production en prenant en considération le volume de production, les alternatives associées au processus de mise en production, ainsi que la disponibilité et la capacité des machines.

La performance d'un système de fabrication flexible (FMS) est fortement dépendante de la sélection de la méthode d'ordonnancement pour le système de commande. Une telle méthode permet d'exploiter autant que possible la flexibilité du système. La politique d'ordonnancement, peut être exprimée par une règle de répartition choisie, détermine quel travail sur les tâches en attente d'être traitées devrait être sélectionné en premier. Les règles de répartition utilisées peuvent changer dynamiquement et reflètent le statut de l'atelier, il y a trois étapes de prise de décision dans un FMS: les décisions de conception, de planification, d'ordonnancement et de contrôle.

L'augmentation d'automatisation et la complexité des systèmes de fabrication ont souligné la nécessité pour le développement de la planification et l'amélioration des techniques d'ordonnancement pour les systèmes de fabrication flexibles (FMS). Un ensemble considérable de documents de recherche s'est accumulé dans ce domaine depuis la fin des années 1980 lorsque les premiers articles ont été publiés, période dans laquelle les systèmes flexibles de production ont été adoptés par les pays industriels **[Saygin et Kilic,1995], [Peng et Chen, 1998]**, **[Souier, 2009]** mais beaucoup d'études dans le contrôle et l'ordonnancement des FMS en temps réel ne prennent pas en considération la flexibilité de routages alternatifs **[Kazerooni et *al.*, 1997]**, **[Souier, 2009]** et la plupart des études qui prennent en compte ce point, règlent le problème de la sélection de routages avant le début de la production **[Das et Nagendra, 1997]**, **[Cho et Wysk, 1995]**, **[Souier, 2009]**.

Cette approche n'est pas applicable pour les systèmes flexibles de production aléatoires **[Rachamadugu et Stecke, 1994]**, où on ne peut pas prévoir l'arrivée ou l'entrée des pièces dans le système avant le début de la production. Car les routages des pièces peuvent être différents même pour des pièces de même type **[Rachamadugu et Stecke, 1994]**. Ainsi le système de commande d'un FMS aléatoire est obligé d'utiliser effectivement et efficacement la flexibilité des opérations et de routages en temps réel pour avoir la capacité de s'adapter, avec l'arrivée aléatoire de pièces et d'événements imprévus **[Mamalis et *al.*, 1995]**, **[Rachamadugu et Stecke, 1994].**

Ce chapitre est organisé de la manière suivante. La section 3.1 introduit notre modèle FMS étudié ; la section 3.2 et 3.3 décrit une présentation sur les algorithmes des métaheuristiques ainsi que leurs différentes étapes. L'adaptation de ces approches est donnée dans la section 3.4.

Finalement, les sections 3.5 et 3.6 évaluent une étude comparative sans et avec introduction de pannes, et illustre une analyse de sensibilité de l'algorithme de la recherche dispersée. Nous proposons dans le reste de ce chapitre une conclusion qui récapitule les résultats.

Les résultats des métaheuristiques utilisées comme base de comparaison dans ce travail sont ceux trouvés par Souier **(2009)**.

3.1 Présentation du modèle FMS étudié

Le système étudié est composé de sept machines et deux stations : l'une de chargement et l'autre de déchargement, définies comme suit :

- Deux fraiseuses verticales (FV).
- Deux fraiseuses horizontales (FH).
- Deux tours (T).
- Une toupie (TP).
- Une station de chargement(SC).
- Une station de déchargement (SD).

Chacune des machines a une file d'attente d'entrée et une file d'attente de sortie, la station de chargement a aussi une file d'attente d'entrée. Le système traite six types de pièces différentes. Chaque type de pièces a un certains nombre de routages.

Les opérations sur le système de production étudié sont basées sur les suppositions suivantes :

- Chaque type de pièce a des routages alternatifs connus avant le début de la production.
- Le temps de traitement de chaque type de pièce sur une machine est connu, et il comprend le temps de changement des outils et le temps d'exécution de la machine.
- Le temps de traitement d'une opération est le même sur les machines alternatives identifiées pour cette opération.
- Chaque machine ne traite qu'une seule pièce à la fois.

Le système de production étudié peut être présenté comme suit :

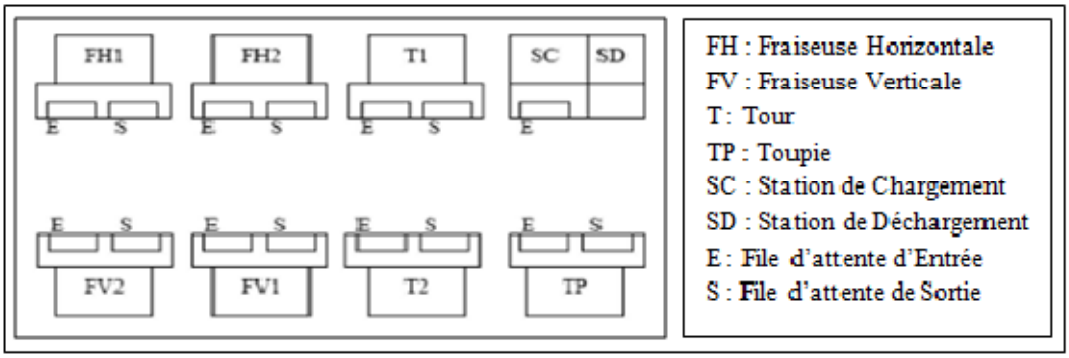


Figure 3.a: Configuration du modèle FMS étudié **[Saygin, 2001].**

Le tableau 3.a présente les différents types de pièces et leurs taux d'arrivée au système, ainsi que leurs routages possibles avec le temps de traitement sur chaque machine.

Type des pièce	Taux d'arrivée	Les numéros de routages	Routages et temps de traitement (min)
A	17%	1	SC-T1(30)-FV1(20)-SD
		2	SC-T1(30)-FV2(20)-SD
		3	SC-T2(30)-FV1(20)-SD
		4	SC-T2(30)-FV2(20)-SD
B	17%	5	SC-T1(20)-TP(1)-FV1(15)-SD
		6	SC-T1(20)-TP(1)-FV2(15)-SD
		7	SC-T2(20)-TP(1)-FV1(15)-SD
		8	SC-T2(20)-TP(1)-FV2(15)-SD
C	17%	9	SC-T1(40)-TP(1)-FV1(25)-SD
		10	SC-T1(40)-TP(1)-FV2(25)-SD
		11	SC-T2(40)-TP(1)-FV1(25)-SD
		12	SC-T2(40)-TP(1)-FV2(25)-SD
D	21%	13	SC-T1(40)-TP(1)-T1(20)-FH1(35)-SD
		14	SC-T1(40)-TP(1)-T1(20)-FH2(35)-SD
		15	SC-T1(40)-TP(1)-T2(20)-FH1(35)-SD
		16	SC-T1(40)-TP(1)-T2(20)-FH2(35)-SD
		17	SC-T2(40)-TP(1)-T1(20)-FH1(35)-SD
		18	SC-T2(40)-TP(1)-T1(20)-FH2(35)-SD
		19	SC-T2(40)-TP(1)-T2(20)-FH1(35)-SD
		20	SC-T2(40)-TP(1)-T2(20)-FH2(35)-SD
E	20%	21	SC-T1(25)-TP(1)-T1(35)-FH1(50)-SD
		22	SC-T1(25)-TP(1)-T1(35)-FH2(50)-SD
		23	SC-T1(25)-TP(1)-T2(35)-FH1(50)-SD
		24	SC-T1(25)-TP(1)-T2(35)-FH2(50)-SD
		25	SC-T2(25)-TP(1)-T1(35)-FH1(50)-SD
		26	SC-T2(25)-TP(1)-T1(35)-FH2(50)-SD
		27	SC-T2(25)-TP(1)-T2(35)-FH1(50)-SD
		28	SC-T2(25)-TP(1)-T2(35)-FH2(50)-SD
F	8%	29	SC-FH1(40)-SD
		30	SC-FH2(40)-SD

Tableau 3.a: Routages alternatifs et temps de traitement des pièces **[Saygin, 2001].**

La présence de machines identiques dans le système offre aux pièces arrivant au système plusieurs routages possibles. Notre problématique est comment choisir le meilleur routage parmi les routages possibles et ceci doit se faire en temps réel. En d'autres termes, à chaque fois qu'une pièce entre dans la station de chargement on doit lui associer le meilleur routage possible selon sont type parmi ceux possibles.

Toutes les métaheuristiques sont des algorithmes itératifs, il est donc très difficile de trouver la meilleur solution dès la première itération, ce qui justifie l'échec de leur adaptation pour la résolution des problèmes complexes en ligne. Pour résoudre notre problème en ligne, il est nécessaire de prendre en considération, à chaque itération, des critères concernant l'état du

système tels que le taux de production, le taux d'utilisation des machines, le temps du cycle, …etc. qui peuvent être évalué à chaque fois qu'une pièce arrive à la station de chargement.

Notre but est d'équilibrer les charges entre les routages non pas en termes de nombre de pièces mais en termes de temps opératoire. Pour aboutir à cet équilibre on doit maximiser le produit des charges de tous les routages.

3.2 Les algorithmes des métaheuristiques

Résoudre un problème n'est pas toujours suffisant. La solution trouvée doit être la meilleure solution possible. En d'autres termes, il faut trouver la solution optimale du problème. Mais dans la pratique, la solution optimale n'est pas toujours faisable à cause de l'existence de certaines contraintes qui la rendent irréalisable dans le monde réel. Différentes techniques sont utilisées pour résoudre des problèmes d'optimisations combinatoires, parmi ces techniques on cite les approches métaheuristiques.

Les métaheuristiques peuvent être appliquées lorsque l'espace de recherche est vaste ou le problème possède énormément de paramètres qui doivent être optimisés simultanément, ou lorsque le problème à résoudre ne peut pas être facilement décrit par un modèle mathématiques précis

Les métaheuristiques sont de bonnes techniques utilisées pour résoudre des problèmes d'optimisation combinatoire qui sont difficiles à solutionner et qui nécessitent un temps de calcul important. Ce sont des techniques peut être définie comme étant un processus itératif, qui permet de subordonner ou cordonner différentes opérations ou procédures pour pouvoir détecter des solutions de bonne qualité. Elles peuvent manipuler une seule solution ou plusieurs solutions à chaque itération, les métaheuristiques s'adaptent facilement à une large variété de problèmes combinatoires et elles convergent vers des solutions optimales qui ne peuvent pas être détectées par les méthodes traditionnelles.

Dans ce cas-là, l'utilisation des métaheuristiques est inutile. Elles possèdent différentes caractéristiques; elles ont de nombreux avantages. Voici quelques points forts et limites des métaheuristiques :

- les métaheuristiques ont la possibilité de s'appliquer à une grande variété de problèmes;
- elles montrent une bonne efficacité;
- elles permettent d'obtenir une bonne performance en tenant compte de la qualité des solutions et du temps de calcul.

En revanche,

- les métaheuristiques ne garantissent pas une solution quasi optimale et il est difficile de prévoir la performance de la méthode utilisée;
- elles nécessitent une adaptation aux problèmes traités;
- elles nécessitent un bon réglage des paramètres afin d'obtenir des résultats de bonne qualité dans un temps de calcul acceptable.

La section suivante est consacrée à l'adaptation des algorithmes des deux métaheuristiques, l'algorithme mémétique et la recherche dispersée pour la résolution de notre problème.

3.3 Adaptation des métaheuristiques pour la résolution de notre problème

Avant de présenter l'adaptation des deux algorithmes il est nécessaire de présenter quelques notions essentielles liées aux deux méthodes :

3.3.1 Chromosomes et évaluation

Parmi les éléments caractéristiques des algorithmes de type génétique, le codage des solutions en chromosomes adéquats est une décision particulièrement critique, surtout pour des problèmes comportant différents niveaux d'information, comme c'est le cas pour le problème d'ordonnancement dans un système flexible de production. En effet, un chromosome doit ici contenir les routages choisis des pièces présentes dans la file infinie.

Nous donnons un exemple de codage pour notre problème. Etant donné que la capacité des files d'attente est 4; et supposons que les premières pièces dans la file infinie sont comme suit : C, A, F, et B.

Les routages possibles sont donc : 9, 10, 11, et 12 pour C ; 1, 2, 3, et 4 pour A ; 29, et 30 pour F ; 5, 6, 7, et 8 pour B, et pour E les routages sont numérotés de 20, à 28.

Le codage est donc une chaine qui comporte l'un des routages possibles pour chaque pièce, Par exemple la figure 3.b donne un codage possible.

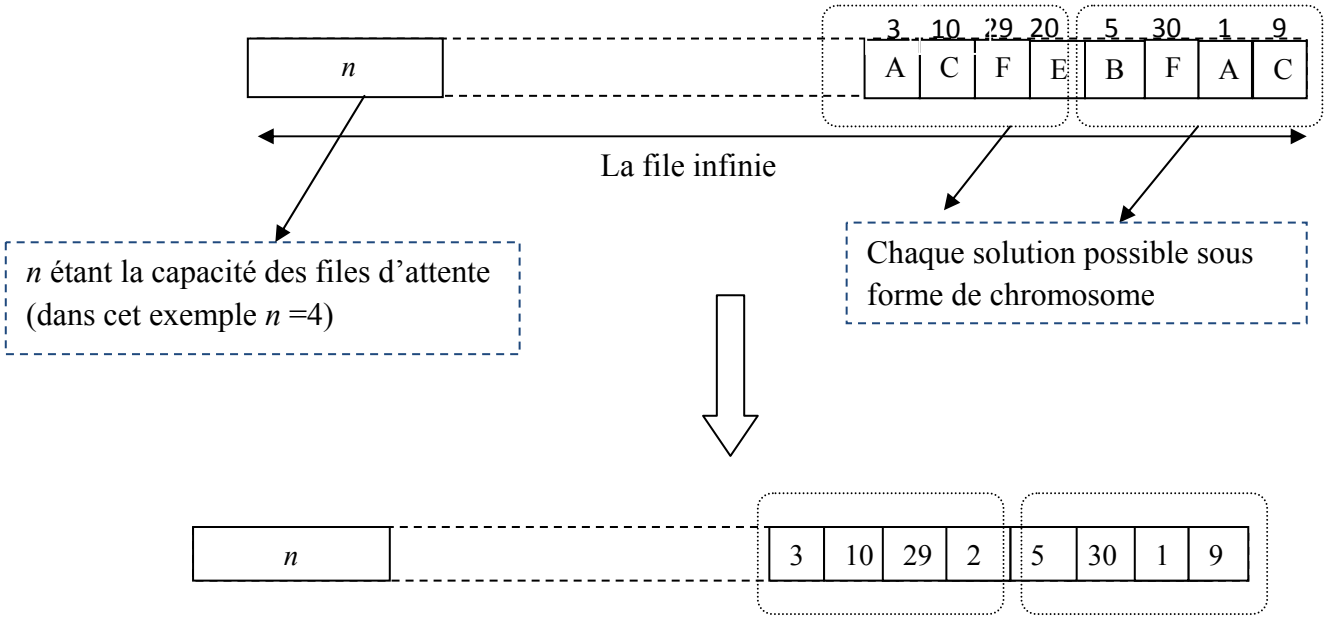


Figure 3.b: Un exemple d'un codage possible pour une capacité de file d'attente=4.

3.3.2 La fonction objectif

Chaque chromosome est évalué par une valeur appelée fitness $F(S)$, représentée par le coût total de la solution réalisable S, on définit une fonction objectif ou fonction de coût que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres et les contraintes concernés, Un ensemble de N points dans un espace de recherche, choisi à priori au hasard, constitue la population initiale; chaque individu x de la population possède une certaine

performance, qui mesure son degré d'adaptation à l'objectif visé: dans le cas de la maximisation d'une fonction objectif *f*, *x* est d'autant plus performant que *f(x)* est plus grand .

Pour trouver le fitness du chromosome *S* dans notre travail, la fonction objectif est le produit de charges des *n* routages, les mathématiciens ont prouvé que plus les valeurs sont très proches plus leur produit est grand (la moyenne géométrique). Le but étant de maximiser cette fonction, afin d'équilibrer les charges des routages, on parle de charges non pas en terme de nombre des pièces mais en terme du temps opératoire (on calcule le temps restant de la pièce au cours du traitement et le temps de traitements des pièces en attente d'être traitées sur cette machine, ces pièces sont celles contenues dans sa file d'attente d'entrée et celles contenues dans la file d'attente de sortie de la machine précédente dans le même routage) donnera donc un bon équilibre entre les routages, la figure 3.c présente cette définition.

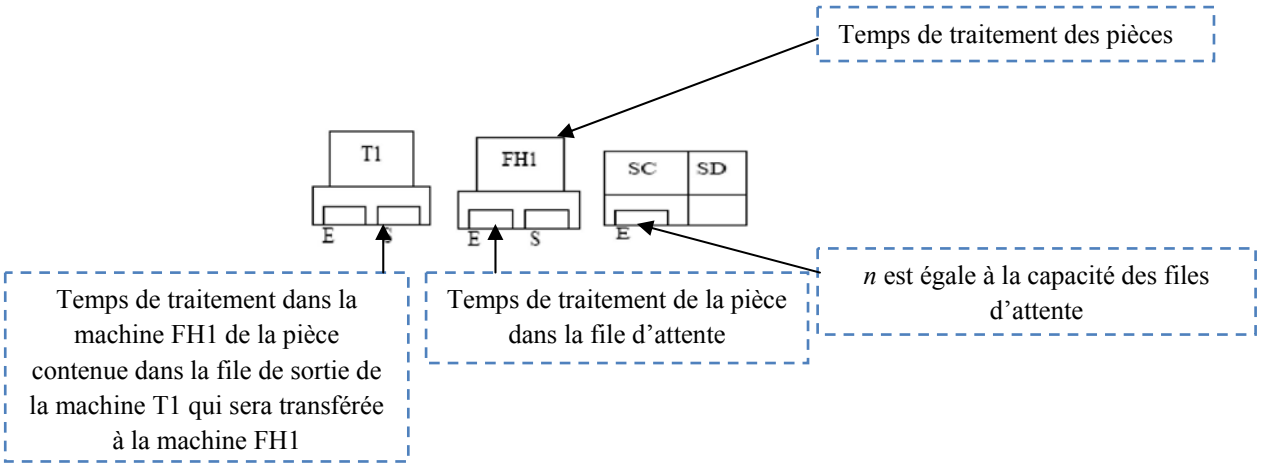


Figure 3.c Un des routages de la pièce type A(1)

La maximisation de cette fonction (le produit des temps opératoires est définie par cette fonction :

$$F(t) = \prod_{j=1}^{30}(\sum_{i=1}^{m} O(i,j))$$

m : le nombre de machines

j : le nombre de routages

O : temps opératoires

Pour ce faire, nous avons pris en considération les types des premières pièces existantes dans la file infinie (*n* est égale à la capacité de la file d'attente de la station de chargement). Par la suite, chaque pièce sera affectée à un routage selon son type.

3.3.3 Sélection des parents et croisement

La sélection des individus favorise les meilleurs individus, cette sélection peut se baser sur la fonction objectif ou se faire aléatoirement. La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. Dans notre cas la sélection est faite aléatoirement.

Grâce au codage des individus (solutions) sous forme de chromosomes, il est possible de générer des enfants par croisement de deux parents de la population.

Le croisement joue un rôle important dans les performances des algorithmes de type génétique. C'est le point fort de ces algorithmes. Il permet de créer de nouveaux individus et ainsi d'explorer l'espace de recherche en combinant les différentes solutions.

Cela se fait en choisissant un endroit de coupure le long des deux chromosomes, puis en reliant la partie gauche de l'un à la partie droite de l'autre, et vice versa. Ce type de croisement à un point sur chaque sous-partie du chromosome permet une transmission génétique à la fois fidèle aux parents (une seule coupure par sous-partie), et suffisamment diversifiant.

Supposons avoir les deux chromosomes d'après la figure 3.d suivants :

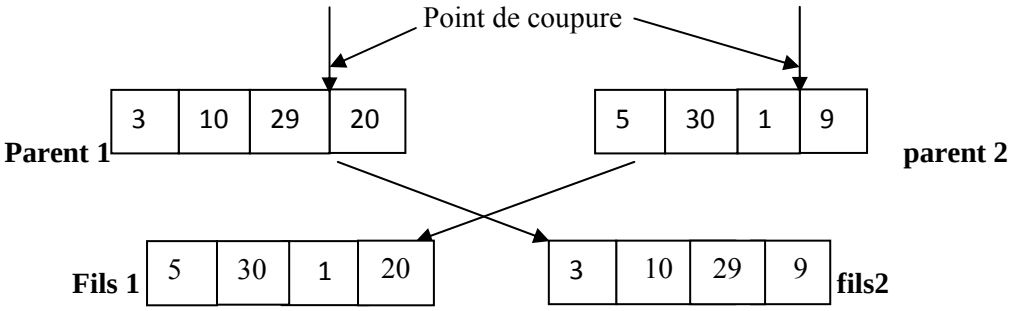


Figure 3.d: Exemple d'un croisement de deux solutions parents. Deux enfants obtenus par ce croisement.

Un point de coupure est généré aléatoirement au niveau des deux parents sélectionnés et leurs parties sont inter changées, reliant maintenant les extrémités.

3.3.4 Les méthodes de recherche locale

Une des particularités du MA/PM est de travailler sur une petite population de solutions de grande qualité. Pour cela, les solutions générées subissent une recherche locale. L'inclusion d'une recherche locale dans une métaheuristique permet d'intensifier la recherche autour d'une solution donnée.

Nous avons choisit comme méthode de recherche locale la méthode de descente. Le résultat dépend du voisinage retenu, de la solution initiale *x* et de façon de voisin d'une solution. En effet, suivant la solution initiale ou l'ordre de visite des voisins, on peut obtenir des solutions différentes. C'est-à-dire qu'à partir d'une solution trouvée on fait une recherche qui passe d'une solution à une autre solution voisine, et à chaque fois qu'on trouve une solution améliorante on l'utilise comme nouvelle solution de départ, et la nouvelle recherche s'effectue dans le voisinage de la nouvelle solution trouvée. Dans l'algorithme MA/PM, cette recherche s'effectue sur chaque individu de la population générée initialement, puis elle s'applique sur chaque nouvel individu obtenu par le croisement, c'est-à-dire sur chaque nouvelle génération de solutions, cela veut dire qu'on tente toujours de choisir les meilleurs individus de la population initiale ou de la génération construite.

Le pseudo code de la méthode de descente est donné comme suit:

9£

Algorithme 3.1: Méthode de descente

(1) Pour chaque individu (qui représente une condition initiale *x* pour cette méthode)
(2) Tant que la condition d'arrêt n'est pas vérifiée faire
(3) Modifier les routages de certaines pièces (trouver une autre solution voisine x' de x)
(4) Si la fonction objectif est améliorée (produit des charges des routages) : $f(x') > f($ **alors**
(5) Remplacer x par x'
(6) Finsi
(7) Fin tant que

3.3.5 La gestion de population

Une autre question importante à nos yeux et indispensable pour le recherche dispersée et l'algorithme mémétique avec gestion de population est la définition d'une mesure de distance entre individus. Celle-ci doit être capable de refléter une similitude ou une différence entre deux individus. On pourra alors en déduire la distance d'un individu à la population toute entière. En fonction du problème, du codage des individus et de la distance retenue, l'évaluation peut-être plus ou moins couteuse en temps de calcul. Mais parfois il peut-être bon de passer du temps pour calculer cette distance si elle conduit au final à une meilleure diversité de la population.

La principale caractéristique du l'algorithme mémétique avec gestion de population vient de la gestion de la population grâce à une mesure de distance dans l'espace des solutions. Cette opération permet de contrôler la diversité des individus en filtrant l'entrée des enfants dans la population. Pour constituer le groupe de solutions-élites, une mesure de distance doit être élaborée, une mesure de dissimilarité ou la ressemblance des individus dans la population, pour éviter une convergence rapide de la population, il est important de maintenir un certain niveau de diversité au sein des populations intermédiaires. Le paramètre de diversité peut-être mesuré de la manière suivante :

$$\left\{\Delta = \sum \Delta(\boldsymbol{S_i}, \boldsymbol{S_j})\right\}$$

La distance est le nombre de caractéristique qui différent entre deux solutions. Si S_1=(9,30,10,5)et S_2=(20,15,10,2) alors $\Delta(S_1, S_2)$=3 . Plus deux individus sont distant, plus ils sont considérés comme étant différents.

3.3.6 La mutation

Son principal rôle est la diversification de la population qui, après plusieurs générations, tend à s'uniformiser. Dans notre cas la mutation est la modification des routages de certaines pièces en lui donnant un routage différent parmi les routages possibles. La modification du routage se fait aléatoirement.

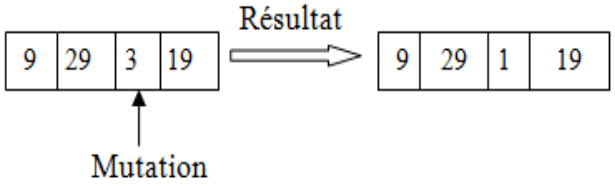


Figure 3.e Exemple de mutation

3.3.7 L'algorithme mémétique avec gestion de population

Le fonctionnement est assez simple et est basé sur un algorithme génétique. Nous supposons que nous savons comparer deux individus entre eux et mesurer leur dissemblance. Nous pouvons donc mesurer la similarité entre un individu et la population existante. Au départ, on génère une population initiale de petite taille et on choisit un paramètre Δ fixant le niveau de dissemblance des solutions entre elles. Ensuite, on procède comme dans un algorithme génétique, on choisit deux individus que l'on croise pour obtenir deux enfants.

Pour chaque individu un on applique une recherche locale de façon à obtenir des optima locaux. S'ils ne répondent pas au critère de diversité, on applique un opérateur de mutation sur ces individus jusqu'à satisfaction de ce critère. Ensuite sous condition, on les insère dans la population à la place d'un autre individu. D'un coté l'algorithme MA/PM applique une recherche locale à toutes les nouvelles solutions et de l'autre, elle maintient une population de petite taille et diversifie. L'autre avantage, c'est l'´evolution du paramètre de diversité Δ qui permet à tout moment d'augmenter ou de réduire la diversité des individus dans la population.

Le pseudo code de l'algorithme MA/PG est donné comme suit :

Algorithme 3.2 : MA/PM

1. **S'il y'a** une place libre dans la station de chargement alors
2. **Initialiser** : générer une population initiale P de solutions (cela revient dans notre cas affecter les *n* premières pièces de la file infinie à des routages d'une façon aléatoire e respectant les types de pièces), Génération d'une population aléatoire.
3. **Tant que** (critère d'arrêt est non atteint : nombre d'itérations)
4. **Pour** chaque individu
5. **Evaluation** de la fitness de cet individu (le produit des charges de routages).
6. **Si** la fonction objectif est améliorée alors mettre à jour la meilleure solution (cel améliorante).
7. **Fin pour**
8. **Fixer** le paramètre de la diversité Δ
9. **Répéter**
10. **Sélection** : choisir deux solutions p_1 et p_2
11. **Croisement** : combiner deux solutions parents p_1 et p_2 pour former des nouvelles solutions c_1 , c_2
12. **Recherche locale** : appliquer une procédure de recherche locale sur c_1, c_2
13. **Pour** toute solution *c* **faire**
14. **Tant que** *c* ne satisfait pas la condition d'addition **Faire**
15. **Mutation** (*c*)
16. **Fin Tant que**
17. **Supprimer** la solution de la population
18. **Addition**: de solution :$\boldsymbol{p} \leftarrow \boldsymbol{p} \cup \boldsymbol{c}$
19. **Fin Pour**
20. **Mettre** à jour le paramètre de diversité Δ
21. **Jusqu'à** satisfaire un critère d'arrêt
22. **Fin Tant que**.
23. **Fin Si**

Dans cet algorithme le facteur d'intensification principal est la recherche locale. Concernant la diversification, c'est la boucle (lignes 14 à 16) imposant un certain niveau de diversité qui joue ce rôle. On force les individus à être de plus en plus éloignés entre eux, on diversifie donc la recherche.

3.3.8 La rechercher dispersée *(scatter search)*

Cette méthode fait également partie de la famille des algorithmes évolutionnaires et s'appuie sur le principe suivant. Elle consiste à générer une population de solutions (assez importante au départ) en essayant de proposer des solutions diverses les unes des autres et chaque individu est amélioré par l'application d'une recherche locale. De cette population on extrait un ensemble B de solutions dispersés à partir d'un ensemble de référence *R* contenant les meilleures solutions de la population initiale. Ces solutions dispersés sont obtenus en effectuant tout d'abord des combinaisons des solutions de l'ensemble de références *R* entre elles (et avec les nouvelles solutions générées) puis améliorées jusqu'à ce qu'il n'y ait plus de nouvelles solutions générées par combinaison. Les solutions de A sont optimisées à l'aide d'une recherche locale pour obtenir l'ensemble B des points dispersés. Le nouvel ensemble de solutions de référence est obtenu en sélectionnant des points dans R ∪ B.

Ensuite une moitié de la population est régénérée (remplacée par des solutions diverses) et le processus recommence jusqu'à satisfaction d'un critère d'arrêt.

Un des points importants, c'est la mesure de la diversité des solutions. Cette mesure doit être prise dans l'espace des solutions (et non dans l'espace des objectifs) et elle doit refléter la différence entre deux solutions.

Le pseudo code de la recherche dispersée est donné comme suit :

Algorithme 3.3 : La recherche dispersée (SS)

1. **Initialiser** : générer une population initiale P de solutions cela revient dans notre cas affecter les *n* premières pièces de la file infinie à des routages d'une façon aléatoire e respectant les types des pièces
2. **Mettre** les meilleures solutions trouvées dans une population de référence R (on calcule le produit des charges de routages)
3. **Tant que** le critère d'arrêt non satisfait (nombre d'itérations) faire
4. A=R → prendre la population de référence
5. **Tant que** A≠∅ faire
6. **Combiner** les solutions (B←R×A)
7. **Améliorer** les solutions B
8. **Mettre** à jour les solutions R
9. **Garder** les meilleures solutions à partir de R ∪ B
10. A←B−R
11. **Fin** tant que
12. Supprimer mauvaise solution dans R
13. Ajouter de nouvelles solutions diverses dans R
14. **Fin** tant que

On garde à chaque itération un ensemble de référence qui soit à la fois composé de bonnes solutions et de solutions diverses. Les facteurs d'intensification se retrouvent alors dans l'application systématique d'une méthode de recherche locale ainsi que dans la combinaison et dans la mise à jour de *R*.

Pour pouvoir évaluer les performances des deux métaheuristiques, nous avons utilisés comme critères de performance :

- **Le taux de production :** c'est l'un des critères les plus importants. Le taux de production, dit aussi le taux de sortie des pièces, se calcule en divisant le nombre de pièces sortantes du système par le nombre des pièces entrantes à la file infinie, afin de faire une normalisation.
- **Le temps de cycle :** le temps de cycle d'une pièce est le temps de présence d'une pièce dans le système depuis qu'elle entre dans la station de chargement jusqu'à ce qu'elle quitte la station de déchargement. Nous nous intéressons au temps du cycle moyen (par minute) de toutes les pièces sortantes du système.
- **Les en-cours :** ce sont les pièces présentes dans le système.
- **Le taux d'utilisation des machines**
- **Le taux d'utilisation de l'AGV**

3.4 Analyse de sensibilité

Tout comme pour les autres méthodes, les performances d'un algorithme à base génétique dépendent fortement des paramètres choisis pour cet algorithme. Une série de tests et de simulations doivent être effectués pour déterminer les valeurs de ces paramètres. Pour des bonnes performances de ces algorithmes il faut :

- Une population de taille modérée.
- Une probabilité de croisement élevée.
- Une probabilité de mutation faible (inversement proportionnelle à la taille de la population).
- Mesurer la diversité.

- **Taille de la population**

La taille de la population (N), doit être fonction de taille du problème et du codage utilisé, il est clair que si la taille de la population est trop petite, il y a convergence prématurée car l'algorithme n'a pas grand échantillon de l'espace de recherche. Par contre, une population trop grande induit un cout important en temps de la fonction d'adaptation.

- **Taux de croisement**

C'est la probabilité avec laquelle sont choisis les individus auxquels est appliqué l'opérateur de croisement. Ce choix est généralement expérimental et sa valeur est très souvent prise entre 0.5 et 0.9 **[Learman, 2000**]. La population subit des changements importants lorsque ce taux est élevé. Il en résulte, par conséquent, une convergence très rapide. Par contre, si le taux est faible, la recherche risque de stagner à cause du faible taux d'exploration.

- **Diversité contrôlée**

Mesurer la diversité entre les solutions est particulièrement important, la métaheuristique peut se concentrer sur les individus divers et ne pas effectuer d'exploration inutile. Le contrôle de la diversité est un facteur tout aussi important. Il permet en fonction de l'ensemble des individus de la population de plus ou moins diversifier et de plus ou moins intensifier la recherche.

- **Taux de mutation :**

C'est la probabilité avec laquelle l'opérateur de mutation est appliqué à un individu de la population. Cet opérateur est considéré comme un mécanisme d'adaptation secondaire pour les algorithmes a base génétique. Des études empiriques conseillent d'utilises un faible taux de mutation de l'ordre de 0.01 **[Golver, 1998**]. Avec un taux plus petit, la population risque de ne pas être assez diversifiée. Par contre, avec un taux élevé, la recherche devient aléatoire puisque l'on perturbe trop les individus générés **[Talbi, 1995].**

3.4.1 Sensibilité par rapport à la taille de la population pour la recherche dispersée

Dans cette section nous étudions la variation des différents critères de performance choisis en fonction de la taille de la population, pour une capacité de la file d'attente égale à deux, rappelons que *n* est la capacité des files d'attente. Dans cet algorithme, chaque chromosome artificiel représente les routages choisis des premières pièces par un code. Après chaque évaluation des individus on modifie la meilleure solution s'il y' a une amélioration.

La mutation consiste à modifier les routages de certaines pièces parmi les *n* premières pièces contenues dans la file infinie. De cette population on extrait un *ensemble de référence R* contenant les meilleures solutions de la population initiale. Ensuite ces solutions sont combinées entre elles (et avec les nouvelles solutions générées) puis améliorées jusqu'à ce qu'il n'y ait plus de nouvelles solutions générées par combinaison.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
N=5	23,07	45,95	68,99	96,64	99,99	99,99	99,99	99,99
N=10	23,54	47,21	71,00	99.99	99.99	99.99	99.99	99.99
N=15	23,88	47,78	71,63	99,99	99,99	99,99	99,99	99,99
N=20	24,17	48,48	72,61	99.99	99.99	99.99	99,99	99,99
N=25	24,34	48,92	723,21	99,99	99,99	99,99	99,99	99,99
N=30	24,64	49,00	73.72	99,99	99,99	99,99	99,99	99,99

Tableau 3.1: Le taux de production en fonction de la taille de la population pour une taille file=2

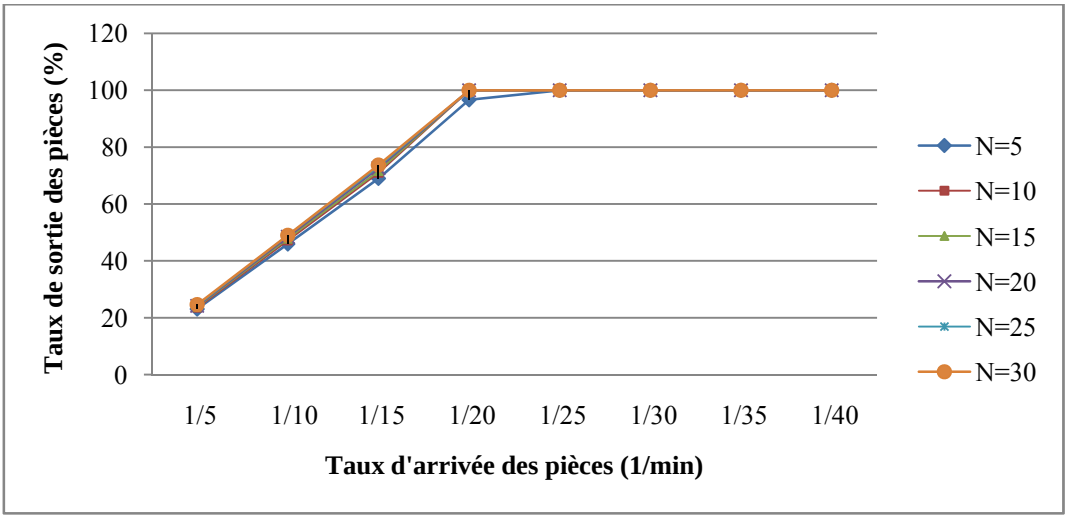


Figure 3.1: Variation du taux de production en fonction de la taille de la population pour une taille file=2

Le tableau 3.1 et la figure 3.1 montrent que pour des taux de création de pièces inférieurs ou égales à 1/25 toutes les valeurs du taux de production sont identiques pour les différentes valeurs de la taille de la population, hors de cet intervalle les valeurs des taux de production sont satisfaisantes pour les différentes valeurs de la taille de la population, mais il est clair que pour une taille de la population à 30 les résultats obtenus sont les meilleurs pour tous les taux de création des pièces. Les plus grandes valeurs sont obtenues pour une taille de la population égale à 30 lorsque la population trop grande, le temps de calcul augmente et demande un espace mémoire important.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
N=5	176,86	172,20	173,54	136,32	92,29	86,72	82,89	88,46
N=10	172,52	169,99	164,16	118,32	94,95	91,03	84,10	91,83
N=15	167,90	169,24	168,03	122,73	97,94	92,46	88,24	95,08
N=20	165,26	165,28	164,24	117,03	94,00	88.11	79,24	82,99
N=25	169,03	167,03	167,44	123,05	101,50	87,92	83,35	85,35
N=30	167,90	161,67	165,71	117,90	109,3	94,51	97,94	92,46

Tableau 3.2: Variation le temps de cycle en fonction de la taille de la population pour une capacité de file d'attente=2

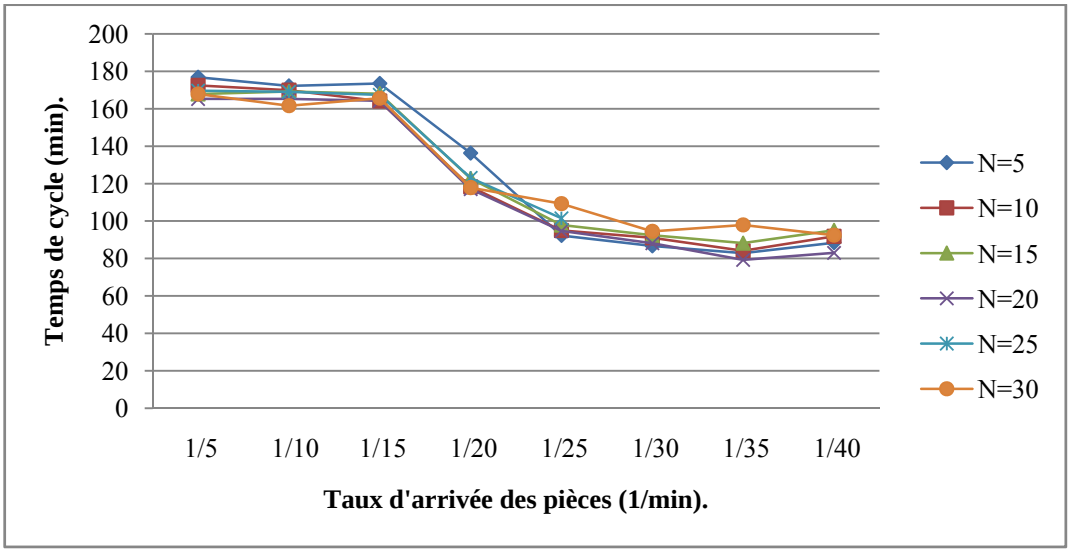


Figure 3.2: Variation le temps de cycle en fonction de la taille de la population pour une capacité de file d'attente=2

Le tableau 3.2 et la figure 3.2 montrent que les valeurs du temps du cycle pour un taux de création des pièces supérieures à 1/20 le meilleur temps de cycle est obtenu pour une taille de la population égale à 30, pour les autres valeurs du taux de création la taille de population *N=5* donne de meilleurs résultats.

3.4.2 Sensibilité par rapport au paramètre de diversité pour MA/PM

Le paramètre de diversité est un facteur tout aussi important. Il permet en fonction de l'ensemble des individus de la population de plus ou moins diversifier et de plus ou moins intensifier la recherche.

paramètre de diversité	Taux de production (%)	Le temps de cycle (min)	Les en-cours (%)
Δ=0	90,04	176,07	8,83
Δ=1	92,41	165,67	8,95

Tableau 3.3 : Les différents critères de performance en fonction de paramètre de diversité pour une taille file=2 et un taux d'arrivée des pièces=1/19.

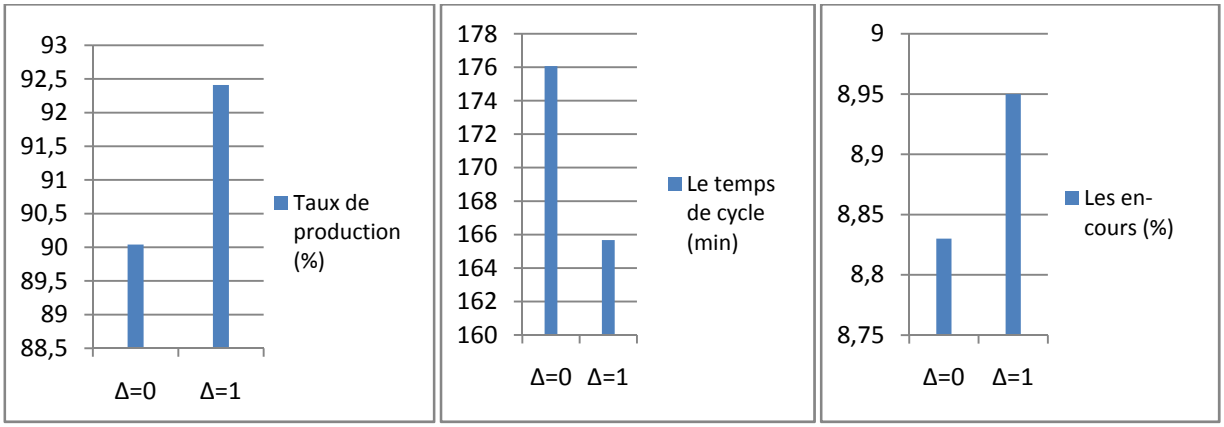


Figure 3.3 : Les différents critères de performance en fonction de paramètre de diversité pour une taille file=2 et un taux d'arrivée des pièces=1/19

Le tableau 3.3 et la figure 3.3 présentent les différents critères de performance en fonction de paramètre de diversité pour une taille de file égale à 2 et un taux d'arrivée des pièces égale à 1/19. Pour un Δ=0 l'algorithme se comporte comme un algorithme mémétique classique, le taux de production est acceptable, ainsi que pour le temps du cycle et les encours, tandis que pour un Δ=1, la valeur du taux de production augmente, et le temps du cycle diminue tandis que les encours restent toujours proches.

paramètre de diversité	Taux de production (%)	Le temps de cycle (min)	Les en-cours (%)
Δ=0	87.71	339,71	19,71
Δ=1	87.86	342,04	19,73
Δ=2	87.76	335,62	19,70
Δ=3	87.84	324,21	19,71
Δ=4	87.94	327,58	19,72
Δ=5	88,03	329,73	19,77

Tableau 3.4 : les différents critères de performance en fonction de paramètre de diversité pour une taille file=6 et un taux d'arrivée des pièces=1/18.

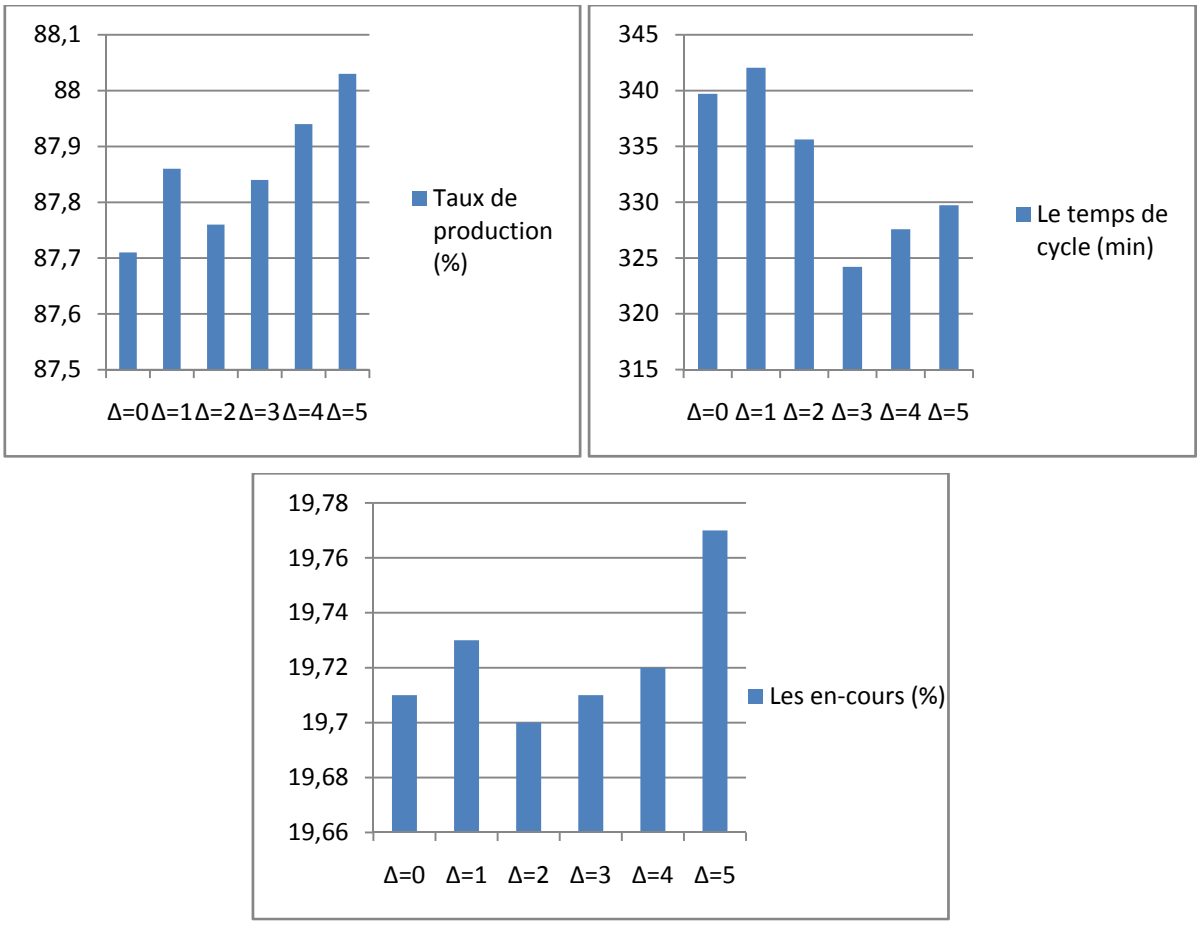


Figure 3.4 : Les différents critères de performance en fonction de paramètre de diversité pour une taille file=6 et un taux d'arrivée des pièces=1/18

Le tableau 3.4 et la figure 3.4 présentent les différents critères de performance en fonction de paramètre de diversité pour une taille de la file d'attente égale à 6 et un taux d'arrivée des pièces égale à1/18. Pour un Δ=0 l'algorithme se comporte comme un algorithme mémétique classique, le taux de production est acceptable, ainsi que pour le temps du cycle et les encours. Pour voir l'effet de l'augmentation des valeurs du paramètre Δ, on augmente a chaque fois et on compare entre les critères de perfermances, on remarque que plus on augmente Δ plus le taux de production augmente et le temps de cycle diminue tandis que les encours restent toujours proches.

3.5 Etude comparative sans introduction de pannes

3.5.1 Taux de production

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/19	1/20	1/21	1/22	1/25	1/30	1/35	1/40
Recherche dispersée	24.64	49,00	73.72	93.51	99,99	99,99	99,99	99,99	99,99	99,99	99,99
mémétique avec gestion de population	24.31	48.62	72.84	92.40	99,99	99,99	99,99	99,99	99,99	99,99	99,99
les algorithmes génétiques	23,70	47,25	71,08	90,36	98,44	99,99	99,99	99,99	99,99	99,99	99,99
les essaims particulaires	23,12	46,10	69,33	88,51	99,22	99,99	99,99	99,99	99,99	99,99	99,99
Les colonies de fourmis	21,51	43,20	64,54	84,66	90,52	98,85	99,99	99,99	99,99	99,99	99,99

Tableau 3.5 : Taux de sortie des pièces pour une capacité de file d'attente = 2

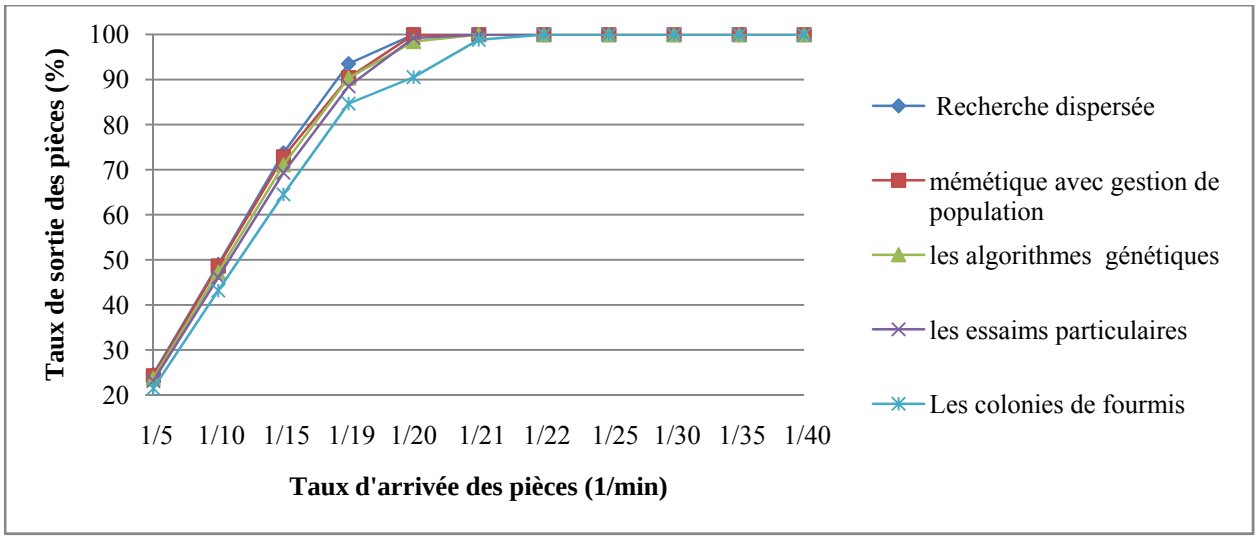


Figure 3.5 : Taux de sortie des pièces pour une capacité de file d'attente = 2

Les courbes de la figure et le tableau 3.5 montrent que pour des tailles de files d'attente petites et un taux de création de pièces supérieur ou égale à 1/21, les résultats obtenus par la recherche dispersée et l'algorithme mémétique avec gestion de population sont meilleurs que celles des algorithmes génétiques et les essaims particulaires sauf pour les colonies de fourmis on remarque que le système sature rapidement par apport aux autres méthodes et qu'en dessous d'un taux de création égale 1/25 le taux de production est pratiquement le même pour toutes les métaheuristiques.

Les courbes de la figure 3.5 montrent que pour de petites capacités de files d'attente et un taux de création de pièces 1/19, la recherche dispersée est plus efficace que les autres. L'algorithme mémétique avec gestion de population montre aussi sa performance par rapport à des algorithmes génétiques et aux essaims particulaires pour une capacité de file d'attente égale à 2. Et pour un taux de création inférieur à 1/21, les résultats obtenus par les métaheuristiques sont les mêmes.

On remarque que le point de saturation dans le système est autour du taux de création des pièces 1/19 et 1/20 pour la recherche dispersée et l'algorithme mémétique avec gestion de population mais pour les algorithmes génétiques et les essaims particulaires il est autour le taux de création des pièces 1/20, le système devient non saturé pour touts les méthodes à taux de création de pièces égale 1/21sauf pour les colonies de fourmis le point de saturation est autour 1/22.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/18**	**1/20**	**1/25**	**1/30**	**1/35**	**1/40**
Rechercher dispersée	24,48	48,95	73.30	88.07	99,99	99,99	99,99	99,99	99,99
mémétique avec gestion de population	24,39	48,86	72,5	87,03	99,99	99,99	99,99	99,99	99,99
Les algorithmes génétiques	24,17	48,26	72,40	87,00	99,99	99,99	99,99	99,99	99,99
Les essaims particulaires	23,78	47,47	71,20	85,34	99,99	99,99	99,99	99,99	99,99
Les colonies de fourmis	22,98	45,88	69,00	82,40	99,99	99,99	99,99	99,99	99,99

Tableau 3.6 : Taux de sortie des pièces pour une capacité de file d'attente = 6

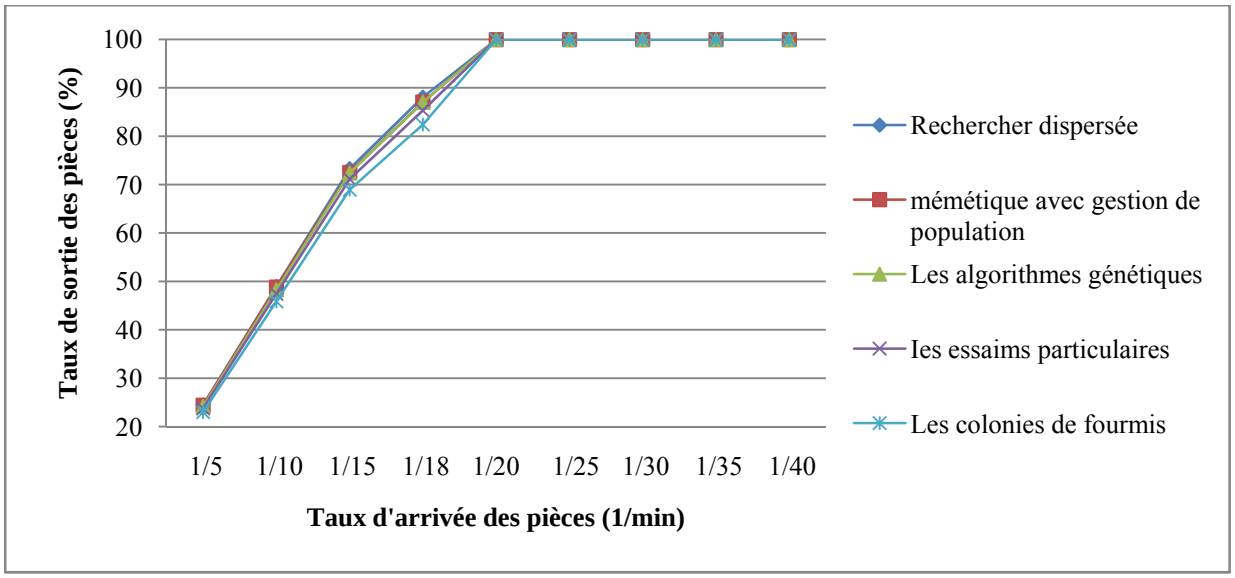


Figure 3.6 : Taux de sortie des pièces pour une capacité de file d'attente = 6

La courbe de figure et le tableau 3.6 montrent que pour de grandes capacités de files d'attente et un taux de création de pièces supérieur à 1/18, la recherche dispersée domine les autres métaheuristiques. Hors de cet intervalle, les taux de production obtenus par les métaheuristiques sont identiques.

A partir des résultats montrés sur la figure et tableau 3.6, nous pouvons remarquer que le système n'est pas saturé pour les métaheuristiques pour un taux de création de pièces égale à 1/20 et que le système sature moins rapidement dés que le taux de création des pièces arrive à 1/18.

Capacité de file d'attente	2	4	6	8
Recherche dispersée	24.64	24,41	24,48	24,55
mémétique avec gestion de population	24.39	24,34	24,39	24,51
les algorithmes génétiques	23,70	23,93	24,17	24,24
les essaims particulaires	23,12	23,40	23,78	24,07
Les colonies de fourmis	21,51	22,85	22,98	23,13

Tableau 3.7 : Taux de sortie des pièces pour un taux de création de pièces = 1/5

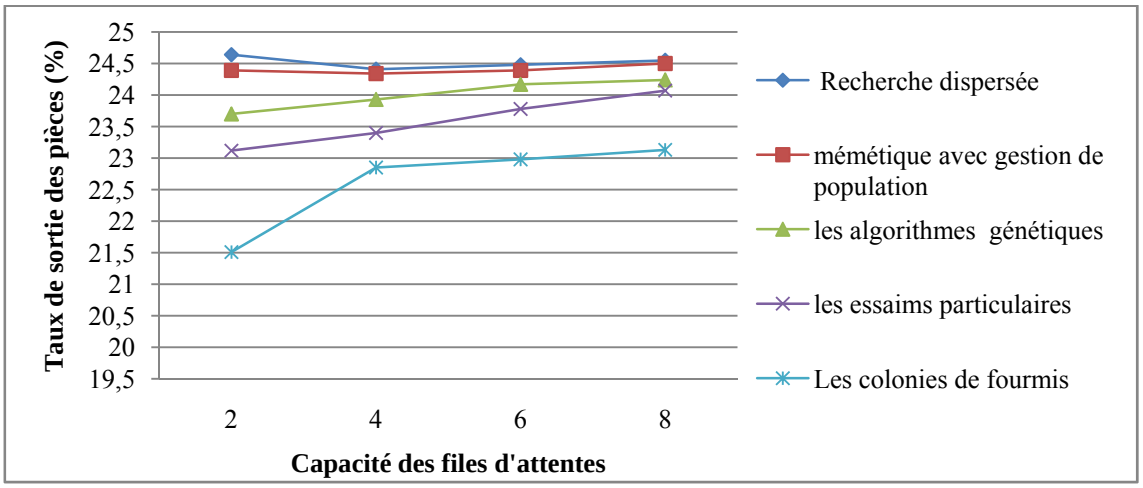


Figure 3.7 : Taux de sortie des pièces pour un taux de création de pièces = 1/5

Capacité des files d'attentes	2	4	6	8
Recherche dispersée	99,99	99,99	99,99	99,99
mémétique avec gestion de population	99,99	99,99	99,99	99,99
les algorithmes génétiques	98,44	99,99	99,99	99,99
les essaims particulaires	99,22	99,99	99,99	99,99
Les colonies de fourmis	90,52	99,99	99,99	99,99

Tableau 3.8 : Taux de sortie des pièces pour un taux de création de pièces = 1/20

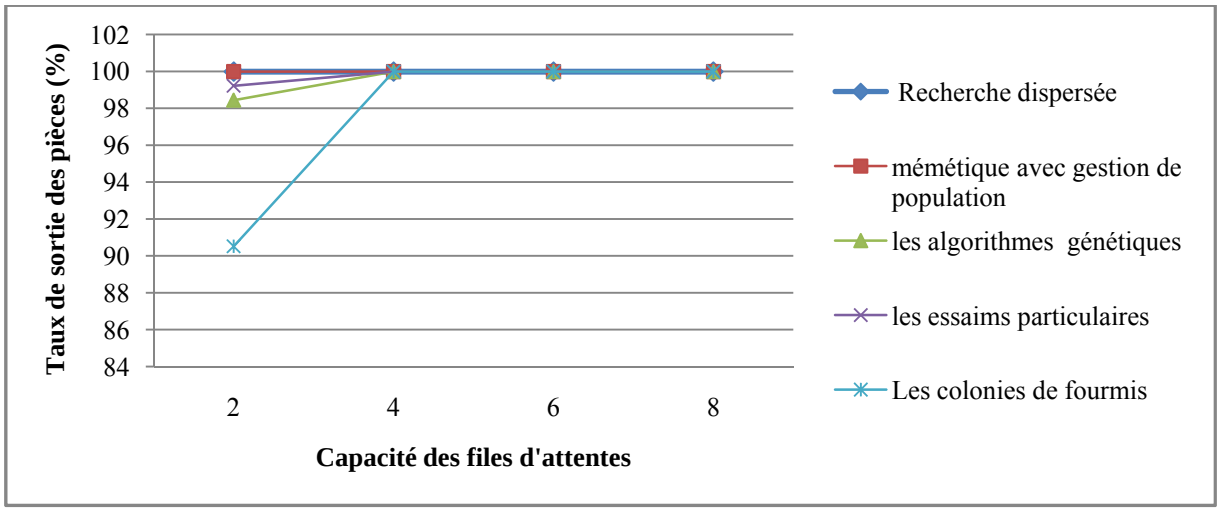


Figure 3.8: Taux de sortie des pièces pour un taux de création de pièces = 1/20

Pour un taux d'arrivée des pièces égale à 1/5, La figures 3.7 nous montre qu'en variant les capacités de files d'attente et en maintenant le temps de création fixe, L'algorithme recherche dispersée est plus efficaces et plus performante.

Pour un taux d'arrivée des pièces égale à 1/20, les résultats obtenus par métaheuristiques sont identiques pour toutes les capacités de la file d'attente sauf pour une capacité de file d'attente égale à 2, où les algorithmes mémétique avec gestion de population et la recherche dispersée donnent les meilleurs résultats.

A partir des résultats montrés sur la figure et tableau 3.8, nous pouvons remarquer que le système n'est pas saturé pour les algorithmes mémétique avec gestion de population et recherche dispersée pour un taux de création de pièces égale à 1/20 et pour toutes les tailles des files d'attentes ce qui veut dire que le système sature moins rapidement si on utilise ces algorithmes.

3.5.2 Le temps de cycle

Dans cette sous section nous allons montrer l'impact du taux de création de pièces, et la capacité des files d'attente sur le temps de cycle

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/19	1/20	1/21	1/22	1/25	1/30	1/35	1/40
recherche dispersée	167,90	169,24	168,03	167,32	117.94	109,34	94,51	97,94	92,46	88,24	95,08
mémétique avec gestion de population	166.72	171.60	170.80	165.97	127.40	110,77	98,30	95,90	95,43	80,78	86,760
les algorithmes génétiques	168,10	168,90	169,80	170,27	132,70	104,37	98,73	98,70	88,60	81,80	89,60
les essaims particulaires	169,40	173,20	173,50	173,17	125,90	112,93	99,86	96,10	91,70	81,20	89,60
Les colonies de fourmis	163,50	165,40	166,60	169,19	155,20	119,74	101,48	98,80	89,00	83,70	90,90

Tableau 3.9: Temps de cycle pour une capacité de file d'attente = 2

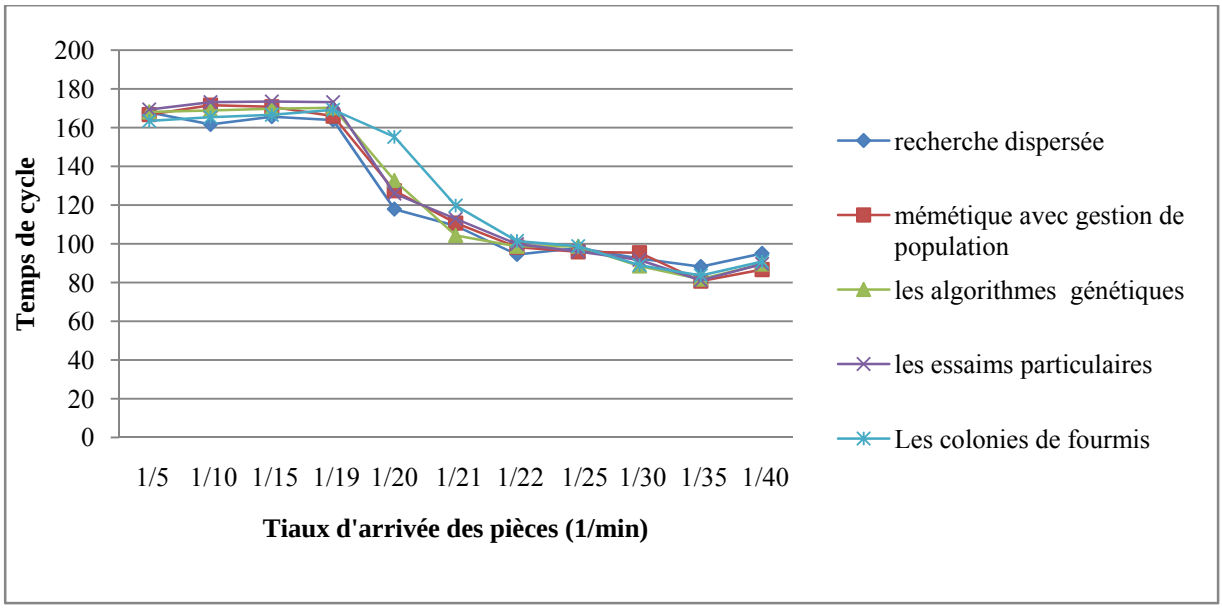


Figure 3.9: Temps de cycle pour une capacité de file d'attente = 2.

Le tableau 3.9 et la figure 3.9 montrent que pour une taille de file d'attente égale à 2 et un taux de création de pièces supérieur à 1/19, les temps de cycle des colonies de fourmis sont meilleures que les autres métaheuristiques.

Pour un taux de création de pièces inférieur ou égale à 1/19, nous ne pouvons pas dire qu'une métaheuristique est la meilleure pour tous les taux, les valeurs du temps de cycle obtenus sont proches pour toutes les métaheuristiques.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/18**	**1/20**	**1/25**	**1/30**	**1/35**	**1/40**
Rechercher dispersée	327,90	330.86	326.84	321.87	137.25	95,12	90,98	79,17	84,55
mémétique avec gestion de population	325.90	340.73	327.32	327.72	118.74	100,10	88,86	85,35	92,17
Les algorithmes génétiques	338,10	332,80	332,00	335,70	121,300	98,90	91,60	83,40	93,10
Les essaims particulaires	331,20	326,00	332,00	327,40	133,60	102,20	91,10	82,50	91,00
Les colonies de fourmis	320,60	324,00	326,00	324,00	121,40	97,00	89,30	82,60	91,10

Tableau 3.10: Temps de cycle pour une capacité de file d'attente = 6

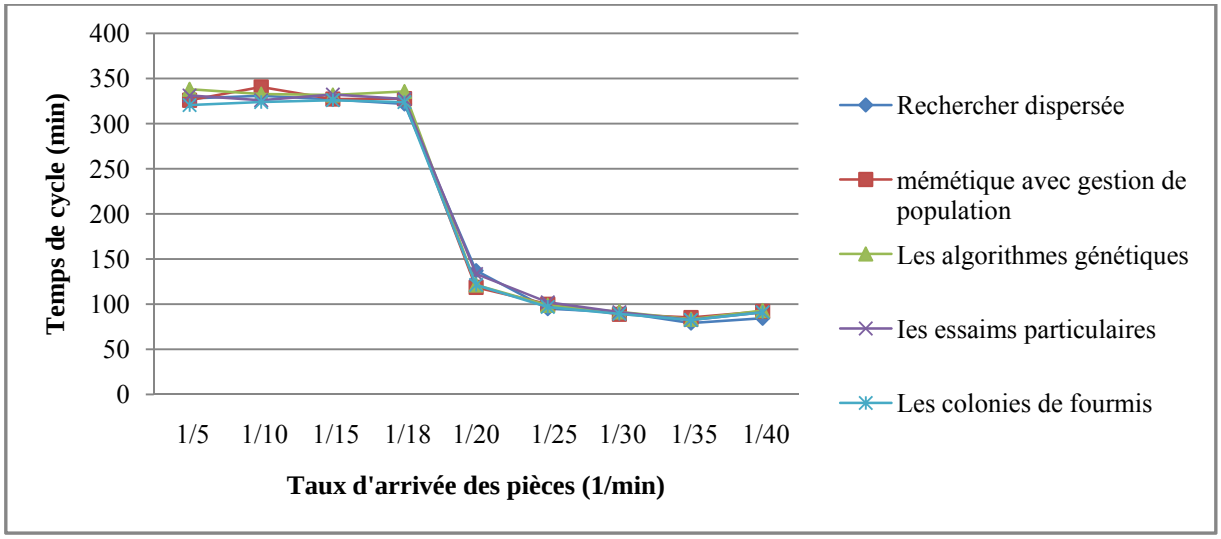


Figure 3.10: Temps de cycle pour une capacité de file d'attente = 6.

La courbe de la figure 3.10 et le tableau 3.10 montrent que pour une taille de file d'attente égale à 6, un taux de création supérieur à 1/18, les temps de cycles obtenus par les colonies de fourmis sont les meilleures que ceux les métaheuristiques, l'algorithme mémétique avec gestion de population est la meilleure pour un taux de création égale à 1/20, les temps de cycle sont les minimaux pour la recherche dispersée pour un taux de création égale à 1/18 et 1/25,et un taux compris entre 1/35 et 1/40, les colonies de fourmis pour un taux égale à 1/30.

Capacité des files d'attentes	**2**	**4**	**6**	**8**
Recherche dispersée	167,90	241,70	327,93	402,10
mémétique avec gestion de population	166072	245020	325,90	427,66
les algorithmes génétiques	168,10	249,90	338,10	417,00
les essaims particulaires	169,40	249,20	331,20	406,90
Les colonies de fourmis	163,50	247,90	320,60	382,40

Tableau 3.11 : Temps de cycle pour un taux de création de pièces =1/5

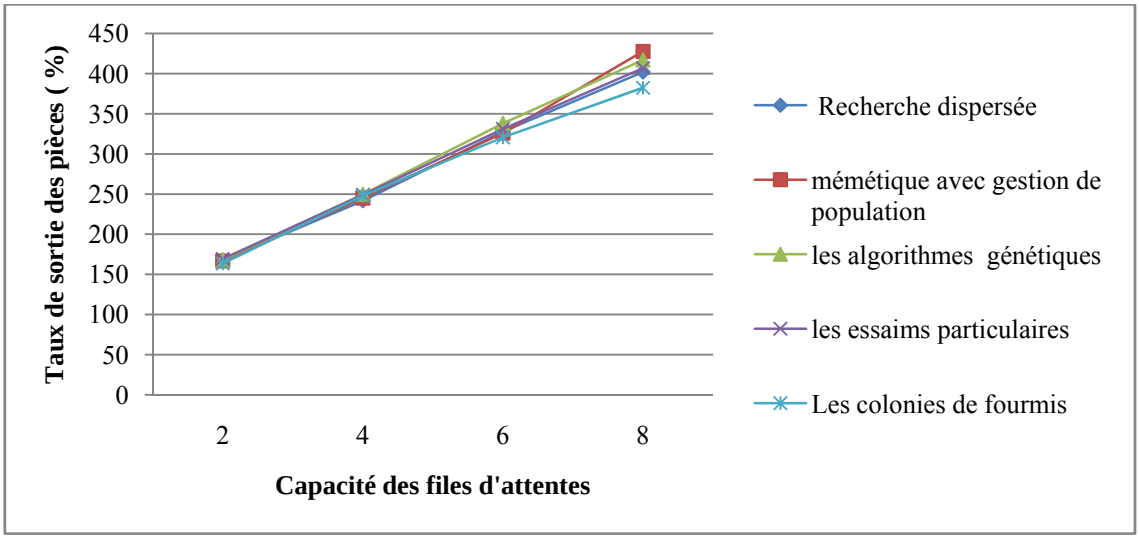


Figure 3.11 : Temps de cycle pour un taux de création de pièces =1/5

Le tableau 3.11 et la figure 3.11 montrent que Pour toutes les capacités de file d'attente et un taux de création 1/5, les colonies de fourmis sont meilleures que les autres métaheuristiques sauf pour capacité de file d'attente est égale 4 la recherche dispersée donne les meilleurs temps de cycle.

Capacité des files d'attentes	**2**	**4**	**6**	**8**
Recherche dispersée	117,94	116,10	137,30	124,40
mémétique avec gestion de population	127,42	122,50	118,70	125,01
les algorithmes génétiques	132,70	118,00	121,30	119,70
les essaims particulaires	125,90	119,9	133,60	117,50
Les colonies de fourmis	155,20	130,00	121,40	119,80

Tableau 3.12 : Temps de cycle pour un taux de création de pièces =1/20

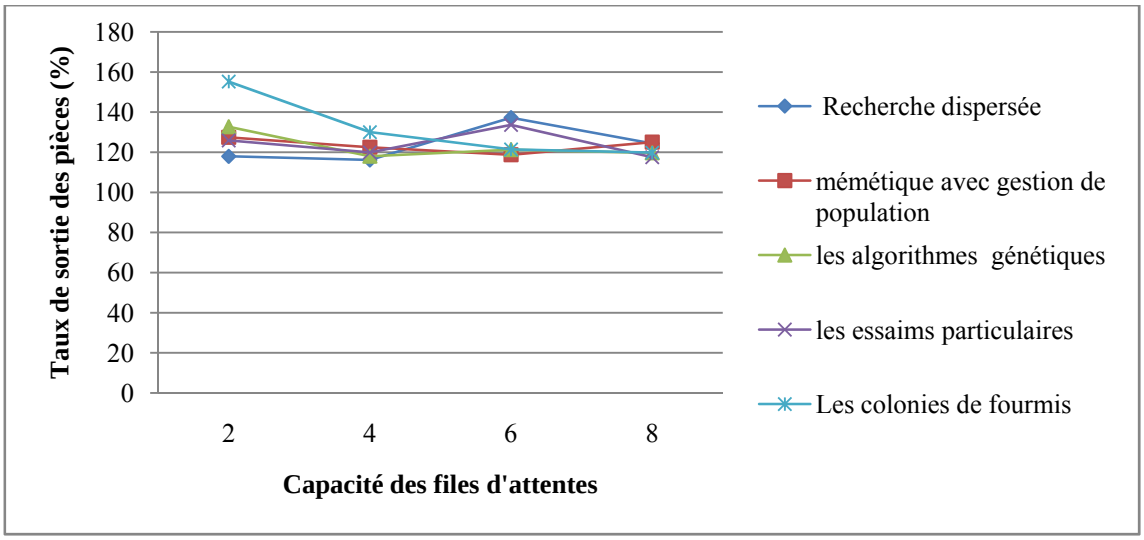


Figure 3.12 : Temps de cycle pour un taux de création de pièces =1/20

Le tableau 3.12 et le figures 3.12, montrent que les valeurs du temps de cycle obtenus sont proches pour toutes les métaheuristiques, nous pouvons dire que la recherche dispersée a pu améliorer le temps de cycle pour un système très saturé et de petites files d'attente où elle donne les meilleurs temps de cycle, et si la taille de file d'attente augmente l'algorithme mémétique avec gestion de population donne les meilleurs temps de cycle pour taille file égale à 6 mais pour taille file égale 8 les temps de cycle sont les minimaux pour les essaims particulaires.

Mais on peut dire que l'algorithme mémétique avec gestion de population en moyenne donne des valeurs acceptables.

3.5.3 Les en-cours

Les en-cours sont les pièces présentes dans le système. Cette sous section est consacrée aux en-cours. L'analyse présentée dans cette sous section montre la variation des en-cours en fonction du taux de création de pièces et en fonction des capacités de files d'attente.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/19**	**1/20**	**1/21**	**1/22**	**1/25**	**1/30**	**1/35**	**1/40**
Recherche dispersée	8,95	8,91	8,91	8,92	7,01	6,30	5,85	5,25	4,50	4,05	4,05
mémétique avec gestion de populati	8,92	8,93	8,94	8,95	7,25	6,39	5,85	5,19	4,51	4,05	3,90
les algorithmes génétiques	8,85	8,85	8,90	8,86	7,45	6,73	5,87	5,23	4,46	4,04	3,90
les essaims particulaires	8,64	8,67	8,67	8,70	7,35	6,53	5,92	5,25	4,45	4,03	3,88
Les colonies de fourmis	8,46	8,44	8,40	8,41	7,29	6,21	6,03	5,21	4,45	4,04	3,90

Tableau 3.13 : Les en-cours pour une capacité de file d'attente = 2

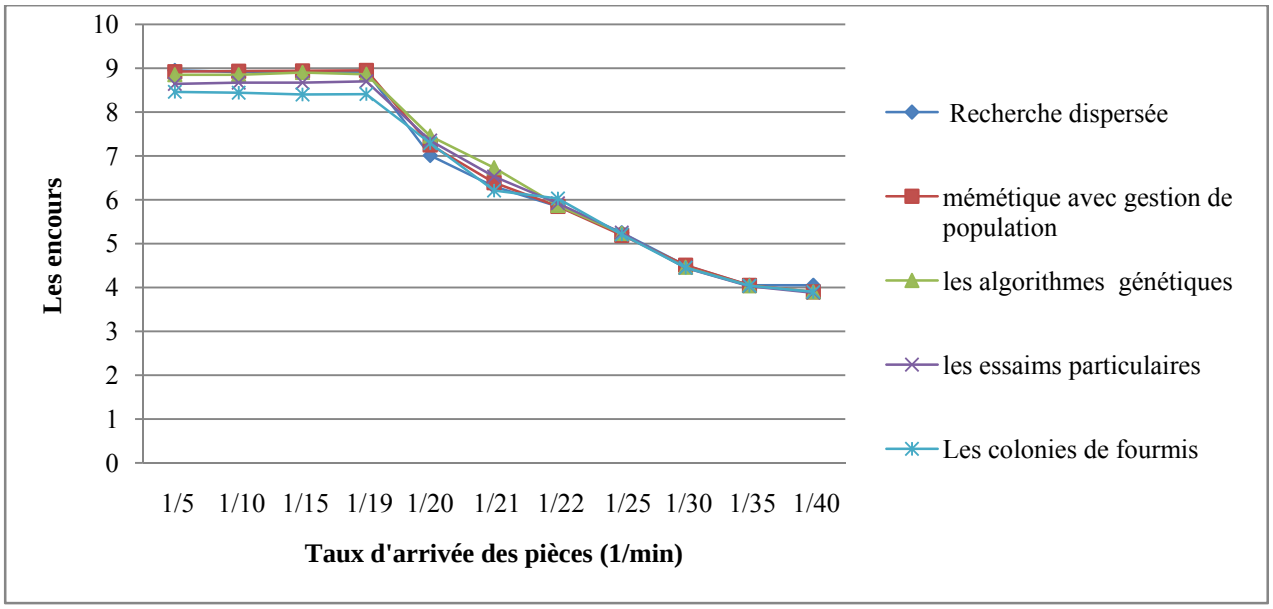


Figure 3.13 : Les en-cours pour une capacité de file d'attente = 2.

La figure et le tableau 3.13 montrent que pour des capacités de files d'attente égale à 2 et quand le système est saturé, le nombre de pièces qui sont restées dans le système des colonies de fourmis sont inférieurs à celui obtenu par les autres métaheuristique sauf pour un taux de création de 1/20, nous pouvons remarquer que la recherche dispersée est le plus efficace, et hors de cet intervalle il n'existe pas une métaheuristique qui donne de meilleurs résultats pour tous les taux.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/18	1/20	1/25	1/30	1/35	1/40
Rechercher dispersée	19,52	19,64	19,62	19,57	11,68	9,23	8,45	8,04	7,84
mémétique avec gestion de population	19,78	19,7	19,79	19,77	11.12	9,25	8,47	8,04	7,90
Les algorithmes génétiques	19,73	19,72	19,70	19,72	11,26	9,26	8,46	8,40	7,89
Les essaims particulaires	19,24	19,28	19,30	19,00	11,16	9,26	8,47	8,04	7,88
Les colonies de fourmis	18,56	18,56	18,50	18,50	11,31	9,20	8,45	8,02	7,81

Tableau 3.14 : Les en-cours pour une capacité de file d'attente = 6

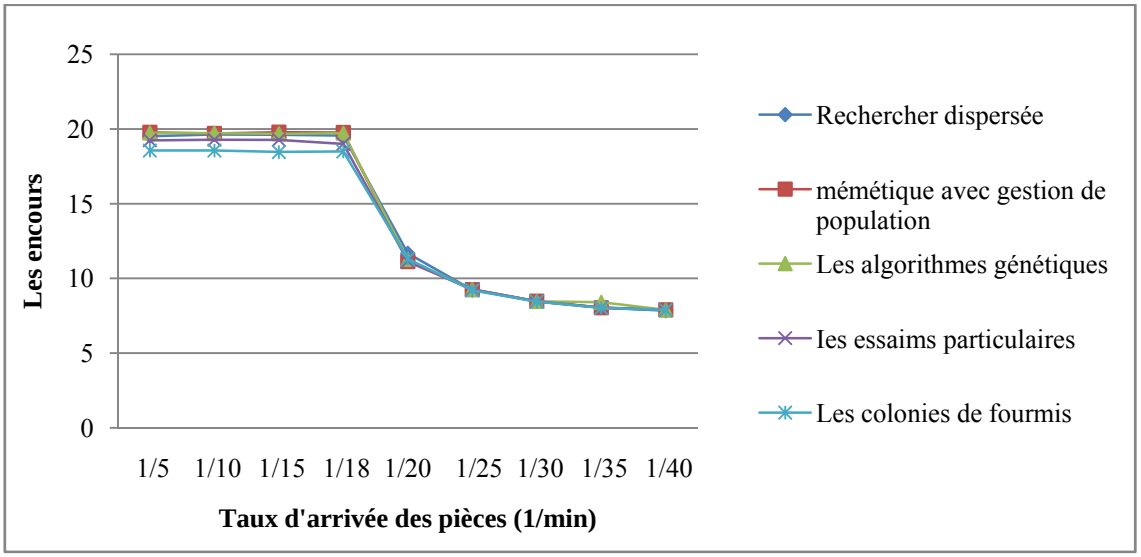


Figure 3.14 : Les en-cours pour une capacité de file d'attente = 6.

A partir des figures et tableaux 3.14, nous pouvons remarquer que si la taille de files d'attente est égale à 6, le nombre de pièces restantes obtenus par les colonies de fourmis sont inférieur à ceux obtenus par les autres métaheuristiques, sauf pour un taux égal à 1/20 on remarque que l'algorithme mémétique avec gestion de population est meilleur que les autres métaheuristiques. Et hors de cet intervalle il n'existe pas une métaheuristique qui donne les meilleurs résultats pour tous les taux.

Capacité des files d'attentes	**2**	**4**	**6**	**8**
Recherche dispersée	7,01	8.92	11,68	13,27
mémétique avec gestion de population	7,25	9,30	11.12	13,19
les algorithmes génétiques	7,45	9,35	11,26	13,15
les essaims particulaires	7,35	9,21	11,16	13,28
Les colonies de fourmis	7,29	9,31	11,31	13,30

Tableau 3.15 : Les en-cours pour un taux de création de pièces =1/20

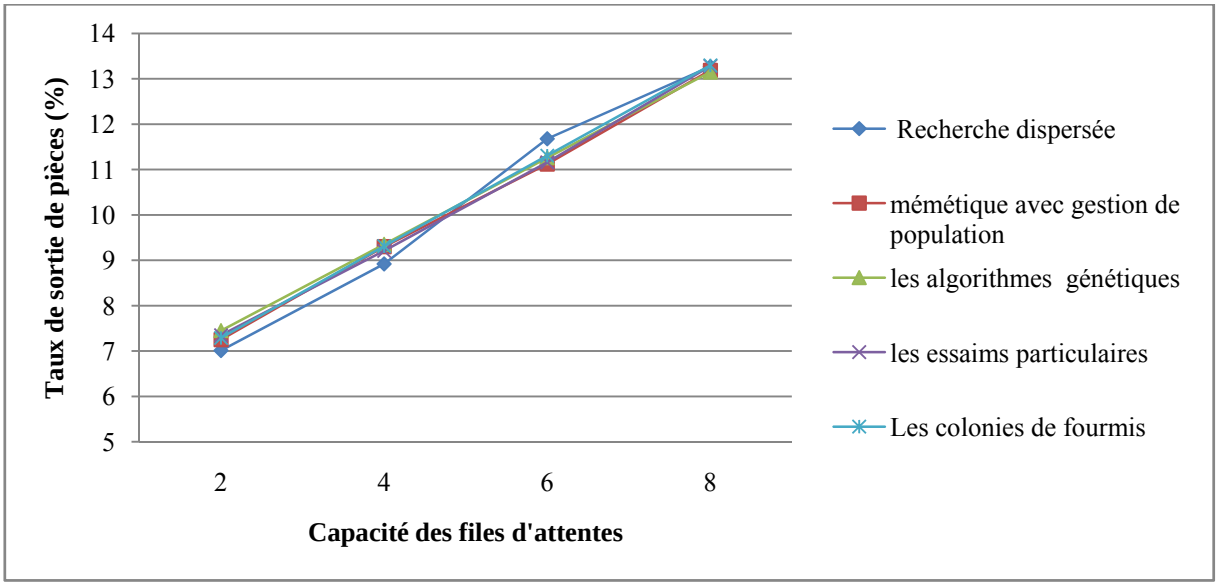


Figure 3.15 : Les en-cours pour un taux de création de pièces =1/20

A partir de tableaux 3.15 et de figure 3.15 on peut remarquer que pour une petite capacité de file d'attente et un taux de création 1/20, la recherche dispersée est meilleure que les autres métaheuristiques, si la taille de file d'attente augmente l'algorithme mémétique avec gestion de population est meilleure que les autres méthodes, car pour ce taux le système n'est pas saturé si on utilise la recherche dispersée.

3.5.4 Taux d'utilisation de l'AGV

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/19**	**1/20**	**1/21**	**1/22**	**1/25**	**1/30**	**1/35**	**1/40**
Recherche dispersée	32,75	32,72	32,75	32,73	33,31	31,87	30,56	27,44	23,41	19,46	17,34
mémétique avec gestion de populat	32,79	32,78	32.82	32,82	33,45	32,26	30,68	27,05	22,95	19,67	17,50
les algorithmes génétiques	32,68	32,67	32,69	32,65	33,07	32,20	30,90	27,04	23,16	19,54	17,43
les essaims particulaires	31,98	31,97	32 ,00	32,18	33,20	31,84	30,85	27,48	22,98	19,64	17,4
Les colonies de fourmis	29,41	29,46	29,37	31,16	30,64	31,8	30,88	27,36	22,89	19,58	17,47

Tableau 3.16: Le taux d'utilisation de l'AGV pour une capacité de file d'attente = 2

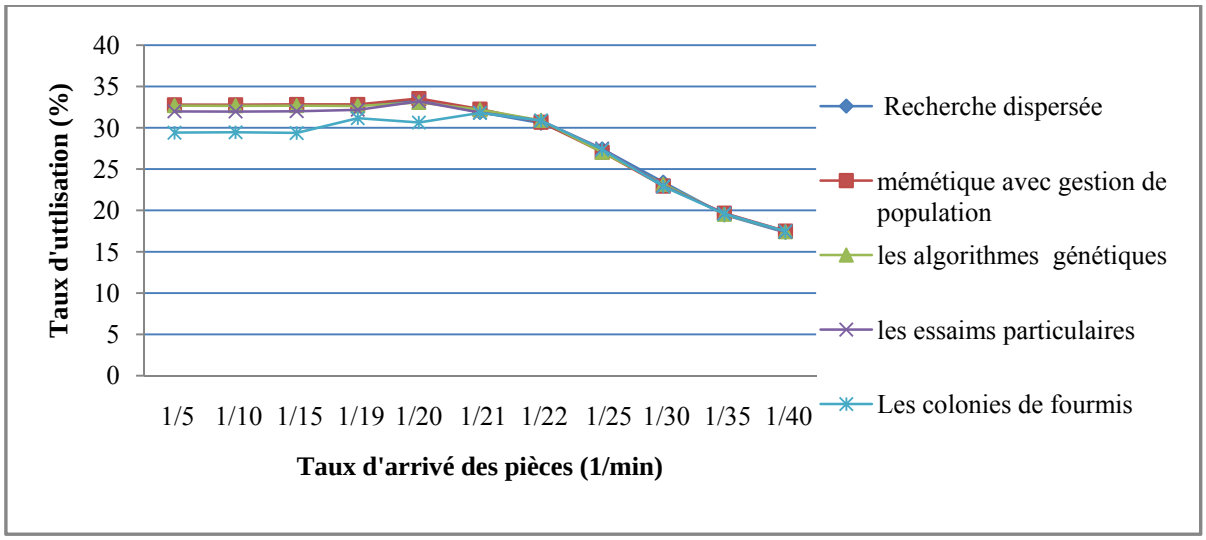


Figure 3.16: Le taux d'utilisation de l'AGV pour une capacité de file d'attente = 2.

La figure 3.16 montre que pour une taille de file d'attente égale à 2 et un taux de création de pièces supérieur à 1/21 et dans l'intervalle [1/40,1/35], le taux d'utilisation de l'AGV pour l'algorithme mémétique avec gestion de population est plus important que pour les autres métaheuristiques.

Pour un taux de création des pièces égale à 1/22 les algorithmes génétiques sont les meilleurs, les essaims particulaires sont les plus efficaces si on les compare avec les autres métaheuristiques pour un taux égale à 1/25, la recherche dispersée est la meilleure pour un taux de création de pièces égale à 1/30.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/18**	**1/20**	**1/25**	**1/30**	**1/35**	**1/40**
Rechercher dispersée	33,33	33,28	33,25	33,29	33,37	27,10	22,83	19,64	17,15
mémétique avec gestion de population	33,59	33,58	33,48	33,49	33,41	27,35	23,12	19,49	17,37
Les algorithmes génétiques	33,56	33,59	33,6	33,55	33,16	27,47	23,10	19,70	17,46
Les essaims particulaires	33,00	32,98	33,00	33,00	33,27	27,41	23,08	19,55	17,46
Les colonies de fourmis	32,33	32,29	32,40	32.20	33,26	27,39	23,05	19,67	17,52

Tableau 3.17: Le taux d'utilisation de l'AGV pour une capacité de file d'attente = 6

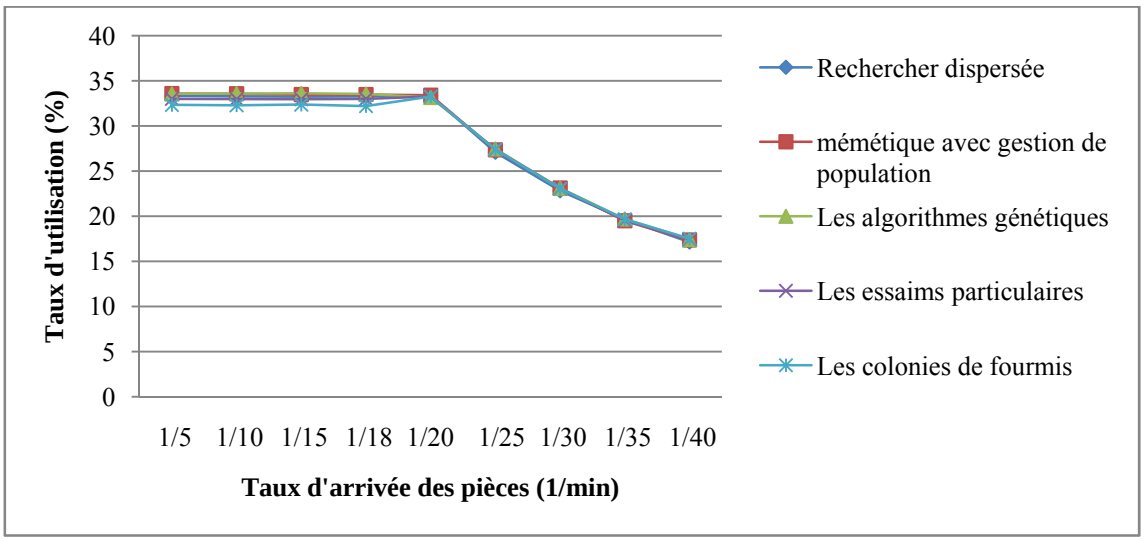


Figure 3.17: Le taux d'utilisation de l'AGV pour une capacité de file d'attente = 6.

La figure 3.17 montre que pour une taille de files d'attente égale à 6 et un taux de création de pièces égale à 1/5 et 1/20 l'algorithme mémétique avec gestion de population donne le meilleur taux d'utilisation d'AGV. Et dans l'intervalle [1/18,1/10] et [1/40,1/25] les taux d'utilisation d'AGV pour les algorithmes génétiques sont meilleurs que ceux obtenus par les autres métaheuristiques

3.5.5 Taux d'utilisation des machines

Le taux d'utilisation des machines est un critère très important dans la mesure des performances d'un système de production.

Les courbes des figures 3.18 et 3.19 nous donnent le taux d'utilisation moyen des machines FV_1 et FV_2.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/19	1/20	1/21	1/22	1/25	1/30	1/35	1/40
Recherche dispersée	32.64	32.39	32.57	32.77	33.15	31.21	29,43	24,52	19,07	17,87	14,85
mémétique avec gestion de populatio	31.18	31.26	31.02	31.20	32,50	30,23	29,15	25,52	20,22	17,33	14,46
les algorithmes génétiques	28,72	28,36	28,60	29,12	31,94	31,27	28,57	25,54	19,70	17,67	14,00
les essaims particulaires	27,72	27,44	27,61	27,60	32,51	30,11	28,71	24,44	20,16	17,42	14,71
Les colonies de fourmis	26,67	26,93	26,77	25,85	28,78	30,11	28,63	24,74	20,66	17,56	14,53

Tableau 3.18: Taux d'utilisation des machines FV_1 et FV_2 pour une capacité de file d'attente=2

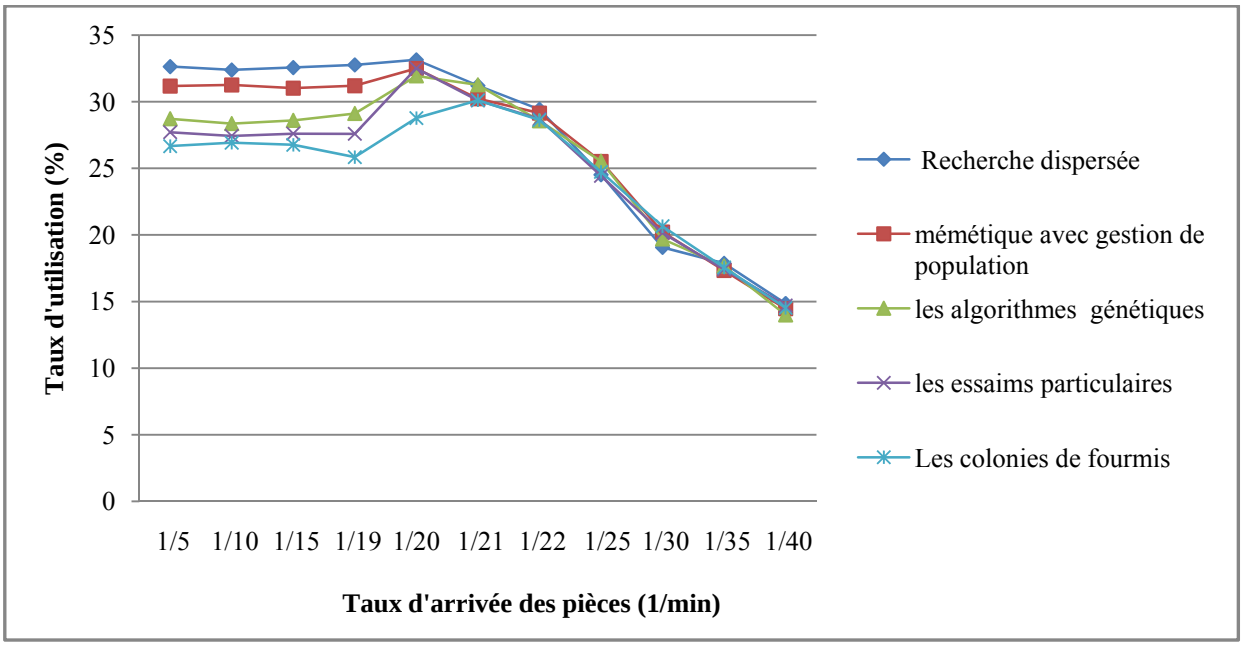


Figure 3.18: Taux d'utilisation des machines FV_1 et FV_2 pour une capacité de file d'attente=2

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/18	1/20	1/25	1/30	1/35	1/40
Rechercher dispersée	30.72	30.76	30.66	30.77	32,99	25,39	20,53	17,42	15,32
mémétique avec gestion de population	29.78	30.01	29.86	30.15	31.43	24,77	19,81	17,78	14,77
Les algorithmes génétiques	28,70	28,38	28,4	28,67	33,51	24,44	19,83	17,28	14,62
Les essaims particulaires	28,23	28,09	28,00	28,00	33,51	24,59	19,91	17,64	14,50
Les colonies de fourmis	26,22	26,12	26,10	26,09	33,27	24,64	20,00	17,57	14,4

Tableau 3.19: Taux d'utilisation des machines FV_1 et FV_2 pour une capacité de file d'attente=6

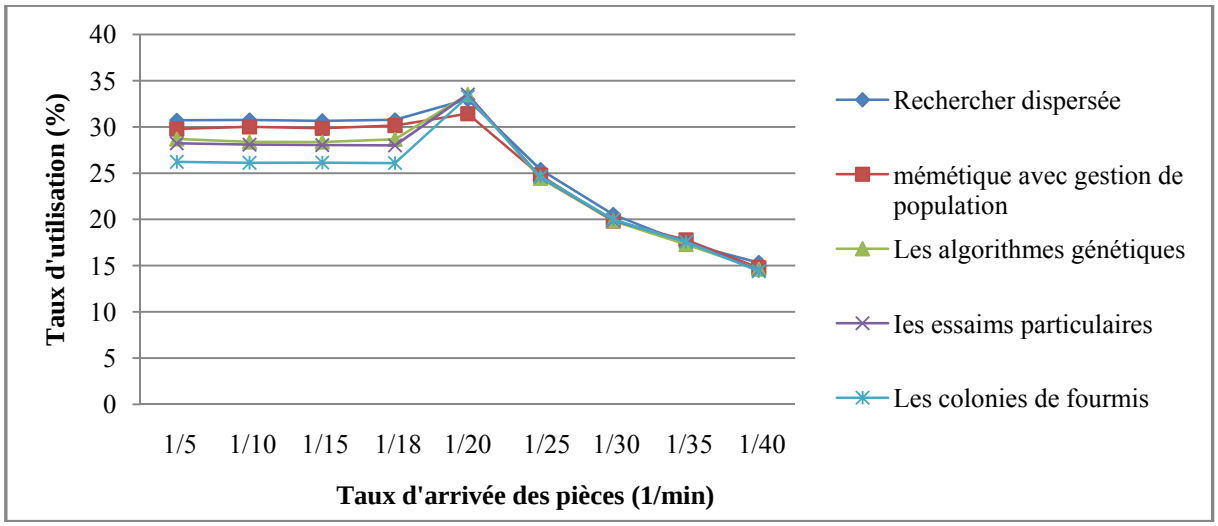


Figure 3.19 : Taux d'utilisation des machines FV1 et FV2 pour une capacité de file d'attente=6

On remarque pour taux d'utilisation des machines FV1 et FV2 pour un système saturé, que la recherche dispersée est plus efficace que les autres métaheuristiques.

- Pour de petites capacités de files d'attente, l'algorithme recherche dispersée est le meilleure pour un taux d'arrivée de pièces compris entre [1/19,1/5] et pour un taux inférieur à 1/35, l'algorithme mémétique avec gestion de population est le plus efficace pour un taux compris entre [1/35,1/25].
- Pour une capacité de file d'attente égale à 6, la recherche dispersée est la meilleure pour un taux d'arrivée de pièces sauf pour un taux égale à 1/35, où l'algorithme mémétique avec gestion de population est plus efficace.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/19**	**1/20**	**1/21**	**1/22**	**1/25**	**1/30**	**1/35**	**1/40**
Recherche dispersée	92.04	91.92	92.04	92,04	93.79	89,45	85,67	76,50	64,83	54,35	48,18
mémétique avec gestion de populat	91,82	91,81	91.87	91.90	94.08	89,87	85,90	75,69	63,91	54,77	48,49
les algorithmes génétiques	90,96	90,85	90,98	90,98	92,74	90,10	86,36	75,69	64,83	54,90	48,36
les essaims particulaires	88,95	88,84	89,20	89,59	93,20	89,40	86,24	76,55	63,96	54,70	48,29
Les colonies de fourmis	82,06	82,26	81,97	86,39	85,73	89,02	86,31	76,31	63,99	54,59	48,44

Tableau 3.20: Taux d'utilisation des machines T_1 et T_2 pour une capacité de file d'attente=2

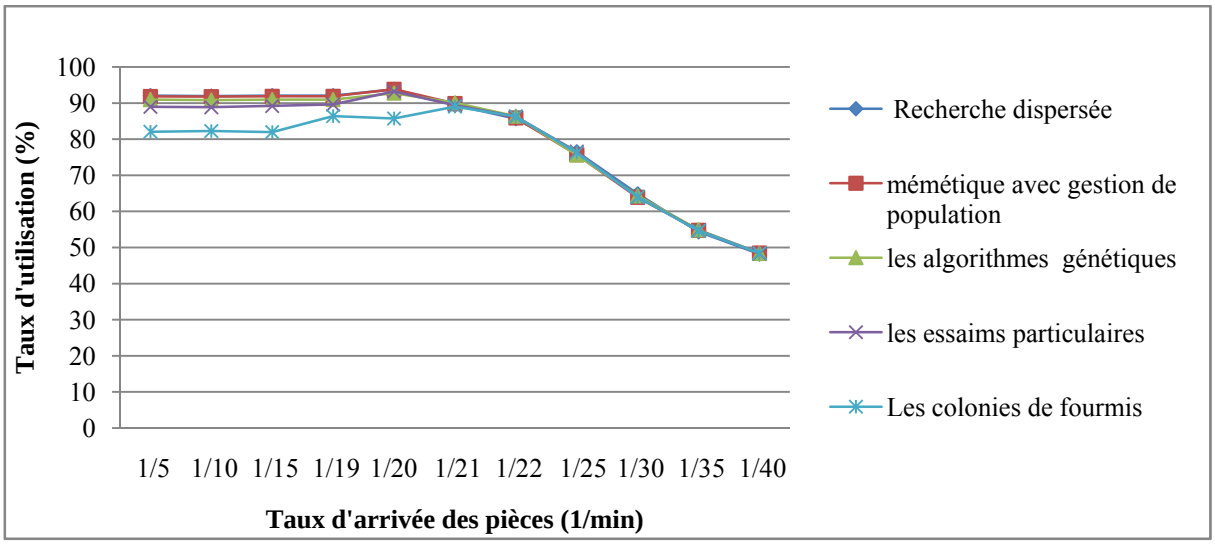


Figure 3.20: Taux d'utilisation des machines T_1 et T_2 pour une capacité de file d'attente=2

Les courbes de la figure et le tableau 3.20 montrent que pour de petites files d'attente et un taux de création de pièces supérieur à 1/20, les résultats obtenus par la recherche dispersée sont meilleurs que ceux obtenus pour les autres métaheuristiques. Pour un taux de création égale à 1/20 l'algorithme mémétique avec gestion de population est la meilleure, et pour un taux de création inférieur à 1/20, les résultats obtenus par les métaheuristiques restent voisines.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/18**	**1/20**	**1/25**	**1/30**	**1/35**	**1/40**
Rechercher dispersée	93.05	93.02	92,88	92,95	93,74	75,79	63,66	54,70	47,80
mémétique avec gestion de population	93,41	93,39	93,30	93,36	93,99	76,30	64,24	54,42	48,24
Les algorithmes génétiques	93,24	93,25	93,30	93,20	93,33	76,55	64,28	54,82	48,37
Les essaims particulaires	91,69	91,59	91,50	93,36	93,54	76,43	64,16	54,53	48,41
Les colonies de fourmis	89,48	89,35	89,60	89,12	93,52	76,39	64,09	54,76	48,54

Tableau 3.21: Taux d'utilisation des machines T_1 et T_2 pour une capacité de file d'attente=6

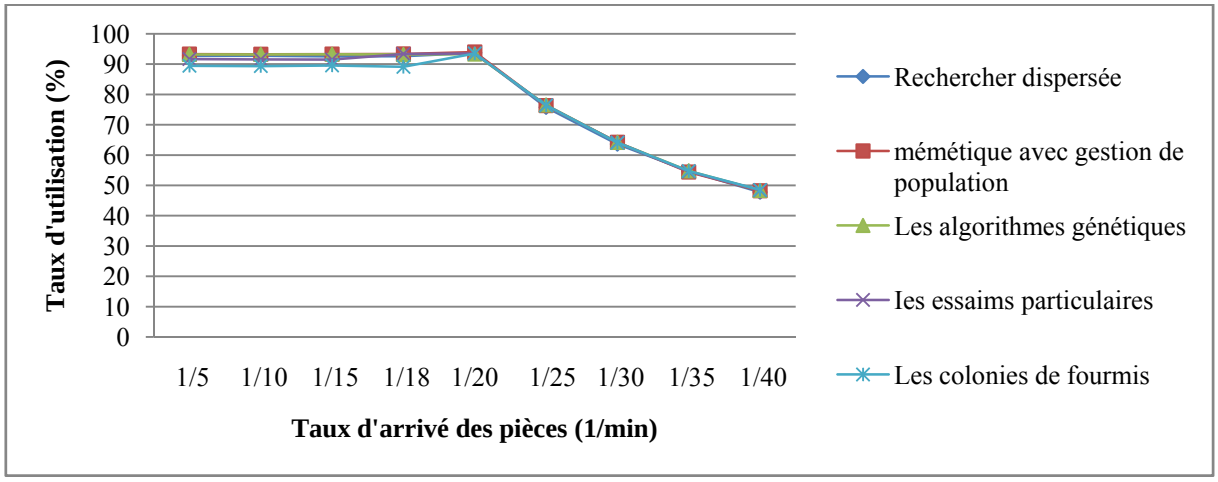


Figure 3.21: Taux d'utilisation des machines T1 et T2 pour une capacité de file d'attente=6

Les courbes de la figure et le tableau 3.21 montrent que le taux d'utilisation pour les machines T1 et T2 est plus important pour l'algorithme mémétique avec gestion de population que pour les autres métaheuristiques étudiées, pour un taux de création important (supérieur ou égale à 1/20). Hors de cet intervalle il devient moins performant mais les résultats obtenus par les cinq métaheuristiques restent voisines, sauf pour certains cas :

- Pour un taux de création compris entre [1/35,1/25] et une taille de files d'attente égale à 6, les algorithmes génétiques sont les meilleures.
- -Pour un taux de création égale à 1/35, les colonies de fourmis donnent les meilleurs taux d'utilisation des machines.

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/19**	**1/20**	**1/21**	**1/22**	**1/25**	**1/30**	**1/35**	**1/40**
Recherche dispersée	2,247	2,253	2,250	2,240	2,287	2,207	2,136	1,989	1,766	1,392	1,281
mémétique avec gestion de populat	2,305	2,301	2,315	2,308	2,333	2,285	2,159	1,910	1,674	1,435	1,312
Les algorithmes génétiques	2,387	2,399	2,393	2,369	2,300	2,275	2,204	1,907	1,711	1,407	1,299
Les essaims particulaires	2,351	2,359	2,357	2,361	2,307	2,202	2,193	1,995	1,679	1,428	1,292
Les colonies de fourmis	2,117	2,113	2,108	2,134	2,169	2,241	2,199	1,970	1,663	1,417	1,307

Tableau 3.22: Taux d'utilisation de la machine TP pour une capacité de file d'attente=2

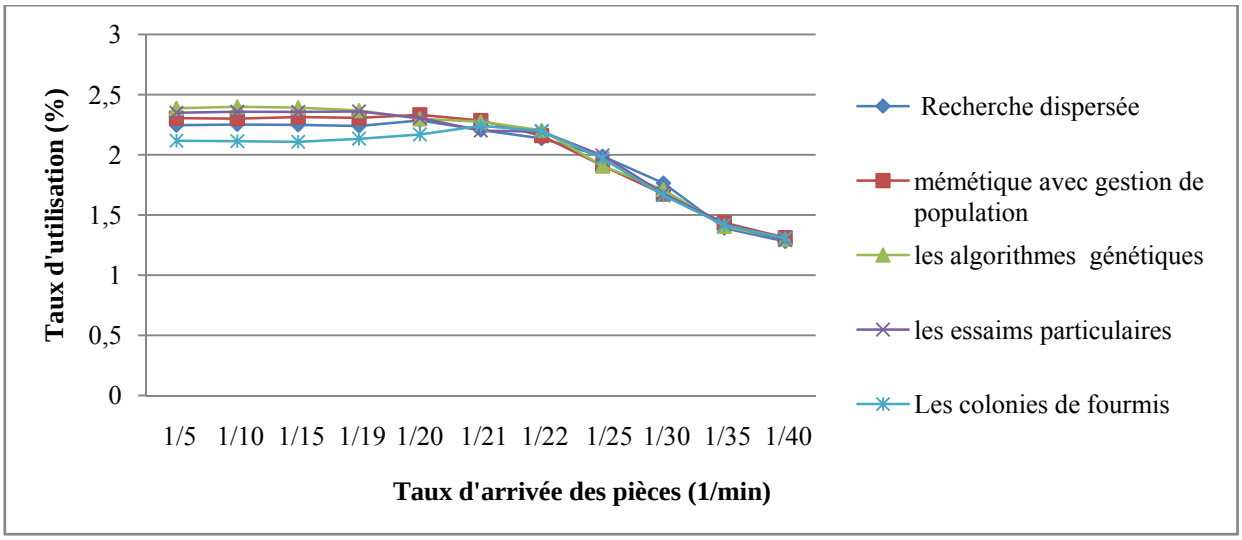


Figure 3.22: Le taux d'utilisation de la machine TP pour une capacité de file d'attente=2

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/18	1/20	1/25	1/30	1/35	1/40
Rechercher dispersée	2,376	2,375	2,374	2,439	2,299	1,919	1,649	1,428	1,243
mémétique avec gestion de population	2,485	2,484	2,424	2,419	2,308	1,969	1,708	1,399	1,287
Les algorithmes génétiques	2,481	2,497	2,500	2,481	2,258	1,995	1,706	1,440	1,299
les essaims particulaires	2,440	2,442	2,440	2,440	2,279	1,983	1,699	1,410	1,304
Les colonies de fourmis	2,443	2,443	2,45	2,235	2,277	1,978	1,693	1,434	1,317

Tableau 3.23: Taux d'utilisation de la machine TP pour une capacité de file d'attente=6

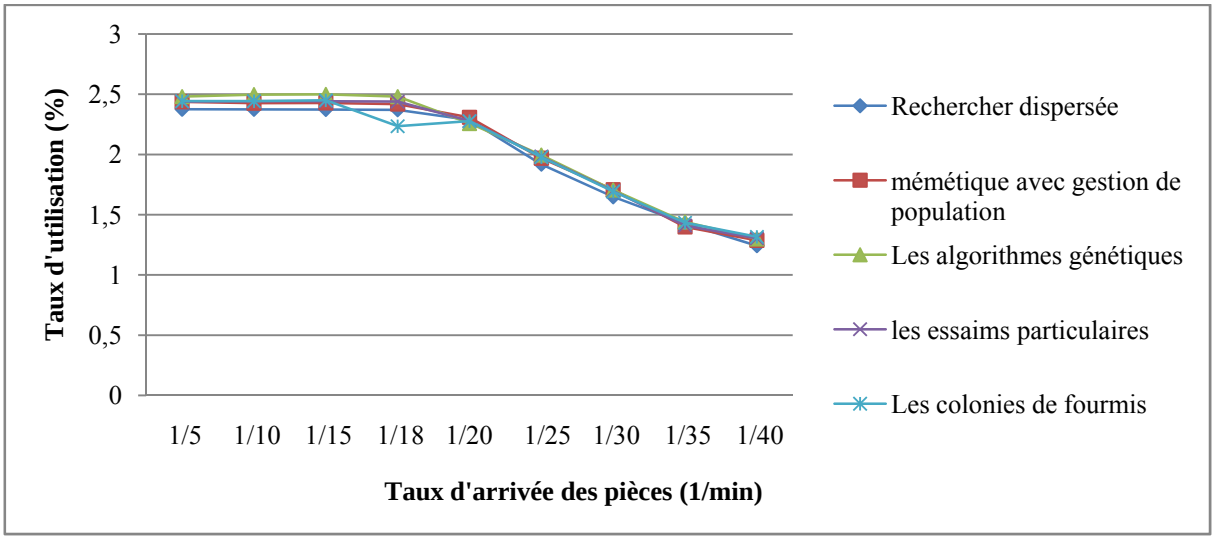


Figure 3.23: Le taux d'utilisation de la machine TP pour une capacité de file d'attente=6

Le taux d'utilisation de la toupie (voir les figures 3.22 et 3.23) est plus faible pour les quatre métaheuristiques que celui obtenu par les algorithmes génétiques pour des taux de création supérieurs à 1/20 et taille file 6, hors de cet intervalle les résultats obtenus par les cinq métaheuristiques restent voisines.

Nous pouvons remarquer qu'il n'existe pas une grande différence entre les métaheuristiques mais nous pouvons voir que pour un système saturé, les algorithmes génétiques sont les meilleurs pour les deux tailles de files d'attente égale à 2 et 6.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/19	1/20	1/21	1/22	1/25	1/30	1/35	1/40
Recherche dispersée	39.67	40.04	40.23	39.82	40.65	39,22	37,95	35,31	31,33	24,73	22,73
mémétique avec gestion de populat	40,76	40,88	41,13	41,01	41,45	39,96	38,35	33,92	29,72	25,47	23,27
les algorithmes génétiques	42,36	42,57	42,47	42,05	40,99	40,41	39,14	33,88	30,45	25,00	23,05
les essaims particulaires	41,71	41,87	41,82	41,80	40,87	39,13	38,94	35,42	29,89	25,35	22,92
Les colonies de fourmis	37,59	37,52	37,44	38,44	38,52	39,81	39,06	35,00	29,52	25,15	23,19

Tableau 3.24: Taux d'utilisation des machines FH_1 et FH_2 pour une capacité de file d'attente=2

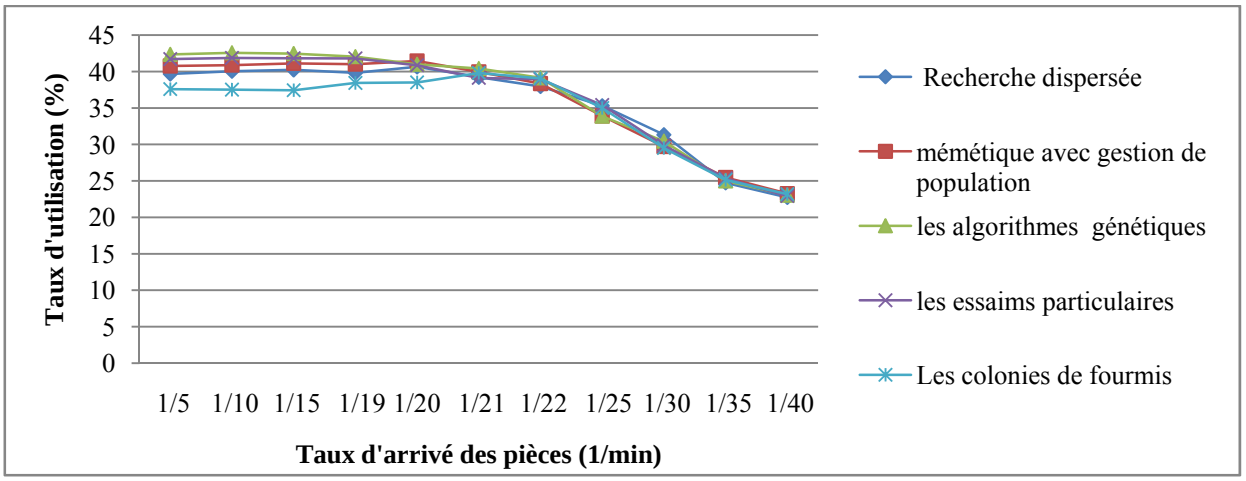


Figure 3.24: Taux d'utilisation des machines FH_1 et FH_2 pour une capacité de file d'attente=2

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/18	1/20	1/25	1/30	1/35	1/40
Rechercher dispersée	42,28	42,54	42,16	42,11	40,65	34,09	29,29	25,35	22,07
mémétique avec gestion de population	43,26	43,07	43,10	42,95	41,01	34,97	30,31	24,85	22,84
Les algorithmes génétiques	44,02	44,29	44,30	44,02	40,14	35,43	30,27	25,55	23,06
Les essaims particulaires	43,28	43,33	43,30	43,00	40,52	35,22	30,16	25,05	23,14
Les colonies de fourmis	43,33	43,32	43,50	43,19	40,48	35,15	30,04	25,45	23,36

Tableau 3.25: Taux d'utilisation des machines FH1 et FH2 pour une capacité de file d'attente=6

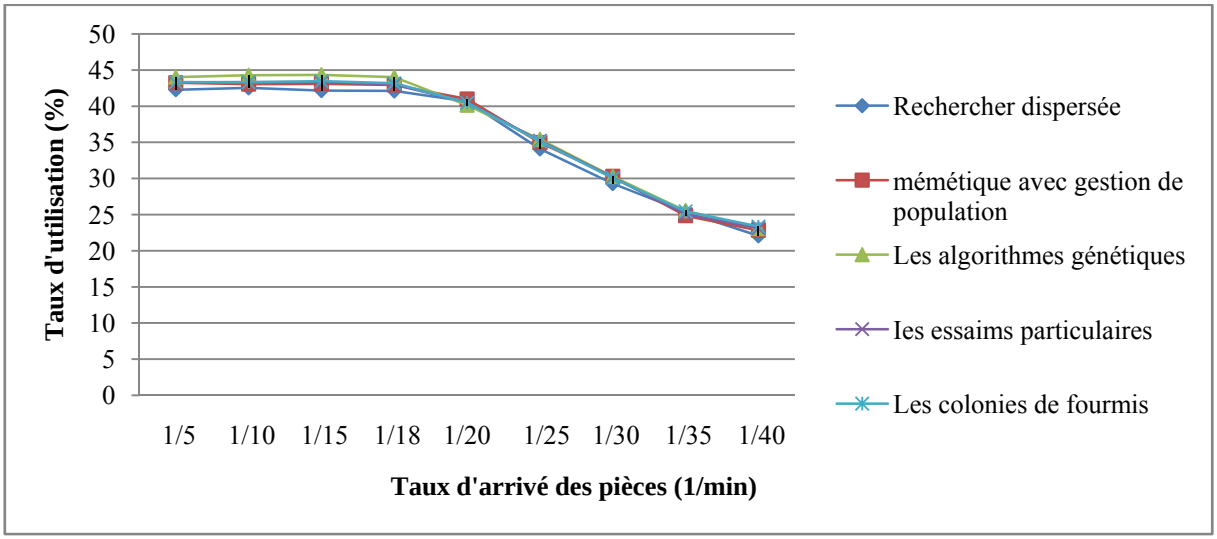


Figure 3.25: Taux d'utilisation des machines FH1 et FH2 pour une capacité de file d'attente=6

L'utilisation des machines FH1 et FH2 (voir les figures 3.24 et 3.25) pour un taux de création supérieur à 1/20, les algorithmes génétiques sont plus efficaces que les autres

métaheuristiques, Pour un système non saturé, nous ne pouvons pas dire qu'une métaheuristique est la meilleure pour tous les taux d'arrivée des pièces mais dans chaque cas certaines métaheuristiques sont meilleures que d'autres.

3.6 Etude comparative avec introduction de pannes

Dans cette section, nous nous proposons d'étudier et de comparer les métaheuristiques étudiées avec l'introduction de pannes. Les pannes sont introduites sur les ressources toutes les 100 heures. La panne va durer 2 heures suivant une loi exponentielle.

3.6.1 Taux de production

La simulation du système en introduisant des pannes, nous montre que les taux de production pour la recherche dispersée étudiée est nettement supérieur à ceux des métaheuristique avec une capacité de file d'attente égale à 2 (voir figure 3.26).

Taux d'arrivée des pièces (1/min)	**1/5**	**1/10**	**1/15**	**1/20**	**1/25**	**1/30**	**1/35**	**1/40**
Rechercher dispersée	23,98	47,90	72,02	95,21	99,99	99,99	99,99	99,99
mémétique avec gestion de populati	23,84	47,42	71,21	94,34	99,99	99,99	99,99	99,99
Les algorithmes génétiques	23,12	46,22	69,02	92,00	99,99	99,99	99,99	99,99
Les essaims particulaires	22,61	44,99	67,5	90,04	99,99	99,99	99,99	99,99
Les colonies de fourmis	21,13	42,11	63,13	84,47	99,98	99,99	99,99	99,99

Tableau 3.26: Taux de sortie des pièces pour une capacité de file d'attente = 2

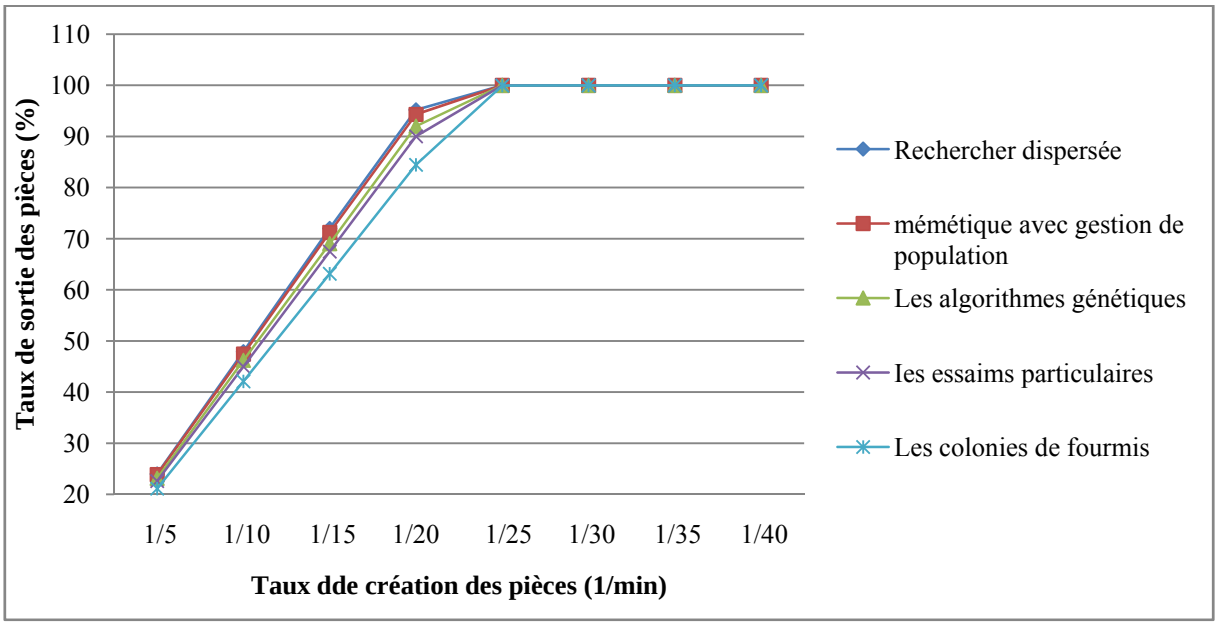


Figure 3.26: Taux de sorite des pièces pour une capacité de file d'attente = 2.

La figure 3.26 nous montre aussi que le taux de production pour une file d'attente égale à 2 est plus important pour la recherche dispersée que pour les autres métaheuristiques pour un taux de création supérieur à 1/20 et hors de cet intervalle les taux de production obtenus par les métaheuristiques sont identiques.

Capacité de file d'attente	2	4	6	9
Rechercher dispersée	95,21	99,99	99,99	99,99
mémétique avec gestion de population	94,34	99.99	99,99	99,99
Les algorithmes génétiques	92,00	98,99	99,99	99,99
Les essaims particulaires	90,04	99,99	99,99	99,99
Les colonies de fourmis	84,47	90,27	99,99	99,99

Tableau 3.27: Taux de sortie des pièces pour un taux de création de pièces = 1/20

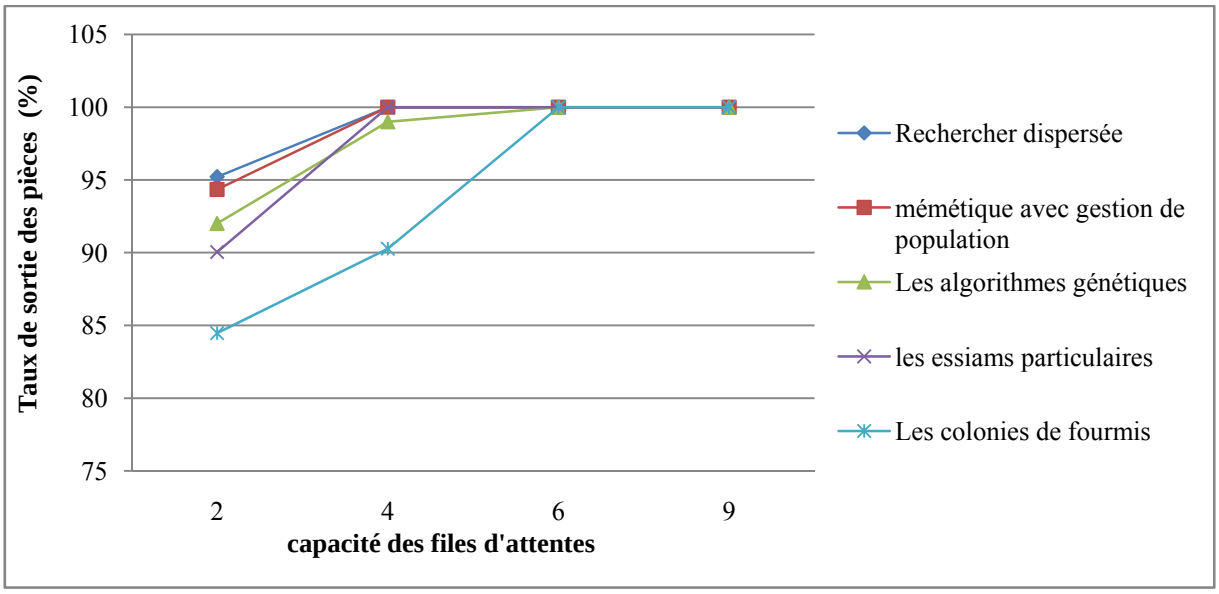


Figure 3.27 : Taux de sortie des pièces pour un taux de création de pièces = 1/20

La figure 3.27 nous montre que pour un taux de création de pièces égale à 1/20, le taux de production est meilleur pour la recherche dispersée par rapport aux autres métaheuristiques pour une taille de files d'attente égale à 2.

Les essaims particulaires et la recherche dispersée et l'algorithme mémétique avec gestion de population sont les plus efficaces pour une taille de files d'attente égale à 4. Pour de grandes capacités de files d'attente les résultats obtenus par les métaheuristiques sont presque les mêmes.

De ces résultats avec et sans présence de pannes, nous pouvons constater que le taux de production en utilisant la recherche dispersée est plus important que celui en utilisant les autres métaheuristiques pour un taux de création de pièces très élevé (supérieur à 1/20). Nous pouvons dire que les essaims particulaires et l'algorithme mémétique avec gestion de population sont aussi les meilleurs pour un taux de création de pièces égale à 1/20.

3.6.2 Le temps de cycle

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Rechercher dispersée	166,93	169,95	167,59	172,98	97,27	91,09	83,61	98,61
mémétique avec gestion de populati	167,97	167,27	174,13	176,57	102,00	100,67	83,36	101,52
Les algorithmes génétiques	176,20	175,20	174,60	170,80	104,60	93,30	85,70	94,20
Les essaims particulaires	173,90	174,60	175,10	170,80	103,20	93,20	84,90	95,50
Les colonies de fourmis	175,40	167,30	169,80	169,80	104,00	90,10	82,60	97,50

Tableau 3.28: Temps de cycle pour une capacité de file d'attente = 2.

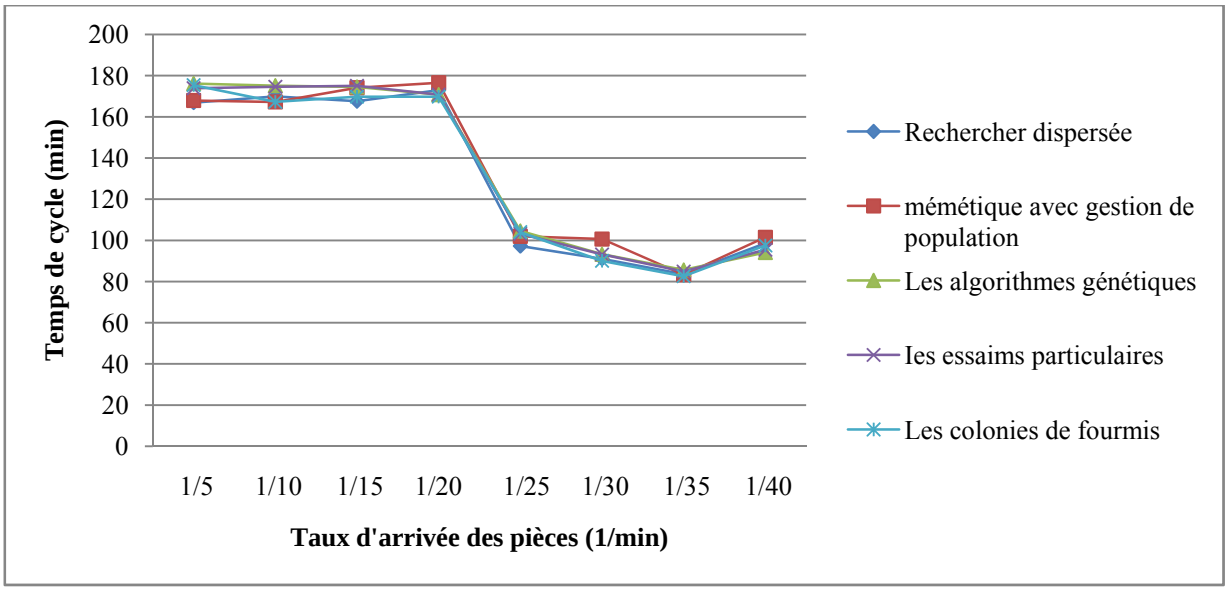


Figure 3.28: Temps de cycle pour une capacité de file d'attente = 2.

La figure et tableau 3.28 montrent que pour un taux de création de pièces égale à 1/5 et une taille de file d'attente égale à 2, la rechercher dispersée est la plus efficace si on s'intéresse aux temps de cycle. Dans l'intervalle [1/10,1/20], les temps de cycles obtenus par l'algorithme mémétique avec gestion population est meilleur par rapport aux autres métaheuristiques. Hors de cet intervalle, nous ne pouvons pas dire que la recherche dispersée est la meilleure par rapport aux autres.

Capacité des files d'attentes	2	4	6	9
Rechercher dispersée	166,93	145,22	150,71	149,16
mémétique avec gestion de population	167.97	150,48	145,94	151,55
Les algorithmes génétiques	170,80	161,00	150,90	145,40
Les essaims particulaires	170,80	157,80	144,70	149,60
Les colonies de fourmis	169,80	239,20	141,70	153,70

Tableau 3.29: Temps de cycle pour un taux de création de pièces = 1/20

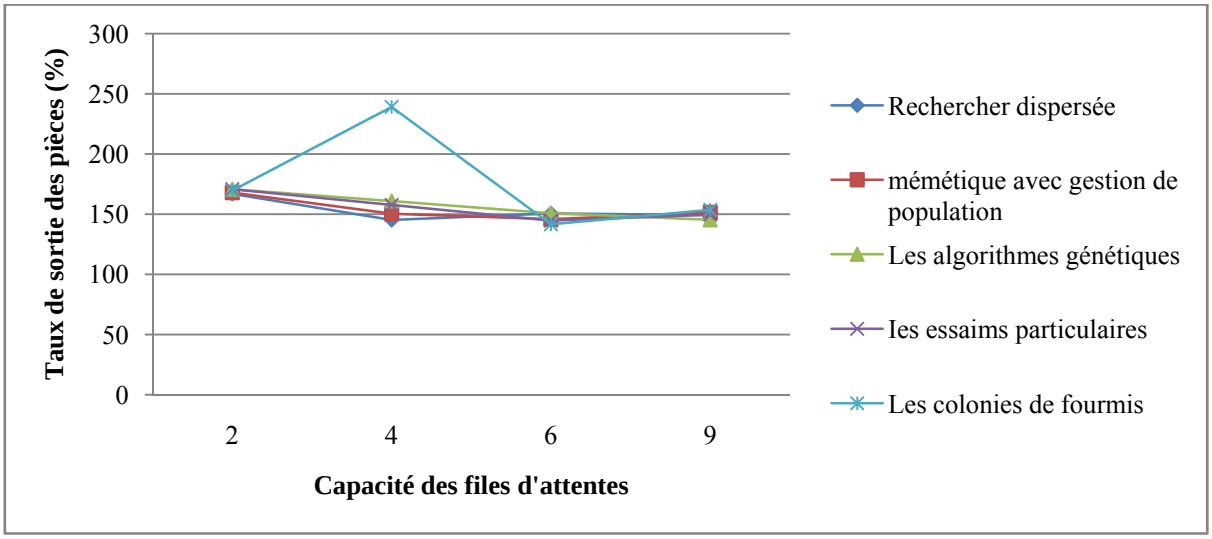


Figure 3.29: Temps de cycle pour un taux de création de pièces = 1/20

La figure 3.29 nous montre qu'en variant les capacités de files d'attente la recherche dispersée est la meilleur une taille de files d'attente égale à 2 et4, mais en moyenne le temps de cycle nous pouvons dire que les deux méthodes étudies sont les meilleures pour toutes les taux de création de pièces.

3.6.3 Les en-cours

Le nombre de pièces qui restent dans le système quand il est saturé plus petite pour les essaims particulaires, et la recherche dispersée vis-à-vis de l'algorithme génétique et mémétique avec gestion de population, Hors de cet intervalle, les valeurs des en-cours obtenues sont, en général, proches pour toutes les méthodes (voir la figure 30)

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	8,94	8,90	8,94	8,94	5,36	4,56	4,09	3,92
mémétique avec gestion de populatior	8,94	8,93	8,91	8,91	5,37	4,60	4,08	3,93
les algorithmes génétiques	8,81	8,85	8,87	8,83	5,34	4,57	4,00	3,92
les essaims particulaires	8,70	8,68	8,68	8,67	5,39	4,55	4,09	3,94
Les colonies de fourmis	8.46	8.43	8.43	8.46	5.40	4.55	4.09	3.95

Tableau 3.30: Les en-cours pour une capacité de file d'attente = 2.

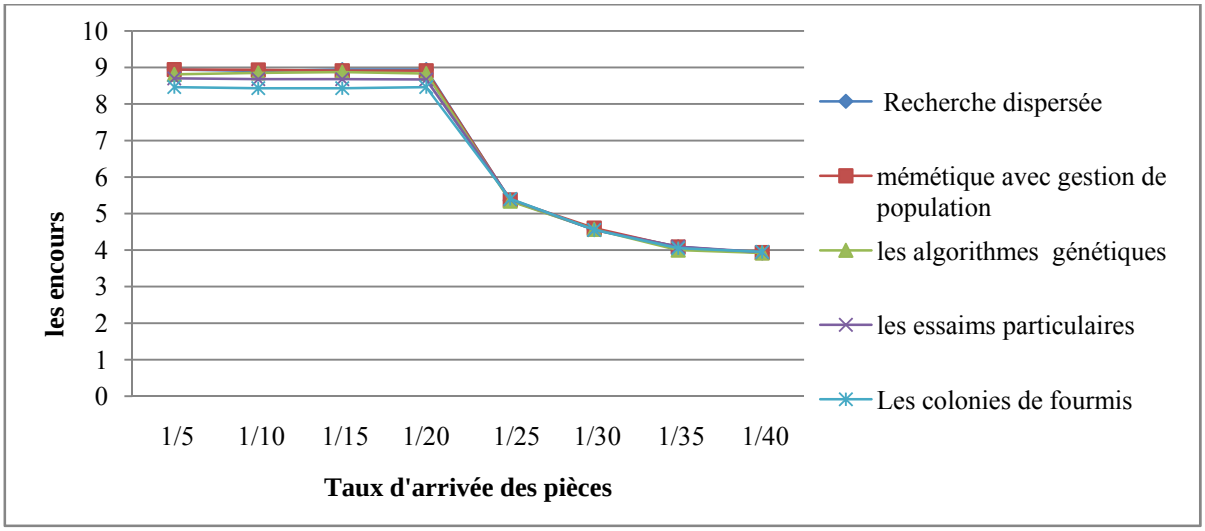


Figure 3.30: Les en-cours pour une capacité de file d'attente = 2.

Capacité des files d'attentes	2	4	6	9
Recherche dispersée	8,94	10,26	12,47	14,28
mémétique avec gestion de population	8,94	10,28	12,06	14,28
les algorithmes génétiques	8,83	11,42	12,28	14,36
les essaims particulaires	8,67	10,13	12,47	14,43
Les colonies de fourmis	8,46	13,37	12,30	14,13

Tableau 3.31: Les en-cours pour un taux de création de pièces = 1/20

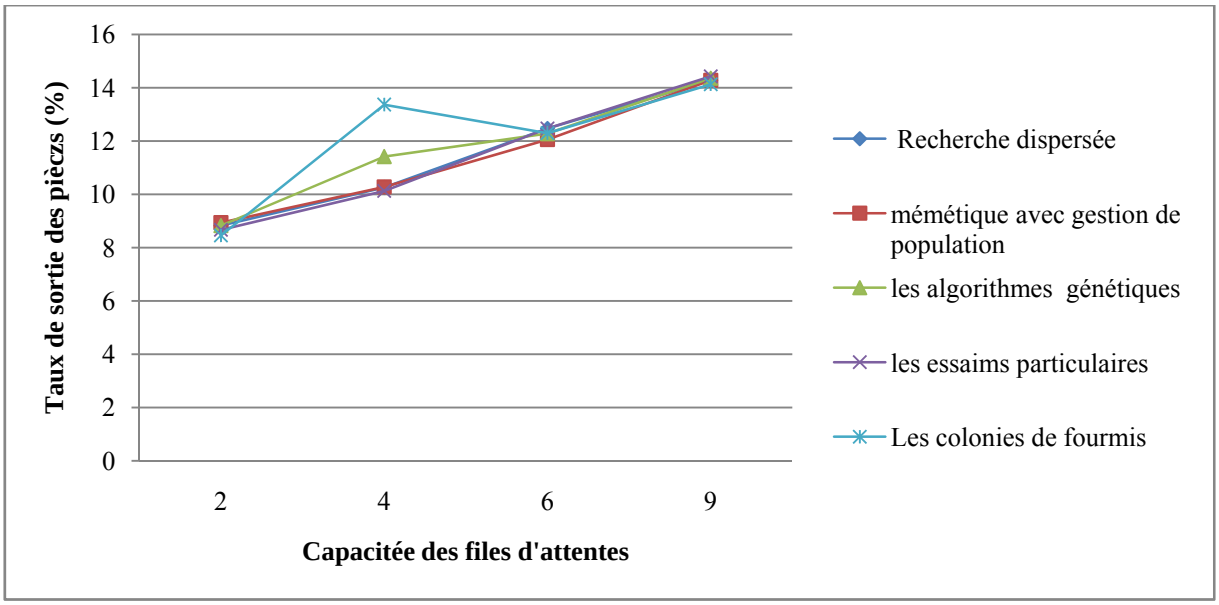


Figure 3.31: Les en-cours pour un taux de création de pièces = 1/20.

La figure 3.31 montre que pour une taille de file d'attente égale à 2 et 8, les colonies de fourmis sont les plus efficace, si on s'intéresse aux en-cours. Si la taille de file d'attente est augmente, nous pouvons dire que les essaims particulaires ont améliorées les résultats qui concernent les en-cours pour une taille égale à 4 tandis que l'algorithme mémétique avec gestion de population est la meilleure pour une taille supérieure à 6.

3.6.4 Le taux d'utilisation de l'AGV

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	31,81	31,84	31,78	31,86	27,49	23,00	19,54	17,26
mémétique avec gestion de populatio	31,98	31,94	32,00	31,87	27,25	23,26	19,44	17,49
les algorithmes génétiques	31,85	31,83	31,79	31,78	27,25	23,02	19,73	17,40
les essaims particulaires	31,24	31,15	31,16	31,20	27,42	23,13	19,68	17,57
Les colonies de fourmis	28,84	28,75	28,74	28,82	27,27	22,95	19,57	17,39

Tableau 3.32: Taux d'utilisation de l'AGV pour une capacité de file d'attente=2

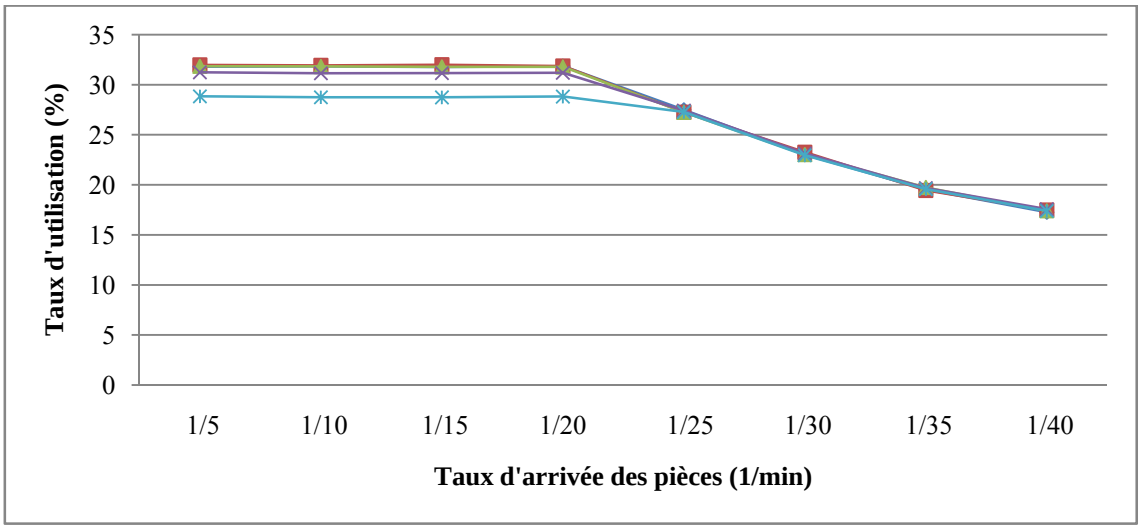


Figure 3.32: Taux d'utilisation de l'AGV pour une capacité de file d'attente=2

Capacité des files d'attentes	2	4	6	9
Recherche dispersée	31,81	32,95	32,55	32,75
mémétique avec gestion de population	31,98	32,64	32,87	32,82
les algorithmes génétiques	31,78	32,82	32,86	32,93
les essaims particulaires	31,20	32,68	32,95	32,77
Les colonies de fourmis	28,82	31,43	32,88	32,69

Tableau 3.33: Taux d'utilisation de l'AGV pour un taux de création de pièces =1/20

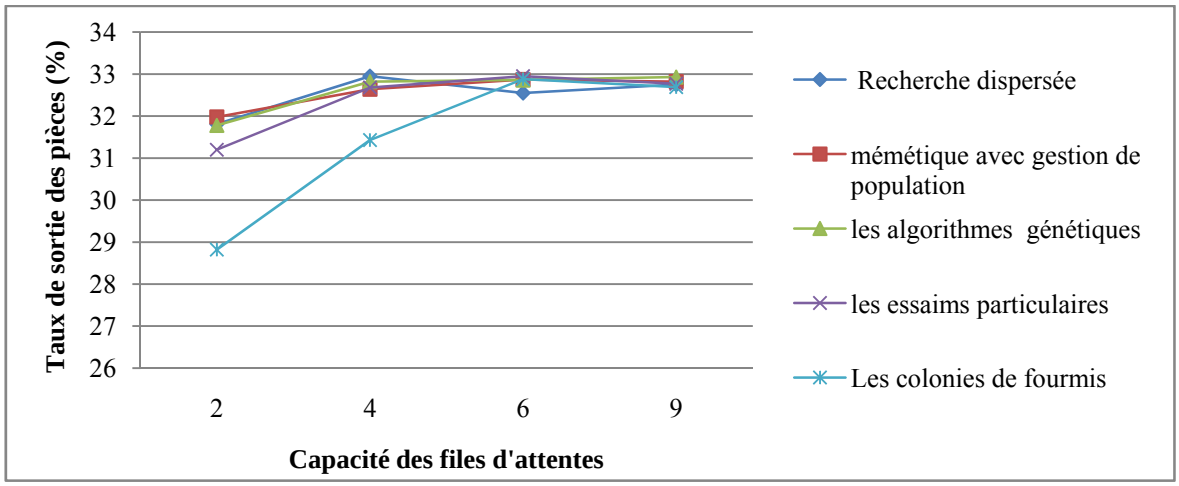


Figure 3.33 : Taux d'utilisation de l'AGV pour un taux de création de pièces =1/20

Les figures 3.32 et 3.33 nous montrent que pour un système saturé, de petites files d'attente égale à 2 et avec la présence de pannes, le taux d'utilisation de l'AGV est plus important pour mémétique avec gestion de population. Si on augmente les files d'attente à 4 la recherche dispersée est la meilleure et pour file d'attente égale à 6 les essaims particulaires sont les plus efficaces mais pour taille file égale à 8 les algorithmes génétiques dépassent les autres. Pour un système non saturé, l'utilisation de l'AGV les résultats obtenus par les cinq métaheuristiques restent voisines.

3.6.5 Taux d'utilisation des machines

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	32,16	31,96	31.75	31.20	24,38	20,19	17.68	15.05
mémétique avec gestion de populatio	30.34	30,50	30,54	30,16	25,02	19,45	17,91	14,48
les algorithmes génétiques	28,04	28,05	27,72	27,10	25,02	20,07	17,18	14,70
les essaims particulaires	27,20	26,86	26,87	26,83	24,57	19,78	17,30	14,29
Les colonies de fourmis	26,33	26,18	26,15	26,30	24,95	20,24	17,58	14,71

Tableau 3.34: Le taux d'utilisation des machines FV_1 et FV_2 pour une capacité de file d'attente=2

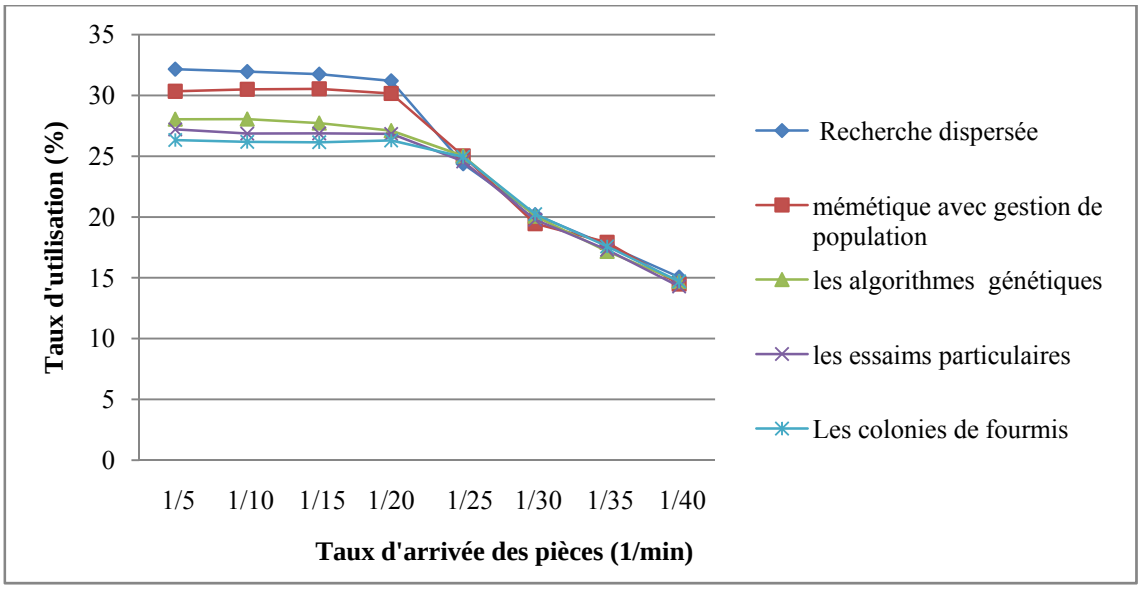


Figure 3.34: Taux d'utilisation des machines FV_1 et FV_2 pour une capacité de file d'attente=2

La figure et tableau 3.34 montrent aussi que pour une capacité de file d'attente égale à 2, l'utilisation des machines FV1 et FV2 est plus importante pour la recherche dispersée par rapport les quatre méthodes et l'algorithme mémétique avec gestion de population est plus efficace pour un taux de création de pièces égale à 1/25 et 1/35. Les colonies de fourmis sont les meilleures, pour un taux de création égale à 1/30.

Capacité des files d'attentes	2	4	6	9
Recherche dispersée	32,16	34,04	35,04	34,54
mémétique avec gestion de population	30,34	34,81	34,25	34,36
les algorithmes génétiques	27,10	33,14	34,28	34,11
les essaims particulaires	26,83	33,81	34,04	34,49
Les colonies de fourmis	26,30	29,53	34,23	34,70

Tableau 3.35: Taux d'utilisation des machines FV_1 et FV_2 pour un taux de création des pièces=1/20

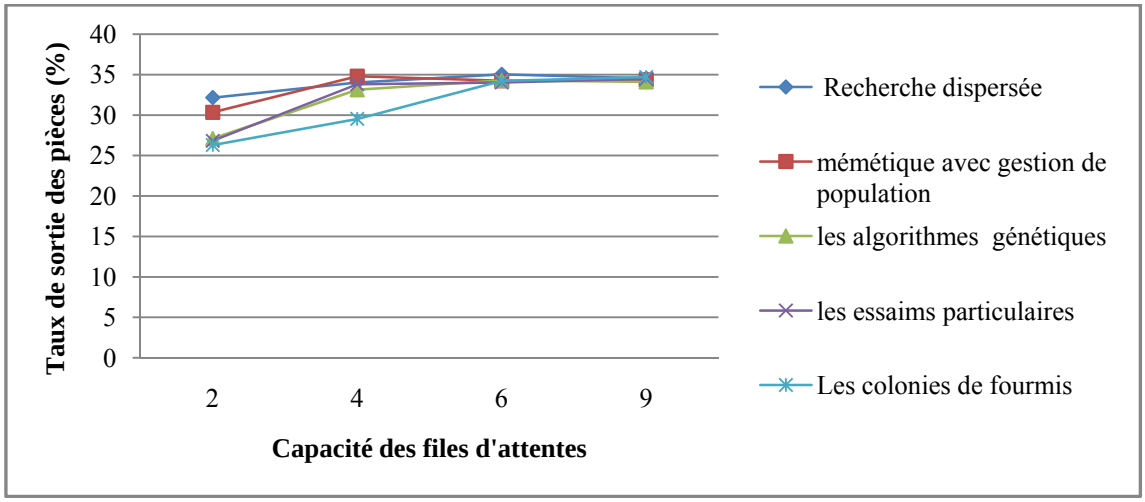


Figure 3.35: Taux d'utilisation des machines FV_1 et FV_2 pour un taux de création des pièces=1/20

La figure et tableau 3.35 nous montrent que pour un taux de création égale à 1/20, nous pouvons dire que la meilleure utilisation des machines FV1 et FV2 est donnée par la recherche dispersée pour une taille de file d'attente égale à 2 et 6, l'algorithme mémétique avec gestion de

population pour une taille égale à 4, les colonies de fourmis pour une taille de file d'attente égale à 8.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	88,24	88,32	88,29	88,53	76,57	63,95	55,09	48,82
mémétique avec gestion de populatio	88,68	88,68	88,55	88,74	76,62	64,12	54,77	48,09
les algorithmes génétiques	88,67	88,63	88,44	88,13	76,10	64,04	54,90	48,30
les essaims particulaires	86,90	86,61	86,63	86,71	76,45	64,26	54,80	48,64
Les colonies de fourmis	80,50	80,24	80,44	76,13	63,90	54,58	54,58	48,29

Tableau 3.36: Taux d'utilisation des machines T_1 et T_2 pour une capacité de file d'attente=2

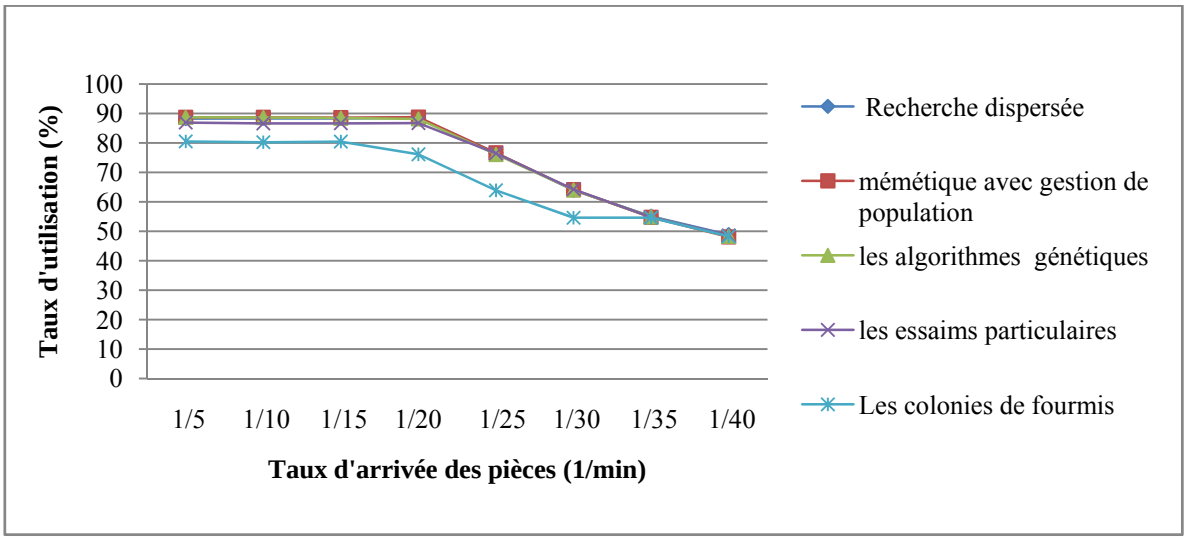


Figure 3.36: Taux d'utilisation des machines T_1 et T_2 pour une capacité de file d'attente=2

A partir de la figure et tableau 3.36, nous pouvons remarquer que pour une capacité de file d'attente égale à 2, l'utilisation des machines T1 et T2 est plus importante pour l'algorithme mémétique avec gestion population par rapport à les autres métaheuristiques pour un taux de création de pièces supérieur à 1/30. Pour un taux de création égale 1/30, les essaims particulaires dépassent les autres métaheuristiques. Pour un taux de création inférieur à 1/30, la recherche dispersée est la meilleure.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	2,165	2,176	2,178	2,207	2,001	1,684	1,408	1,265
mémétique avec gestion de populatio	2,251	2,240	2,246	2,247	1,950	1,736	1,389	1,311
les algorithmes génétiques	2,324	2,322	2,330	2,340	1,950	1,687	1,447	1,293
les essaims particulaires	2,291	2,295	2,295	2,300	1,985	1,710	1,430	1,320
Les colonies de fourmis	2,069	2,065	2,068	2,068	1,954	1,673	1,415	1,292

Tableau 3.37: Taux d'utilisation de la machine TP pour une capacité de file d'attente=2

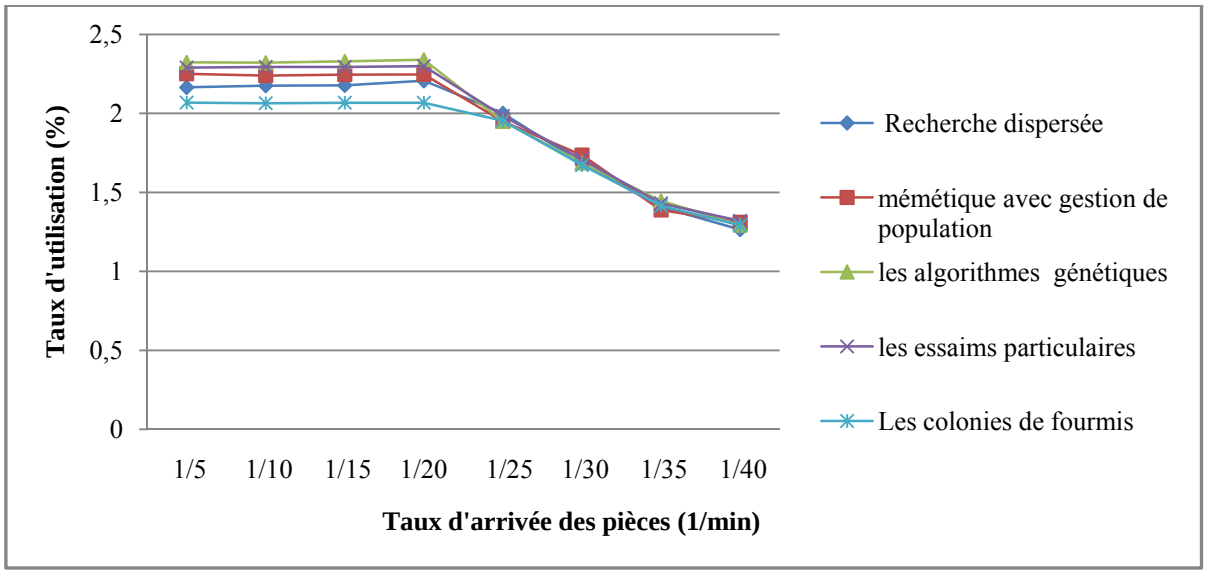


Figure 3.37: Taux d'utilisation de la machine TP pour une capacité de file d'attente=2

Comme nous l'avons bien remarqué (voir la figure et tableau 3.37) l'utilisation de la machine TP est meilleure pour les algorithmes génétiques par rapport à les autre métaheuristiques, pour un taux de création supérieur à 1/25, Hors de cet intervalle, les valeurs obtenues sont, en général, proches pour toutes les méthodes.

Taux d'arrivée des pièces (1/min)	1/5	1/10	1/15	1/20	1/25	1/30	1/35	1/40
Recherche dispersée	38,50	38,68	38,91	39,22	35,51	29,89	25,00	22,63
mémétique avec gestion de populatio	39,99	39,80	39,91	39,91	34,62	30,80	24,66	23,25
les algorithmes génétiques	41,25	41,22	41,36	41,63	34,62	29,94	25,69	22,93
les essaims particulaires	40,66	40,72	40,73	40,82	35,24	30,34	25,51	23,52
Les colonies de fourmis	36,74	36,67	36,66	36,72	34,70	29,70	25,12	22,92

Tableau 3.38: Taux d'utilisation des machines FH_1 et FH_2 pour une capacité de file d'attente=2

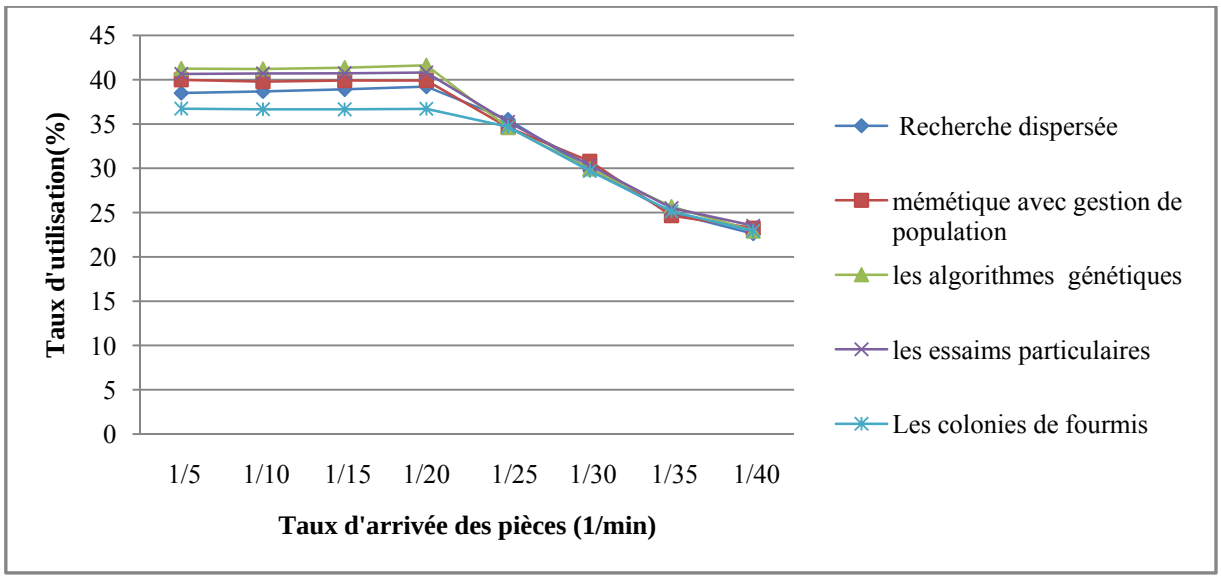


Figure 3.38: Taux d'utilisation des machines FH_1 et FH_2 pour une capacité de file d'attente=2

Les figures et tableaux 3.38 montrent que pour une capacité de files d'attente égale à 2, si le taux de création des pièces est supérieur à 1/25, le taux d'utilisation des machines FH1 et FH2

est plus important pour les algorithmes génétiques par rapport aux autres métaheuristiques , et qu'en dessous de taux de création 1/25 l'utilisation de ces machines les valeurs obtenues sont, en général, proches pour toutes les méthodes.

3.7 Conclusion

Dans ce chapitre, nous avons présentés notre adaptation de deux métaheuristiques à base de population de solutions. Ce sont des méthodes évolutionnaires appelées algorithme mémétique avec gestion de la population (Memetic Algorithm with Population Management – MA/PM) et recherche dispersée (scatter search). Pour le première, Il s'agit d'un algorithme génétique hybridées avec des techniques de recherche locale et de mesure de distance permettant de contrôler la diversification des solutions dans la population, dans seconde on extrait un ensemble de référence R contenant les meilleures solutions de la population initiale.

Nous avons présenté une analyse de sensibilité de ces algorithmes, en étudiant l'influence de variation des paramètres internes de la métaheuristique sur ses performances, pour pouvoir tirer des informations sur les meilleurs valeurs des paramètres de la méthode.

Ces approches ont été comparées avec d'autres méthodes évolutionnaires, pour pouvoir tirer une idée sur l'efficacité de chacune des méthodes utilisées. Ensuite nous avons donné les résultats et les interprétations obtenus des différentes simulations effectuées en ajoutant des pannes aux machines du système.

Les simulations ont été faites en variant, à chaque fois, l'un des critères du système et en gardant l'autre fixe, (le taux d'arrivée des pièces, et la capacité des files d'attente). De cette simulation, un certain nombre de conclusions peuvent être tirés :

- L'étude de sensibilité de l'algorithme MA/PG et SS a permis de définir les paramètres qui améliorent les performances des métaheuristiques considérées.
- Les résultats obtenus ont tous montré que les métaheuristiques que nous avons étudiées ont donné de meilleures performances pour le taux de production, pour un système flexible de production saturé surtout pour de petites capacités de files d'attente même avec la présence de pannes.
- Pour le temps de cycle, nous ne pouvons pas dire qu'une métaheuristique donne un temps de cycle meilleur que les autres pour toutes les capacités de files d'attente, mais on peut dire que l'algorithme mémétique avec gestion de population donne un temps de cycle raisonnable pour toutes les capacités des files d'attentes.
- Dans le cas sans panne, les en-cours dans les colonies de fourmis sont les meilleurs pour un taux de création supérieur à 1/20 et de petites files d'attente, pour de grandes files d'attente et un système saturé, la recherche dispersée et l'algorithme mémétique avec gestion de la population sont les plus efficaces que les autres métaheuristiques. Dans le cas avec panne, nous ne pouvons pas dire qu'une métaheuristique est la meilleure pour toutes les capacités de files d'attente.
- le taux d'utilisation de l'AGV, l'algorithme mémétique avec gestion de population est plus efficace que les autres métaheuristiques pour un taux de création supérieur à 1/20 et de petites files d'attente et pour un taux de création des pièces égale à1/20 la recherche dispersée est meilleure que les autre.

- le taux d'utilisation des machines, l'algorithme mémétique avec gestion de population et la recherche dispersée en génerale sont les plus efficaces pour un taux de création de pièces supérieur à 1/20 tandis que pour les machines FH1 et FH2, les algorithmes génétiques donnent les meilleures utilisations pour ces deux machines.

Enfin, la question qui ce pose toujours c'est « Mais pourquoi n'avez-vous pas utilisé la méthode X plutôt que la méthode Y? ». Le seul impératif qui ressort c'est qu'il faut essayer. On ne peut assurément pas répondre à une question demandant le résultat d'une autre métaheuristique sans l'avoir implémenté et testé au paravent.

CONCLUSION GENERALE

Dans ce mémoire, nous avons abordé la résolution d'un problème d'ordonnancement dans un système flexible de production à l'aide d'algorithmes évolutionnaires. Afin d'évaluer les différentes approches proposées, nous avons adapté ces méthodes pour la résolution du problème de sélection de routages alternatifs en temps réel dans un système flexible de production.

Le choix de ces deux approches est justifié par la complexité du problème d'ordonnancement faisant partie des problèmes d'optimisation combinatoire, et présentant un intérêt dans le monde industriel.

Le domaine des algorithmes évolutionnaires a connu un essor important ces dernières années et possède un important potentiel de développement et d'application aux problématiques industrielles réelles.

L'aspect itératif de ces techniques peut rendre leur utilisation difficile pour la résolution des problèmes complexes en temps réel, mais devant le succès rencontré par ces techniques dans le cadre de résolution des autres problèmes NP-difficiles, tout au long de ce travail notre but a été leur adaptation pour résoudre un problème d'ordonnancement dans un FMS.

Par conséquent, nous avons proposé des algorithmes sur la base de certaines métaheuristiques : La recherche dispersée et l'algorithme mémétique avec gestion de population pour résoudre notre problème afin d'avoir une idée sur leur efficacité et de sélectionner la plus fiable. Pour cela, nous les avons appliqué sur un système flexible de production et nous avons choisi les algorithmes évolutionnistes (les colonies de fourmis, les algorithmes génétiques, les essaims particulaires) comme base de comparaison des performances de nos algorithmes.

Le premier chapitre expose la problématique d'ordonnancement dans les systèmes flexibles de production, et donne quelques notions générales : les systèmes de production flexibles, les problèmes d'ordonnancement, l'ordonnancement de type job shop, et leurs méthodes de résolution.

Après avoir présenté les méthodes approchées d'une manière générale, le deuxième chapitre était réservé à une présentation détaillée des métaheuristiques, dans laquelle nous avons présenté les plus connues, nous avons exposé leurs origines, principes de bases, et leurs algorithmes. La dernière section du chapitre était réservée aux deux méthodes que nous avons adapté, à savoir, l'algorithme mémétique avec gestion de la population, et celui de la recherche dispersée.

La seconde étape de notre étude est abordée dans le troisième chapitre, en premier lieu on présente le modèle FMS de test et les algorithmes proposés pour résoudre le problème de sélection de routages alternatifs en temps réel. En deuxième lieu, nous avons validé nos algorithmes pour l'ordonnancement dans les systèmes flexibles de production par la simulation. Un modèle de simulation a été implémenté sur Core (TM) 2 Duo CPU avec 2.1 GHz et 2 Gode RAM, en variant le taux d'arrivée des pièces et en gardant la capacité de la file d'attente fixe. Les critères de performance considérés sont : le taux de production (taux de sortie des pièces), le temps de cycle, les en-cours, le taux d'utilisation de l'AGV, et le taux de production des machines.

D'après l'analyse des tests, les résultats suivants ont été trouvés:

- Les résultats obtenus ont tous montré que les métaheuristiques que nous avons étudiée ont donné de meilleures performances pour le taux de production, le taux d'utilisation des machines et du système de transport pour un système flexible de production saturé surtout pour de petites capacités de files d'attente même avec la présence de pannes.
- Pour le temps de cycle, nous ne pouvons pas dire qu'une métaheuristique donne un temps de cycle meilleur que les autres pour toutes les capacités de files d'attente, mais on peut dire que le recherche dispersée donne un temps de cycle raisonnable pour un système saturé et de petites files d'attente.

Par ailleurs, ce travail a permis d'ouvrir de nouvelles perspectives pour les études futures sur le plan théorique (recherche), méthodologique ou pratique (application).

La première perspective scientifique concerne les métaheuristiques hybrides qui, s'efforçant de tirer partie des avantages spécifiques de métaheuristiques différentes en les combinant peuvent améliorer les performances de notre système.

- Hybrider l'algorithme génétique avec celui des essaims particulaires : Le croisement de type optimisation par essaims particulaires (OEP) est conçu suivant le principe de l'OEP qui combine les propriétés de trois particules afin d'en construire une nouvelle. Un vecteur de vitesse initialise suivant la méthode décrite pour l'algorithme OEP à chaque chromosome. La meilleure solution globale g de l'algorithme représente le troisième chromosome du croisement. Les deux autres chromosomes, nommés P_1 et P_2, sont sélectionnés comme dans l'algorithme AG (Le croisement à deux points).
- L'algorithme mémétique avec gestion population : Utiliser une méthode de recherche locale plus évoluée que la méthode de descente dans MA/PG, telle que la méthode multistart descent, le recuit simulé, ou la recherche tabou

La deuxième perspective ou plutôt le début d'un nouveau travail consiste à traiter les problèmes d'ordonnancement de système flexible de production par les réseaux de Hopfield (un type des réseaux de neurones) mais en utilisant une autre fonction objectif (fonction objectif de DMM modifiée **[Hassam, 2006]**). Les réseaux de Hopfield sont des réseaux récurrents et entièrement connectés. Dans ce type de réseau, chaque neurone est connecté à chaque autre neurone et il n'y a aucune différenciation entre les neurones d'entrée et de sortie. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. La fonction d'énergie est exprimée de la manière suivante :

$$E = -\frac{1}{2}\overrightarrow{V^t}.w.\vec{V} - \vec{I}.\vec{V} \qquad (1)$$

- Le réseau peut être caractérisé par une fonction d'énergie
- Une fonction objectif remplace la fonction d'énergie
- L'optimisation consiste à minimiser la fonction objectif

L'idée de Hopfield est que, puisqu'un réseau neuronal va chercher à minimiser la fonction d'énergie, il sera possible de construire un réseau pour la minimisation de cette fonction en associant les variables du problème d'optimisation aux variables de la fonction d'énergie.

En d'autres termes, cela revient à mettre en équation le problème d'optimisation sous une forme équivalente à la fonction d'énergie (1). Ce dernier évolue à partir d'un état initial (pris par

exemple aléatoirement) de façon à réduire son énergie ; l'évolution se fait ainsi jusqu'à ce que le réseau atteigne son équilibre. L'ensemble des états des neurones sur lesquels il s'est stabilise correspondra à une solution au problème.

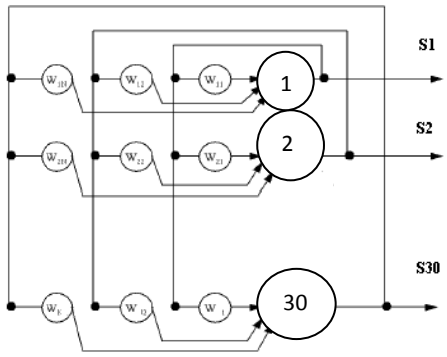
- **Construction du réseau :**

On considère un réseau de 30 routage (état des routages), un neurone n_i représente une possibilité ou bien l'état de chaque routage.

Maintenant on détermine la matrice des poids de connexion W pour cela, il suffit de mettre la fonction objective sous la forme de la fonction d'énergie du réseau (on na pas de contraintes donc on na pas d'entrées externes).

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}V_I V_j$$

En identifiant l'expression de cette fonction d'énergie celle de la fonction objective (fonction objectif de DMM modifiée) :



$$E = -\frac{1}{2}\sum_{i=1}^{30}\sum_{j=1}^{30} D_{ij}S_I S_j$$

X (i) représente le coefficient de disponibilité du routage *i*, il prend la valeur 0 ou 1

$$S(j) = \sum_{i=1}^{q} X(i) * D(i, j)$$

les coefficients de dissimilitude *Dij* comme suit :

$$D_{ij} = \frac{\text{nombre des machines comun entre les routages i et}}{\textit{nombretotal des machine entre les routage i et j}}$$

Hopfield a montré que l'énergie du réseau décroît à chaque changement d'état (l'état de S) donc converge vers un état stable qui est appelé attracteur dépendant de l'état initial. Cet état stable correspond à un minimum local de la fonction d'énergie.

Références bibliographiques

[Abada, 1997] Abada A, Contribution a la résolution des problèmes d'ordonnancement par réseaux de neurones. *Thèse de doctorat Laboratoire d'Automatique de Grenoble - L.A.G - I.N.P.G*, 1997.

[Adamou, 1997] Adamou, M., Contribution à la modélisation en vue de la conduite des systèmes flexibles d'assemblage à l'aide des réseaux de Petri orientés objet. *Thèse de doctorat, Université de Franche-Comté*, 1997

[Ammi, 2007] Ammi I., Résolution du problème de l'ordonnancement conjoint de la production et de la maintenance par colonies de fourmis. *Mémoire de fin d'études soutenue à Institut National de formation en Informatique I.N.I* ,2007.

[Bäck et al., 1997] Bäck T. and Eschelman L.J., Rudolph G., Porto V., Hansdbook Heuristic Method, chapter Rearch : Toward Self-tuning Heuristics. *Pages 61-83. John Wiley and Sons Ltd,* 1997.

[Benalia, 2005] Benalia M., Parallèlisation de la recherche dispersée Étude du modèle des îles : Application au problème MAX-W-SAT. *Mémoire de magister en informatique soutenue à l'institut national d'informatique-INI-,* 2005.

[Benatchba, 2005] Benatchba K., Modèle d'exécution pour l'aide à la solution problème MAX-SAT. *Thèse de doctorat soutenue à l'institut national d'informatique Alger*, 2005.

[Blum and Roli, 2003] Blum C. and Roli A., Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Computing Surveys*, 35(3) :268-308, 2003.

[Boukef, 2010] Boukef B.O., Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques Optimisation par algorithmes génétiques et essaims particulaires. *Thèse de doctorat soutenue à l'Université de Tunis El manar*, 2010.

[Bourdeaud'huy et Korbaa 2006] Bourdeaud'huy T., Korbaa O., Un modèle mathématique pour la résolution du problème d'ordonnancement cyclique avec minimisation de l'en-cours, *6ème Conférence de Modélisation et SIMulation, MOSIM'06, Rabat, Maroc,* 2006.

[Brauner et al., 2004] Brauner N.et Castagna P., Les Systèmes Flexibles de Production. *Journal Européen des Systèmes Automatisés*, 2004.

[Browne et al, 1984] Browne J., Dubois D., Rathmill K., Sethi S., Stecke E., Classification of Flexible manufacturing systems. *FMS Magazine*, vol. 2, p. 114 117, 1984.

[Carlier, 1988] Carlier J., et Chretienne, P., Problèmes d'ordonnancement - modélisation - complexité - algorithmes, *ERI Masson*, 1988.

[Colorni et al., 1991] Colorni, A., Dorigo, M., Maniezzo,V., et Trubian, M., Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science, 34, 39-53*, 1991.

[**Chaari, 2010]** Chaari T., Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride. *Thèse de doctorat soutenue à l'Université de Valenciennes et du Hainaut-Cambrésis,* 2010.

[Cavique, 2001] Cavique L., Rego C. and Themdo I., A Scatter Search Algorithm For the Maximum Clique Problem , In Ribeiro, C.C, Hansen, P. (Eds.), Essays and Surveys in Metaheuristics,. *Kluwer-Academic Publishers*, 2001.

[Cho et Wysk, 1995] Cho, H., & Wysk, R.A., Intelligent workstation controller for computer-integrated manufacturing: problems and models. *Journal of Manufacturing Systems,* 14(4), pp. 252-263, 1995.

[Cung et al, 2001] Cung V. –D., Martins S.L., Ribeiro C.C., Roucairol C., Strategies for the parallel implementation of metaheuristics, in Essays and Surveys in Metaheuristics (C. C. Robeiro et P. Hansen, editors), 263-308, Kluwer, 2001.

[Darwin, 1959] Darwin, C., On The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life. *Murray, London,*1859.

[Das et Nagendra, 1997] Das, S.K., & Nagendra, P., Selection of routes in a flexible manufacturing facility. *International Journal of Production Economics,* 48, pp. 237-247, 1997.

[Dorigo M. et al. 1999] Dorigo M. et Di Caro G., The Ant Colony Optimization Meta-Heuristic. In Corne D., Dorigo M. and Glover F., New Ideas in Optimization, McGraw-Hill, 1953, 1999.

[Dupas ,2004] Dupas R., Amélioration de performance des systèmes de production : apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles. *Habilitation à Diriger des Recherches présentée à L'Université d'Artois*, 2004.

[Esquirol, 1999] Esquirol P. et Lopez P., L'ordonnancement. *Economica,* 1999.

[Fabre, 2009] Farbe F., Conduite orientée ordonnancement d'un simulateur dynamique hybride : application aux procédés discontinus. *Thèse de doctorat soutenue à l'Institut National Polytechnique de Toulouse*, 2009.

[Fogel, 1993] Fogel D.B., On the philosophical differences between evolutionary algorithms and genetic algorithms. In D.B. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on EvolutionaryPr ogramming*, pages 23–29. Evolutionary Programming Society, La Jolla, 1993.

[Giard, 1988] Giard V., Gestion de production, *Economica,* 1988.

[Glover et al., 2003] Golver F. and Kochenberger G.A., Handbook of Metaheuristics, chapter Sctter search and path relinking : advances and applications, *pages 1-36. Kluwer Academic*, 2003.

[Glover, 1998] Glover F., Atemplate for scatter search and path relinking , J.K. Hao, E.Lutton, E. Ronald, M. Schoenauner, D. Snyers (Eds.), Artificial Evolution*, Lecture Notes in Computer Sciences 1363 :3-51,Spinger-Verlag*, 1998.

[Glover, 1995] Glover F., Scatter search and star-paths: beyond the genetic metaphor. *OR Spektrum*, 17:125–137, 1995.

[Goldberg, 1989] Goldberg D.E., Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley, Reading*, 1989.

[Ha, Gainier *et al.* 1999] Galinier et Habib M., Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes*. Revue d'Intelligence Artificielle*, 13(2): pp. 283-324,1999.

[Hassam, 2006] Hassam A., Manipulation des routages alternatifs en temps réel dans les systèmes flexibles de production. *Mémoire de magister, université Abou Bekr Belkaid, Tlemcen*, 2006*.*

[**Hernandez, 2008]** Hernandez J., Algorithmes Métaheuristiques Hybrides Pour La Sélection De Génes Et La Classification De Données De Biopuces. *Laboratoire d'Étude et de Recherche en Informatique d'Angers*, 2008.

[Hoffmeister and T. Back, 1991] Hoffmeister F.and Back T.. Genetic algorithms and evolution strategies: Similarities and differences. In H.-P. Schwefel and R. M¨anner, editors, *Parallel Problem Solving from Nature – PPSN I*, Lecture Notes in Computer Science 496, pages 455–469. Springer, Berlin, 1991.

[Holland, 1975] Holland J.H., Adaptation in natural and artificial systems . *PhD, Michigan Press Univ., Ann Arbor, MI*, 1975.

[Hoos et Stützle, 2004] Hoos H.H. and Stützle T., Stochastic local search: Foundations and applications. Morgan Kaufmann, 2004.

[Jambu, 1999] Jambu M., Introduction au data-mining. *Eyrolles*, 1999.

[Kaufmann, 1992] Kaufmann A, Introduction à la logique floue. *Techniques de l'Ingénieur, A 120, Mathématiques pour l'ingénieur*, 1992.

[Kazerooni et *al.*, 1997] Kazerooni, A., Chan F.T., & Abhary, K., A fuzzy integrated decision-making support system for scheduling of FMS using simulation, Computer Integrated Manufacturing Systems, 10(1), pp27-34, 1997.

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

[Krasnogor and Smith, 2005] Krasnogor N. and Smith J., A tutorial for competent memetic algorithms : Model, taxonomy, and design issues. *IEEE Transaction on Evolutionry computation,* 9(5) : 474-488, 2005.

[Kusiak, 1985] Kusiak A., Flexible manufacturing systems : a structural approach. *Int. J. of Pro. Res*, vol. 23, p. 1057-1073, 1985.

[**Lawler *et al.*, 1989]** Lawler E.L., Lenstra J.L., Rinnooy Kan, A.H.G., Schmoys, D.B.. Sequencing and scheduling: algorithms and complexity. Rapport BS-R8909, *Centre for Mathematics and Computer Science, Amsterdam*, 1989.

[Learman, 2000] Learman et Ngouenet, Algorithmes génétiques séquentiels et parallèles pour une représentation affine des raxilitée. *rapport de recherche N°2570*, 2000.

[Lee In, 2000] Lee In, Shaw M.J., A neural-net approach to real time flow-shop sequencing. *Computers and Industrial Engineering*, 2000.

[Lemamou, 2009] Lemamou, E.A., Ordonnancement de projets sous contraintes de ressources à l'aide d'un algorithme génétique à croisement hybride de type OEP. *Mémoire présentée à l'université du Québec à Chicoutimi comme exigence partielle de la maitrise en informatique, Université du Québec*, 2009

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

[Lopez et al., 2001] Roubellat F., Ordonnancement de la production. *Hermès Sciences publications*, 2001.

.

[MacCarthy et Liu, 1993] MacCarthy B.L. et Liu J, Anew Classification Scheme For Flexible Manufacturing Systems. *International Journal of Production Research*, 31(2), 299-309, 1993.

[Maccarthy, 1993] Maccarthy B., Liu J., A New Classi_cation Scheme For Flexible Manufacturing Systems. *International Journal of Production Research*, vol. 31, n° 2, p. 299-309, 1993.

[Mebarki, 1995] Mebarki N., Une approche d'ordonnancement temps réel basée sur les règles de priorité des files d'attente. *Thèse de doctorat soutenue à l'Université de Claude Bernard Lyon1,* 1995.

[Mehenni, 2006] Mehenni T., Utilisation des Métaheuristiques Pour Résoudre Un Problème D'ordonnancement sur Machine a Contrainte de Ressource Non Renouvelable. *Mémoire de magister soutenue à l'Université Mohamed Boudiaf de M'Sila*, 2006.

[Meignan, 2004] Meignan D., une approche organisationnelle et multi- agent pour la modélisation et l'implantation métaheuristiques : Application aux problèmes d'optimisation de réseaux de transports. *Thèse de doctorat soutenue à l'Université de Franche-Comté*, 2004.

[Merhoum et al., 2007] Merhoum K. et Djeghaba M., Algorithme génétique pour le problème d'ordonnancement de type job-shop. *Université Badji Mokhtar, BP12, Annaba,* 2007

[Metropolis N. et al., 1953] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E., Equation of state calculations by fast computing machines. *Journal f Chemical Physics, 21,*1953.

[Mirdamadi, 2009] Mirdamadi S., Modélisation du processus de pilotage d'un atelier en temps réel à l'aide de la simulation en ligne couplée à l'exécution. *Thèse de doctorat soutenue à l'Institut National Polytechnique de Toulouse*, 2009.

[Moscato et Cotta, 2003] Moscato P. and Cotta C., The k-feature set problem is w[2]-complete. *Journal of computer ans system sciences*, 67:686–690, 2003.

[Moscato, 1999] Moscato P., New Ideas in Optimization, chapter Memetic Algorithms: A Short Introduction. *McGraw-Hill*, 1999.

[Moscato, 1993] Moscato P., An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Annals of Operations Research*, 41:85–121, 1993.

[Moscato, 1989] Moscato P., On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. *Technical report, Caltech Concurrent Compulation Program*, 1989.

[Paris, 2004] Paris J.L, Apport des algorithmes évolutionnistes et de l'apprentissage pour l'analyse et l'optimisation via simulation des systèmes de production. *Mémoire d'habilitation à diriger des recherches à Université Blaise Pascal*, 2004.

[Peng et Chen, 1998] Peng C., and Chen F.F., Real-time control and scheduling of flexible manufacturing systems: a simulation based ordinal optimization approach. *International Journal of Advanced Manufacturing Technology,* 14(10), pp. 775-786,1998.

[**Prodhon, 2008]** Prodhon C.**,** Le Problème De Localisation-Routage. *Thèse de doctorat soutenue à l'Université de Technologie de Troyes*, 2008.

[Prodhon, 2006] Prodhon C., et Wol-er-Calvo R., A memetic algorithm with population management (*MA/PM*) for the capacitated location-routing problem. In J.Gottlieb et G. R. Raidl (Eds.), *Lecture Notes in Computer Science*, 2006.

[Rachamadugu et Stecke, 1994] Rachamadugu R., Stecke K.E., Classification and review of FMS scheduling procedures. Production Planning and control, 5, pp. 2-20, 1994.

[Rebreynd, 1999] Rebreynd P., Algorithmes Génétiques Hybrides en Optimisation Combinatoire, *Thèse Ph.D, Ecole Normale Supérieure de Lyon*, 1999.

[Remboldu, 1993] Rembold U., Nnajy B., Storr A., Computer Integrated Manufacturing and Engineering. *Addison-Wesley*, 1993.

[Saygin et Kilic, 1995] Saygin C., and Kilic, S.E., Dissimilarity Maximization Method for Real-time Routing of Parts in Random FlexiblenManufacturing Systems. *The International Journal of Flexible Manufacturing Systems,* 16, pp. 169-182, 2004.

[Sabuncuoglu, 1998] Sabuncuoglu I., Scheduling with neural networks: a review of literature and new research directions, *Production Planning and Control,* vol.9, n°1, pp. 2-12, 1998.

[Saygin, 2001] Saygin C., Chen F.F., Singh J., Real time manipulation of alternative routings in flexible manufacturing systems: A simulation study. *International journal of advanced manufacturing technology*, p. 755-763, 2001.

[Schwefel and T. Bäck, 1998] Schwefel H.-P and Bäck T., Artificial evolution: How and why? In D. Quagliarella, J.P´eriaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategyin Engineering and Computer Science: Recent Advances and Industrial Applications*, pages 1–19. Wiley, Chichester, 1998.

[Sebastien, 2006] Sebastien N, Métaheuristiques Hybridespour la Résolution du Problème D'ordonnancement de Voitures dans une Chaine D'assemblage Autonobile. *Thèse de doctorat soutenue à l'Université de Québec à Montréal*, 2006.

[Senties, 2007] Senties B., Methodologie D'aide a La Decision Multicritère Pour L'ordonnancement D'ateliers Discontinus. *Thèse de doctorat soutenue à L'institut National Polytechnique De Toulouse*, 2007.

[Sevaux, 2004] Sevaux M., 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*, 2004.

[Sörensen et Sevaux, 2004] Sörensena[a] K., Sevauxb[b] M., MA|PM: memetic algorithms with population management, [a]*University of Antwerp, Faculty of Applied Economics, Prinsstraat 13, B-2000 Antwerp, Belgium* [b]*University of Valenciennes,CNRS, UMR 8530, LAMIH-SP, Le Mont Houy-Bat Jonas 2, F-59313 Valenciennescedex 9, France* , 2004.

[Souier, 2009] Souier, M., Métaheuristiques pour la manipulation de routages alternatifs en temps réel dans un Job Shop, *Mémoire de Magister,Université Abou Bakr Belkaid, Tlemcen, 2009.*

[Smith, 1992] Smith, S. F., Knowledge-Based production management: approaches, results and prospects, *Production Planning and Control*, vol.3, n°4, 350-380, 1992.

[Sik, 2003] Sik H., Yih Y., Development of a real-time multi-objective scheduler for a semiconductor fabrication system, *Taylor & Francois group*, 2003.

[Talbi, 2004] Talbi E., Sélection et Réglage de Paramètres pour l'Optimisation de Logiciels d'Ordonnancement Industriel. *Thèse de doctorat, Institut National Polytechnique de Toulouse*, 2004.

[Talbi, 1995] Talbi E.G., Algorithmes génétique parallèles : Techniques et Applications. *Editions Hermés*, 1998.

[Tamani, 2008] Tamani K., Développement d'une méthodologie de pilotage intelligent par régulation de flux adaptée aux s.ystèmes de production. *Thèse de doctorat soutenue à l'université de savoie*, 2008.

[Widmer, 2001] Widmer M., Hertz A., and Costa D., Les métaheuristiques, *chapter 3, pages 55-93. hermes edition*, 2001.

Résumé

Les problèmes d'ordonnancement étant *NP-difficiles.* Les métaheuristiques sont de plus en plus utilisées pour résoudre de tels problèmes. C'est pour cela que l'utilisation d'heuristiques et de métaheuristiques est amplement justifiée.

Dans ce travail, nous avons adaptés des métaheuristiques pour la sélection de routages alternatifs en temps réel. La première métaheuristique est l'algorithme mémétique avec gestion de population, il s'agit d'un algorithme génétique hybride avec une technique de recherche locale et de mesure de distance permettant de contrôler la diversification des solutions dans la population, et la deuxième est l'algorithme de recherche dispersée basé sur l'extraction d'un ensemble de référence *R* contenant les meilleurs solutions de la population initiale. Les résultats obtenus sont comparés avec ceux de d'autres métaheuristiques à base de population.

Les résultats ont montré que les deux métaheuristiques adaptées ont amélioré le taux de production, le aux d'utilisation des différentes machines et le taux d'utilisation du système de transport, pour un système de production saturé et même en présence de pannes.

Mots clés

Ordonnancement, métaheuristiques, intelligence artificielle, algorithme mémétique avec gestion de population, la recherche dispersée, simulation.

Abstract

Scheduling problems are generally NP-hard. Metaheuristics approaches are being increasingly used to solve such problems.

In this work, we adapted metaheuristics to solve the real time alternative routings selection problem. The first metaheuristic is the memetic algorithm with population management, which is an hybridization of a genetic algorithm and a local search, and works by measuring and controlling the diversity of a small population of high-quality solutions, and the second is the scatter search method based on the extraction of a reference set *R* which contains best solutions of the initial population. The obtained results are compared to those of other metaheuristics based population.

The obtained results showed that both algorithms have improved the production rate, the utilization rate of different machines and the transport system, even in the presence of breakdowns.

Key words

Scheduling, métaheuristics, artificial intelligence, mémétic algorithm with population management, scatter search, simulation.

ملخص

مشاكل الجدولة في الغالب من الإشكاليات كثيرات الحدود الصعبة. فوقيات الاستدلال من الطرق التي تعرف استعمالا متزايدا لحل هذا النوع من الإشكاليات.

في هذا البحث، عملنا على تكييف فوقيات الاستدلال لحل إشكالية اختيار المسارات التناوبية في الوقت الحقيقي. فوقية الاستدلال الأولى هو خوارزمية المثلية مع إدارة الجيل والمتمثلة الخوارزمية الجينية الهجينة مع تقنيات البحث المحلية، وقياس المسافة في السيطرة على تنويع الحلول في الجيل ، والثانية هي خوارزمية البحث التبعثري التي تعتمد على استخراج مجموعة مرجعية R التي تحتوي على أفضل الحلول من الجيل الأولي. قورنت النتائج المتحصل عليها مع تلك المترتبة عن فوقيات استدلال أخرى مرتكزة على عدد الجيل.

أظهرت النتائج التي تم التوصل إليها أن كلا من الخوارزميات حسنت نسبة الإنتاج، ومعدل استخدام الآلات، ونسبة استخدام نظام النقل، حتى في حالة وجود الأعطال.

الكلمات المفتاحية

الجدولة، فوقيات الاستدلال، الذكاء الصناعي، خوارزمية المثلية مع التحكم في الجيل ، البحث التبعثري، محاكاة.