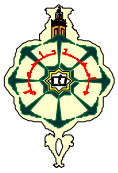
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.



Université Abou Bakr Belkaid –Tlemcen –

Faculté de Technologie.

Département de Génie Electrique et Electronique.

Mémoire de Magister en Automatique

Option : Modélisation et commande des systèmes

Thème

Modélisation et Ordonnancement temps réel d'un Job shop à l'aide des métaheuristiques

Présenté par :

Melle. HOUBAD Yamina

Soutenu en décembre 2011 devant le jury composé de :

Président :		
Mr. CHIKH Mohamed Amine	Maître de conférences	UABB, Tlemcen
Examinateurs :		
Mme. NOUREDDINE Myriam	Maître de conférences	USTO, Oran
Mr. BELKADI Khaled	Maître de conférences	USTO, Oran
Mlle GHOMRI Latéfa	Maître assistante A	UABB, Tlemcen
Encadreur :		
Mr. SARI Zaki	Professeur	UABB, Tlemcen
Co-encadreurs :		
Mr. HASSAM Ahmed	Maître assistant A	UABB, Tlemcen
Mr. SOUIER Mehdi	Maître assistant B	EP-ECG, Tlemcen

Remerciements

La rédaction de ce manuscrit est l'aboutissement d'une année de travail. Ce travail lui-même n'aurait pu être mené sans l'aide de plusieurs personnes auxquelles je souhaite exprimer ma gratitude.

Ce n'est pas par tradition que cette page figure au préambule de ce mémoire, mais c'est plutôt un devoir moral et une reconnaissance sincère qui me pousse à la faire.

Je tiens tout d'abord à remercier mon encadreur monsieur SARI Zaki, professeur à l'Université de Tlemcen, et le directeur de Laboratoire de productique de Tlemcen, pour m'avoir accueilli au sein de son Laboratoire MELT, encadré, soutenu et prodigué de nombreux conseils tout au long de mon année de mémoire. Je lui suis particulièrement reconnaissante de m'avoir laissé une grande liberté scientifique. Je voudrai lui remercier l'encadrement qu'il m'a offert et sa disponibilité appréciable tout au long de mon travail dans ce mémoire. En plus de ses qualités scientifiques et pédagogiques, j'ai pu apprécier sa grande sympathie qui rend le travail à ses cotés très agréable.

J'aimerai particulièrement adresser mes remerciements les plus vifs et ma reconnaissance à mes Co-encadreurs:

- Monsieur HASSAM Ahmed Maître assistant à l'Université de Tlemcen, pour sa disponibilité, conseils, et ses orientations judicieuses.

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

Ils ont toute ma gratitude pour leurs encouragements et leur soutien moral, et leurs précieuses orientations.

Toute ma gratitude va ensuite aux membres du Jury, qui ont accepté de consacrer une partie de leur temps à l'évaluation de mon travail :

Je tiens à remercier Monsieur CHIKH Mohamed Amine, Maître de conférences à l'Université de Tlemcen, pour m'avoir fait l'honneur de présider mon jury.

De même, j'adresse des remerciements chaleureux à madame NOUREDDINE Myriam, Maître de conférences à l'Université d'Oran, monsieur BELKADI Khaled, Maître de conférences à l'Université d'Oran, et mademoiselle GHOMRI Latéfa, Maître assistante à l'Université de Tlemcen, qui ont accepté d'être les examinateurs de ce travail.

Je tiens aussi à remercier tout les membres de Laboratoire de Productique (MELT), pour leurs encouragements. Ce fut un plaisir de vous côtoyer quotidiennement, j'ai particulièrement apprécié l'ambiance chaleureuse qui règne au laboratoire. Je vous souhaite les meilleures des chances dans vos futurs projets. Que chacun trouve ici l'expression de mon amitié.

Mes remerciements les plus vifs s'adressent à ma spéciale Amel qui m'a soutenu tout au long de mes études universitaires, pour son amitié, et sa présence dans des moments difficiles.

Je profite de l'occasion pour remercier mes plus chères amies, Nassima, Souhila, Fatima, Nadia, Ghania, et Habiba pour leurs encouragements.

Je dédie ce travail à ma promotion de Magister d'Automatique, avec tous mes souhaits de réussite.

Mes derniers mots seront à ma famille. Il m'est difficile d'exprimer en quelques mots tous mes remerciements, mon amour, et ma gratitude à deux personnes qui sans eux je n'aurai jamais pu atteindre la réussite, mes parents, qui m'ont toujours fait confiance, soutenu, et conseillé, à mon père qui m'a toujours aidé, et soutenu, et ma très chère mère qui a su faire preuve de beaucoup de patience tout au long de mon travail de mémoire.

Mes remerciements s'adressent aussi à mon frère Mekki pour son aide, ses conseils, et ses encouragements qui étaient indispensables durant toutes mes études.

A tous, je dédie ce mémoire qui n'aurait jamais vu le jour sans vous.

Table des matières

Table des matières Chapitre 3

Liste des algorithmes

Liste des tableaux

Liste des figures

INTRODUCTION GENERALE

La concurrence internationale et la globalisation des marchés sont des facteurs contraignants pour les entreprises de nos jours. Afin de demeurer compétitives dans le contexte économique actuel, elles doivent adopter des méthodes de gestion plus efficaces.

L'évolution croissante des besoins d'une entreprise fait que la conception du système de production est de plus en plus orientée vers des familles de produits et non vers un seul type de produit. Les systèmes correspondants à une telle exigence doivent se révéler flexibles.

Un système flexible de production représente plusieurs cellules reliées entre elles par des véhicules filoguidés composant les diverses zones de production. Le système flexible est un système capable de s'adapter à toute nouvelle contrainte imposée par l'environnement dont lequel la production est gérée par un système informatique (changement de produit, modification de flux de production) **[Lemlouma, 2001]**.

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 de travail et tient compte des diverses contraintes de la production.

La théorie de l'ordonnancement est une branche de la recherche opérationnelle. Elle occupe un rôle important dans de nombreux secteurs de l'économie, notamment en gestion de production des entreprises de biens ou de services.

La fonction ordonnancement vise à définir les dates d'exécution d'un ensemble de travaux en tenant compte de la disponibilité des ressources, de manière à optimiser un ou plusieurs critère(s) donné(s), tout en respectant les contraintes de fabrication et d'organisation.

Les problèmes d'ordonnancement sont présents dans tous les secteurs d'activités de l'économie depuis l'informatique jusqu'à l'industrie manufacturière. C'est pour cette raison qu'ils ont fait et continuent de faire l'objet de nombreux travaux de recherche.

Résoudre un problème d'ordonnancement consiste à ordonnancer, c'est-à-dire, programmer ou planifier, dans le temps l'exécution de tâches en leur attribuant les ressources nécessaires matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation **[Dupas, 2004]**.

Les méthodes de résolution des problèmes d'ordonnancement puisent dans toutes les techniques de l'optimisation combinatoire (programmation mathématique, programmation dynamique, théorie des graphes…).

Ces méthodes garantissent en général l'optimalité de la solution fournie, mais certains problèmes nécessitent la construction de méthodes de résolution approchées, qu'on appelle méthodes heuristiques, efficaces pour les problèmes de type NP-difficile. L'appartenance à la classe des problèmes de type NP-Difficiles signifie qu'on ne pourra probablement jamais résoudre ce problème d'une manière optimale.

Le développement de la théorie de complexité des algorithmes a permis de clarifier la difficulté du problème. La plupart des problèmes d'ordonnancement sont NP-difficiles **[Talbi, 2004]**. De tels problèmes nécessitent de construire des méthodes de résolution approchées, dites heuristiques ou métaheuristiques. Ces méthodes ne fournissent aucune garantie sur la qualité des résultats, mais ont permis de résoudre avec succès plusieurs problèmes d'optimisation et en particulier une multitude de problèmes d'ordonnancement.

L'intérêt croissant apporté par les métaheuristiques est tout à fait justifié par le développement des machines avec des capacités calculatoires énormes, ce qui a permis de concevoir des métaheuristiques de plus en plus complexes qui ont fait preuve d'une certaine efficacité lors de la résolution de plusieurs problèmes à caractère NP-difficile.

L'objet de cette thèse est de proposer et de tester 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 mimétique qui combine un algorithme génétique avec une méthode de recherche locale (la méthode de descente). La deuxième est l'algorithme une métaheuristique basée sur le comportement de recherche de nourriture d'une espèce de fourmis primitives, dites les Pachycondyla apicalis, l'algorithme inspiré du comportement de ces fourmis est nommé l'algorithme API (comme APIcalis). 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.

Le manuscrit est organisé de la manière suivante :

Dans le premier chapitre, nous présentons des 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.

Dans le deuxième chapitre nous nous intéressons de manière approfondie aux métaheuristiques tout en décrivant certaines méthodes à solution unique et à base de population. Nous présentons d'une manière plus détaillée les six algorithmes qui ont déjà été utilisés pour résoudre le problème d'ordonnancement du même système que nous étudions **[Souier 2009], [Souier et *al.*, 2010]**. Ensuite nous présentons les deux métaheuristiques que nous avons adapté pour la résolution de notre problème d'ordonnancement ; l'algorithme mémétique et l'algorithme API basé sur le comportement de l'espèce de fourmis Pachycondyla apicalis. Pour chaque métaheuristique présentée, nous donnons son origine et son algorithme de base.

Le troisième chapitre est consacré à notre contribution proprement dite, nous commençons par présenter le modèle FMS étudié pour tester nos algorithmes et l'adaptation de ces techniques pour résoudre notre problème. Puis nous présentons les résultats obtenus et leurs interprétations après plusieurs simulations et la comparaison de ces techniques avec deux métaheuristiques (les algorithmes génétiques, et les colonies de fourmis) utilisées pour résoudre le même problème **[Souier 2009], [Souier et *al.*, 2010].**

Enfin, nous terminons par une conclusion générale qui fera la synthèse de nos différentes contributions, et donne quelques perspectives de recherche.

Chapitre 1

Ordonnancement des systèmes flexibles de production

Chapitre 1

Ordonnancement des systèmes flexibles de production

Nous proposons dans ce chapitre des notions de base utiles pour la compréhension de ce mémoire. La première partie du chapitre présente les notions fondamentales concernant les systèmes de production, et particulièrement les systèmes flexibles de production. La seconde partie est consacrée aux notions liées à l'ordonnancement, à savoir la définition d'un problème d'ordonnancement, les objectifs de l'ordonnancement, et les méthodes de résolution.

1.1 Introduction

Le système de production rassemble tous les moyens permettant de donner une valeur ajoutée aux produits ou aux services. Cette transformation est commandée par un système de gestion qui doit respecter un ensemble de contraintes en vue d'atteindre des objectifs définis **[Akrout et Masmoodi, 2009]**.

Un système de production est dit flexible s'il peut assurer la production simultanée de plusieurs types de pièces avec des quantités variables et s'il est capable de s'adapter à la production de nouveaux produits pour lesquels le système n'a pas été étudié **[Bourdeaud'huy et Korbaa, 2006], [Esquirol et Lopez, 1999].**

La flexibilité de routages est l'un des types de flexibilité étudiés par la communauté scientifique, cette flexibilité offre au système les moyens d'un aiguillage plus souple, de façon à servir les différents segments de procédés libres ou sous engagés.

Les problèmes liées à la technologie des systèmes flexibles de production (FMS) sont relativement complexes comparés aux systèmes de production traditionnels. C'est la raison pour laquelle les problèmes d'ordonnancement dans ces systèmes sont généralement NP-difficiles. Il n'existe donc pas d'algorithme capable de résoudre ces problèmes de manière exacte **[Souier et *al*., 2010]**.

Depuis la fin des années 80, les méthodes approchées, et plus particulièrement les métaheuristiques, suscitent un intérêt croissant pour leur capacité à fournir des solutions de bonne qualité pour une consommation en temps de calcul réduit.

Les heuristiques sont beaucoup plus adaptées au contexte industriel, où il ne s'agit pas tant de satisfaire complètement tel ou tel objectif que de fournir une solution d'ordonnancement réaliste, qui procure une satisfaction jugée acceptable d'un ensemble d'objectifs. Pour cette raison, les logiciels d'ordonnancement disponibles sur le marché font tous appel à des méthodes heuristiques **[Talbi, 2004]**.

Les besoins industriels ont entraîné l'apparition et le développement de nouvelles technologies pour mieux gérer ces systèmes productifs : d'où l'importance du rôle de la gestion de production. Cette dernière a pour objet *« la recherche d'une organisation efficace dans l'espace et dans le temps de toutes les activités relatives à la production afin d'atteindre les objectifs de l'entreprise »* **[Giard, 1988], [Khalouli, 2010]**. La fonction ordonnancement fait partie des fonctions de la gestion de production, dans ce chapitre nous nous intéressons à la fonction ordonnancement, et plus particulièrement aux problèmes d'ordonnancement dans les systèmes flexibles de production, et leurs méthodes de résolutution.

La deuxième section de ce chapitre donne quelques définitions et notions liés aux systèmes flexibles de production. La troisième section est consacrée aux problèmes d'ordonnancement, dans cette section nous présentons l'ordonnancement des systèmes de production, ses contraintes, ses objectifs, et les problèmes d'ordonnancement liés aux ateliers multi machines.

La dernière section présente les méthodes de résolution des problèmes d'ordonnancement.

1.2 Les systèmes de production

« *Produire c'est transformer* » **[Sassine, 1998]**, donc le système de production représente le lieu et les moyens de transformation des matières premières en un produit sous une autre forme. Les systèmes de production diffèrent par les objectifs tracés par les décideurs du système, ces objectifs sont dynamiques et ils évoluent suivant l'évolution du marché et des technologies sur lesquelles s'appuie le système.

Les systèmes de production ont été classés dans **[Sassine, 1998], [Amodeo, 1999]** en trois grandes catégories :

-**Les processus continus :** tels que la production électrique.

-**Les processus discrets :** tels que l'usinage et toutes les activités d'assemblage.

-**Les processus discontinus :** qui se situe à mi-chemin entre les processus continus et les processus discrets, les deux types de processus sont couplés : la production et continue mais il y a un conditionnement discret des produits.

Les systèmes de production regroupent l'ensemble des moyens qui permettent de transformer les ressources en produits ou en services.

Un système de production est soumis à une charge qui définit l'ensemble des biens ou des services que le système doit réaliser. Pour écouler cette charge, le système utilise une ou plusieurs ressources **[Caumond et *al.*, 2006a].** Plus précisément, la charge d'un système de production est constituée de jobs. Un job suit la réalisation d'un produit dans toutes les étapes de production (de la matière première jusqu'au produit fini). Chaque job a une gamme qui décrit la suite des opérations qu'il doit réaliser. La gamme décrit les opérations à réaliser en donnant :

- Leur durée,
- La ou les machines qui doivent réaliser cette opération,
- Les ressources consommées (type de ressource et quantité utilisée),
- L'ordre entre les opérations à réaliser (facultatif) **[Caumond et *al.*, 2006].**

1.2.1 Systèmes de production flexibles

Caumond et al., (2006) définissent un système flexible de production comme étant : « U*n 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* ».

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.

Un système flexible de production (Flexible Manufacturing System : FMS) représente plusieurs cellules flexibles reliées entre elles par des véhicules filoguidés composant les diverses zones de production. Le système flexible est un système capable de s'adapter à toute nouvelle contrainte imposée par l'environnement dont lequel la production est gérée par un système informatique (changement de produit, modification de flux de production) **[Lemlouma, 2001]**.

Les systèmes flexibles de production sont constitués d'un certain nombre d'équipements qui peuvent être divisés en quatre grandes familles :

- Les stations de travail : chaque station de travail est composée d'une machine à commande numérique, d'un magasin d'outils avec un système automatique de changement d'outils, d'un magasin à palettes et supports, d'un système automatique de chargement / déchargement des palettes dans la machine, d'un système automatique d'inspection et d'un système de contrôle.

- Les systèmes de manutention et de transport, comprenant les systèmes de transport, les robots manipulateurs…, servent à déplacer les produits et éventuellement les outils entre les machines suivant les chemins planifiés pendant la conception du FMS. Il existe différents moyens de transport et différentes stratégies. On peut par exemple transporter les produits d'une aire de stockage vers une station de travail, puis vers une aire de stockage, ou par contre faire les déplacements entre les stations de travail, ce qui diminue le nombre de déplacements. On peut aussi déplacer les produits par petits lots au lieu que ce soit par unité.

Les systèmes flexibles de production sont constitués d'un ensemble de machines à commande numérique, de stations de travail connectées par un système de transport automatisé; le tout commandé par ordinateur.

Le choix de l'une de ces méthodes dépend du système global. On favoriserait le déplacement à l'unité lorsque les distances entre les stations sont petites. Par contre, pour les grandes distances, le déplacement par lots sera plus intéressant. De plus, la taille et le poids des produits comparés au système de transport peuvent limiter le choix d'une méthode ou d'une autre.

- Les systèmes de stockages qui renferment les aires de stockages, les systèmes de chargement déchargement…, peuvent en plus du stockage, servir à d'autres activités, comme l'emballage, le contrôle de qualité...

- Les systèmes de contrôle et de communication : chaque système de contrôle peut être composé d'un ordinateur central de commande, il donne des instructions et reçoit des états des situations de chaque équipement du FMS. Il peut garder en mémoire les gammes d'usinage des produits si les ordinateurs subalternes sont saturés. C'est le système de contrôle qui décide quand et/ ou comment les produits doivent se déplacer entre les différentes machines et être chargés ou déchargés **[Sari, 2003]**.

1.2.2 Notion de flexibilité

Le mot flexibilité est souvent utilisé dans des contextes différents. On parle de la flexibilité d'un équipement pour souligner ses capacités multifonctions. Un atelier flexible désigne un atelier capable de fabriquer différents types de produits.

La notion de flexibilité est l'aptitude d'un système à répondre aux modifications de son environnement afin d'assurer le respect de ses objectifs. Ces modifications peuvent être internes telles que les pannes des équipements, les pannes des systèmes informatiques (logiciels de gestion), variation des temps de fabrication, absentéisme des travailleurs… ou externes, telles que 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].**

Plusieurs types de flexibilités **[Adamou, 1997]** ont été mis en évidence suivant leurs incidences sur l'objectif qui est le produit fini et sur les moyens de production permettant la résolution de ce produit :

-*flexibilité de produits :* offre la possibilité d'une reconfiguration du système pour la prise en compte d'un nouveau produit ou famille de produits permettant ainsi un gain de productivité ;

La flexibilité d'un système de production se caractérise par sa capacité d'adaptation à la production des nouveaux produits pour lesquels le système n'a pas été étudié.

-*flexibilité de mélange :* c'est la possibilité de produire simultanément un ensemble de produits ayant des caractéristiques de base communes ; cette flexibilité peut être mesurée par le nombre de produits différents qui peut être fabriqués simultanément.

-*flexibilité de quantité :* il s'git de la capacité du système à faire face aux fluctuations de la quantité des produits à fabriquer en modifiant les rythmes, ainsi que les temps de passage et d'engagement des outils ;

-*flexibilité de routage :* offre au système les moyens d'un aiguillage plus souple, de façon à servir les différents segments de procédés libres ou sous-engagés ;

-*flexibilité d'ordre des opérations :* permet de changer l'ordre des opérations en cours de production (ce qui suppose l'existence d'une gamme principale et des gammes secondaires) ou de choisir la destination suivante après chaque opération ;

-*flexibilité d'expansion :* autorise une extension et une modification de l'architecture du système et elle exige une modélisation ;

-*flexibilité des ressources :* c'est la capacité des ressources à effectuer plusieurs tâches élémentaires et de permettre la programmation.

1.3 L'ordonnancement

Ordonnancer la fabrication d'un ensemble de produits, c'est en répartir la réalisation dans le temps et l'espace.

L'ordonnancement de la production a fait ces dernières années l'objet de recherches très approfondies, pour de nombreux problèmes identifiés dont la résolution repose sur une politique d'ordonnancement qui soit la plus optimale possible, chose qui a permis de mettre au point de nombreuses méthodes de résolution.

Les problèmes d'ordonnancement se rencontrent très souvent dans le milieu industriel, notamment dans l'optimisation de la gestion des systèmes de

L'ordonnancement trouve son application dans différents domaines allant du plus simple au complexe, dont on peut citer :

- Emplois du temps,
- Les problèmes d'affectation, de transport,…
- Organisation des prises de vue d'un satellite,
- Les systèmes distribués et les systèmes embarqués,
- Dans les ateliers de production,

Etablir un ordonnancement revient donc à coordonner l'exécution de toutes les tâches, en utilisant au mieux les ressources disponibles **[Esquirol et Lopez, 1999]**. En d'autres termes, il s'agit de :

« Déterminer ce qui va être fait, quand, où et avec quels moyens, étant donné un ensemble de taches à accomplir, le problème d'ordonnancement consiste à déterminer quelles tâches doivent être exécutées et à assigner des dates et des ressources à ces tâches de façon à ce que les tâches soient, dans la mesure du possible, accomplies en temps utile, au moindre coût et dans les meilleurs conditions » **[Parunak, 1985].**

Les tâches, les ressources, et les contraintes sont les notions fondamentales en ordonnancement.

Donc L'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles et de contraintes liées à la disponibilité des ressources, afin de satisfaire un ou plusieurs critères d'optimisation.

D'après cette définition, on remarque que dans un problème d'ordonnancement interviennent trois notions fondamentales : les tâches, les ressources, et les contraintes.

- Les tâches

« Une tâche est le processus le plus élémentaire. Elle est constituée d'un ensemble d'actions à accomplir, dans des conditions fixées, pour obtenir un résultat attendu et identifié, en termes de performances, de coûts et de délais».

Une tâche est un travail élémentaire localisé dans le temps, et dont la réalisation nécessite un certain intervalle de temps appelé durée.

Les tâches représentent toutes les opérations à effectuer pour la fabrication des produits. Une tâche, c'est à dire un ensemble d'opérations, requiert pour son exécution certaines ressources qu'il faut programmer de façon à optimiser un certain objectif.

Si l'ensemble des tâches à exécuter au cours du temps est donné a priori, c'est-à-dire leurs dates (date de début et date de fin) et leurs ordres d'exécution sont connus à l'avance, le problème d'ordonnancement est dit statique. Dans le cas contraire, si l'ensemble des tâches à exécuter évolue dans le temps, le problème est dit dynamique.

Deux types de tâches sont distingués :

* *les tâches morcelables* (préemptives) qui peuvent être exécutées en plusieurs fois facilitant ainsi la résolution de certains problèmes,

* *les tâches non morcelables* (indivisibles) qui sont exécutées en une seule fois et ne peuvent pas être interrompues avant qu'elles soient complètement terminées.

- **Les Ressources**

Une ressource est un moyen technique ou humain utilisé pour permettre la réalisation des tâches. Ce moyen technique est donc nécessaire et indispensable pour le bon fonctionnement du cycle de fabrication.

Dans un atelier, plusieurs types de ressources sont distingués :

- *les ressources renouvelables*, qui, après avoir été allouées à une tâche, redeviennent disponibles et qui peuvent être réutilisées (machines, personnel, etc.),
- *les ressources consommables*, qui, après avoir été allouées à une tâche, ne sont plus disponibles, et sont donc épuisées (argent, matières premières, etc.),
- *les ressources partageables* qui peuvent être partagées entre plusieurs tâches.

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

Ces ressources peuvent être classées d'une autre manière :

- *les ressources de type disjonctif*, qui ne peuvent exécuter qu'une opération ou une tâche à la fois,
- *les ressources de type cumulatif*, qui peuvent exécuter plusieurs opérations simultanément.

Une machine est considérée comme un type de ressource qui n'est caractérisé que par son horaire de travail **[Mesghouni, 1999]**.

- **Les contraintes**

Les contraintes représentent les conditions à respecter lors de la construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter lors de la résolution de ces problèmes. Ces contraintes peuvent être classées en deux types : endogène et exogène.

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement les variables représentant les relations reliant les tâches et les ressources.

➢ ***Les contraintes endogènes***

Elles constituent des contraintes liées directement au système de production et à ses performances telles que :

- Les dates de disponibilité des machines et des moyens de transport,

- Les capacités des machines et des moyens de transport,

- Les séquences des actions à effectuer ou les gammes des produits.

➢ ***Les contraintes exogènes***

Ces contraintes sont imposées extérieurement. Elles sont indépendantes du système de production ; on distingue :

- Les dates de fin de fabrication au plus tard du produit imposées généralement par les commandes,

On distingue les contraintes liées directement au système (endogènes) et celles imposées extérieurement (exogènes).

- Les priorités de quelques commandes et de quelques clients,

- Les retards possibles accordés pour certains produits.

Une autre classification consiste à distinguer :

➢ Les contraintes de gamme ou de précédence, caractérisant l'ordre d'exécution des tâches selon la gamme,

➢ Les contraintes de capacités, caractérisées par un ensemble de conflits pour l'utilisation des ressources, pouvant se diviser en :

- Contraintes disjonctives où une seule ressource est partagée par deux ou plusieurs tâches à la fois,
- Contraintes cumulatives où il y'a partage de plusieurs ressources par plusieurs tâches,

➢ Les contraintes liées directement aux produits finis (dates de livraison des commandes, priorités, dates de péremption, etc.)

1.3.1 Les objectifs d'ordonnancement

Dans la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions, ou plus simplement à leur admissibilité.

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques, construits sur la base d'indicateurs de performances. On cherchera donc à minimiser ou maximiser de tels critères. On note par exemple ceux

- liés au temps :
  - Le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches
  - Le stock d'en-cours de traitement
  - Différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées ;
- liés aux ressources :
  - La quantité totale ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches
  - La charge de chaque ressource ;
  - Liés à une énergie ou un débit ;
  - Liés aux coûts de lancement, de production, de transport, etc., mais aussi aux revenus, aux retours d'investissements.

1.3.2 Les problèmes d'atelier multi-machines

Les problèmes d'ordonnancement d'ateliers se décomposent en trois grandes classes de problèmes 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*.

1.3.2.1 Le type flow-shop

Les ateliers de type « *flow-shop* » pour lequel la ligne de fabrication est constituée de plusieurs machines en série ; toutes les opérations de toutes les tâches passent par toutes les machines dans le même et unique ordre. Ce type d'atelier est dit à cheminement unique. Dans les ateliers de type « *flowshop* hybride », une machine peut exister en plusieurs exemplaires identiques et parallèles.

Dans ce type d'atelier (Figure 1.2), on dispose de *n* pièces qui doivent s'exécuter suivant le même ordre sur les *m* machines qui composent l'atelier.

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.

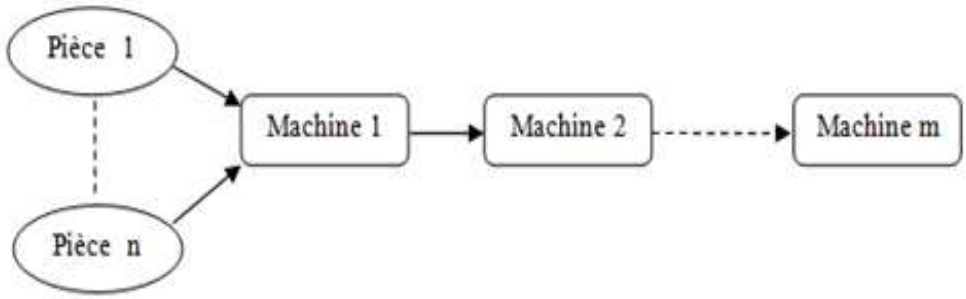


Figure1.1: Atelier *Flowshop.*

On distingue quatre types de *Flowshop* :

- ***Flowshop pur :*** tous les temps opératoires sont positifs.

- ***Flowshop généralisé :*** les temps opératoires peuvent être nuls si une tâche ne doit pas subir un traitement sur une machine particulière.

- ***Flowshop de permutation :*** toutes les tâches sont disponibles à l'instant 0, les tâches s'exécutent dans l'ordre défini à l'instant 0, le dépassement n'est pas autorisé.

- ***Flowshop hybride :*** c'est un atelier *Flowshop* dans lequel chaque machine est remplacée par un étage composé de plusieurs machines qui ne sont pas forcément identiques et disposées en parallèle.

Chaque travail visite successivement chaque étage et ne doit passer que par une seule machine par étage.

1.3.2.2 Le type job-shop

Dans les ateliers de type « *job-shop* », les opérations sont réalisées selon un ordre total bien déterminé, variant selon la tâche à exécuter.

Ce type d'ateliers est nommé aussi atelier à cheminements multiples. Dans ce cas, plusieurs changements d'outils sont à envisager.

- ***Le type job-shop flexible***

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.

Le *job-shop* flexible est une extension du modèle *jobshop* classique. Sa particularité essentielle réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations.

Plus précisément, une opération est associée à un ensemble contenant toutes les machines pouvant effectuer cette opération.

Dans un atelier *Jobshop*, aucune séquence d'opérations n'est fixe (Figure 1.2), et une pièce peut circuler plusieurs fois sur la même machine pour des tâches différentes.

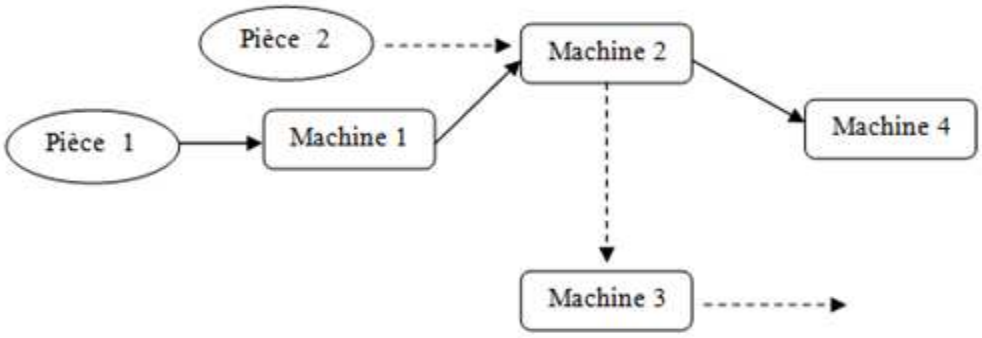


Figure 1.2: Atelier *Jobshop*

1.3.2.3 Le type open-shop

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.

Aucun ordre de fabrication n'est imposé, dans ce cas ; l'acheminement de toutes les opérations est multiple et libre; ces opérations peuvent être exécutées dans n'importe quel ordre. Des extensions à ces types de base ont été définies dans le but de se rapprocher des ateliers réels de production. Par exemple, la flexibilité des machines qui consiste à associer à chaque opération un ensemble de machines c'est-à-dire que cette opération peut être traitée par une machine quelconque de cet ensemble. Un problème d'affectation est alors ajouté au problème initial d'ordonnancement.

Les types suivants sont distingués :

- les problèmes à « machines parallèles » où les opérations sont indépendantes et sont traitées par le même ensemble de machines,

- Les problèmes de type «flow-shop hybride » constituant une extension du problème du type « flow-shop », où la première opération de chaque produit est traitée par le premier ensemble, la deuxième opération par le deuxième ensemble, et ainsi de suite.

1.4 Les méthodes de résolution des problèmes d'ordonnancement

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

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

La complexité d'un problème est la complexité de son meilleur algorithme connu. On peut ainsi classer les problèmes suivant leur complexité. Cependant, on distingue deux familles de problèmes : les problèmes qui ont une complexité polynomiale, et les autres, ceux dont la complexité est exponentielle (problèmes NP-complets ou NP-difficiles) **[Rebreynd, 1999]**.

Les problèmes d'ordonnancement sont, dans le cas général, des problèmes de type NP-difficile.

Définition : *on appelle algorithme polynomial, un algorithme dont le nombre d'opérations est un polynôme de n ; par exemple $O(n^a)$ avec a une constante.*

Tous les problèmes NP-difficiles ne sont pas de difficulté identique. On rencontre par exemple des problèmes qui ne peuvent pas être résolus en temps polynomial, ces problèmes sont dits NP-difficiles au sens ordinaire ou simplement NP-difficile, pour d'autres problèmes, on peut ne pas trouver d'algorithme polynomial, ils sont dits NP-difficiles au sens fort.

L'appartenance à la classe des problèmes NP-difficiles signifie qu'on ne pourra probablement jamais résoudre les problèmes de grande taille de façon optimale. Les méthodes de résolution des problèmes d'ordonnancement représentent une grande partie des problèmes d'optimisation combinatoire. En effet pour de tels problèmes, les méthodes exactes requièrent un effort calculatoire qui croit exponentiellement avec la taille des instances du problème (explosion combinatoire) et, rapidement, les heuristiques ou métaheuristiques deviennent l'unique moyen d'obtenir une bonne solution en un temps raisonnable.

Dans la plupart des cas, les problèmes d'ordonnancement ont été démontrés NP-difficiles **[Khalouli, 2010]** et le temps de résolution par des méthodes optimales augmente de façon exponentielle avec la taille du problème.

Dans la littérature, nous retrouvons plusieurs approches de résolution des problèmes d'ordonnancement qui font appel aux techniques de l'optimisation combinatoire.

1.4.1 Méthodes exactes

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

Ces méthodes ont l'avantage d'être capable de garantir l'optimalité de la solution obtenue, mais les problèmes classés NP-Difficiles demeurent hors de portée de ces méthodes.

- La programmation dynamique établie par Bellman (1954). Cette méthode a été appliquée pour la résolution de certains problèmes d'optimisation à l'aide de formules de récurrence.

- Le Branch an Bound (Procédure de séparation et évaluation), consistant à appliquer une technique de séparation et évaluation. La séparation permet de décomposer le problème en un ensemble de sous-problèmes de taille réduite. Les sous problèmes résultants sont ensuite évalués d'une relaxation (continue ou lagrangienne principalement) jusqu'à ne plus avoir que des problèmes faciles à résoudre ou dont on sait avec certitude qu'ils ne peuvent pas contenir de solution optimale **[Jourdan, 2010].**

Les méthodes exactes ont l'avantage de fournir une solution optimale au problème traité, mais elles sont généralement couteuses en terme du temps de calcul pour les problèmes de types NP-Difficiles. Au contraire, les méthodes approchées visent à générer des solutions de haute qualité en un temps de calcul raisonnable, mais il n'existe aucune garantie de trouver la solution optimale. Ces méthodes approchées sont elles mêmes divisées en deux sous- catégories : les *algorithmes d'approximation* et *les heuristiques*. Contrairement aux heuristique, les algorithmes d'approximation garantissent la qualité de la solution trouvée par rapport à l'optimal. Enfin, il existe encore deux sous- classes de méthodes heuristiques : les heuristiques spécifiques à un problème donné, et les *métaheuristiques* qui regroupent des méthodes plus génériques.

1.4.2 Méthodes approchées :

Depuis la fin des années 80, les méthodes approchées, et plus particulièrement les métaheuristiques, suscitent un intérêt croissant pour leur capacité à fournir des solutions d'excellente qualité pour une consommation en temps de calcul réduite.

Les heuristiques sont des méthodes approchées conçues de manière simple, rapide et ciblée sur un problème particulier.

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

Elles semblent être tout à fait adaptées à l'optimisation de systèmes de production et c'est donc ce type de méthode qui sera utilisé dans la suite de ce travail.

- **Les métaheuristiques :**

Parmi les heuristiques, certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors de métaheuristiques.

- **Présentation générale:**

Les métaheuristiques sont apparues dans les années 80. Ce sont des méthodes d'optimisation, de type stochastique, de problèmes difficiles. De telles méthodes sont incomplètes. En effet, il n'est jamais garanti que leur utilisation aboutisse à la découverte d'une solution. Il est encore moins sûr qu'une solution optimale soit atteinte.

Lorsqu'une heuristique est adaptable à un grand nombre de problèmes, elle est alors dite métaheuristique.

Le terme métaheuristique provient de la concaténation des mots méta (« au-delà » en grec) et heuristique ("trouver" en grec). L'interprétation de ces mots peut signifier que l'on effectue des recherches à un très haut niveau. En effet, les métaheuristiques sont très génériques, et permettent de s'adapter à une grande classe de problèmes. Le but des métaheuristiques est de minimiser (ou de maximiser) une (ou plusieurs) fonction(s) objectif(s). Par exemple, on peut vouloir trouver le plus court chemin entre un point A et un point B, sachant que des obstacles sont présents (minimisation). On peut aussi vouloir trouver comment effectuer le plus grand nombre de tâches en un temps donné (maximisation). L'évolution des métaheuristiques se fait de manière itérative. Ceci a l'avantage de permettre l'arrêt de l'algorithme quand on le souhaite, et de récupérer la meilleure solution trouvée jusqu'à présent. Cela permet par exemple de ne jamais arrêter l'algorithme tant qu'il n'a pas atteint de solution satisfaisante sauf si l'utilisateur le demande. Il n'est pas obligé d'attendre la fin de l'exécution pour obtenir un résultat. Cependant, plus l'algorithme travaille longtemps, plus il y a de chances que la solution s'améliore.

Un point important des métaheuristiques est qu'elles font évoluer des solutions en les améliorant à chaque itération. Il faut qu'il soit possible d'évaluer ces solutions. En effet, il est toujours nécessaire de pouvoir les comparer afin de retenir la meilleure. Une solution est considérée comme meilleure si elle est plus proche du résultat attendu qu'une autre. En général, une part d'aléatoire est présente afin d'empêcher l'algorithme de tomber dans un minimum local. La part d'aléatoire est différente selon les algorithmes. Il existe des métaheuristiques à méthode implicite, où la part d'aléatoire ne suit pas une loi donnée (algorithmes génétiques, par exemple), et des métaheuristiques à méthode explicite, où une distribution de probabilité suit une loi définie à chaque itération (les colonies de fourmis, par exemple). La solution initiale est souvent choisie au hasard. Elle évolue ensuite pour s'améliorer.

1.5 Conclusion :

Dans ce chapitre, nous nous sommes intéressés à quelques notions concernant les systèmes flexibles de production, l'ordonnancement de la production, et les méthodes dédiées à ces types de problèmes.

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

L'ordonnancement de type jobshop est généralement de type complexe démontré NP-difficile. Pour de tels problèmes, les méthodes exactes se trouvent incapables de les résoudre.

Comme il est souvent très difficile de trouver la meilleure solution possible (optimale), on se contente souvent de chercher une solution sous optimale. Une solution est dite sous optimale quand son optimalité n'a pas pu être prouvée. Il s'agit donc d'une solution approchée.

Six métaheuristiques (les algorithmes génétiques, les essaims particulaires, les colonies de fourmis, la recherche tabou, le recuit simulé, et l'électromagnétisme) ont été adaptées par Souier (2009) pour la résolution du problème d'ordonnancement dans le modèle FMS que nous étudions. Le chapitre suivant est consacré à une présentation détaillée des métaheuristiques, leurs origines, et leurs algorithmes.

Chapitre 2

Les métaheuristiques

Chapitre 2

Les Métaheuristiques

Ce chapitre est consacré aux métaheuristiques, méthodes formant une famille d'algorithmes d'optimisation combinatoire, visant à résoudre les problèmes de type NP-difficiles pour lesquels on ne connait pas de méthodes classiques plus efficaces. Nous commençons par une présentation générale, suivie d'une section consacrée aux principes des métaheuristiques les plus répondues. Enfin nous présentons d'une manière détaillée les deux algorithmes, l'algorithme mimétique, et l'algorithme API, que nous avons adapté pour la résolution de notre problème.

2.1. Introduction

Les approches traditionnelles, pour résoudre les problèmes NP-difficiles, consistent à appliquer des techniques d'optimisation à une formulation analytique. Cependant, ces modèles ont montré leurs limites **[Buzzcot et Yao, 1986], [Ben-Arieh, 1988]**, **[Newman, 1988]**. 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 **[Talbi, 2004]**.

Les chercheurs se sont alors orientés vers l'utilisation de méthodes approchées appelées « heuristiques ». Contrairement à une méthode exacte, qui vise à l'obtention d'une solution optimale, l'objectif d'une heuristique est de trouver une « bonne » solution en un temps raisonnable. Les heuristiques sont beaucoup plus adaptées au contexte industriel, où il ne s'agit pas tant de satisfaire complètement tel ou tel objectif que de fournir une solution d'ordonnancement réaliste, qui procure une satisfaction jugée acceptable d'un ensemble d'objectifs. Pour cette raison, les logiciels d'ordonnancement disponibles sur le marché font tous appel à des méthodes heuristiques **[Talbi, 2004]**.

Les métaheuristiques constituent une classe de méthodes qui fournissent des solutions de bonne qualité en temps raisonnable à des problèmes combinatoires réputés difficiles pour lesquels on ne connait pas de méthode classique plus efficace **[Ayari, 2010]**.

Ce chapitre est consacré aux métaheuristiques, nous allons présenter celles les plus connues, leurs origines, leurs principes de base, et leurs algorithmes, ainsi que les deux approches que nous avons adapté dans notre travail, à savoir, l'algorithme mimétique et l'algorithme API basé sur le comportement de fourragement d'une espèce de fourmis primitives (Les Pachycondyla apicalis).

2.2. Généralités sur les méthodes approchées

Depuis toujours, les chercheurs ont tenté de résoudre les problèmes NP-difficiles le plus efficacement possible. Pendant longtemps, la recherche s'est orientée vers la proposition d'algorithmes exacts pour des cas particuliers polynomiaux **[Sevaux, 2004]**. Certains problèmes demeurent hors de portée des méthodes exactes. Les méthodes approchées, dites aussi d'approximation, constituent une alternative intéressante pour traiter les problèmes d'ordonnancement de grande taille. Elles sont définies comme étant des procédures exploitant au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible **[Nicholson, 1971]**. Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle ***heuristique***.

Les heuristiques sont des algorithmes qui peuvent résoudre les problèmes de type NP-difficiles.

L'apparition des heuristiques a permis de trouver des solutions en général de bonne qualité pour résoudre les problèmes **[Sevaux, 2004]**. Lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de ***méta-heuristique*** **[Khalouli, 2010]**.

Lorsque les premières métaheuristiques apparaissent, beaucoup de chercheurs se sont lancés dans l'utilisation de ces méthodes. Cela a conduit à une avancée importante pour la résolution pratique de nombreux problèmes. Cela a aussi créé un engouement pour le développement même de ces méthodes. Il existe des équipes entières qui ne travaillent qu'au développement des métaheuristiques **[Sevaux, 2004]**.

Les solutions trouvées par les heuristiques sont des solutions approchées, l'optimalité n'est pas garantie.

2.3. Les métaheuristiques

Les métaheuristiques constituent une partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire **[Ayari, 2010]**. On appelle métaheuristiques (du grec, méta=qui englobe) des méthodes conçues pour échapper aux minima locaux. Le terme méta s'explique aussi par le fait que ces méthodes sont des structures générales dont il faut instancier les composants en fonction du problème par exemple, le voisinage, les solutions de départ ou les critères d'arrêt.

Les métaheuristiques sont apparues au début des années 1980 avec une ambition commune : résoudre au mieux les problèmes dits d'optimisation difficile. En effet, celles-ci s'appliquent à toutes sortes de problèmes discrets, et elles peuvent s'adapter aussi aux problèmes continus.

L'un des intérêts majeurs des métaheuristiques est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable.

Un des enjeux de la conception des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter à un problème donné **[Dréo, 2004]**.

Les heuristiques qui peuvent être adaptées à des problèmes différents sans changements majeurs dans l'algorithme de base sont dites métaheuristiques.

Les métaheuristiques les plus classiques sont celles fondées sur l'exploration d'un voisinage, et forment la première famille. L'algorithme part d'une solution et la fait évoluer à chaque itération sur l'espace de recherche. Les méthodes les plus connues dans cette catégorie sont la méthode GRASP (Greedy Randomized Adaptive Search Procedure), le recuit simulé (Simulated Annealing, SA), et la recherche tabou (Tabu Search, TS).

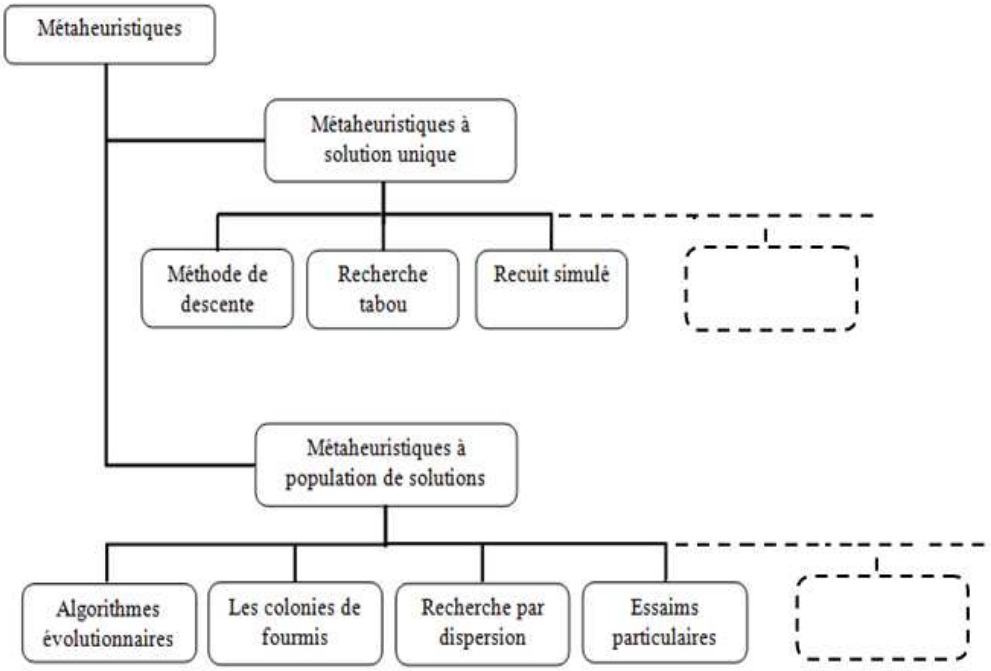
L'autre approche possible est basée sur l'utilisation d'une population de solutions, et comme leur nom l'indique ces approches travaillent sur un ensemble de solutions. Ce type d'approche constitue la seconde famille. A chaque itération, la métaheuristique fait évoluer un ensemble de solutions en parallèle. Les algorithmes génétiques (Genetic Algorithms, GA), de recherche dispersée (Scatter Search, SS) ou les algorithmes de colonies de fourmis (Ant Colony, AC) sont des exemples courants de ce type de méthodes.

2.3.1. Intensification et diversification

Toutes les métaheuristiques s'appuient sur un équilibre entre l'intensification de la recherche et la diversification de celle-ci. D'un côté, l'intensification permet de rechercher des solutions de plus grande qualité en s'appuyant sur les solutions déjà trouvées et de l'autre, la diversification met en place des stratégies qui permettent d'explorer un plus grand espace de solutions et d'´echapper à des minima locaux. 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) **[Sevaux, 2004]**.

La métaheuristique doit trouver les solutions de bonne qualité dans chaque zone de l'espace de recherche mais en s'échappant des minima locaux.

Il existe de nombreuses métaheuristiques allant de la simple recherche locale à des algorithmes plus complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à un large éventail de problèmes d'optimisation combinatoire. On peut partager les métaheuristiques en deux grandes classes : les métaheuristiques à solution unique (c.à.d. évoluant avec une seule solution) et celles à solutions multiples ou population de solutions. Les méthodes d'optimisation à population de solutions améliorent, au fûr et à mesure des itérations, une population de solutions. L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité.



On trouve les métaheuristiques à solution unique, qui travaillent sur une seule solution et celles à population de solutions qui travaillent sur un ensemble de solutions.

Figure 2.1: Classification des métaheuristiques.

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

On appelle méthode de recherche (ou algorithme de recherche) locale celle qui converge vers un optimum local. Les méthodes itératives à solution unique sont toutes basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale, puis l'améliore pas à pas en choisissant une nouvelle solution dans son voisinage **[Bachelet, 99]**.

Les métaheuristiques à solutions uniques sont celles qui partent d'une solution et tentent de l'améliorer.

Le processus s'arrête lorsqu'on ne peut plus améliorer la solution courante ou parce que le nombre maximal d'itérations (fixé au départ) est atteint. Quoique, ces méthodes ne soient pas complètes (rien n'assure qu'elles méthodes parviennent très souvent à trouver des solutions de bonne qualité dans des temps de calcul raisonnables). Elles sont souvent les premières méthodes testées sur les nouveaux problèmes combinatoires émergeant des applications réelles et académiques. On trouve dans la littérature de nombreuses méthodes locales. Les plus anciennes et les plus utilisées sont : la méthode de descente, le recuit simulé, la recherche tabou,…etc.

2.3.2.1. Principe des méthodes de recherche locale

Les méthodes de recherche locale sont basées sur l'amélioration progressive d'une solution dans un voisinage.

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]**.

Généralement, les méthodes de recherche locale donnent des optima locaux.

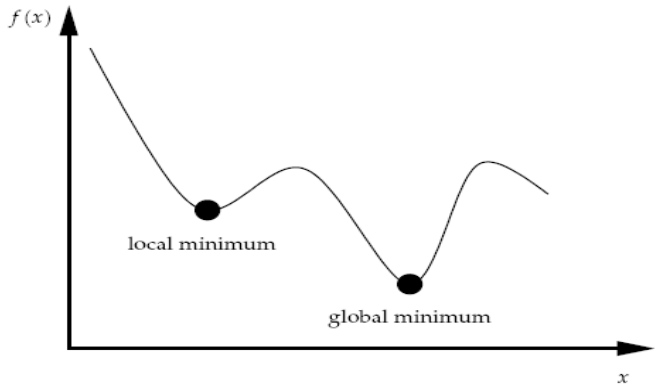


Figure 2.2: Minimum local et global **[Sevaux 2004]**.

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.

Il existe des méthodes de recherche locale qui utilisent des techniques permettant de s'échapper aux minima locaux.

2.3.2.2. Le recuit simulé (Simulated Annealing)

La méthode du recuit simulé a été introduite en 1983 par **Kirkpatrick et *al*., (1983)**. Cette méthode originale est basée sur les travaux bien antérieurs de **Metropolis et *al*., (1953)**. Cette méthode que l'on pourrait considérer comme la première métaheuristique 'grand public' a reçu l'attention de nombreux travaux et principalement de nombreuses applications **[Sevaux 2004].**

L'analogie historique s'inspire du recuit des métaux en métallurgie : un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Si on le refroidit lentement, les atomes se réarrangent, les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalente à un optimum global **[Jourdan, 2003]**.

Le recuit simulé est un algorithme inspiré de la métallurgie pour l'amélioration de la qualité du métal.

La méthode du recuit simulé, appliquée aux problèmes d'optimisation, considère une solution initiale et recherche dans son voisinage une autre solution de façon aléatoire. L'originalité de cette méthode est qu'il est possible de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle. Ceci permet d'échapper aux minima locaux **[Jourdan, 2003]**.

L'évolution de l'algorithme est gérée par un paramètre appelé énergie, l'énergie désigne la valeur de la fonction objectif, dite aussi fonction coût de la solution courante. La température T est un paramètre de l'algorithme : elle détermine la probabilité avec laquelle une dégradation de la solution courante est acceptée. La température décroit graduellement au cours du temps.

Au début de la recherche, la température élevée permet au recuit d'explorer l'espace de recherche sans être piégé par les optima locaux. A mesure que la température décroît, le recuit est de moins au moins capable d'explorer l'espace de recherche, et la recherche se concentre sur une zone déterminée : l'exploration prend le dessus sur l'exploration. Par conséquent la vitesse de décroissance de la température détermine la vitesse à laquelle le recuit « converge » vers une solution **[Duvivier, 2000].**

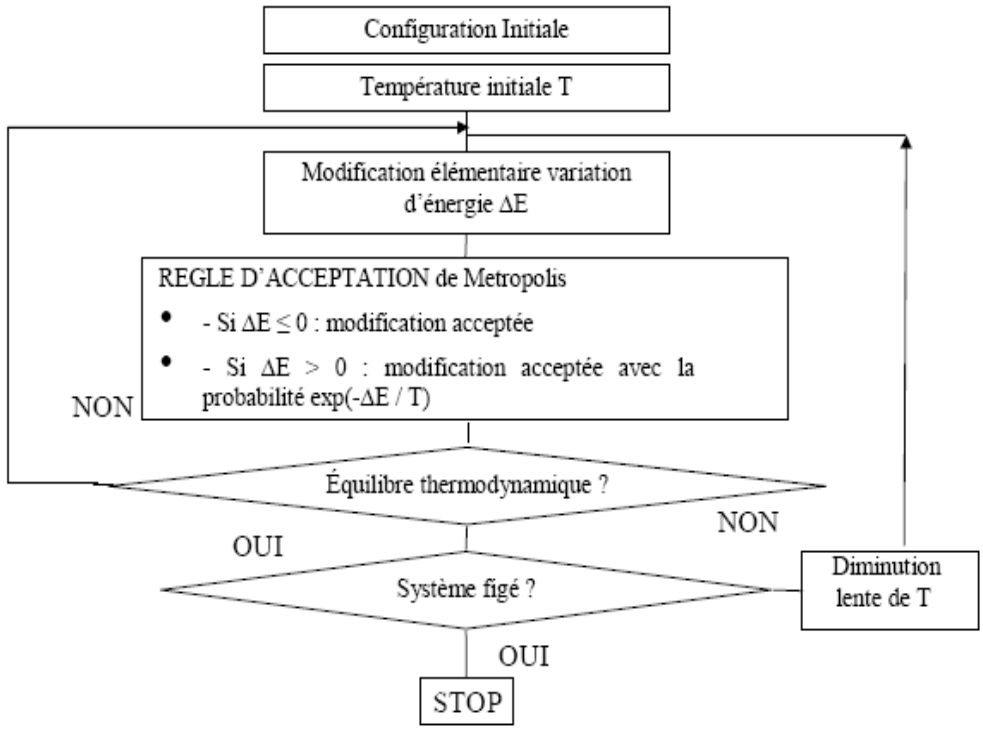


Figure 2.3: L'organigramme du recuit simulé **[Dréo, 2004]**

2.3.2.3. La recherche tabou (Tabu Search)

Dans un article présenté par **Glover (1986)**, on voit apparaître pour la première fois, le terme *tabu search.* Contrairement au recuit simulé qui ne génère qu'une seule solution x_0 'aléatoirement' dans le voisinage *N(x)* de la solution courante *x*, la méthode tabou, dans sa forme la plus simple, examine le voisinage *N(x)* de la solution courante *x* **[Sevaux, 2004]**. Cette méthode utilise la notion de mémoire pour éviter de tomber dans un optimum local.

La recherche tabou tente de trouver la meilleure solution dans un voisinage, en évitant d'examiner les mêmes solutions plusieurs fois.

Le principe de l'algorithme est le suivant : à chaque itération, le voisinage (complet ou sous ensemble de voisinage) de la solution courante est examinée et la meilleure solution est sélectionnée. En appliquant ce principe, la méthode autorise de remonter vers des solutions qui semblent moins intéressantes mais qui ont peut être un meilleur voisinage. Le risque est de cycler entre deux solutions.

L'algorithme tabou est similaire à celui du recuit simulé mais dans lequel on interdit d'effectuer des cycles.

Pour éviter ce phénomène, la méthode a l'interdiction de visiter une solution récemment visitée. Pour cela, une liste taboue contenant les attributs des dernières solutions visitées et tenue à jour. Chaque nouvelle solution considérée enlève de cette liste la solution la plus anciennement visitée **[Jourdan, 2003].**

Les solutions déjà examinées seront mémorisées dans une liste dite la liste tabou.

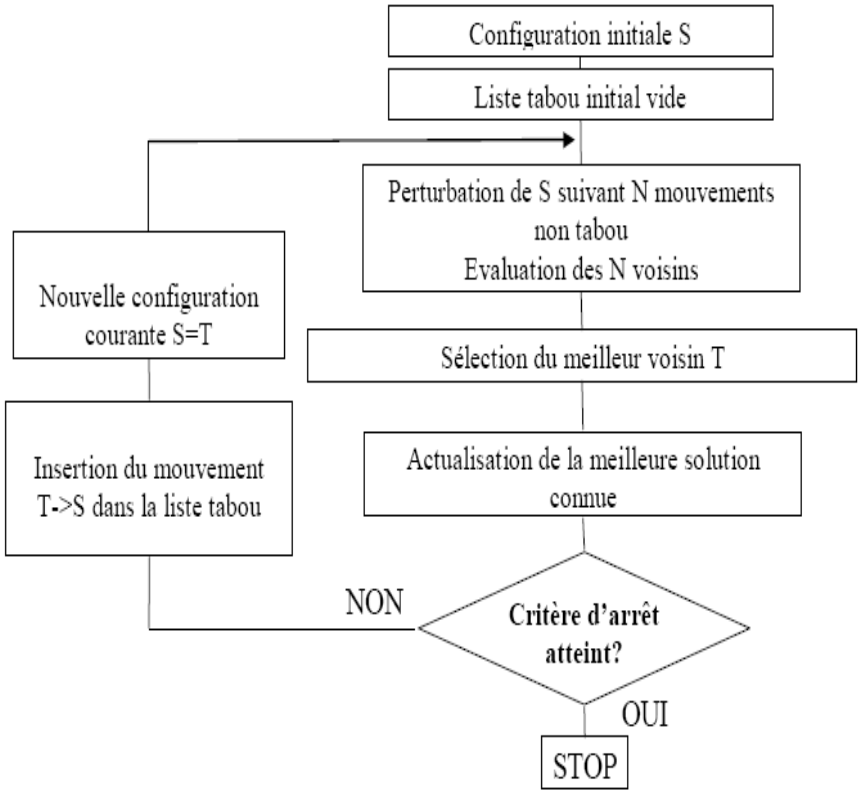


Figure 2.4: L'organigramme de la recherche tabou **[Dréo, 2004].**

2.3.3. Métaheuristiques à base de population :

Les métaheuristiques à population de solutions travaillent sur tout un ensemble de solutions qui évoluent au cours des itérations.

Contrairement aux méthodes à solution unique qui font intervenir une seule solution, les méthodes de recherche à population, comme leur nom l'indique, travaillent sur un ensemble de solutions appelé population. Le principe général de toutes ces méthodes consiste à combiner des solutions entre elles pour en former de nouvelles en essayant d'hériter des 'bonnes' caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit atteint (nombre de générations maximum, nombre de générations sans améliorations, temps maximum, borne atteinte,…). Parmi ces algorithmes à population, on trouve deux grandes classes qui sont les algorithmes génétiques et les colonies de fourmis. Les algorithmes génétiques ont beaucoup fait parler d'eux et depuis longtemps. Les colonies de fourmis sont des techniques plus récentes.

Ces méthodes se différencient par leur manière de représenter les problèmes à résoudre et par leur façon de faire évoluer la population d'une génération à l'autre.

Il existe bien d'autres techniques que nous allons décrire quelques unes telles que les essaims particulaires, l'algorithme mimétique, et les colonies de fourmis d'une espèce de fourmis primitives dite ' Pachycondyla apicalis'.

2.3.3.1. Les colonies de fourmis (Ant Colony Optimization)

Des constatations ont montré que les fourmis, bien qu'elles aient individuellement des capacités cognitives très limitées et qu'elles soient pour certaines espèces aveugles (aucune information visuelle du monde environnant ne leur parvient), sont capables de sélectionner collectivement le plus court chemin entre une source de nourriture et leur nid. La coordination des activités collectives chez les fourmis est due à des mécanismes de communications indirectes via une substance volatile chimique appelée « phéromone », à laquelle les fourmis sont très sensibles **[Khalouli, 2010].**

Les messages chimiques utilisés par les fourmis sont un moyen de sélectionner le plus court chemin afin de transporter leur nourriture.

En se déplaçant du nid à la source de nourriture et vice-versa (ce qui, dans un premier temps, se fait essentiellement d'une façon aléatoire), les fourmis déposent au passage sur le sol une substance odorante appelée phéromone, ce qui a pour effet de créer une piste chimique. Les fourmis peuvent sentir ces phéromones qui ont un rôle de marqueur de chemin : quand les fourmis choisissent leur chemin, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones. Cela leur permet de retrouver le chemin vers leur nid lors du retour. D'autre part, les odeurs peuvent être utilisées par d'autres fourmis pour retrouver les sources de nourriture détectées par leurs consœurs **[Jourdan, 2003].**

La particularité des algorithmes des colonies des fourmis est qu'elles se basent sur la totalité de la population pour la recherche des solutions.

De plus, comme la phéromone s'évapore avec le temps, le chemin le plus long est de moins en moins emprunté et sa trace disparaît presque complètement. Par conséquent, les sentiers les plus courts seront les plus concentrés en phéromones **[Khalouli 2010].**

Au début, les fourmis explorent différents chemins en effectuant des déplacements aléatoires. Une fois qu'un chemin menant à une source de nourriture est découvert, elles y déposent une quantité de phéromone renforçant ainsi son importance et la probabilité d'être choisi par d'autres fourmis de la colonie. D'un autre côté, les mauvais chemins auront tendance à être oubliés, voire même, disparaître avec l'évaporation de la phéromone. Ce procédé est basé sur le mécanisme de rétroaction positive. Il assure que les fourmis utilisent la voie d'accès la plus courte car elle sera la plus imprégnée par la phéromone **[Khalouli, 2010].**

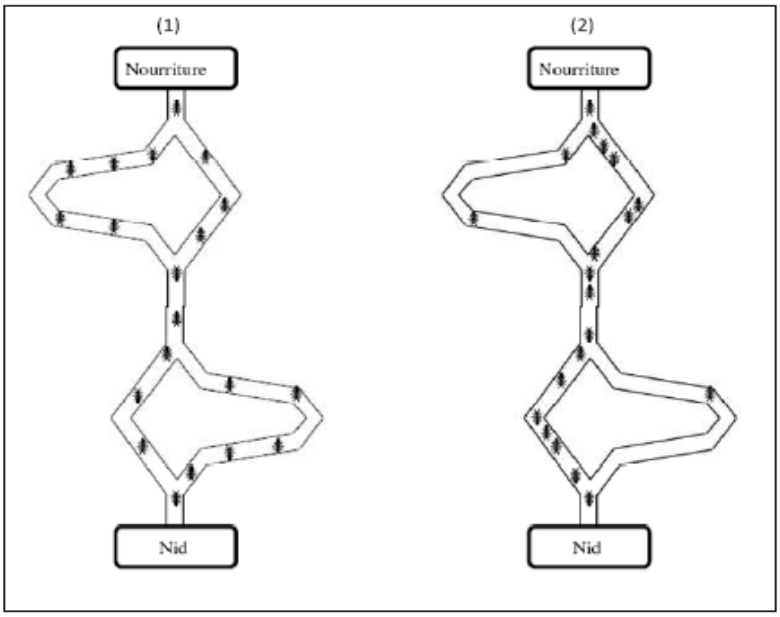


Figure 2.5: Expérience de sélection du plus court chemin **[Dréo, 2004].**

La base du fonctionnement de cet algorithme repose sur les phéromones que déposent les fourmis lors de leurs trajets. La forte concentration des phéromones poussera les individus à utiliser majoritairement un chemin plutôt que l'autre.

(1) Au début de l'expérience les fourmis explorent indifféremment les deux branches du pont.

(2) A la fin de l'expérience les fourmis ont tendance à emprunter le même chemin (qui est le plus court).

Les algorithmes de colonies de fourmis ont été introduits par Dorigo et al. (1996). Une des applications principales de la méthode originale était le problème du voyageur de commerce et depuis elle a considérablement évolué.

L'ACO consiste à travailler sur une population de solutions, chacune correspondant à une fourmi artificielle. Une structure de donnée commune, qui renferme des informations sur les quantités de phéromones artificielles accumulées dans l'espace de recherche, est utilisée pour coordonner le fonctionnement de la population**.** Ainsi, par analogie aux comportements des fourmis réelles, les algorithmes de colonie de fourmis sont basés sur la communication indirecte d'une colonie de fourmis artificielles. La quantité de phéromones représente une information utilisée par les fourmis pour former une solution de manière stochastique à un problème donné. Il s'agit donc d'une métaheuristique de construction aléatoire biaisée par un ensemble de données globales représentant une mémoire sur la qualité des solutions via les traces (ou quantités) de phéromones. Cet ensemble est régulièrement mis à jour par un mécanisme simulant l'évaporation de la phéromone. Il effectue des décisions probabilistes fonction des quantités de phéromones artificielles et d'une vision locale du problème via une heuristique d'information dite aussi visibilité **[Khalouli, 2010].**

Algorithme 2.1: Un simple algorithme de colonie de fourmis

(1) **Initialiser**
(2) **Répéter**
(3) **Pour** chaque fourmi **faire**
(4) **Construire** une solution basée sue une procédure de construction, dépendant de la trace de phéromone
(5) **Mettre à jour** la trace de phéromone en se basant sur la qualité de la solution trouvée
(6) **Fin pour**
(7) **Jusqu'à** satisfaction d'un critère d'arrêt

2.3.3.2. Les algorithmes génétiques (Genetic Algorithms)

2.3.3.2.1. Principes des algorithmes évolutionnaires

Le paradigme des algorithmes évolutionnaires **[De Jong et *al*., 2000]**, **[Back et *al*., 2000]** fondé dans les années 1960, consiste à s'inspirer des mécanismes de l'évolution naturelle et à utiliser le concept de populations d'individus ou solutions pour résoudre des problèmes du monde réel **[Dupas, 2004]**.

Les algorithmes génétiques reposent sur le principe de la sélection naturelle et de l'évolution des espèces.

Les algorithmes évolutionnaires sont des techniques d'optimisation itératives et stochastiques inspirées de la théorie de l'évolution. Cette théorie a été élaborée par Charles Darwin (1859) et fondée sur le modèle biologique. Un algorithme évolutionnaire simule le comportement d'évolution des êtres vivants. Dans ce cas, on considère que l'évolution tend à produire des organismes plus adéquats à leur environnement en ayant recourt à des phénomènes comme la sélection naturelle et l'héritage génétique. Au cours du cycle de simulation, trois opérateurs interviennent généralement : la recombinaison (ou croisement), la mutation et la sélection. La recombinaison et la mutation créent de nouvelles solutions candidates, tandis que la sélection élimine les candidats les moins prometteurs. Plusieurs types d'évolution ont été développés, parmi lesquelles nous citons : les stratégies d'évolution, la programmation évolutionnaire, et les algorithmes génétiques. Les plus connus et les plus utilisés en optimisation et en ordonnancement sont les algorithmes génétiques **[Khalouli, 2010].** Cette section est consacrée à la présentation des algorithmes génétiques.

Les algorithmes génétiques sont faciles à utiliser, efficaces et facilement modifiables.

Il s'agit d'une famille d'algorithmes basés sur l'évolution des espèces. L'évolution d'une espèce est simplement une suite successive d'améliorations afin qu'elle soit la mieux adaptée au milieu dans lequel elle évolue. Cette adaptation est réalisée grâce à la sélection naturelle et aux mécanismes de reproduction **[Rebreyend, 1999]**.

La première description du processus des algorithmes génétiques a été donnée par Holland en 1975, puis **Goldberg (1989)** les a utilisés pour résoudre des problèmes concrets d'optimisation.

Le but des algorithmes génétiques est d'optimiser une fonction prédéfinie, appelée fonction objectif, ou fitness, ils travaillent sur un ensemble de solutions candidates, appelé « population » d'individus ou chromosomes. Ces derniers sont constitués d'un ensemble d'éléments appelés « gènes » **[Chaari, 2010].**

Il n'existe pas de représentation fixée mais, en général, les chromosomes ont la forme de chaîne(s).

Il est nécessaire d'introduire un certain nombre de termes avant d'aborder plus précisément les algorithmes génétiques. Par conséquent, nous définissons les termes suivants :

Définition 1 : (Gène)

Un gène est une suite de bases azotées qui contient le code d'une protéine donnée. On appellera gène la suite de symboles qui codent la valeur d'une variable.

Définition 2 : (Chromosome)

Un chromosome est constitué d'une séquence finie de gènes qui peuvent prendre des valeurs appelées allèles qui sont prises dans un alphabet qui doit être judicieusement choisi pour convenir au problème étudié.

Pour chacun des individus on mesure la faculté d'adaptation par la fitness.

Un chromosome est, donc un élément de l'espace de recherche, généralement, un chromosome est représenté sous forme de chaînes.

Définition 3 : (Individu)

Un individu est une solution potentielle. Dans la plupart des cas un individu sera représenté par un seul chromosome.

Définition 3 : (Population)

Une population est un groupe d'individus sur lequel l'algorithme génétique effectue un certain nombre d'opérations.

Définition 3 : (Génération)

Une génération est l'ensemble des opérations qui permettent de passer d'une population à une autre. Ces opérations sont généralement la sélection des individus de la population courante, le croisement des individus, et la mutation.

En optimisation, la fitness est souvent la valeur de la fonction objectif de la solution associée à l'individu.

Définition 4 : (fonction objectif)

Pour évaluer la pertinence d'une solution par rapport à une autre, on introduit ce que l'on nomme une fonction d'adaptation (ou bien fonction d'évaluation ou de fitness) qui correspond à l'utilité de la solution par rapport au problème.

Définition 5 : (sélection des survivants)

Cette étape consiste à ne garder que les solutions les plus intéressantes. Par convention, le nombre d'individus est le même d'une génération à l'autre. En d'autres termes, il y a autant de morts que de nouveau-nés.

L'algorithme génétique est basé sur les fonctions de sélection, de croisement et de mutation.

L'algorithme génétique manipule une population d'individus qui évolue au cours du temps grâce aux divers opérateurs (croisement, mutation, sélection) en favorisant la reproduction des individus performants. Il est efficace tant que les individus de la population sont différents **[Duvivier, 2000]**.

Une première population est choisie soit aléatoirement, soit par des heuristiques ou par des méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques. Cette population doit être suffisamment diversifiée pour que l'algorithme ne reste pas bloqué dans un optimum local. C'est ce qui se produit lorsque trop d'individus sont semblables **[Chaari, 2010].**

La sélection favorise les individus qui ont une meilleure fitness.

Les algorithmes génétiques génèrent de nouveaux individus, dits solutions parents, de telle sorte qu'ils soient plus performants que leurs solutions parents. Le processus d'amélioration des individus s'effectue par utilisation d'opérateurs génétiques, qui sont la sélection, le croisement et la mutation.

2.3.3.2.2. Les étapes de l'algorithme génétique

L'algorithme génétique commence par générer une population initiale d'individus, pour chacun d'entre eux nous calculons la fitness (qui représente la valeur de la fonction objectif ou fonction d'évaluation de la solution associée à l'individu considéré), et nous appliquons par la suite un opérateur de sélection pour la reproduction. Ces opérateurs seront soumis ensuite à un opérateur de croisement qui va combiner les individus sélectionnés pour donner naissance à des nouvelles solutions (nouveaux individus). Les individus résultants peuvent être mutés par un opérateur de mutation.

Le croisement génère des individus, qui sont le résultat de la combinaison de deux solutions parents.

Les trois opérateurs de sélection, croisement, et mutation permettent de générer une nouvelle population d'individus (dite nouvelle génération). Nous évaluons la fitness des individus obtenus. De génération en génération, on obtient des individus plus performants que ceux des anciennes générations. Le processus s'arrête lorsqu'un critère d'arrêt est atteint.

La mutation est une modification aléatoire d'un individu, c'est un opérateur qui permet d'ajouter de la diversité à la population.

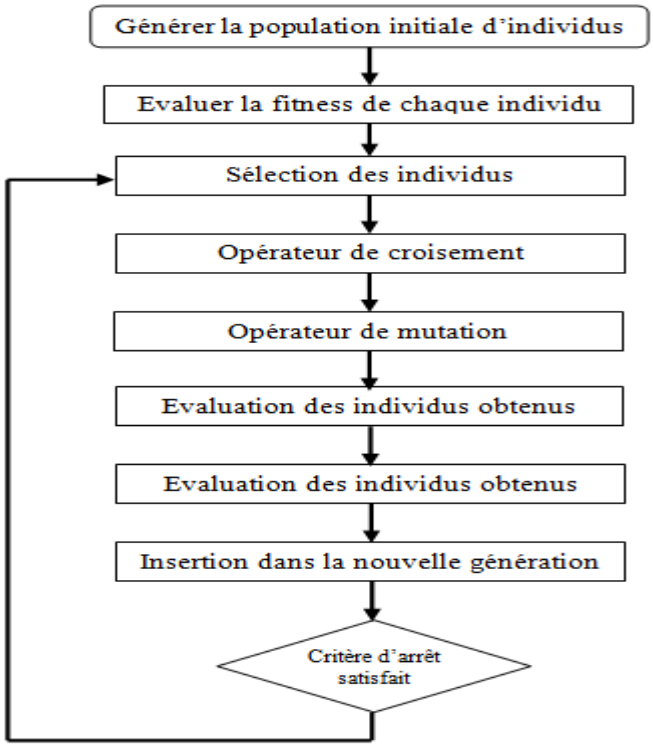


Figure 2.6: Principales étapes d'un algorithme génétique.

2.3.3.3. Les essaims particulaires (Particle Swarms Optimization)

L'optimisation par essaims particulaires (OEP) est une méthode proposée en 1995 aux Etats Unis sous le nom de Particle Swarms Optimisation (PSO) par Eberhart et Kennedy **(1995)**. Cette métaheuristique s'inspire des interactions entre individus au sein d'une population telle que les nuées d'oiseaux, les bancs de poissons, et les essaims d'abeilles. L'algorithme permet de faire converger les solutions vers des optimums locaux. Bien que cette approche ait été développée pour les milieux continus, elle a aussi été appliquée aux milieux discrets.

PSO est une métaheuristique inspirée des systèmes naturels (tels que les nués d'oiseaux).

En effet, ces groupes d'animaux montrent des dynamiques de déplacements relativement complexes, alors qu'individuellement, ils n'ont accès qu'à des informations limitées, comme la position et la vitesse de leurs plus proches voisins. Ce type d'approches utilise une population de solutions potentielles connue sous l'appellation d'essaim, les individus sont appelés particules. Elle se base sur la collaboration des particules entre elles en échangeant des informations permettant ainsi l'émergence de comportements complexes. Chaque particule est caractérisée par sa position courante et 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 10]**.

A chaque itération de l'algorithme, les particules se déplacent dans l'espace de recherche selon une certaine vitesse. Le calcul de de cette vitesse dépend de plusieurs facteurs tels que la vitesse actuelle pondérée par un coefficient d'inertie *w*, ainsi que l'écart par rapport à leur meilleure position connue et à celle de leurs voisines pondérées respectivement par des coefficients de confiance c_1 et c_2. Les particules voisines d'une autre particule sont appelées ses informatrices **[Lemamou, 2009].**

La structure de base d'un algorithme d'OEP est illustré par la figure 2.7 à chaque itération de l'algorithme, la nouvelle vitesse et la nouvelle position de chaque particule sont calculées. Les particules sont ensuite évaluées et leurs mémoires, c.à.d, leurs meilleures positions connues, sont mises à jour. La diversité au sein de la population permet à l'algorithme de ne pas rester bloqué dans un optimum local **[Lemamou, 2009].**

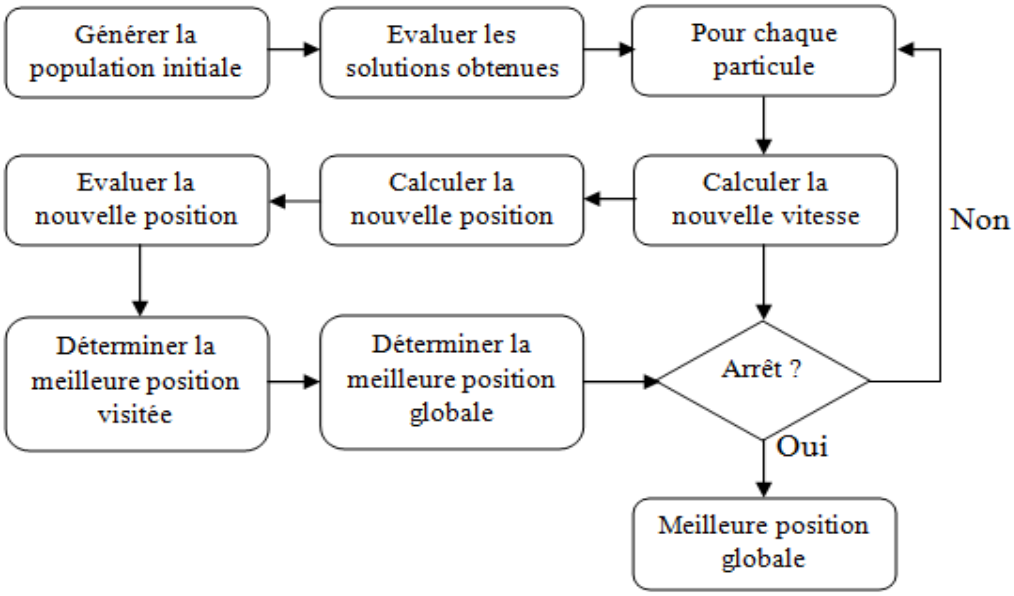


Figure 2.7: Principe de l'algorithme d'optimisation par essaims particulaires **[Lemamou, 2009].**

2.3.3.4. L'électromagnétisme [Souier, 2009]

Cette métaheuristique a été proposée par Birbil et Fang en 2003 **[Birbil et Fang, 2003]** pour résoudre les problèmes d'optimisation difficile continus efficacement. Elle a été inspirée d'une analogie du mécanisme attraction-répulsion de la théorie d'électromagnétisme, elle est une métaheuristique basée sur une population de solutions.

Dans cette approche la charge de chaque point est relative à la valeur de la fonction objectif que nous essayons d'optimiser. Cette charge détermine également l'attraction ou la répulsion du point. D'ailleurs, la force électrostatique entre deux points est directement proportionnelle aux charges de ces points et inversement proportionnelle au carré de la distance entre ces points. La charge fixe de chaque particule est donnée comme suit :

$$q_i = exp\left(-n * \left(f(x_i) - f(x_{best}) / \left(\sum_{k=1}^{m} \left(f(x_k) - f(x_{best})\right)\right)\right)\right), \forall\, i \qquad 2.6$$

Où q_i est la charge de particule *i*, $f(x_i)$, $f(x_{best})$, et $f(x_k)$ sont les valeurs des fonctions objectives des particules *i*, la meilleure particule, et la particule *k*. *m* est la taille de la population et *n* la dimension du problème.

La qualité de solution ou la charge de chaque particule détermine l'effet d'attraction et de répulsion dans la population. Une meilleure solution encourage d'autres particules à converger aux vallées attrayantes tandis qu'une mauvaise solution décourage les autres particules pour se déplacer vers cette région. Ces particules se déplacent avec toute la force et ainsi des solutions diversifiées sont produites. La formule suivante donne la force de la particule *i*.

L'électromagnétisme est une métaheuristique à population de solutions inspirée de la physique.

$$F_i = \sum_{j \neq i}^{m} \begin{cases} (x_j - x_i) * \frac{(q_i * q_j)}{\|x_j - x_i\|^2} : f(x_j) < f(x_i) \\ (x_i - x_j) * \frac{(q_i * q_j)}{\|x_j - x_i\|^2} : f(x_j) \geq f(x_i) \end{cases}, \forall i \qquad 2.7$$

Le pseudo-code de cette métaheuristique est le suivant:

Algorithme2.2 : L'électromagnétisme

(1) **Initialiser** les points
(2) **Tant que** (critère d'arrêt non atteint) faire
(3) **Effectuer** une recherche locale pour chaque particule.
(4) **Calculer** la force total *F ()* de chaque particule.
(5) **Déplacer** la particule par *F ()*.
(6) **Evaluer** les particules.
(7) **Fin Tant que**

2.3.3.5. L'algorithme mimétique

Pour être performante, une méthode de recherche doit associer judicieusement exploration et exploitation de l'espace de recherche. Or, une méthode de recherche est rarement aussi efficace pour exploiter que pour explorer l'espace de recherche. Une solution consiste à ajouter des mécanismes complémentaires dans une méthode de recherche donnée. De ce fait, il peut être extrêmement bénéfique d'associer une méthode de recherche dont la capacité d'exploration est très élevée à une méthode de recherche dont le point fort est l'exploitation de l'espace de recherche. Par ailleurs, une seule méthode ne permet pas d'obtenir de très bons résultats sur un large ensemble de problèmes **[Wolpert et Macready, 1997].** Dans ce cas, pourquoi se limiter à l'utilisation d'une seule méthode alors qu'il serait possible s'utiliser (simultanément) plusieurs méthodes de recherche pour améliorer les performances ? D'où l'idée de fédérer ces méthodes de recherche en une méthode hybride **[Duvivier, 2000].**

Un algorithme hybride est le mélange de deux (ou plus) algorithmes distincts.

L'hybridation revient à combiner plusieurs algorithmes de nature différente. Par exemple, on peut croiser un algorithme génétique avec une heuristique. En général, le résultat est meilleur que celui qui serait obtenu par une des méthodes choisies **[Sevaux, 2004]**.

Les métaheuristiques hybrides sont apparues en même temps que le paradigme lui-même, mais la plupart des chercheurs n'y accordaient que peu d'intérêt **[Cotta et al., 2005]**. Elles deviennent maintenant populaires, car les meilleurs résultats trouvés pour plusieurs problèmes d'optimisation combinatoire ont été obtenus par des algorithmes hybrides **[Noël, 2007].**

Un algorithme mimétique est une hybridation entre un algorithme génétique et une recherche locale.

Les algorithmes génétiques tels que décrits précédemment ont montré leur efficacité dans un nombre de problèmes d'horizons très divers **[Sevaux, 2004].** En effet, la puissance des algorithmes génétiques provient du fait qu'ils sont capables de balayer de manière globale l'espace des solutions contrairement aux méthodes de descente ou aux heuristiques qui explorent une petite zone de cet espace **[Rebreyend, 1999]**. Un inconvénient d'un algorithme génétique est que les opérateurs standards de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche **[Hoos et Stüzle, 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 **[Hernandez, 2008].** C'est le cas des algorithmes mimétiques de **Moscato (1989)** présentés dans cette section.

Moscato **(1989)** introduit pour la première fois les algorithmes mimétiques. On rencontre aussi le nom d'*algorithmes génétiques hybrides* ou celui de *genetic local search.* Quelque soit le nom qu'on lui donne, l'idée principale de cette technique est de rendre plus agressif un algorithme génétique par l'ajout d'une recherche locale en plus de la mutation **[Sevaux, 2004].** Un algorithme mimétique est donc une combinaison, ou encore une coopération entre un algorithme génétique et une méthode de recherche locale.

On peut rendre un algorithme génétique plus performant en le croisant avec d'autres méthodes.

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].** Ainsi, on peut explorer rapidement les zones intéressantes de l'espace de recherche pour les exploiter en détail **[Hernandez, 2008].**

L'idée de Moscato est donc d'ajouter une recherche locale qui peut être une méthode de descente ou une recherche locale plus évoluée (recuit simulé ou recherche tabou par exemple). Cette recherche locale sera appliquée à tout nouvel individu obtenu au cours de la recherche. Cette simple modification entraine de profonds changements dans le comportement de l'algorithme lui-même. Après avoir créé un nouvel individu à partir de deux parents sélectionnés, on applique une recherche locale et sous certaines conditions on applique un opérateur de mutation à cet individu **[Sevaux, 2004].**

Algorithme 2.3: L'algorithme mimétique

(1) Initialiser : générer une population initiale P de solutions
(2) Appliquer une procédure de recherche locale sur chaque solution de P
(3) Répéter
(4) **Sélection :** choisir deux solutions *x* et *x'*
(5) **Croisement** : combiner deux solutions parents *x* et *x'* pour former une solution *y*
(6) **Recherche locale :** appliquer une procédure de recherche locale sur *y*
(7) **Mutation :** appliquer un opérateur de mutation sur
(8) **Choisir** un individu *y'* pour être remplacé dans la population
(9) **Remplacer** *y'* par *y* dans la population
(10) Jusqu'à satisfaire un critère d'arrêt.

Dans cet algorithme, l'opérateur de mutation assure la diversification de la méthode, d'autre part, l'intensification est produite par l'application de la méthode de recherche locale.

La méthode de recherche locale utilisée dans un algorithme mimétique 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.

- **Méthodes de descente**

A partir d'une solution trouvée par heuristique par exemple, on peut très facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante.

Dans les deux méthodes, simple descent et deepest descent la diversification est totalement absente.

L'algorithme 2.4 présente le squelette d'une méthode de descente (simple descent). 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 2.4: 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 $f(y) \geq f(x), \forall y \in N(x)$

Une version plus « agressive» de la méthode de descente est la méthode de plus grande descente **[Sevaux, 2004].** Au lieu de choisir une solution x_0 dans le voisinage de *x*, on choisit toujours la meilleure solution x_0 du voisinage de *x*. L'algorithme 2.5 donne une description de cette méthode.

Algorithme 2.5: La méthode de grande descente

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

Ces deux méthodes sont évidemment sujettes à de nombreuses critiques. Elles se basent toutes les deux sur une amélioration progressive de la solution et donc resteront bloquées dans un minimum local dès qu'elles en rencontreront un. Il existe de manière évidente une absence de diversification. L'´equilibre souhaité entre intensification et diversification n'existe donc plus **[Sevaux, 2004].**

2.3.3.6. L'algorithme API

La plupart des animaux sont beaucoup moins intelligents que l'homme, mais sont cependant capables de réalisations assez évoluées. Considérons par exemple les insectes. Individuellement, une fourmi ou une abeille n'est pas capable de s'adapter aux situations nouvelles. Pourtant, le groupe réalise des tâches très évoluées, comme la construction d'une ruche ou d'une fourmilière. Les insectes sociaux semblent compenser leurs faiblesses individuelles par une coordination globale qui donne à la colonie une forme d'intelligence bien supérieure à celles de ses membres.

L'algorithme API est une métaheuristique inspirée du comportement de fourragement d'une espèce de fourmis classées primitives dites les Pachycondyla apicalis.

Dans cette section nous présentons l'algorithme API qui modélise le comportement de fourragement d'une population de fourmis primitives (Pachycondyla apicalis). Dans les parties sous sections suivantes de cette section, nous allons présenter cette espèce de fourmis, son comportement de recherche de nourriture, ainsi que sa modélisation telles que présentées et décrites par Monmarché (2000).

Cette espèce de fourmis est caractérisée par sa stratégie de recherche de proie relativement simple d'un point de vue local et global. Dans cette espèce de fourmis les individus chassent en solitaire en essayant de couvrir uniformément un espace donné autour du nid en le partitionnant en sites de chasses. Le nid change d'emplacement périodiquement. En optimisation, cela correspond à un algorithme qui effectue plusieurs recherches aléatoires en parallèle, dans un sous espace centré en un point. Le déplacement du nid correspond à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé.

2.3.3.6.1 Pachycondyla apicalis

Les fourmis sont des insectes qui vivent en sociétés. Elles peuplent tous les continents du monde et ont des comportements très variés suivant l'habitat dans lequel elles évoluent.

Le comportement social des fourmis constitue un formidable modèle bio-inspiré d'auto-organisation. Nous présentons dans cette section une nouvelle métaheuristique s'inspirant de l'auto-organisation chez l'espèce de fourmis néotropicale Pachycondyla apicalis. Ceci revient à utiliser la métaphore de cette espèce de fourmis pour concevoir des métaheuristiques d'optimisation adaptées aux problèmes combinatoires.

Pachycondyla apicalis est une espèce de fourmis primitives qui utilisent des principes relativement simples pour la recherche de nourriture de point de vue locale et globale.

Cette section présente un algorithme pour l'optimisation basé sur le comportement des fourmis l'espèce Pachycondyla apicalis. Cette étude trouve son origine dans les travaux de Dominique Fresneau sur la stratégie de fourragement originale de cette espèce de fourmis ponérines **[Fresneau, 1985]**, **[Fresneau, 1994]**.

Pour notre étude nous nous sommes basés sur la modélisation algorithmique faite par Monmarché (2000). Nous allons commencer tout d'abord par présenter les caractéristiques biologiques qui ont pu être exploitées du point de vue d'une modélisation algorithmique avec comme objectif de l'appliquer à notre problème d'ordonnancement.

Monmarché (2000) a donné la grande part à cette espèce de fourmis et a énoncé que l'intérêt de ces fourmis pour l'optimisation vient du fait qu'elles utilisent des principes relativement simples d'un point de vue globale et local pour rechercher leurs proies.

L'auto-organisation qui caractérise cette espèce de fourmis peut être reprise pour concevoir des métaheuristiques d'optimisation.

L'auteur a fait une comparaison entre l'API et les algorithmes génétiques et il a démontré théoriquement la force de l'API pour diversifier les solutions. Dans les AG, l'effet cumulé de la sélection et du croisement a l'inconvénient de faire disparaître la diversité nécessaire à l'exploration : après un certain nombre d'itérations la population se concentre sur les points les plus sélectionnés, seule la mutation introduit alors une certaine diversité.

Dans l'API, le processus est inverse : les fourmis explorent à partir d'un point central qui est le Nid et tendent à s'éloigner au fur et à mesure des itérations.

2.3.3.6.2 Biologie des Pachycondyla apicalis

Pachycondyla apicalis est une ponérine néotropicale que l'on rencontre en Amérique du Sud, en particulier au Mexique.

Figure : 2.8 : L'espèce de fourmis Pachycondyla apicalis.

Contrairement à d'autres espèces de fourmis, les Pachycondyla apicalis n'ont pas un comportement collectif direct pour la recherche de nourriture.

Sa morphologie et le peu de communication entre individus la classe parmi les fourmis dites primitives mais certains aspects de son adaptation démentent ce jugement. Les fourmis *Pachycondyla apicalis* vivent en petites colonies composées de quelques dizaines d'individus (40 à 100 ouvrières).

Ces fourmis n'utilisent pas de messages chimiques pour la recherche de nourriture, c'est le repère visuel qui est utilisé.

Les fourmis de cette espèce installent leur nid dans de vieilles souches ou des arbres morts en décomposition qui offrent ainsi un habitat instable pour des fourmis qui ne savent pas construire de fourmilière. Après un certain temps, lorsque la vétusté de leur nid devient trop importante, elles doivent déménager et chercher un nouveau refuge. Les petits insectes et les cadavres d'insectes présentent la source de nourriture de cette espèce de fourmis. Les fourmis de cette espèce ne participent pas toutes à la recherche de nourriture, certaines ouvrières restent au nid pour s'occuper du couvain, et d'autres prospectent individuellement autour de la fourmilière et ramènent les proies au nid.

Le comportement de recherche de nourriture de ces fourmis est composé d'un ensemble de recherches locales parallèles sur des sites de chasse construits atour du nid.

Le comportement de recherche de nourriture de cette espèce de fourmis n'est pas un comportement collectif direct ; c'est-à-dire que les ouvrières ne déposent pas de message chimique sur le sol pour indiquer aux autres fourrageuses le chemin menant à une source de nourriture.

On peut supposer que la nature des proies recherchées n'encourage pas spécialement ce genre de communication inter-individus : la présence des proies étant relativement aléatoire, la capture d'une proie ne donne que peu d'informations sur la stabilité spatiale de la source de nourriture. Autrement dit, la probabilité de retrouver une proie dans la même zone qu'une précédente capture n'est pas suffisante pour y canaliser les forces de fourragement de la colonie. Cependant, d'un point de vue individuel, les ouvrières mémorisent leur site de capture et lors de leur prochaine sortie du nid, elles retournent systématiquement sur le dernier site de chasse fructueux. Cette spécialisation sectorielle est une réponse pour l'adaptation nécessaire à la découverte et l'exploitation de sources de nourriture.

Ce type de fourragement solitaire se retrouve particulièrement chez les espèces peu populeuses, les espèces à population importante utilisent des mécanismes de recrutement massif beaucoup plus couramment **[Holldobler et Wilson, 1990]**. Les fourrageuses solitaires développent en conséquence des mécanismes d'apprentissage plus évolués.

De sorties en sorties, les ouvrières s'éloignent du nid car la probabilité de trouver une proie est inversement proportionnelle à la densité de fourrageuses qui décroit évidemment quand la distance au nid augmente. Les fourrageuses ont donc un comportement collectif indirect puisque de manière statistique elles coopèrent pour couvrir au mieux leur espace de recherche que constitue le voisinage du nid. Elles construisent de cette façon une mosaïque de zones de chasse qui couvre la périphérie du nid.

La figure 2.9 présente les aires de fourragement d'une colonie *Pachycondyla apicalis*.

L'algorithme API effectue des recherches aléatoires parallèles dans le voisinage de certains points dits « sites de chasse ».

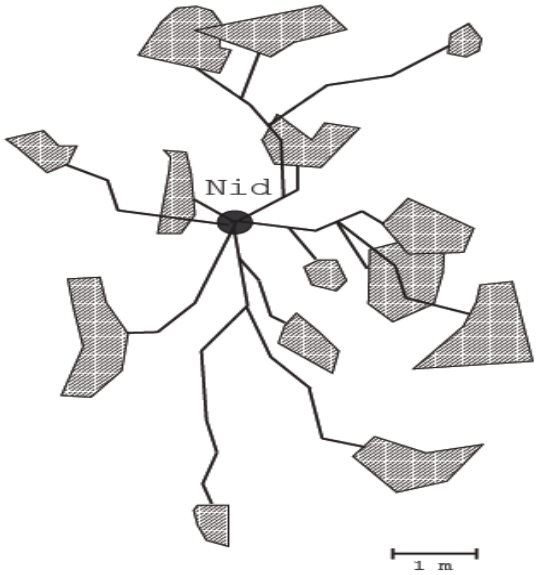


Figure 2.9: Exemple (fictif) de carte des trajets et aires de récolte des fourrageuses (inspiré de **[Fresneau, 1994]**).

Les sites de chasse sont créés autour d'un point dit le nid, généré initialement d'une façon uniformément aléatoire.

Le comportement de fourragement de *Pachycondyla apicalis* peut être résumé en trois règles **[Fresneau, 1994]**:

1. La découverte d'une proie entraine toujours le retour sur ce site lors de la sortie suivante, c'est là que la fourrageuse reprend ses nouvelles prospections ;
2. La découverte d'une proie pèse sur la décision de sortie des fourrageuses en réduisant l'intervalle de temps qu'elles passent au nid ;
3. Les fourrageuses semblent progressivement apprendre une association entre une direction de recherche opposée au nid et l'augmentation de la probabilité de succès.

Ces règles ont l'avantage évident d'être très simples. On peut cependant préciser quelques points :

– Lors de ses premières sorties, la fourmi sort du nid et prend une direction aléatoire mais s'éloigne peu du nid et retourne à l'abri de celui-ci à la moindre alerte. Son trajet est relativement sinueux ;

– Si la fourmi capture une proie, elle retourne directement au nid en ligne droite et mémorise visuellement le chemin qu'elle emprunte ;

– Après une capture, la fourmi utilise le chemin qu'elle a mémorisé pour retourner au site de capture ;

– Le retour sur le site de chasse se soldant par un échec peut se produire plusieurs fois de suite (en moyenne quatre fois) ;

– Un site de chasse ne produisant plus le renforcement que constitue la capture d'une proie est abandonné mais n'est pas obligatoirement oublié par la fourrageuse ;

– L'exploration d'un site de capture privilégie les directions qui éloignent la fourmi du nid dans une limite de périmètre imposée par le coût énergétique prohibitif que représente un échec à une grande distance du nid.

On peut parler d'apprentissage ; car la fourmi apprend les sites qui lui rapportent de la nourriture et elle est capable de les oublier quand elle n'y trouve plus de proies ou lorsque le nid a changé de localisation.

Comme nous l'avons déjà mentionné, la fragilité du nid impose des déménagements réguliers. Des fourmis éclaireuses partent alors à la recherche d'un nouvel abri. Le déménagement s'effectue grâce à des mécanismes de *recrutement en tandem* où une fourmi se fait guider par une de ses congénères jusqu'au nouvel emplacement. Ce changement de nid a pour effet de « réinitialiser » la mémoire des fourrageuses.

C'est à cette occasion que l'utilisation de repères visuels prend toute son importance : le marquage de chemins avec des phéromones perturberait l'adaptation des fourmis à la situation de leur nouveau nid, car les anciens chemins interfèreraient avec les nouveaux. Avec une mémoire visuelle, les fourrageuses reconstruisent un réseau de sites de chasse autour du nouveau nid plus aisément, plus rapidement.

L'aspect individuel de la recherche est particulièrement adapté à la fréquence d'apparition des proies : si une proie tombe sur le sol et est capturée, la fourrageuse reviendra explorer le site toute seule. Si la proie est trop lourde, elle sera découpée puis transportée en plusieurs voyages par la même fourmi qui utilise de cette façon sa mémoire. Si le site de chasse se trouve être un gisement (par exemple des termites) la fourrageuse reviendra systématiquement jusqu'à épuisement du site.

L'intérêt de la stratégie de fourragement des fourmis *Pachycondyla apicalis* réside dans sa simplicité et la bonne couverture de l'espace de recherche qui en résulte. On a ici un phénomène d'émergence : de règles de recherche simples et individuelles, qui ne tiennent pas compte du travail des autres fourmis, on obtient une exploration radiale de l'espace centrée sur le nid.

Dans la réalité, l'efficacité de la stratégie n'est pas parfaite si on considère le nombre de proies trouvées relativement au nombre de proies présentes. Cependant, la taille d'une colonie de *Pachycondyla apicalis* étant relativement réduite, les besoins en nourriture sont relativement modestes. Il semble que l'adaptation de ces fourmis ne réside pas seulement dans leur comportement de fourragement mais aussi dans le maintien de colonies peu populeuses. Ceci permet aux *Pachycondyla apicalis* de survivre dans des secteurs comportant de nombreux prédateurs concurrents et de ne pas nécessiter une grande quantité d'insectes. L'intérêt de ces fourmis pour l'optimisation vient du fait qu'elles utilisent des principes relativement simples à la fois d'un point de vue global et local pour rechercher leurs proies. A partir de leur nid, elles couvrent globalement une surface donnée en la partitionnant en sites de chasse individuels.

Partant du nid, les fourmis couvrent un espace donnée en le partitionnant en plusieurs sites de chasse et chaque fourmi explore ses propres sites.

Pour une fourmi donnée, on observe une stratégie d'exploration aléatoire des sites sensible au succès rencontré. Ces principes peuvent être repris pour résoudre un problème analogue qui est la recherche d'un minimum global, par exemple d'une fonction *f* définie de $\mathbb{R}^l$ dans $\mathbb{R}$.

2.3.3.6.3 Modélisation algorithmique

Dans cette section nous présentons la modélisation algorithmique proposée par Monmarché (2000) pour la résolution des problèmes d'optimisation.

Le nid est périodiquement déplacé, ce qui correspond en optimisation à un opérateur de réinitialisation dans les recherches parallèles où le point central est déplacé.

Monmarché considère une population de n fourmis fourrageuses $a_1,\ldots,a_n$ de l'espèce Pachycondyla apicalis. Ces agents sont positionnés dans l'espace de recherche, noté *S*, et vont tenter de minimiser ou de maximiser une fonction d'évaluation ou bien une fonction objectif f définie de *S* dans *R*.

Chaque point $s \in S$ est une solution valide du problème, ce qui signifie que *f* est définie en tout point de *S*. cet espace de recherche peut être un espace continu, binaire ou un espace de permutations.

La définition des deux opérateurs suivants est suffisante pour déterminer le déplacement des fourmis :

1. L''opérateur O_{rand} qui génère un point de *S* de manière uniformément aléatoire ;
2. L'opérateur O_{explo} qui génère un point s' dans le voisinage d'un point s.

L'opérateur O_{rand} est un opérateur qui permet de générer l'emplacement initial du nid d'une manière uniformément aléatoire. Tandis que le deuxième opérateur O_{explo} peut être une exploration aléatoire tout comme une heuristique inspirée par le domaine de recherche.

Le comportement des fourmis pour la recherche des proies peut être réparti en deux phases, la première présente le comportement local des fourmis, et la deuxième présente le comportement global.

2.3.3.6.4 Comportement local des fourmis

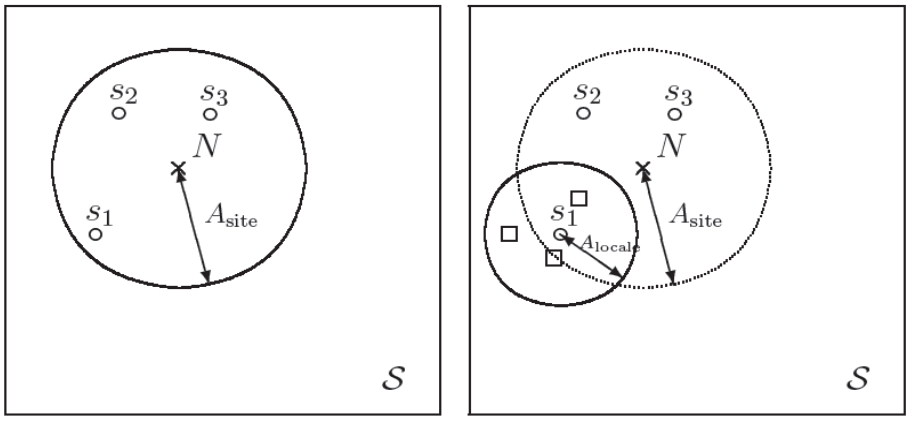


Figure 2.10: Exploration des sites de chasse.
(a) – recherche de sites de chasse. (b) – Exploration locale de la fourmi autour du site s_1.

Pour éviter de rester bloqué dans un minimum local, après chaque P_N itérations l'algorithme API place le nid sur la meilleure solution atteinte.

Le comportement local des fourmis est le comportement individuel de chaque fourmi lors de la génération de l'emplacement initial du nid et à chaque déplacement de celui-ci.

Chaque fourmi a_i quitte le nid pour se constituer une liste de p sites de chasse qu'elle mémorise. Un site de chasse est un point de S construit par l'opérateur O_{explo} avec une amplitude $A_{\text{site}}(ai)$ dans le voisinage de N. La fourmi a_i va ensuite procéder à une exploration locale autour d'un de ses sites de chasse (figure 2.10).

Initialement, quand l'intérêt des sites est inconnu, la fourmi choisit un site s au hasard parmi les p dont elle dispose. L'exploration locale consiste à construire un point s' de S dans le voisinage de s grâce à l'opérateur O_{explo} avec une amplitude $A_{\text{locale}}(a_i)$. La fourmi a_i capture une proie si cette exploration locale a permis de trouver une meilleure valeur de f, ce qui revient à avoir $f(s') < f(s)$ dans le cas de minimisation (et $f(s') > f(s)$ dans le cas de maximisation). Une amélioration de f modélise donc la capture d'une proie. A chaque fois qu'une fourmi parvient à améliorer $f(s)$ elle mémorise s' à la place de s et sa prochaine exploration locale aura lieu dans le voisinage de s'. Si l'exploration locale est infructueuse, pour la prochaine exploration, la fourmi choisira un site au hasard parmi les p sites qu'elle a en mémoire.

Quand un site a été exploré successivement plus de P_{locale} fois sans avoir rapporté de proie, il est définitivement oublié et sera remplacé par un nouveau site à la prochaine itération (c'est-à-dire la prochaine sortie du nid). Le paramètre P_{locale} représente une patience locale.

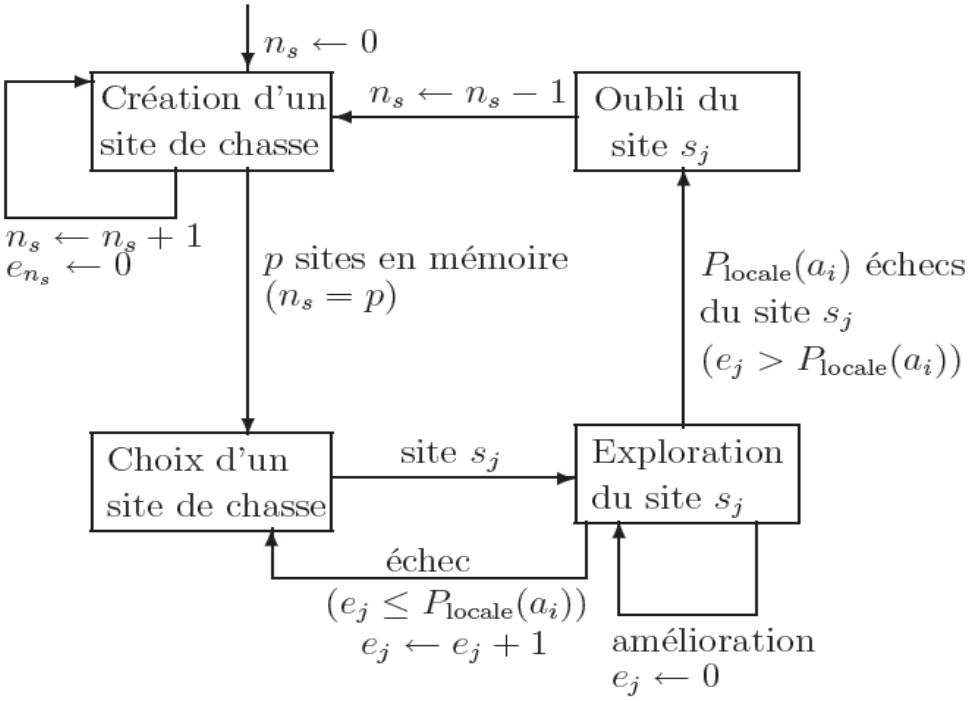


Figure 2.11: Le comportement de fourragement d'une fourmi.

a_i *:* une fourmi.
n_s est le nombre de sites de chasse dans la mémoire de la fourmi.
e_j : Le nombre d'échecs successifs rencontrés sur le site s_j.
P_{locale} : La patience locale de la fourmi, elle représente le nombre d'échecs rencontrés par la fourmi sur le même site de chasse.

2.3.3.6.5 Exploration globale

D'un point de vue global, API place le nid à une position *N* de *S* et procède à l'exploration de *S* autour de *N*. L'exploration est déterminée par le comportement local des fourmis.

A chaque pas de l'algorithme les *n* fourmis sont simulées en parallèle. A l'initialisation, le nid est placé dans *S* de manière uniformément aléatoire par l'opérateur O_{rand}. Puis le nid est déplacé toutes les P_N déplacements des *n* fourmis. Il est alors placé sur le meilleur point *s+* trouvé depuis son dernier déplacement. A chaque déplacement du nid les fourmis reprennent leur exploration à partir de la nouvelle position du nid.

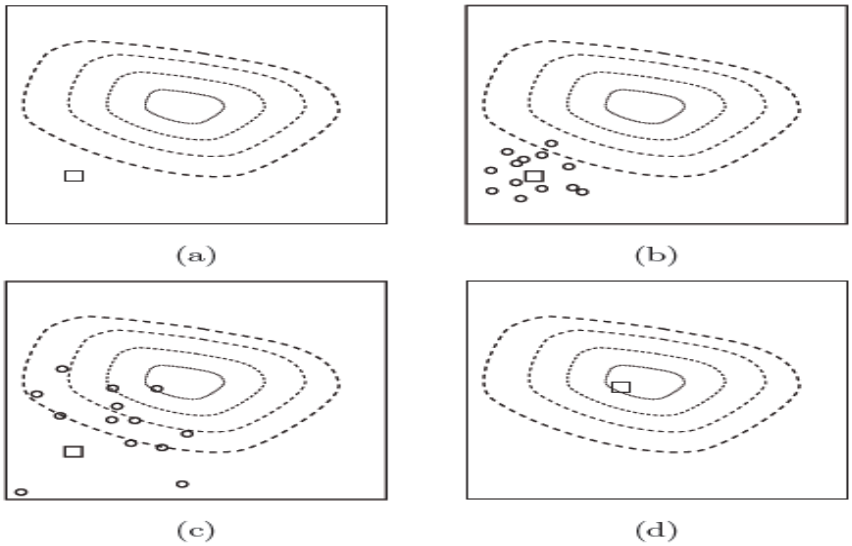


Figure 2.12: Exploration globale : Déplacement du nid [Monmarché, 2000].

(a) : le nid (le carré) est placé aléatoirement dans l'espace de recherche.
(b) : les sites de chasse (cercles) créés autour du nid.
(c) : l'exploration locale déplace cause le déplacement des sites vers des zones plus intéressantes de l'espace de recherche.
(d) : le nid est déplacé sur la position du meilleur site de chasse, les sites sont ensuite générés à partir de cette nouvelle position, come dans (b), et ainsi de suite.

L'algorithme API modélise le comportement global des fourmis, c'est-à-dire le déplacement du nid.

P_N représente un paramètre de patience pour le nid. On peut fixer cette patience suivant la patience locale d'une fourmi (P_{locale}) et la taille de la mémoire d'une fourmi (p) :

$$P_N = 2 \times (P_{locale} + 1) \times p$$

Ce calcul a pour but de laisser suffisamment d'itérations entre chaque déplacement de nid pour que les fourmis puissent créer et explorer leurs p sites de chasse. Une autre solution serait de ne déplacer le nid qu'uniquement si l'optimum n'a pas été amélioré depuis un certain nombre d'itérations, disons P_N, ou encore en utilisant certaines informations sur la population comme la dispersion des fourmis dans l'espace de recherche par exemple.

Enfin, à chaque déplacement du nid, la mémoire des fourmis est vidée et elles doivent reconstruire leurs p sites de chasse.

Du point de vue de l'optimisation, cela permet d'éviter des minima locaux dans lesquels les fourmis resteraient enfermées. Cela permet aussi de rassembler les fourmis autour du meilleur point trouvé et ainsi de concentrer les recherches.

On pourrait cependant procéder d'une manière plus « douce » : il suffirait de placer le nid à la position du minimum global trouvé par la colonie à chaque fois que celui-ci est amélioré sans réinitialiser toutes les fourmis.

Ainsi, quand une fourmi crée un nouveau site de chasse, elle le ferait dans le voisinage de l'optimum global.

On se débarrasse alors du choix de la patience du nid. Une autre méthode douce de déplacement du nid serait de le placer à une position intermédiaire entre sa position actuelle et s^+ : $N \leftarrow (1-\gamma)N + \gamma s^+$ o`u $\gamma \in [0, 1]$ (à condition que l'espace de recherche S le permette).

Les principales étapes de la simulation de la colonie de fourmis sont données par l'algorithme 2.6.

L'algorithme API en phase d'exploration globale, qui modélise le nid et le comportement global de toutes les fourmis ; incluant une phase d'exploration locale qui modélise le comportement individuel de chaque fourmi.

Algorithme 2.6: Algorithme API

(1) **Choisir** aléatoirement l'emplacement initial du nid : $N \leftarrow O_{rand}$
(2) $T \leftarrow 0$ /*indice du nombre d'itérations*/
(3) **Tant que** la condition d'arrêt n'est pas vérifiée faire
(4) **Pour** tout $a_i \in A$ faire
(5) API-FOURRAGEMENT (a_i)
(6) **Fin pour**
(7) **Si** le nid doit être déplacé alors
(8) $N \leftarrow s^+$ /meilleure solution atteinte*/
(9) **Vider** la mémoire de toutes les fourmis
(10) **Fin si**
(11) $T \leftarrow T + 1$
(12) **Fin Tant que**
(13) **Retourner** s^+ et $f(s^+)$

La condition d'arrêt peut être l'une des suivantes :
– s^+ n'a pas été améliorée depuis un certain nombre d'itérations ($T1$) ;
– T a atteint une valeur limite ($T2$) ;
– un certain nombre d'évaluations de solutions de f a été atteint ($T3$).

Dans la majorité des tests présentés par la suite, nous appliquons la dernière condition. D'une part cela suppose que l'évaluation des solutions est une opération coûteuse en temps de calcul, et d'autre part cela permet de comparer plus facilement API avec d'autres méthodes.
Le comportement local de fourragement d'une fourmi est donné par l'algorithme 2.7.

Algorithme 2.7: API-Fourragement(a_i)

(1) **Si** $n_s(a_i) < p$ alors
(2) /*La mémoire de la fourmi n'est pas pleine*/
(3) $n_s(a_i) \leftarrow n_s(a_i) + 1$
(4) **Construction** d'un site de chasse autour du nid :
$s_{n_s}(a_i) \leftarrow O_{explo}(N, A_{site})$
(5) **Initialisation** du compteur d'échecs du site construit : $e_{n_s}(a_i) \leftarrow 0$
(6) **Sinon**
(7) **Soit** s_j le site que la fourmi a exploré à sa dernière sortie
(8) **Si** $e_j > 0$ alors
(9) /* la dernière exploration de s_j a été infructueuse*/
(10) **Choisir** aléatoirement un site s_j de chasse ($j \in \{1, \dots, p\}$)
(11) **Finsi**
(12) **Exploration locale** autour du site s_j: $s' \leftarrow O_{explo}(s_j, A_{locale})$
(13) **Si** $f(s') < f(s_j)$ alors
(14) $s_j \leftarrow s'$
(15) $e_j \leftarrow 0$
(16) **Sinon**
(17) $e_j \leftarrow e_j + 1$
(18) **Si** $e_j > P_{locale}$ alors
(19) Effacer le site s_j de la mémoire de la fourmi
(20) $n_s(a_i) \leftarrow n_s(a_i) - 1$
(21) **Finsi**
(22) **Finsi**
(23) **Finsi**

L'algorithme API-FOURRAGEMENT modélise le comportement de chaque fourmi lors de l'exploration de sa propre zone de recherche.

2.4 Conclusion :

Ce chapitre a décrit les principales métaheuristiques, leurs origines, principes de fonctionnement, et leurs algorithmes de bases. De plus nous avons présenté les deux méthodes que nous avons adapté pour la résolution de notre problème, à savoir, l'algorithme mimétique et l'algorithme API basé sur le comportement de fourragement de l'espèce de fourmis Pachycondyla apicalis.

Le chapitre suivant est consacré à l'adaptation de ces deux algorithmes pour la résolution d'un problème d'ordonnancement dans un système flexible de production (FMS), à la présentation d'une étude de sensibilité du deuxième algorithme, et aux résultats obtenus et leurs interprétations. Les résultats obtenus seront comparés à ceux de l'algorithme génétique et de l'ACO utilisés par Souier (2009), et Souier et *al.*, (2010) pour résoudre le même problème.

Chapitre 3

Les algorithmes API et mimétique pour la résolution d'un problème d'ordonnancement

Chapitre 3

L'algorithme mimétique et les Pachycondyla apicalis pour l'ordonnancement d'un FMS

Dans ce chapitre, nous présentons le problème d'ordonnancement dans un système flexible de production qui va nous préoccuper dans le reste de ce document. Dans les sections de ce chapitre, nous présentons notre adaptation des deux algorithmes proposés, à savoir, l'algorithme mimétique noté MA, et l'algorithme basé sur le comportement de fourragement de l'espèce des fourmis Pachycondyla apicalis noté API,suivi par une étude un peu détaillée de l'algorithme API_m qui présente une version de l'algorithme API auquel nous avons introduit une petite modifacation qui a entrainé un changement majeur dans les résultats obtenus. Cette étude comporte une comparaison entre l'algorithme API original et celui que nous avons modifié noté API_m, ainsi qu'une analyse de sensibilité de l'algorithme API_m. Le reste du chapitre représente les résultats obtenus par simulations pour l'algorithme mimétique et l'algorithme API_m, comparés à ceux de l'algorithme génétique et de l'algorithme ACO.

3.1 Introduction :

Le contrôle et l'ordonnancement temps réel des systèmes flexibles de production sont devenus un domaine de recherche populaire depuis le début des années 80, 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 des pièces et des événements imprévus **[Mamalis et *al*., 1995]**, **[Rachamadugu et Stecke, 1994].**

Dans ce mémoire nous nous intéressons à deux métaheuristiques à base de population, l'algorithme mimétique qui combine un algorithme génétique avec une méthode de recherche locale (la méthode de descente) et l'algorithme API_m qui modélise le comportement de recherche de nourriture d'une espèce de fourmis primitives. Ces deux métaheuristiques sont utilisées pour résoudre le problème de sélection de routages alternatifs dans un système flexible de production. Ce chapitre est consacré à l'adaptation de ces deux méthodes et à la présentation des résultats obtenus, qui sont comparés avec ceux obtenus par l'algorithme génétique, et l'algorithme des colonies de fourmis (ACO) **[Souier, 2009], [Souier et *al*., 2010].**

3.2 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 sue 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 déterminé, 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 :

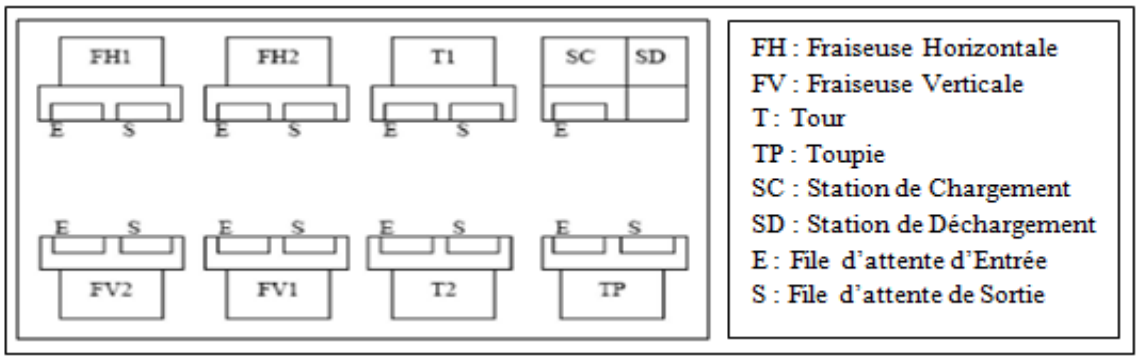


Figure 3.1: Configuration du modèle FMS étudié **[Saygin, 2001].**

Le tableau 3.1 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 pièces	**Taux d'arrivée**	**Routages et temps de traitement (min)**
A	17%	SC-T1(30)-FV1(20)-SD
		SC-T1(30)-FV2(20)-SD
		SC-T2(30)-FV1(20)-SD
		SC-T2(30)-FV2(20)-SD
B	17%	SC-T1(20)-TP(1)-FV1(15)-SD
		SC-T1(20)-TP(1)-FV2(15)-SD
		SC-T2(20)-TP(1)-FV1(15)-SD
		SC-T2(20)-TP(1)-FV2(15)-SD
C	17%	SC-T1(40)-TP(1)-FV1(25)-SD
		SC-T1(40)-TP(1)-FV2(25)-SD
		SC-T2(40)-TP(1)-FV1(25)-SD
		SC-T2(40)-TP(1)-FV2(25)-SD
D	21%	SC-T1(40)-TP(1)-T1(20)-FH1(35)-SD
		SC-T1(40)-TP(1)-T1(20)-FH2(35)-SD
		SC-T1(40)-TP(1)-T2(20)-FH1(35)-SD
		SC-T1(40)-TP(1)-T2(20)-FH2(35)-SD
		SC-T2(40)-TP(1)-T1(20)-FH1(35)-SD
		SC-T2(40)-TP(1)-T1(20)-FH2(35)-SD
		SC-T2(40)-TP(1)-T2(20)-FH1(35)-SD
		SC-T2(40)-TP(1)-T2(20)-FH2(35)-SD
E	20%	SC-T1(25)-TP(1)-T1(35)-FH1(50)-SD
		SC-T1(25)-TP(1)-T1(35)-FH2(50)-SD
		SC-T1(25)-TP(1)-T2(35)-FH1(50)-SD
		SC-T1(25)-TP(1)-T2(35)-FH2(50)-SD
		SC-T2(25)-TP(1)-T1(35)-FH1(50)-SD
		SC-T2(25)-TP(1)-T1(35)-FH2(50)-SD
		SC-T2(25)-TP(1)-T2(35)-FH1(50)-SD
		SC-T2(25)-TP(1)-T2(35)-FH2(50)-SD
F	8%	SC-FH1(40)-SD
		SC-FH2(40)-SD

Tableau 3.1: Routages alternatifs et temps de traitement des pièces **[Saygin, 2001]**.

Les métaheuristiques sont des algorithmes pouvant être appliqués à la résolution d'un grand nombre de problèmes d'optimisation. Elles peuvent être vues comme des méthodologies de niveau supérieur servant à guider la conception d'heuristiques implicitement dédiées à la résolution d'un problème spécifique. Elles sont donc composées d'éléments génériques ou invariants, ainsi que d'éléments spécifiques au problème considéré, tels que la représentation ou l'évaluation d'une solution.

3.3 La fonction objectif :

Toutes les métaheuristiques tentent de trouver une solution visant à minimiser ou maximiser (selon le problème à résoudre) une fonction dite 'fonction objectif'.

Les métaheuristiques sont des algorithmes itératifs, qui tentent d'améliorer une ou plusieurs solutions (une solution dans le cas des méthodes à solution unique, et plusieurs dans le cas des méthodes à population de solutions) au cours des itérations. Il est très difficile de trouver la solution optimale dès la première itération, ce qui justifie l'échec d'adaptation de ces techniques pour la résolution des problèmes complexes en ligne.

Pour pouvoir résoudre notre problème en ligne, nous avons pris en considération à chaque itération des critères concernant l'état du système, à chaque fois qu'une nouvelle pièce entre dans la station de chargement.

Dans notre travail, la fonction objectif est le produit de charges des *n* routages, 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. La maximisation de cette fonction (le produit des temps opératoires) donnera donc un bon équilibre entre les routages. Pour ce faire, nous avons pris en considération les types des premières pièces existantes dans la station 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.

3.4 L'algorithme mimétique :

Cet algorithme est basé sur le même principe des algorithmes génétiques : une population d'individus (solutions) qui évoluent en même temps, chaque solution est représentée sous forme d'un chromosome, on parle alors du codage des solutions. Chaque solution possible du problème est représentée par un chromosome artificiel (généralement sous forme de chaîne). Dans notre cas, un chromosome artificiel représente les routages choisis des pièces.

Avant de présenter le pseudo code de cet algorithme, nous allons tout d'abord expliquer trois points importants pour la compréhension de notre adaptation : le codage d'une solution, le croisement des solutions, et la mutation.

3.4.1 Le codage :

Dans notre cas, une solution possible est un ensemble de routages possibles correspondant aux pièces présentes dans la file infinie.

Le codage consiste à représenter chaque solution possible sous forme de chromosome. Dans notre cas, chaque chromosome représente les routages choisis des pièces présentes dans la file infinie.

Examinons l'exemple suivant :
Initialement, les routages seront choisis d'une façon aléatoire.
n étant la capacité des files d'attente.

Supposons 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.

C	A	F	B

Figure 3.2: Un exemple d'un lot de pièces pour une capacité de file d'attente=4.

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

Le codage est donc une chaine qui comporte l'un des routages possibles pour chaque pièce.

Par exemple la figure 3.3 donne un codage possible.

10	1	30	6

Figure 3.3: Un exemple d'un codage possible pour une capacité de file d'attente=4.

3.4.2 Le croisement :

Le croisement consiste à faire la combinaison de deux parents pour donner naissance à deux enfants qui vont hériter certaines caractéristiques de leurs parents.

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.

Supposons avoir les deux chromosomes suivants :

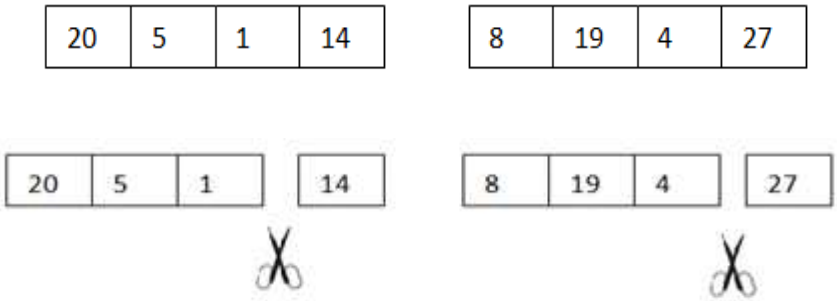


Figure 3.4: Exemple d'un croisement de deux solutions parents.

Choisissant un endroit de coupure le long des deux chaines.
Reliant maintenant les extrémités.

20	5	1	27

8	19	4	14

Figure 3.5: Deux enfants obtenus par croisement des deux parents.

3.4.3 La mutation :

La mutation est une modification qui intervient dans le chromosome, elle a pour rôle de maintenir une certaine diversité dans la population. 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.

n étant la capacité de la file d'attente.
Le pseudo code de l'algorithme est donné comme suit :

Algorithme 3.1: Algorithme mimétique

(1) S'il y'a une place libre dans la station de chargement **alors**
(2) Génération d'une population aléatoire (cela revient dans notre cas à affecter les *n* premières pièces de la file infinie à des routages d'une façon aléatoire en respectant les types des pièces)
(3) Faire la procédure de recherche locale sur chaque individu de la population
(4) Tant que le critère d'arrêt n'est pas atteint faire
(5) Appliquer un opérateur de sélection (c'est-à-dire choisir deux solutions $x\ et\ x'$)
(6) Appliquer l'opérateur de croisement sur les deux parents x et x' pour former deux solutions $y\ et\ y'$
(7) Appliquer la procédure de recherche locale sur $y\ et\ y'$
(8) Appliquer l'opérateur de mutation sur $y\ et\ y'$
(9) Appliquer l'opérateur de remplacement
(10) Fin tant que

3.4.4 La méthode de descente :

Nous avons choisit comme méthode de recherche locale la méthode de descente. C'est une méthode basée sur l'exploration du voisinage d'une solution. 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 mimétique, 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:

Algorithme3.2: 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) : $f(x') > f(x)$ **alors**
(5) Remplacer *x* par *x´*
(6) Finsi
(7) Fin tant que

3.5 L'algorithme API :

n est la capacité de la file d'attente.

L'algorithme API modélise le comportement de l'ensemble de fourmis de l'espèce Pachycondyla apicalis dans l'espace.

Algorithme 3.3: Algorithme API

(1) **Affecter** les *n* premières pièces de la file infinie à des routages d'une façon aléatoire (c'est la construction du nid)
(2) **Tant que** le critère d'arrêt n'est pas vérifié **faire**
(3) **Pour** tout $a_i \in A$ **faire**
(4) API-FOURRAGEMENT
(5) **Fin pour**
(6) **Si** $P_N = P_{max}$ (le nid doit être déplacé) **alors**
(7) $Nid \leftarrow s^+$ /*meilleure solution atteinte*/
(8) **Fin si**
(9) **Fin tant que**

L'algorithme API-FOURRAGEMENT modélise le comportement de chaque fourmi pour pouvoir explorer sa propre zone de recherche, en construisant un ensemble de solutions dites les sites de chasse. L'exploration d'un site consiste à rechercher des solutions améliorantes dans le voisinage de ce site.

Algorithme 3.4: API-FOURRAGEMENT

(1) **Si** $n_s(a_i) < p$ **alors**
(2) $n_s(a_i) \leftarrow n_s(a_i) + 1$
(3) **Exploration** autour du nid$\rightarrow s_{n_s}(a_i) \leftarrow O_{explo}$ (ceci revient à modifier certains routages des pièces de la solution initiale (le nid) aléatoirement selon leurs types)
(4) **Initialiser** le compteur d'échecs : $e_{n_s}(a_i) \leftarrow 0$
(5) **Sinon**
(6) Soit s_j le site que la fourmi a exploré à sa dernière sortie (dernière solution visitée)
(7) **Si** $e_j > 0$ alors
(8) **Choisir** aléatoirement un autre site parmi les sites possibles
(9) **Finsi**
(10) **Exploration** locale autour du site choisi (modifier les routages de certaines pièces de la solution examinée pour trouver une autre solution (site))
(11) **Evaluation** de la fonction objectif (calcul du produit de charges de routages)
(12) **Si** la fonction objectif est améliorée **alors**
(13) **Remplacer** le site exploré par la solution améliorante $s_j \leftarrow s'$
(14) $e_j \leftarrow 0$
(15) **Sinon**
(16) $e_j \leftarrow e_j + 1$
(17) **Si** $e_j > P_{locale}$ **alors**
(18) **Effacer** le site s_j de la mémoire de la fourmi
(19) **Fin si**
(20) **Fin si**
(21) **Fin si**

Une exploration du site consiste à trouver une nouvelle solution dans le voisinage de la première solution, puis à faire une évaluation en calculant la fonction objectif, rappelons qu'un site est une solution possible du problème, l'exploration va donner donc un autre site (une autre solution possible).

Lorsque nous avons adapté l'algorithme tel qu'il est, les résultats obtenus n'étaient pas satisfaisants, pour cela nous avons fait une petite modification.

L'idée est de permettre à l'algorithme de bien explorer le voisinage du nid, c'est-à-dire, bien explorer le voisinage de la solution vers laquelle la population converge.

- **API_m (API modifié) :**

L'algorithme 3.3 reste tel qu'il est, la modification est introduite au niveau de l'algorithme API-FOURRAGEMENT.

Algorithme 3.5: API-FOURRAGEMENT modifié

(1) Si $n_s(a_i) < p$ **alors**
(2) $n_s(a_i) \leftarrow n_s(a_i) + 1$
(3) **Si** $n_s(a_i) \neq 1$ **alors**
(4) **Exploration** autour du nid$\rightarrow s_{n_s}(a_i) \leftarrow O_{explo}$ (ceci revient à modifier certains routages des pièces de la solution initiale (le nid) aléatoirement selon leurs types)
(5) **Initialiser** le compteur d'échecs : $e_{n_s}(a_i) \leftarrow 0$
(6) **Finsi**
(7) Sinon
(8) Soit s_j le site que la fourmi a exploré à sa dernière sortie (dernière solution visitée)
(9) **Si** $e_j > 0$ alors
(10) **Choisir** aléatoirement un autre site parmi les sites possibles
(11) **Finsi**
(12) Exploration locale autour du site choisi (modifier les routages de certaines pièces de la solution examinée pour trouver une autre solution (site))
(13) Evaluation de la fonction objectif (calcul du produit de charges de routages)
(14) **Si** la fonction objectif est améliorée **alors**
(15) **Remplacer** le site exploré par la solution améliorante $s_j \leftarrow s'$
(16) $e_j \leftarrow 0$
(17) **Sinon**
(18) $e_j \leftarrow e_j + 1$
(19) **Si** $e_j > P_{locale}$ **alors**
(20) **Effacer** le site s_j de la mémoire de la fourmi
(21) **Fin si**
(22) **Fin si**
(23) Fin si

Dans cet algorithme, on choisit un emplacement aléatoire du nid, puis, pour chaque fourmi on fait l'exploration autour de celui-ci pour obtenir de nouvelles solutions dans le voisinage (les sites de chasse), ensuite chaque fourmi va explorer l'ensemble de ses propres sites.

La modification que nous l'avons faite est que, pour chaque fourmi, l'exploration se fait autour du nid et autour des sites de chasse, c'est-à-dire que le nid sera considéré comme un site de chasse commun pour toutes les fourmis (ligne 3 dans l'algorithme API-FOURRAGEMENT modifié). Cette simple modification a entrainé des changements majeurs dans les résultats obtenus.

Dans la section suivante, nous allons présenter les résultats obtenus par l'algorithme API_m comparés à ceux de l'algorithme API, pour les mêmes paramètres (le nombre de sites, la patience locale, et la patience globale), suivi par la présentation d'une analyse de sensibilité de l'algorithme API_m.

Afin de montrer les améliorations apportées par les deux métaheuristiques, et les améliorations apportées à l'algorithme API après modification, nous avons fait plusieurs simulations avec des variations sur les critères du système étudié. Les deux critères du système auxquels nous nous sommes intéressés sont la taille de la file d'attente d'entrée et de sortie des stations et le taux de création (d'arrivée) des pièces.

Pour certaines sections telles que la comparaison des deux algorithmes API et API_m l'analyse de sensibilité de l'API_m nous nous sommes intéressés aux cas où le système est en état de saturation (c'est-à-dire une capacité de la file d'attente petite et un taux d'arrivée des pièces élevé).

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**

3.6 Comparaison entre API_m et API :

Nous avons mentionné précédemment que la modification faite dans l'algorithme API a conduit à un changement important des résultats. Dans cette section nous nous intéressons à la comparaison entre l'algorithme API original noté API et celui modifié noté API_m. Les résultats présentés sont obtenus en fixant la taille de la file d'attente (taille file=2) et en variant le taux de création des pièces.

Les paramètres internes de la métaheuristique sont fixés pour les deux algorithmes comme suit :

- La patience locale P_{locale} =1;
- Le nombre des sites de chasse=8 ;
- La patience du nid (dite aussi la patience globale) $P_N = 2 * (P_{locale}) * p$

3.6.1 Le taux de production :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	99.99	99.99	99.99	99.99	88.51	46.70	30.77	15.37
API_m	99.99	99.99	99.99	99.99	97.68	70.74	47.10	23.63

Tableau 3.2: Le taux de production pour une capacité de la file d'attente=2.

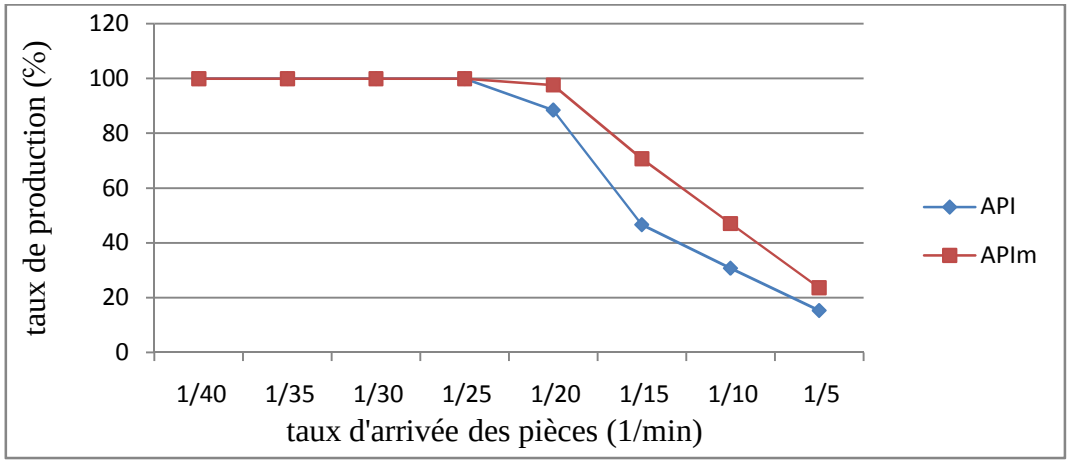


Figure 3.6: Le taux de production pour API et API_m pour une capacité de la file d'attente=2.

Le tableau 3.2 et la figure 3.6 montrent que l'algorithme API_m est plus performant que l'algorithme API original pour des taux de création des pièces supérieurs à 1/25 c'est-à-dire pour des cas saturés; pour des valeurs inférieures, les deux algorithmes donnent des valeurs identiques, donc l'algorithme API_m a pu améliorer le taux de production du système par rapport à l'API.

3.6.2 Le temps du cycle :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	91.24	81.50	90.66	98.68	133.24	165.87	174.31	169.38
API_m	94.29	82.57	90.93	98.95	139.61	168.35	173.93	167.47

Tableau 3.3: Le temps du cycle pour une capacité de la file d'attente=2.

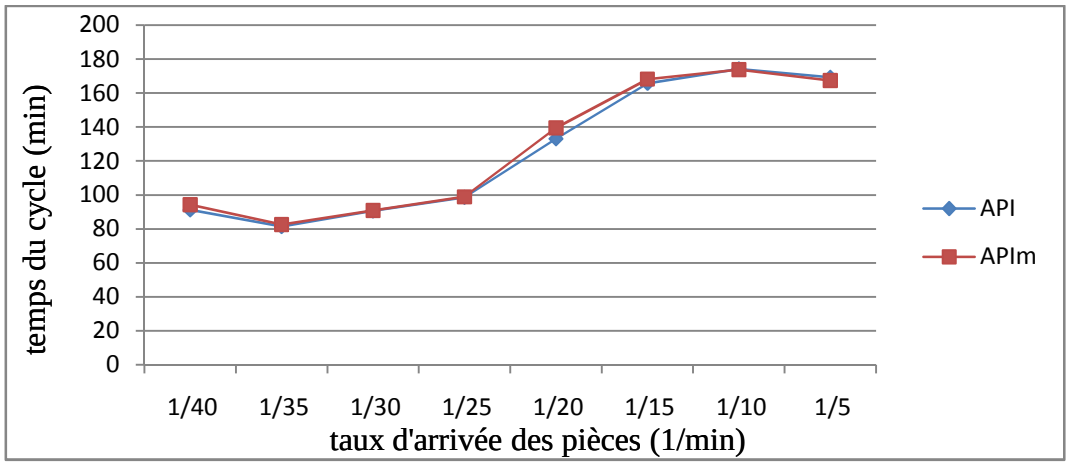


Figure 3.7: Temps du cycle pour API et API_m pour une capacité de la file d'attente=2.

Le tableau 3.3 et la figure 3.7 donnent la variation du temps du cycle des deux algorithmes API et API_m pour une capacité de la file d'attente égale à 2 et pour différentes valeurs du taux de création des pièces. L'algorithme API_m n'a pas pu améliorer le temps du cycle pour certaines valeurs du taux de création des pièces ; tandis que pour certaines valeurs les résultats sont identiques pour les deux algorithmes, ceci revient à l'augmentation du temps du taux de production.

3.6.3 Les en-cours :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	3.89	4.04	4.47	5.25	6.77	6.47	6.48	6.46
API_m	3.89	4.04	4.48	5.23	7.07	8.77	8.78	8.78

Tableau 3.4: Les en-cours pour une capacité de la file d'attente=2.

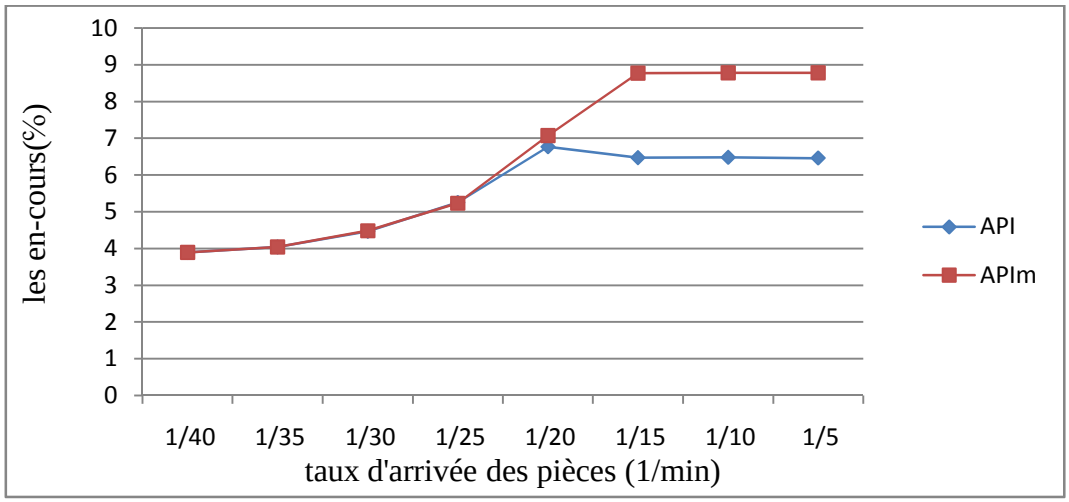


Figure 3.8: Les en-cours pour API et API_m pour une capacité de la file d'attente=2.

Le tableau 3.4 et la figure 3.8 montrent que pour des taux de création des pièces inférieurs ou égales à 1/25 les nombres de pièces présentes dans le système sont presque égaux pour les deux algorithmes, hors de cet intervalle l'algorithme API donne des meilleurs résultats, cela peut être justifié par le fait que le taux de production de l'API_m est supérieur à celui de l'API, i.e: le nombre de pièces traitées dans le système pour l'API_m est supérieur à celui de l'API.

3.6.4 Le taux d'utilisation de l'AGV :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	17.37	19.61	23.16	27.37	30.01	22.12	21.91	21.90
API_m	17.40	19.65	23.01	27.26	32.96	32.36	32.36	32.40

Tableau 3.5: Le taux d'utilisation de l'AGV pour une capacité de la file d'attente=2.

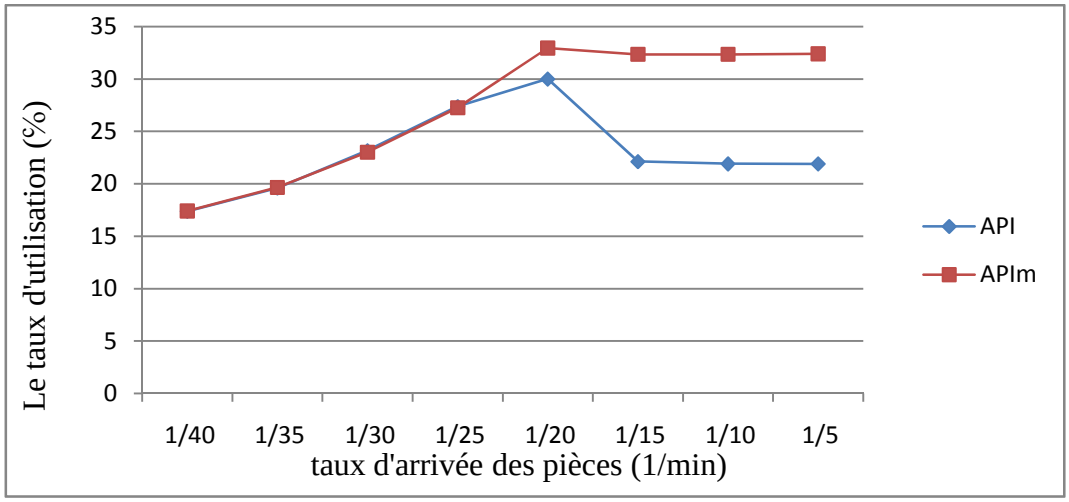


Figure 3.9: Le taux d'utilisation de l'AGV pour API et API_m pour capacité de la file d'attente=2.

Le tableau 3.5 et la figure 3.9 montrent que pour des valeurs de taux de création des pièces inférieures à 1/25 les deux algorithmes donnent presque les mêmes valeurs, et pour des valeurs supérieures l'algorithme API_m montre sa performance, il est clairement remarquable que l'API_m est plus performant dans le cas de saturation.

3.6.5 Le taux d'utilisation des machines :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	48.24	54.65	63.33	76.35	83.91	61.06	60.44	60.41
API_m	48.29	53.85	64.03	76.11	92.31	90.20	90.91	90.82

Tableau 3.6: Le taux d'utilisation des machines T1 et T2 pour une taille de la file=2.

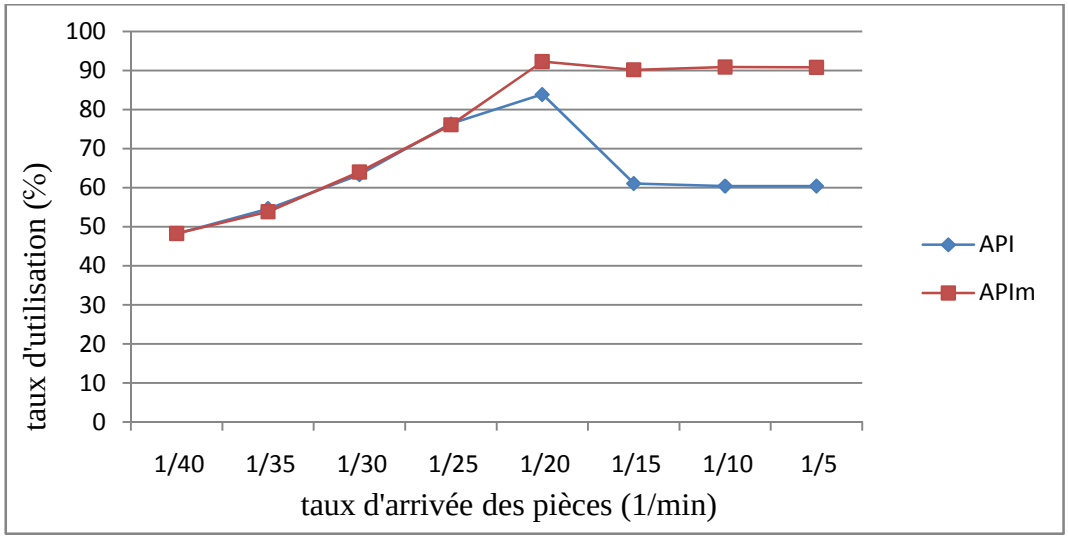


Figure 3.10: Le taux d'utilisation des machines T1 et T2 pour une taille de la file d'attente=2.

Les tableaux 3.6 et la figure 3.10 montrent que pour des valeurs de taux d'arrivée des pièces inférieures ou égales à 1/25 les deux algorithmes donnent des résultats proches. Hors de cet intervalle l'algorithme API_m montre sa performance, en particulier pour les cas saturés.

Concernant le taux d'utilisation des machines FH, FV, et la machine TP, les résultats sont donnés dans l'annexe A.

Finalement, nous pouvons dire que la modification introduite a conduit à des changements remarquables. Cela peut être justifié comme suit :

Le nid sera exploré à chaque itération par chaque fourmi, ce qui permettra donc de bien explorer le voisinage du nid, et puisque le nid sera toujours déplacé vers la meilleure solution atteinte, on aura donc une meilleure exploration de la zone vers laquelle la population converge. En d'autres termes, on permet de rassembler les fourmis autour du meilleur point trouvé, et ainsi de concentrer (intensifier) la recherche dans cette zone.

3.7 Analyse de sensibilité de l'algorithme API_m :

La performance de chaque métaheuristique peut être influencée par le choix ou bien la variation de ses paramètres internes.

Cette section est consacrée à l'étude de l'influence des paramètres internes de la métaheuristique étudiée API_m sur sa performance. Cette étude se fait en variant l'un des paramètres et en fixant les autres.

Les paramètres internes dans l'algorithme API_m sont :

- Le nombre de sites (p): c'est le nombre de solutions que la fourmi explore.
- La patience locale (P_{locale}): qui représente le nombre d'échecs rencontrés par la fourmi sur le même site. C'est-à-dire, le nombre d'échecs rencontrés pendent l'exploration d'une solution.
- La patience globale P_N (dite aussi patience du nid) : c'est en fonction de cette valeur que le nid se déplace, c'est donc un changement de la condition initiale dans l'algorithme, on a déjà mentionné que cette valeur sera fixée en lui donnant une valeur constante ou bien en la calculant par la fonction 3.1 suivante, cela signifie que la patience du nid est soit une constante indépendante des autres paramètres, soit elle est dépendante de la patience locale et du nombre de sites :

  $$P_N = 2 \times (P_{locale} + 1) \times p \qquad 3.1$$

  Pour le reste du document, nous notons :

  $$2 \times (P_{locale} + 1) \times p = f(P_{locale}, p) \qquad 3.2$$

3.7.1 Sensibilité par rapport aux nombre de sites (p) et à la patience locale des fourmis (P_{locale}) :

Dans cette section nous étudions la variation des différents critères de performance en fonction de la patience locale et de nombre de sites de chasse, pour une capacité de la file d'attente égale à deux, et un taux de production des pièces égale à 1/5. La patience globale est fixée comme suit : $P_N=f(P_{locale},p)$. Rappelons que le nombre de sites de chasse est le nombre de solutions possibles construites au voisinage du nid pour chaque fourmi. Dans notre travail la construction est faite d'une façon aléatoire. Le nid étant les routages des premières pièces contenues dans la file infinie (initialement ces routages sont affectés à ces pièces aléatoirement), la construction d'un site de chasse consiste à rechercher des solutions dans le voisinage du nid, ce qui revient à modifier les rouages de certaines pièces contenues dans la file infinie (cette modification se fait aléatoirement).

P_{locale} / p	1	2	4	10	20
1	15.98	15.71	15.46	15.46	15.47
3	23.14	22.86	23.28	22.12	22.07
5	23.13	23.58	23.21	23.15	23.15
8	23.63	23.66	23.54	23.62	23.57

Tableau 3.7: Le taux de production en fonction du nombre de sites et de la patience locale.

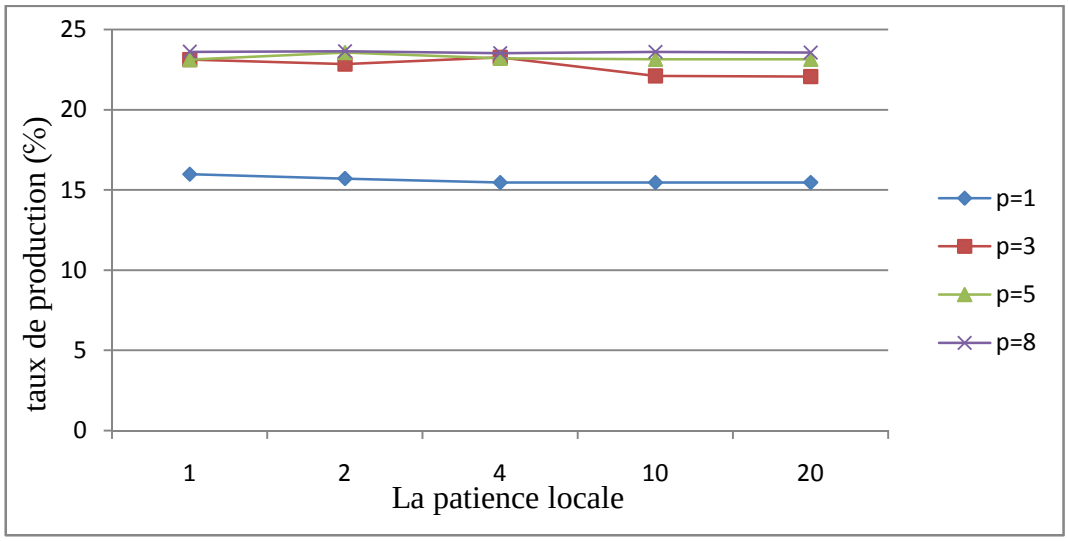


Figure 3.11: Variation du taux de production en fonction de la patience locale P_{locale} pour différentes valeurs de nombre des sites.

Pour un seul site, les valeurs du taux de production sont petites, car la recherche dans le voisinage d'une seule solution va conduire à des minima locaux. Les résultats obtenus pour un nombre de sites égale à 8 sont meilleurs que ceux obtenus par des nombres de sites égales à 3 et 5, d'autre part la variation du taux de production pour un nombre de sites égale à 8 est uniforme en fonction des valeurs de la patience locale P_{locale}.

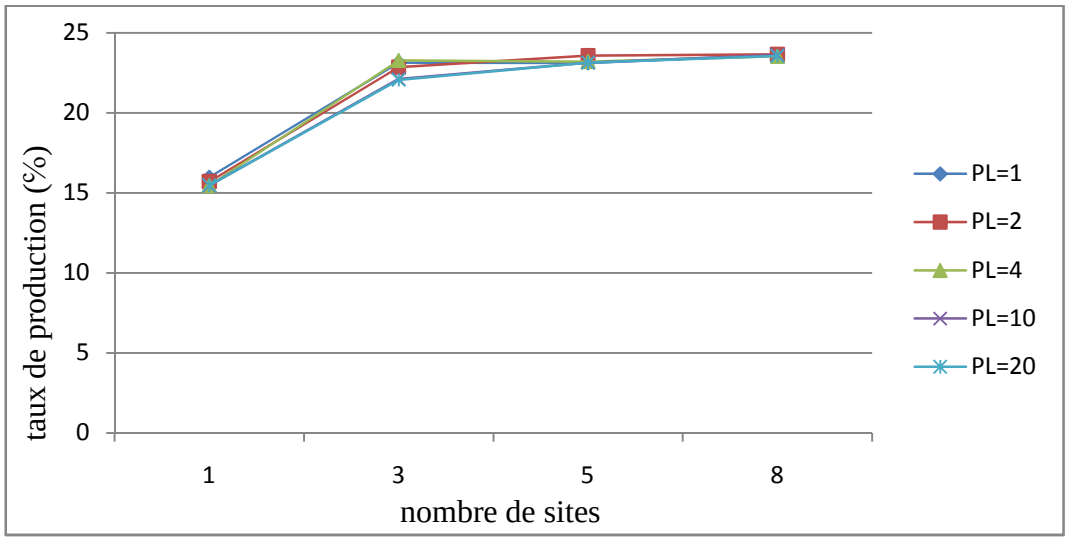


Figure 3.12: Variation du taux de production en fonction de nombre des sites pour différentes valeurs de la patience locale P_{locale}.

La figure 3.12 montre que les valeurs du taux de production sont peu sensibles à la variation de la patience locale, mais on remarque que pour des valeurs petites de la patience locale (1 et 2) le taux de production est un peu plus élevé.

P_{locale} / p	1	2	4	10	20
1	169.88	172.95	173.57	175.67	170.39
3	169.66	167.01	171.75	165.32	169.72
5	169.19	168.20	168.97	171.87	170.24
8	167.47	168.06	165.75	166.03	170.42

Tableau 3.8: Le temps du cycle en fonction de la patience locale P_{locale} et de nombre des sites.

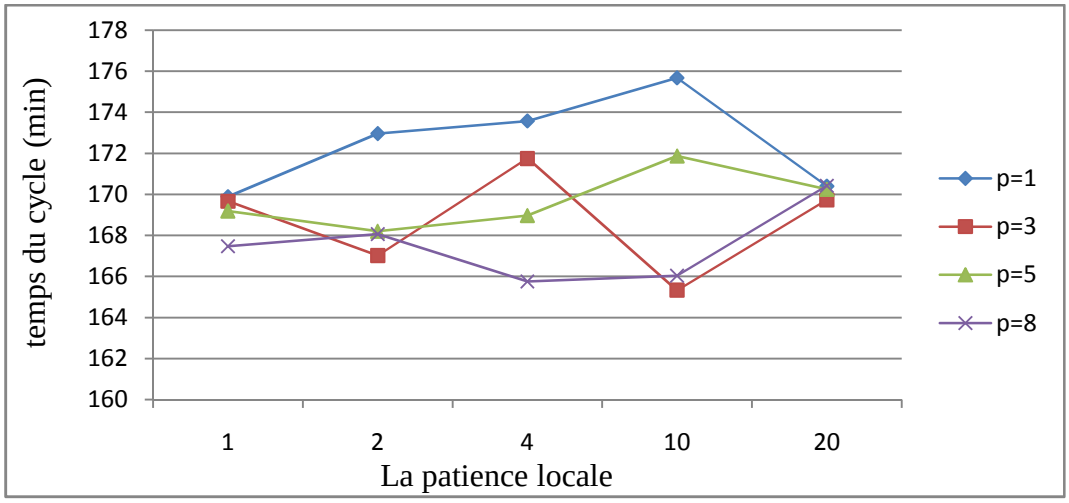


Figure 3.13: Variation du temps du cycle en fonction de la patience locale P_{locale} pour différentes valeurs de nombre des sites.

Le tableau 3.8 et la figure 3.13 montrent que les valeurs du temps de cycle sont meilleures pour des nombres de sites élevés. Pour huit sites de chasse les valeurs obtenues sont les meilleures.

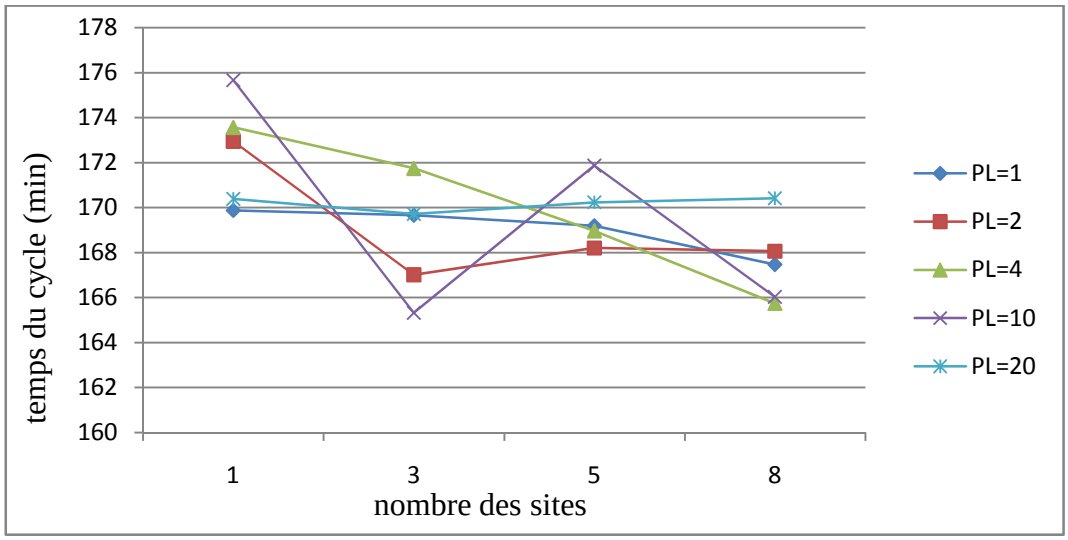


Figure 3.14: Variation du temps de cycle en fonction de nombre des sites de chasse pour différentes valeurs de la patience locale Plocale.

La figure 3.14 montre que pour une patience locale P_{locale}=2, le temps du cycle obtenu est meilleur, sauf pour un nombre de sites égale à un, ce qui est évident, car le nombre de sites représente les possibilités de solutions examinées, donc il est clair que pour une seule solution on va obtenir des solutions correspondants à un optimum locale.

p \ P_{locale}	1	2	4	10	20
1	6.60	6.55	6.48	6.47	6.47
3	8.61	8.45	8.67	8.34	8.32
5	8.64	8.75	8.66	8.65	8.64
8	8.78	8.80	8.78	8.78	8.79

Tableau 3.9: Variation des en-cours en fonction de la patience locale P_{locale} et de nombre des sites.

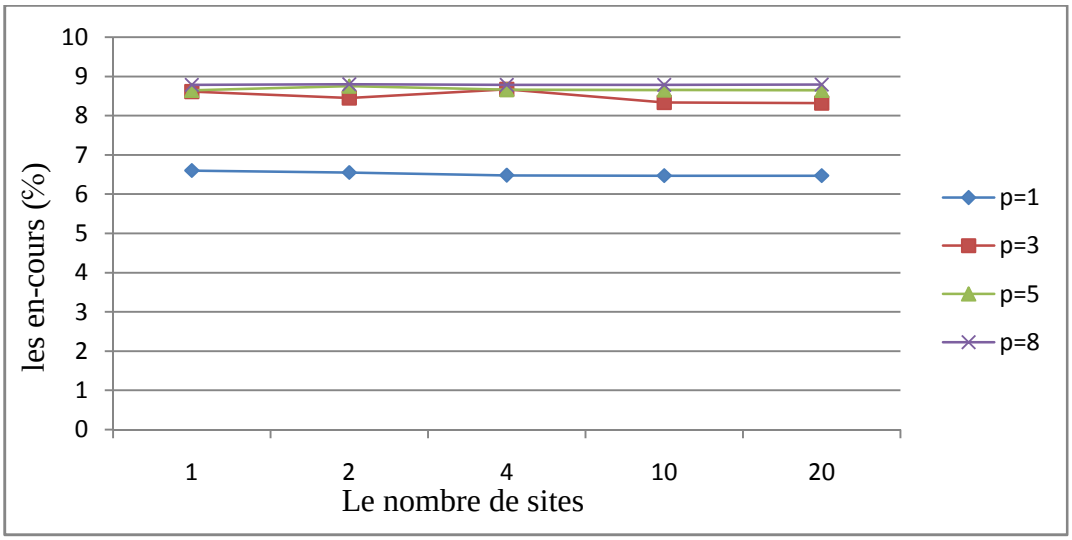


Figure 3.15: Variation des en-cours en fonction de la patience locale P_{locale} pour différentes valeurs de nombre des sites.

Le tableau 3.9 et la figure 3.15 montrent la variation des en-cours en fonction de la patience locale pour différents nombres de sites de chasse, les valeurs obtenus des en-cours pour un seul site de chasse sont plus petites, cela peut être justifié par le fait que le taux de production pour ce cas est petit, donc il est évident que le nombre de pièces restantes dans le système est moins. D'autre part, pour des nombre de sites égales à 3, 5, et 8 sites, les résultats obtenus des en-cours sont voisines, les plus grande valeurs sont obtenus pour un nombre de sites égale à 8, cela peut être justifié par l'augmentation du taux de production pour ce nombre de sites.

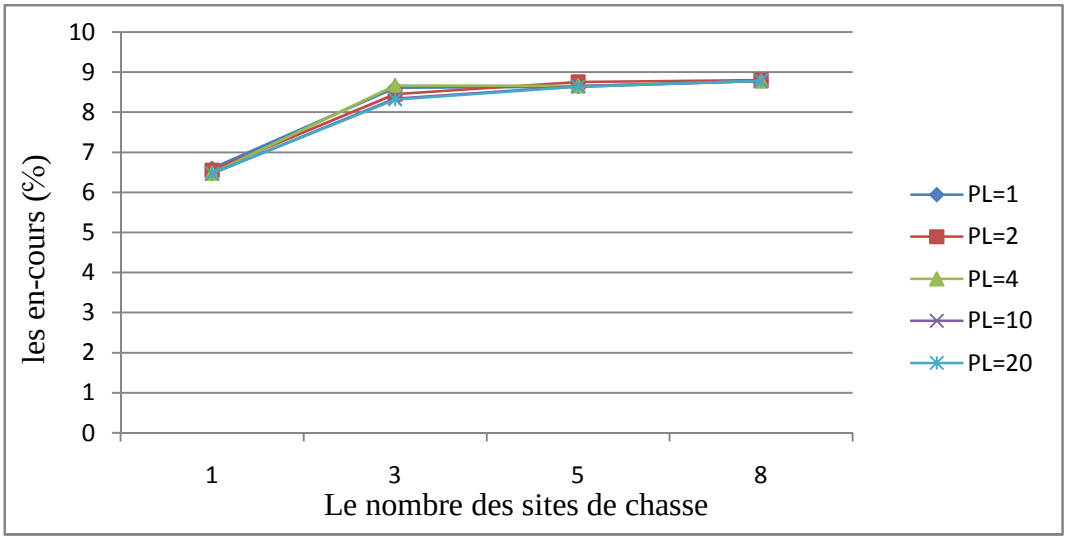


Figure 3.16: Variation des en-cours en fonction de nombre des sites de chasse pour différentes valeurs de la patience locale P_{locale}.

La figure 3.16 présente la variation des en-cours en fonction du nombre de sites pour différentes valeurs de la patience locale. On remarque que la le taux de production n'est pas très influencé par la variation de la patience locale, car les résultats trouvées sont voisines.

En conclusion, on peut dire que les meilleurs résultats sont, en général, obtenus pour un nombre de site de chasse égale à 8. Avoir un nombre important de sites de chasse permet à chaque fourmi d'explorer une grande zone de l'espace, c'est à dire que le nombre de solutions examinées augmente, ce qui donne une possibilité plus importante de trouver des solutions plus améliorantes. Dans ce travail nous nous sommes limités à un nombre de sites égale à 8, comme perspectives, nous avons proposé en conclusion d'augmenter le nombre de site de chasses.

Concernant la patience locale, les meilleurs résultats sont obtenus pour des valeurs de la patience locale petites (1 et 2).

Dans la section suivante, nous allons nous intéressé à l'étude de sensibilité de l'algorithme API_m en fonction de la Patience globale. Ceci peut être fait en fixant les autres paramètres et en variant la patience globale.

3.7.2 Sensibilité par rapport à P_N :

La patience globale, dite aussi patience du nid notée P_N, représente le paramètre en fonction duquel le nid change de localisation, en optimisation cela revient à une réinitialisation de la condition initiale de l'algorithme, le nid se déplace toujours sur le meilleur site de chasse, c'est-à-dire sur la meilleure solution obtenue, et les sites sont ensuite générés à partir de cette position. La patience globale P_N est soit une constante indépendante des autres paramètres, soit elle se calcule en fonction de la patience locale et du nombre de sites par la formule 3.1, le but étant de laisser suffisamment d'itérations entre chaque déplacement du nid.

Pour ce faire, nous avons fixé le nombre de sites p à 8, et la patience locale P_{locale} à 2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
P_N=1	99.99	99.99	99.99	99.99	98.70	70.95	47.30	23.70
P_N=2	99.99	99.99	99.99	99.99	99.43	70.89	47.01	23.52
P_N=3	99.99	99.99	99.99	99.99	98.18	70.80	47.12	23.64
P_N=4	99.99	99.99	99.99	99.99	99.23	70.92	47.13	23.50
P_N=10	99.99	99.99	99.99	99.99	99.99	71.97	47.96	23.72
P_N=20	99.99	99.99	99.99	99.99	97.72	70.78	47.10	23.49
P_N=30	99.99	99.99	99.99	99.99	99.39	70.24	47.00	23.79
P_N=40	99.99	99.99	99.99	99.99	98.99	70.98	47.09	23.64
P_N=f(P_{locale}, p)	99.99	99.99	99.99	99.99	98.54	70.31	46.70	23.56

Tableau 3.10: Le taux de production en fonction de la patience globale P_N pour une taille file=2 et un taux d'arrivée des pièces=1/5

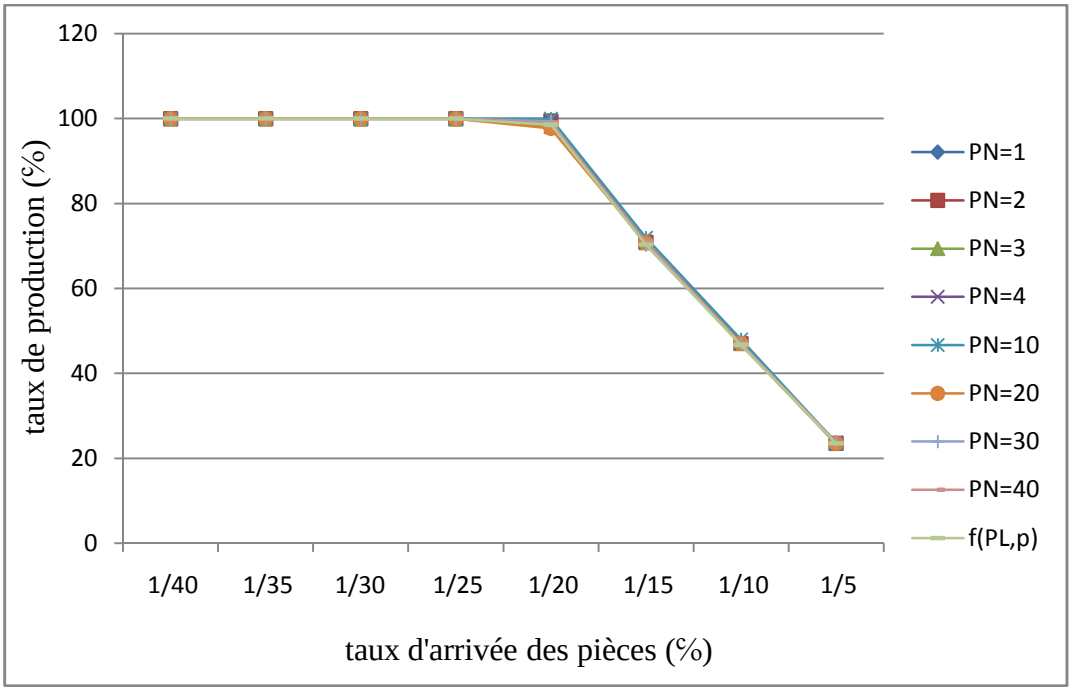


Figure 3.17: Variation du taux de production en fonction de la patience globale.

Le tableau 3.10 et la figure 3.17 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 patience globale, hors de cet intervalle les valeurs des taux de production sont proches pour les différentes valeurs de la patience globale, mais il est clair que pour une patience globale égale à 10 les résultats obtenus sont les meilleurs sauf pour un taux de création des pièces égale à 1/5 où les résultats sont meilleurs pour une patience globale égale à 30.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
P_N=1	92.00	81.21	89.00	94.21	131.35	166.20	169.97	170.59
P_N=2	90.30	83.86	90.47	95.81	123.20	167.62	171.29	169.02
P_N=3	94.93	82.32	89.71	95.74	141.50	169.25	168.17	170.33
P_N=4	92.89	83.76	89.31	99.00	128.66	168.76	170.71	168.80
P_N=10	93.27	82.08	87.50	95.92	116.78	169.78	169.31	170.69
P_N=20	92.10	83.00	86.87	98.61	136.65	170.77	171.13	169.07
P_N=30	91.55	84.38	88.12	96.31	124.01	170.75	170.20	169.87
P_N=40	94.47	80.89	92.42	99.00	130.06	168.38	168.85	167.73
P_N=f(P_{locale}, p)	91.34	82.99	89.87	99.96	133.05	170.21	171.29	170.35

Tableau 3.11: Le temps de cycle pour différentes valeurs de la patience globale P_N pour une taille file=2.

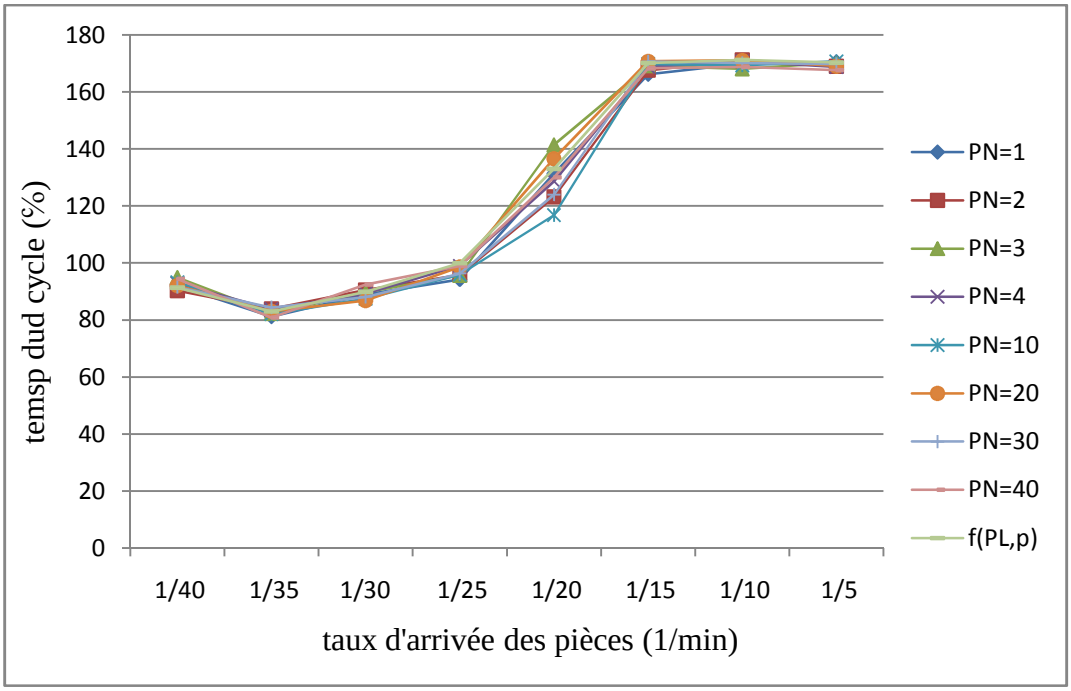


Figure 3.18: Le temps du cycle pour différentes valeurs de la patience globale P_N pour une taille file=2.

Le tableau 3.11 et la figure 3.18 montrent que les valeurs du temps du cycle sont voisines pour toutes les valeurs de P_N. Pour un taux de création des pièces égale à 1/20 le meilleur temps de cycle est obtenu pour une patience globale P_N égale à10, pour les autres valeurs du taux de création aucune valeur de P_N ne donne de meilleurs résultats pour tous les cas.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
P_N=1	3.91	4.03	4.45	5.00	7.50	8.78	8.78	8.80
P_N=2	3.88	4.04	4.48	5.25	7.17	8.79	8.76	8.77
P_N=3	3.91	4.03	4.49	5.26	7.72	8.79	8.77	8.78
P_N=4	3.89	4.04	4.48	5.27	7.36	8.77	8.77	8.77
P_N=10	3.90	4.04	4.45	5.21	7.01	8.84	8.85	8.81
P_N=20	3.91	4.03	4.47	5.27	7.71	8.78	8.75	8.76
P_N=30	3.90	4.05	4.47	5.25	7.16	8.76	8.76	8.81
P_N=40	3.92	4.02	4.44	6.0	7.42	8.78	8.78	8.78
P_N=f(P_{locale}, p)	3.91	4.03	4.48	5.27	7.51	8.74	8.74	8.77

Tableau 3.12: Les en-cours pour différentes valeurs de P_N pour une taille file=2.

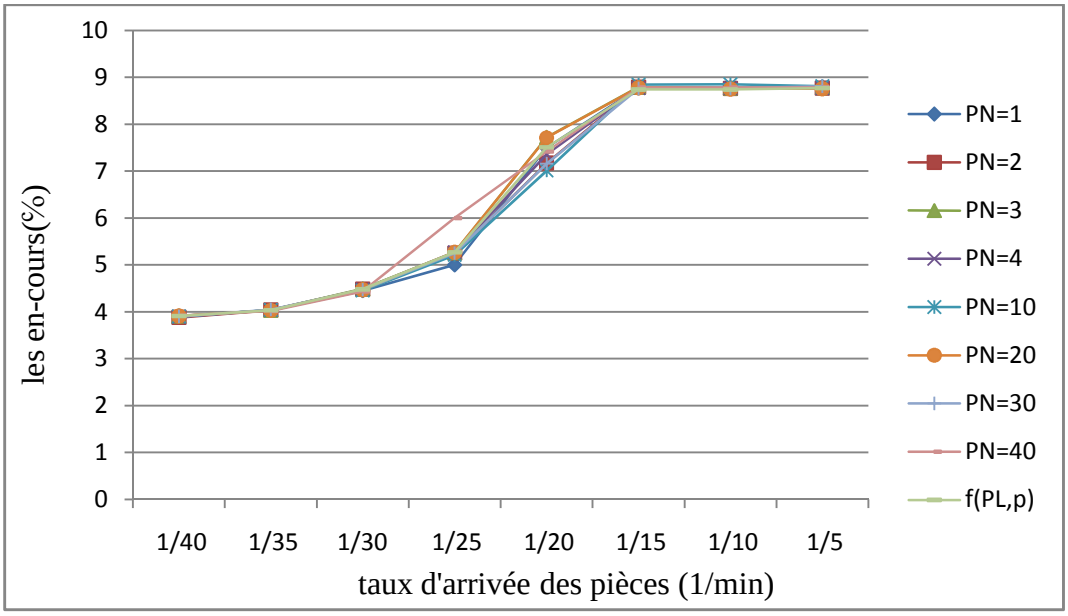


Figure 3.19: Les en-cours pour différentes valeurs de la patience globale P_N pour une taille file=2.

Le tableau 3.12 et la figure 3.19 montrent que les valeurs des en-cours sont proches l'une de l'autre pour les mêmes taux de création des pièces. On remarque que pour certaines valeurs de P_N les résultats obtenus sont meilleurs, mais aucune valeur de P_N ne donne de bons résultats pour toutes les valeurs des taux de création.

En conclusion, on a remarqué que la variation de la patience globale n'influx pas beaucoup sur les performances de l'algorithme API_m, mais pour le taux de production, les meilleurs résultats sont obtenus pour une patience globale P_N égale à 10.

3.8 Résultats et interprétations :

Dans cette section nous présentons les résultats obtenus par l'algorithme mimétique et par l'algorithme API_m. Les résultats de l'algorithme API_m sont obtenus pour un nombre de sites p égale à 8, une patience locale P_{locale} égale à 10, et une patience globale P_N égale à 10. Ces résultats sont comparés avec ceux obtenus par l'algorithme génétique et l'algorithme ACO basé sur le comportement des fourmis conçu par Dorigo **[Dorigo et *al.*, 1996]** et adapté par Souier (2009), et Souier et *al.*, (2010) pour la résolution du problème de sélection de routages alternatifs dans un

FMS. Dans l'annexe nous présentons une comparaison entre l'algorithme mimétique, l'algorithme API_m, et les six métaheuristiques (le recuit simulé, les essaims particulaires, la recherche tabou, l'électromagnétisme, l'algorithme génétique, et les colonies de fourmis) adaptées par Souier (2009) et Souier et *al*., (2010) pour la résolution de ce même problème.

3.8.1 Taux de production :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	99.99	99.99	99.99	99.99	99.99	73.68	49.21	24.62
GA	99.99	99.99	99.99	99.99	98.44	71.08	47.25	23.70
API_m	99.99	99.99	99.99	99.99	99.99	71.97	47.96	23.72
ACO	99.99	99.99	99.99	99.99	90.52	64.54	43.20	21.51

Tableau 3.13: Le taux de production pour une taille de file=2.

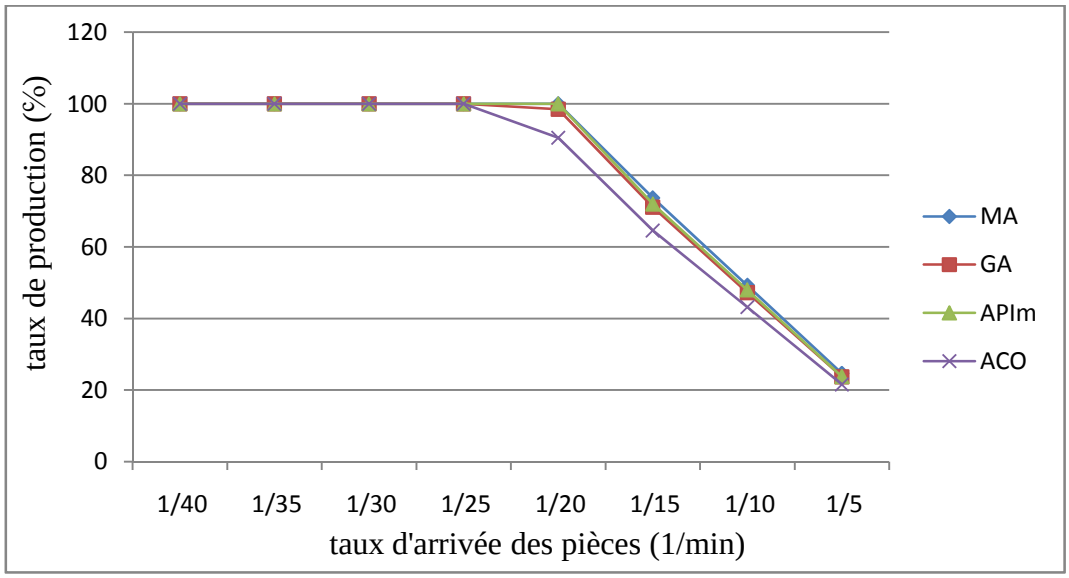


Figure 3.20: Le taux de production pour une taille de file=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	99.99	99.99	99.99	99.99	99.99	74.17	49.56	24.84
GA	99.99	99.99	99.99	99.99	99.99	72.41	48.26	24.17
API_m	99.99	99.99	99.99	99.99	99.99	71.20	47.52	23.71
ACO	99.99	99.99	99.99	99.99	99.99	68.97	45.88	22.98

Tableau 3.14: Taux de production pour une taille file=6.

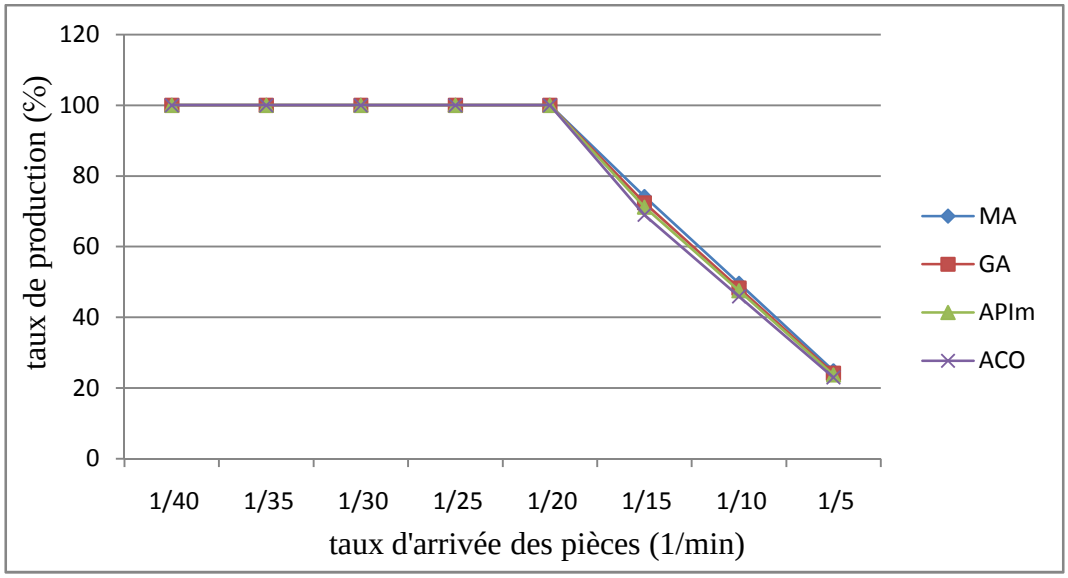


Figure 3.21: Le taux de production pour une capacité de file d'attente =6.

Les tableaux 3.13, 3.14 et les figures 3.20, 3.21 montrent que pour des taux de création de pièces inférieurs ou égales à 1/25 toutes les valeurs des taux de production sont identiques, hors de cet intervalle l'algorithme mimétique domine les autres métaheuristiques. L'algorithme API_m montre aussi sa performance par rapport à l'algorithme ACO pour une capacité de file d'attente égale à 2, mais il est moins performant pour une capacité de file d'attente égale à 6.

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	24.59	24.72	24.84	24.82
GA	23.70	23.93	24.17	24.24
API_m	23.72	23.70	23.80	23.88
ACO	21.51	22.85	22.98	23.13

Tableau 3.15: Le taux de production pour un taux de création des pièces=1/5.

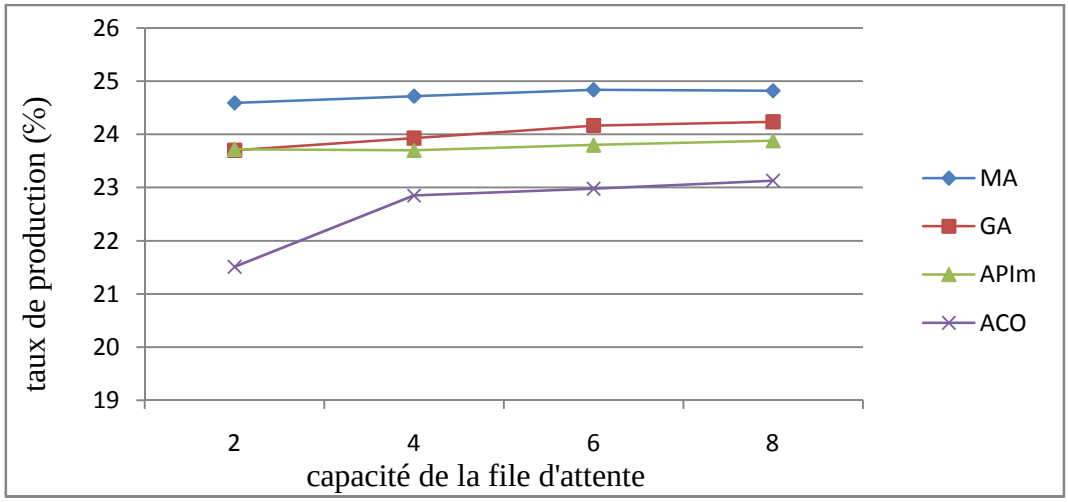


Figure 3.22: Le taux de production pour un taux de création des pièces=1/5.

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	99.99	99.99	99.99	99.99
GA	98.44	99.99	99.99	99.99
API_m	99.99	99.99	99.99	99.99
ACO	90.52	99.99	99.99	99.99

Tableau 3.16: Le taux de production pour un taux de création des pièces=1/20.

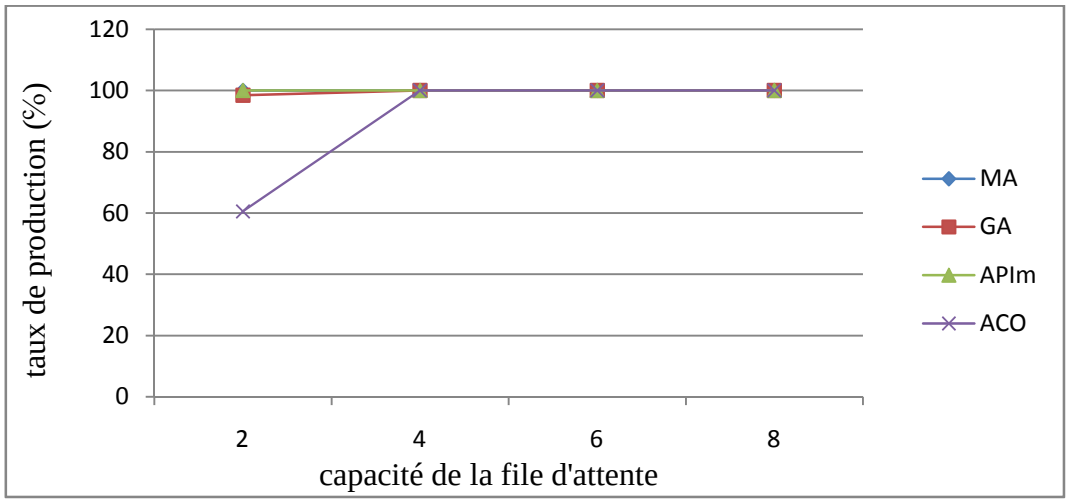


Figure 3.23: Le taux de production pour un taux de création des pièces=1/20

Dans les tableaux 3.15, 3.16 et les figures 3.22, 3.23 on a fixé le taux d'arrivée des pièces à 1/5 puis à 1/20, et nous avons varié la capacité de la file d'attente. Pour un taux d'arrivée des pièces égale à 1/5, l'algorithme mimétique domine les autres métaheuristiques, l'algorithme API_m donne de bons résultats comparé à l'algorithme ACO et même à l'algorithme génétique.

Pour un taux d'arrivée des pièces égale à 1/20, les résultats obtenus par les quatre 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 mimétique et API_m donnent les meilleurs résultats.

3.8.2 Le temps du cycle :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	94.16	83.01	91.02	95.97	124.60	168.13	168.46	171.13
GA	89.60	81.80	88.60	98.70	132.70	169.80	168.90	168.10
API_m	93.27	82.08	87.50	95.92	116.78	169.78	169.31	170.69
ACO	90.90	83.70	89.00	98.80	155.20	166.60	165.40	163.50

Tableau 3.17: Le temps de cycle pour une taille file=2.

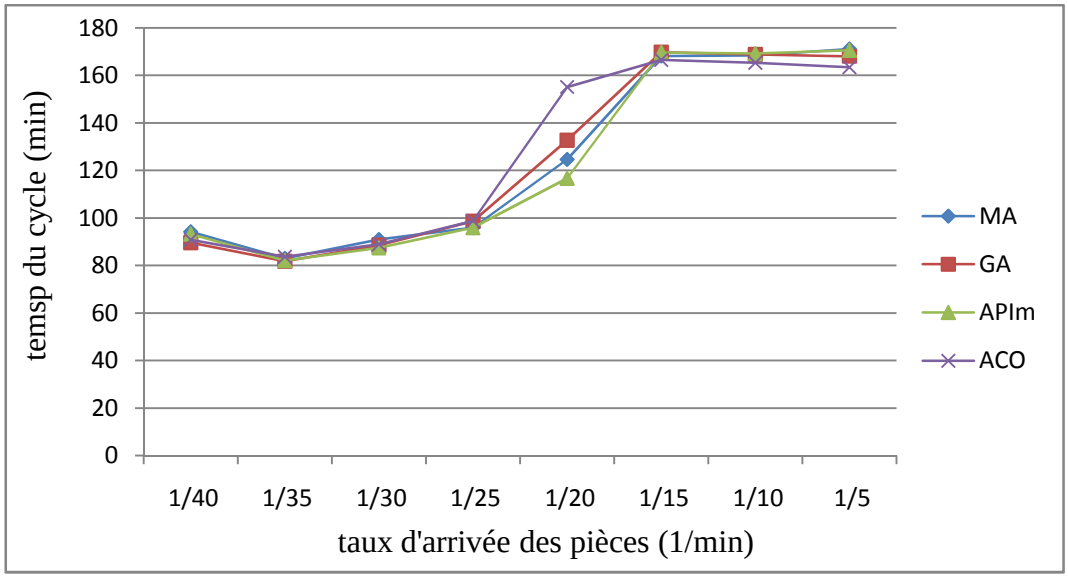


Figure 3.24: Le temps de cycle pour une taille file=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	92.40	84.06	88.84	97.43	123.53	323.50	320.26	327.75
GA	93.10	83.40	91.60	98.90	121.30	331.70	332.80	338.10
API_m	92.48	83.42	87.54	96.61	124.46	327.08	325.66	330.46
ACO	91.10	82.60	89.30	97.00	121.40	326.00	324.00	320.60

Tableau 3.18: Temps du cycle pour une taille file=6.

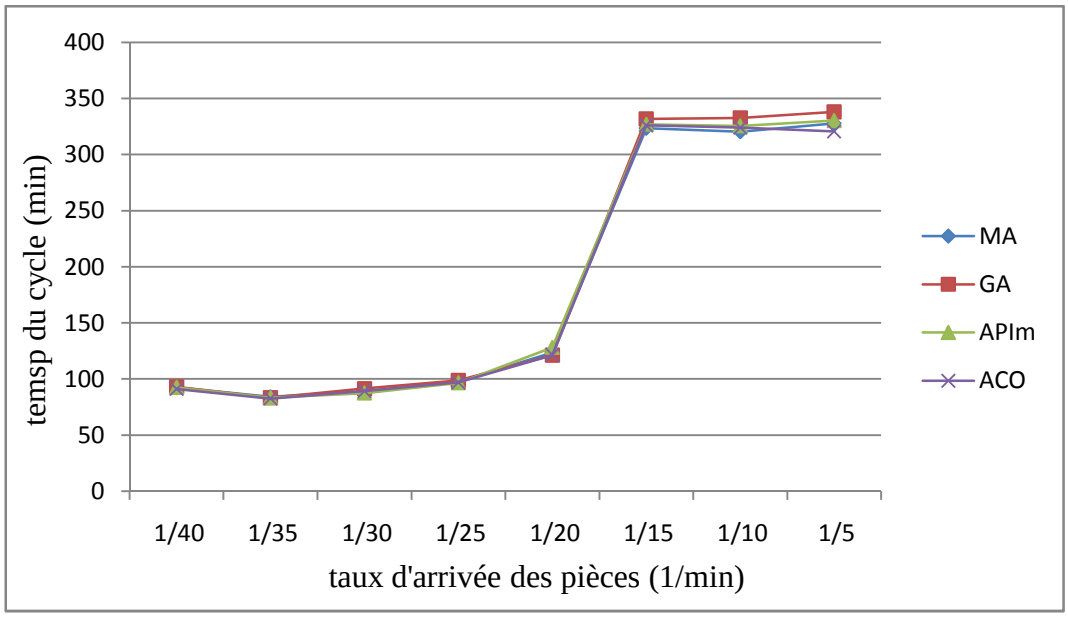


Figure 3.25: Le temps du cycle pour une capacité de file d'attente=6.

Les tableaux 3.17, 3.18 et les figures 3.24, 3.25 montrent que les valeurs du temps de cycle obtenus sont proches pour toutes les métaheuristiques, mais il n'y a pas une métaheuristique qui donne des meilleurs résultats pour tous les taux.

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	174.37	248.60	327.75	410.65
GA	168.10	249.90	338.10	417.00
API_m	170.69	246.77	329.68	406.21
ACO	163.50	247.90	320.60	382.40

Tableau 3.19: Le temps de cycle pour un taux de création des pièces=1/5.

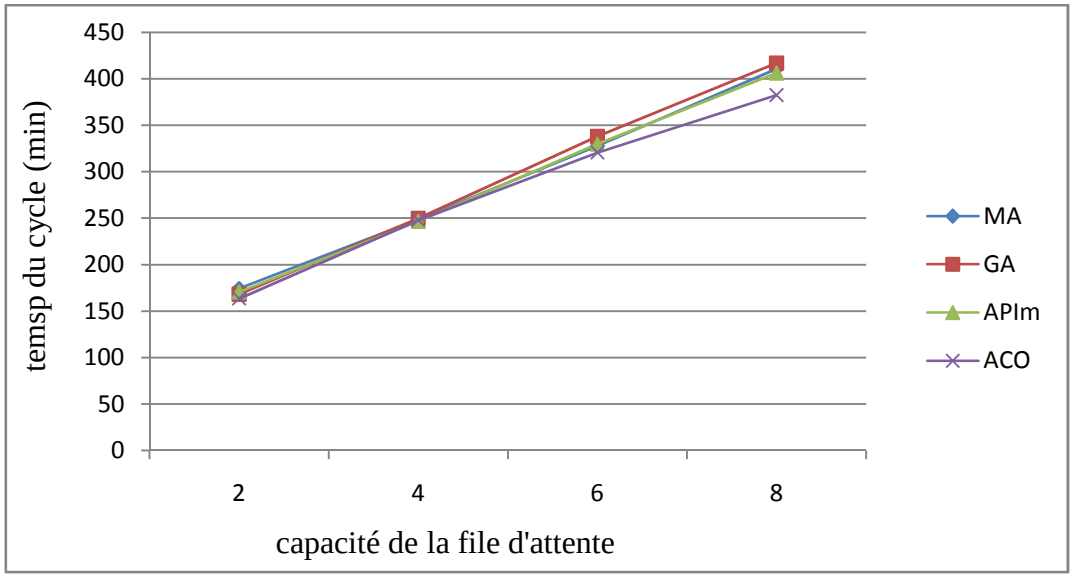


Figure 3.26: Le temps de cycle pour un taux de création des pièces=1/5.

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	124.60	124.41	125.97	128.87
GA	132.70	118.00	121.30	119.70
API_m	116.78	130.70	124.46	124.22
ACO	155.20	130.00	121.40	119.80

Tableau 3.20: Le temps de cycle pour un taux de création des pièces=1/20.

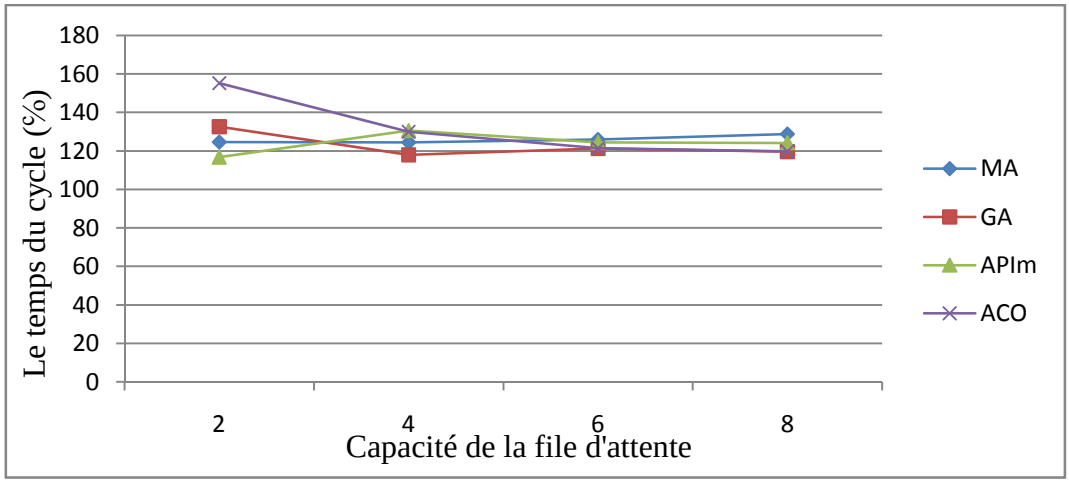


Figure 3.27: Le temps de cycle pour un taux de création des pièces=1/20.

D'après les tableaux 3.19, 3.20 et les figures 3.26, 3.27 le temps les meilleurs résultats sont en général obtenus par l'algorithme ACO et l'algorithme génétique, mais pour certaines valeurs,

les résultats des quatre métaheuristiques restent proches, ceci revient à l'augmentation du taux de production pour l'algorithme mimétique et l'algorithme API_m.

3.8.3 Les en-cours :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	3.91	4.03	4.48	5.22	7.20	9.00	9.00	9.25
GA	3.90	4.04	4.46	5.23	7.45	8.90	8.85	8.85
API_m	3.90	4.04	4.45	5.21	7.01	8.84	8.85	8.81
ACO	3.90	4.04	4.45	5.21	7.29	8.40	8.44	8.46

Tableau 3.21: Les en-cours pour une taille file=2.

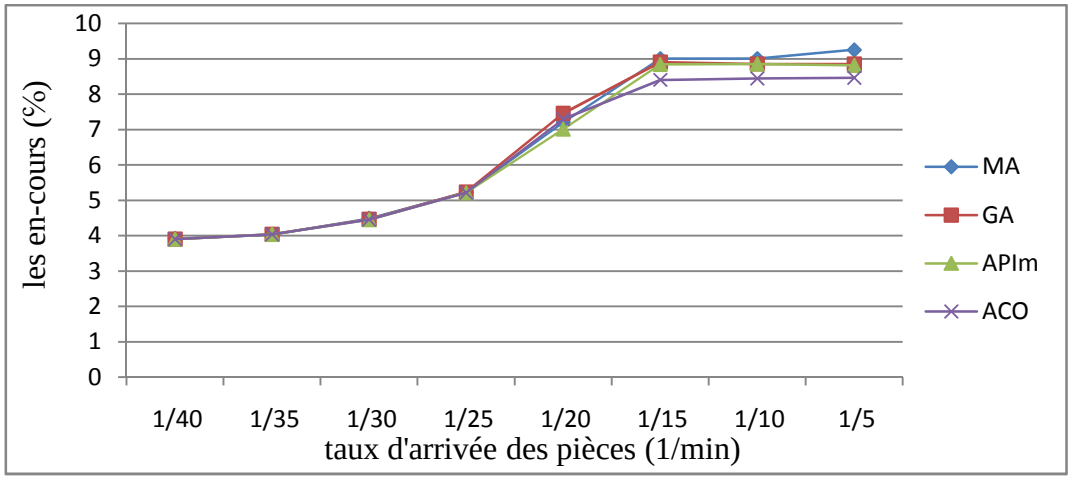


Figure 3.28: Les en-cours pour une taille de la file=2

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	7.88	8.03	8.46	9.26	11.15	23.00	20.00	21.00
GA	7.89	8.04	8.46	9.26	11.26	19.68	19.72	19.73
API_m	7.90	8.03	8.44	9.24	11.41	19.14	19.18	19.13
ACO	7.89	8.02	8.45	9.20	11.31	18.46	18.56	18.56

Tableau 3.22: Les en-cours pour une taille file=6.

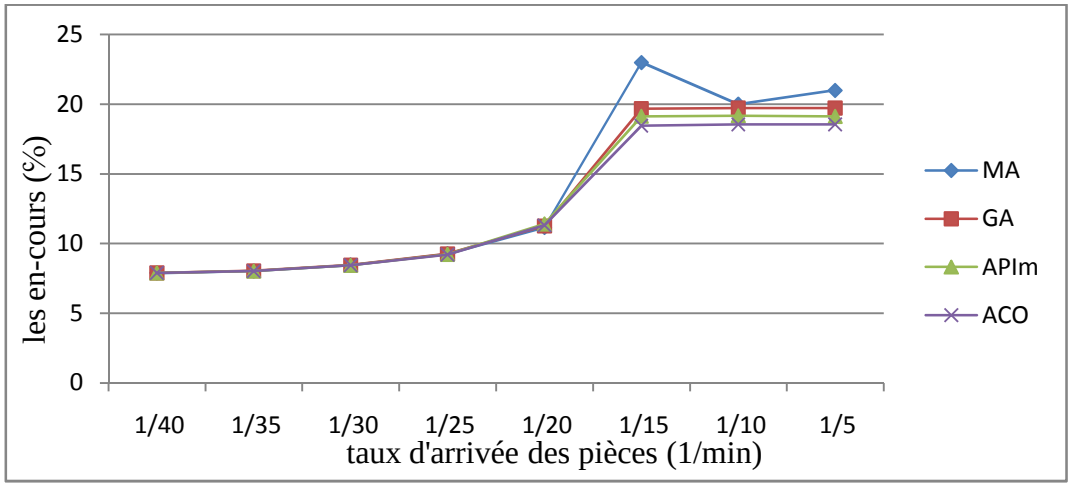


Figure 3.29: Les en-cours pour une capacité de fila d'attente=6.

Les tableaux 3.21, 3.22 et les figures 3.28, 3.29 montrent que les résultats obtenus sont proches, mais aucune des métaheuristiques ne donne de bons résultats pour tous les taux, sauf pour une capacité de file d'attente égale à 6, où l'ACO est la plus performante.

3.8.4 Le taux d'utilisation de l'AGV :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	17.48	19.59	23.13	27.28	33.25	32.84	32.90	32.88
GA	17.43	19.54	23.16	27.04	33.07	32.69	32.67	32.68
API_m	17.51	19.64	22.91	27.25	33.14	32.52	32.51	32.41
ACO	17.47	19.58	22.89	27.36	30.64	29.37	29.46	29.41

Tableau 3.23: Le taux d'utilisation de l'AGV pour une taille file=2.

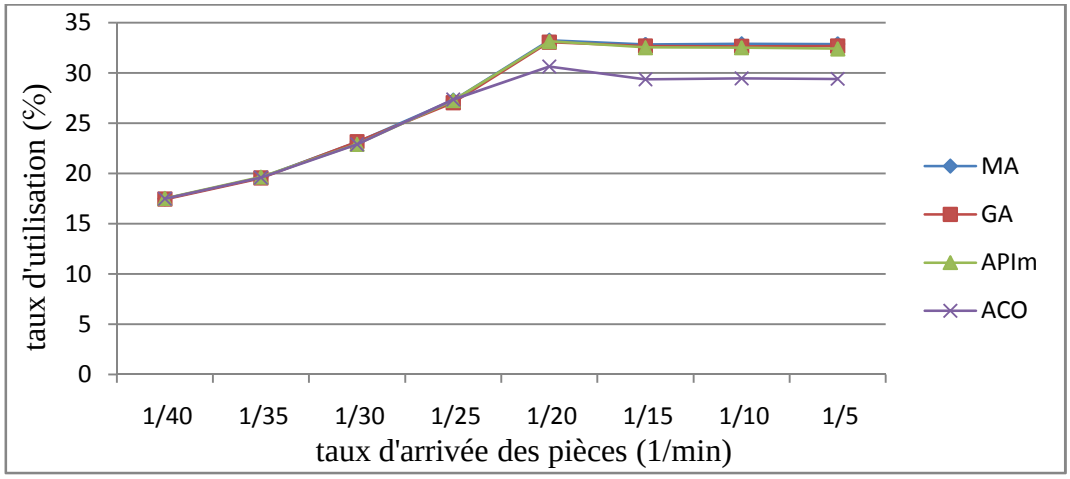


Figure 3.30: Le taux d'utilisation de l'AGV pour une taille file=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	17.35	19.52	22.96	27.28	33.42	33.30	33.35	33.34
GA	17.43	19.70	23.11	27.47	33.16	33.62	33.59	33.56
API_m	17.48	19.44	22.91	27.31	33.17	32.88	32.90	32.85
ACO	17.52	19.67	23.05	27.39	33.26	32.37	32.29	32.33

Tableau 3.24: Taux d'utilisation de l'AGV pour une taille file=6.

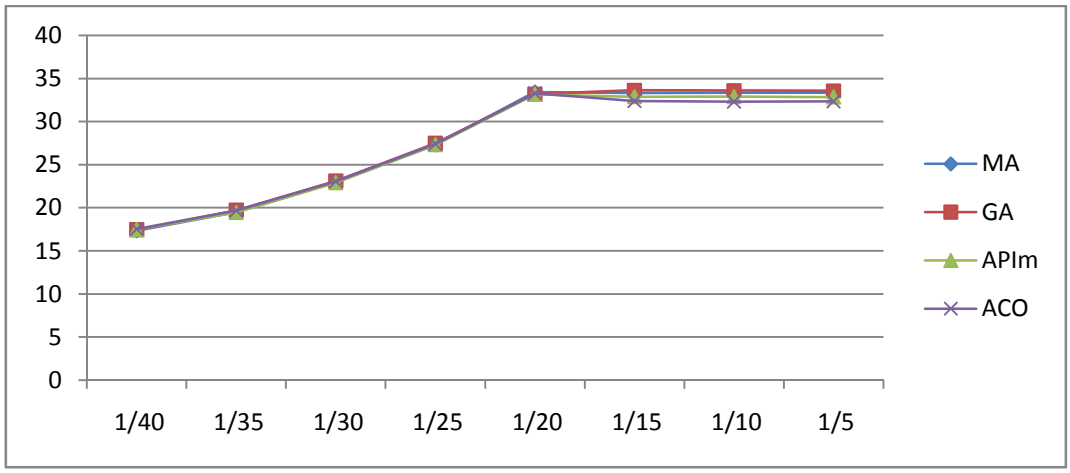


Figure 3.31: Le taux d'utilisation de l'AGV pour une capacité de file d'attente=6.

A partir des tableaux 3.23, 3.24 et des figures 3.30, 3.31 on peut remarquer que pour un système saturé l'algorithme mimétique donne les meilleurs résultats. Hors de cet intervalle, les valeurs des en-cours obtenues sont, en général, proches pour toutes les méthodes.

3.8.5 Le taux d'utilisation des machines :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	48.47	54.62	64.26	76.15	93.51	92.22	92.39	92.66
GA	48.36	54.51	64.32	75.67	92.72	90.98	90.85	90.96
API_m	48.52	54.71	63.82	76.11	93.28	90.95	90.93	90.45
ACO	48.44	54.59	63.99	76.31	85.73	81.97	82.26	82.06

Tableau 3.25: Le taux d'utilisation des machines T1 et T2 pour une taille de la file=2.

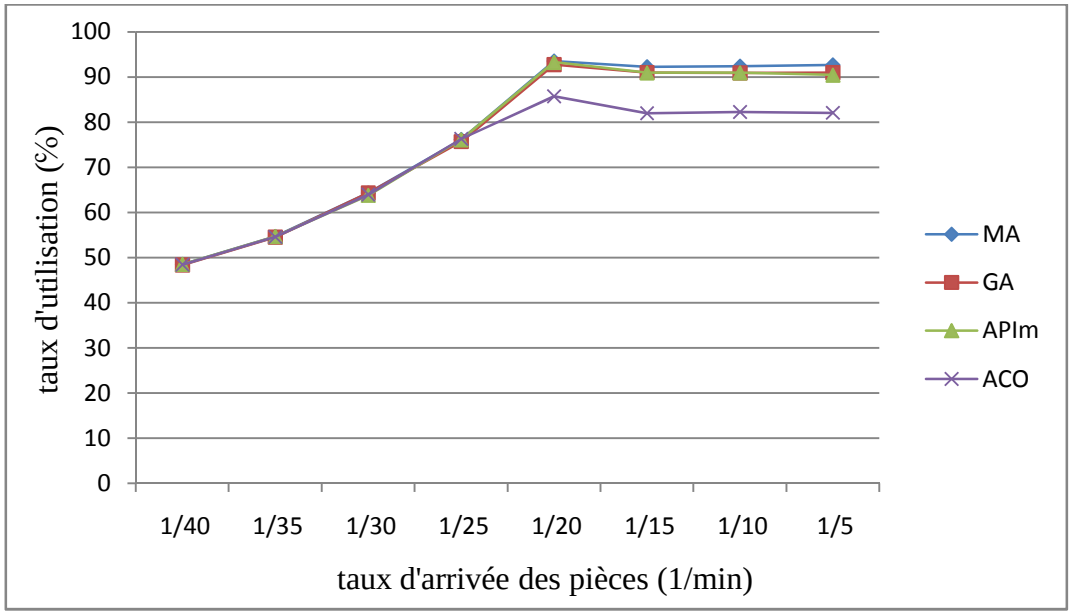


Figure3.32: Le taux d'utilisation des machines T1 et T2 pour une taille de la file d'attente=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	48.19	54.48	63.93	76.63	93.33	93.32	93.45	93.64
GA	48.37	54.82	64.23	76.55	93.33	93.30	93.25	93.24
API_m	48.47	54.30	63.82	76.23	93.34	91.39	91.46	91.32
ACO	48.54	54.76	64.09	76.39	93.52	89.57	89.35	89.48

Tableau 3.26: Le taux d'utilisation des machines T1 et T2 pour une taille file=6.

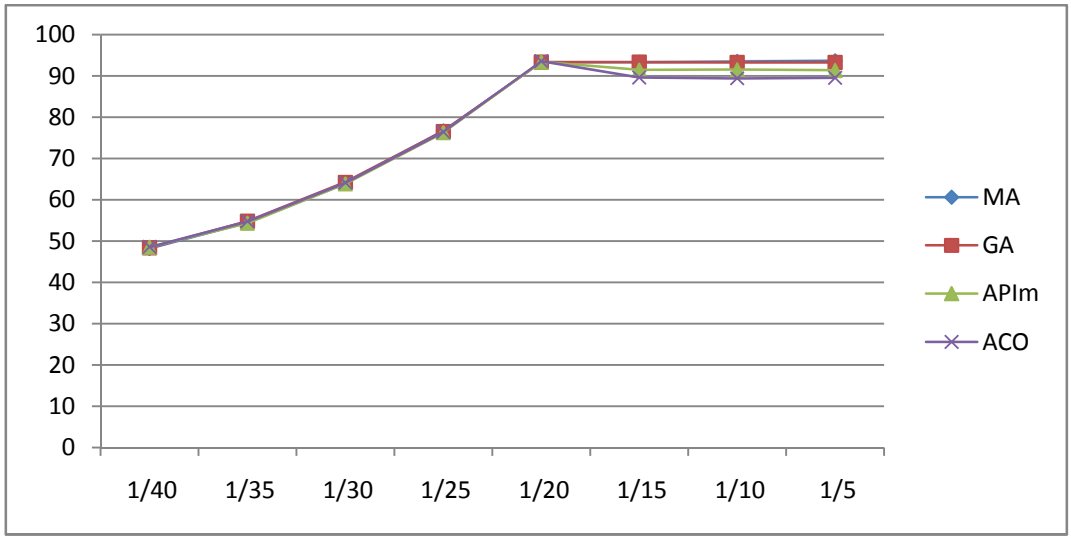


Figure 3.33: Le taux d'utilisation des machines T1 et T2 pour une capacité de file d'attente=6.

A partir des tableaux 3.25, 3.26 et des figures 3.32, 3.33 on peut remarquer que pour une capacité de file d'attente égale à 2 l'algorithme mimétique donne les meilleurs résultats pour un taux de création supérieur ou égale à 1/20, hors de cet intervalle il devient moins performant mais les résultats obtenus par les quatre métaheuristiques restent voisines. Concernant l'algorithme API_m, les résultats obtenus sont satisfaisants comparés à ceux obtenus par l'ACO. Pour une capacité de file d'attente égale à 6, l'algorithme mimétique reste performant pour un taux de création supérieur ou égale à 1/20, hors de cet intervalle il devient moins performant comparé à l'algorithme génétique. De même pour l'API_m, en le comparant à l'ACO pour une capacité de file d'attente égale à 6, les meilleurs résultats pour un taux de création supérieur à 1/20.

3.9 Conclusion :

Dans ce chapitre, nous avons présentés notre adaptation de deux métaheuristiques à base de population de solutions. Ces approches ont été comparées avec deux autres métaheuristiques, pour pouvoir tirer une idée sur l'efficacité de chacune des méthodes utilisées.

Nous avons tout d'abord commencé par présenter une comparaison entre l'algorithme API_m et l'algorithme API. L'algorithme API_m représente une extension de l'algorithme API, l'idée est de tenter de bien explorer la zone de l'espace de recherche vers laquelle la population converge, en permettant au nid de bien être exploré à chaque itération par chaque fourmi.

Ensuite nous avons présenté une analyse de sensibilité de cet algorithme, 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.

Et pour valider notre travail, nous avons consacré une section pour faire une comparaison des deux métaheuristiques étudiées avec deux autres métaheuristiques déjà utilisées pour résoudre le même problème.

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 :

- La comparaison entre l'API et l'API_m montre que la modification introduite au niveau de l'algorithme API a entrainer des améliorations remarquables dans les résultats obtenus.

- L'étude de sensibilité de l'algorithme API_m a permis de définir les paramètres qui améliorent les performances de la métaheuristique considérée.

- Les résultats obtenus ont montré que pour le taux de production et certains d'autres critères de performances, l'algorithme mimétique a donné les meilleurs résultats comparé à l'algorithme génétique.

Concernant l'algorithme API_m, les résultats obtenus sont, en général, meilleurs que ceux obtenus par l'ACO, sauf pour certains critères ou pour certaines valeurs.

CONCLUSION GENERALE

Ce mémoire se situe dans le cadre d'adaptation des métaheuristiques pour la résolution d'un problème d'ordonnancement dans un système flexible de production.

Les métaheuristiques sont des méthodes dédiées à la résolution des problèmes d'optimisation de type NP-difficile. Malgré qu'elles ne garantissent aucune optimalité sur la solution obtenue, les métaheuristiques ont permis de résoudre un grand nombre des problèmes d'optimisation combinatoire, et une multitude de problèmes d'ordonnancement.

Le premier chapitre expose la problématique d'ordonnancement dans les systèmes flexibles de production, et de donner quelques notions générales : les systèmes de production flexibles, les problèmes d'ordonnancement, 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 de du chapitre était réservé aux deux méthodes que nous avons adapté, à savoir, l'algorithme mimétique et l'algorithme API basé sur le comportement de fourragement d'une espèce de fourmis primitives.

Le troisième chapitre est réservé à l'adaptation des deux méthodes (l'algorithme API et l'algorithme mimétique), des simulations ont été effectuées dans un Core (TM) 2 Duo CPU avec 2.2 GHz et 2 GO de RAM, en variant le taux d'arrivée des pièces et en gardant la capacité de la file 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. Les résultats obtenus ont été comparés avec ceux de l'algorithme génétique et l'ACO. Les résultats de ces simulations ont mené aux conclusions suivantes :

- La modification introduite au niveau de l'algorithme API et la comparaison effectuée entre l'algorithme API et l'algorithme API_m a conduit à des changements majeurs dans les résultats obtenus, cela est clair pour le taux de production et le taux d'utilisation des machines.
- L'algorithme mimétique a donné les meilleurs résultats du taux de production. concernant le temps du cycle, l'algorithme mimétique a donné les meilleurs résultats pour le cas saturés.

Les perspectives :

Les perspectives ouvertes pour ce travail :

- Dans l'algorithme mimétique :
  - Utiliser une méthode de recherche locale plus évoluée que la méthode de descente dans l'algorithme mimétique, telle que la méthode multistart descent, le recuit simulé, ou la recherche tabou

- Dans l'algorithme API :
  - Utiliser une heuristique ou une métaheuristique à solution unique pour la création des sites de chasse.
  - Etudier l'influence de l'augmentation des nombres de sites de chasse sur les performances de l'algorithme API_m, car dans ce travail on s'est limité à un nombre de site de chasse égale à 8, il sera intéressant d'étudier la sensibilité de l'algorithme pour des nombres de sites plus importants.
  - Etudier la sensibilité par rapport à la taille de la population, c'est-à-dire, par rapport aux nombre de fourmis.
- Les règles de priorité sont utilisés en industrie sont très diverses. Les règles les plus connues sont First In Fierst Out (FIFO), Shortest Processing Time (SPT), Earliest Due Date (EDD), Minimum Slack Time (MST)…
  Durant ce travail nous avons choisi la règle FIFO pour le séquencement de chaque machine. Comme perspective, nous pouvons proposer de combiner les deux algorithmes proposés avec d'autres règles de priorité.

ANNEXE A

A.1 Le taux de production

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	99.99	99.99	99.99	99.99	99.99	73.68	49.21	24.62
API_m	99.99	99.99	99.99	99.99	99.99	71.97	47.96	23.72
ACO	99.99	99.99	99.99	99.99	90.52	64.54	43.20	21.51
SA	99.99	99.99	99.99	99.99	94.73	61.90	41.21	20.61
PS	99.99	99.99	99.99	99.99	99.22	69.33	46.10	23.12
GA	99.99	99.99	99.99	99.99	98.44	71.08	47.25	23.70
TS	99.99	99.99	99.99	99.99	95.28	64.93	43.29	21.78
EM	99.99	99.99	99.99	99.99	98.52	66.94	44.75	22.42

Tableau A.1: Taux de production pour une taille de la file d'attente =2

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	99.99	99.99	99.99	99.99	99.99	74.12	49.45	24.72
API_m	99.99	99.99	99.99	99.99	99.99	70.99	47.35	23.63
ACO	99.99	99.99	99.99	99.99	99.99	68.78	45.73	22.85
SA	99.99	99.99	99.99	99.99	99.99	64.69	43.44	21.71
PS	99.99	99.99	99.99	99.99	99.99	70.29	46.92	23.40
GA	99.99	99.99	99.99	99.99	99.99	72.06	47.88	23.93
TS	99.99	99.99	99.99	99.99	99.99	69.65	46.38	23.19
EM	99.99	99.99	99.99	99.99	99.99	69.08	46.03	23.02

Tableau A.2: Taux de production pour une taille de la file d'attente = 4

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	99.99	99.99	99.99	99.99	99.99	74.17	49.56	24.84
API_m	99.99	99.99	99.99	99.99	99.99	71.20	47.52	23.71
GA	99.99	99.99	99.99	99.99	99.99	68.97	45.88	22.98
ACO	99.99	99.99	99.99	99.99	99.99	67.48	45.17	22.70
SA	99.99	99.99	99.99	99.99	99.99	71.15	47.47	23.78
PS	99.99	99.99	99.99	99.99	99.99	72.41	48.26	24.17
TS	99.99	99.99	99.99	99.99	99.99	71.36	47.60	23.80
EM	99.99	99.99	99.99	99.99	99.99	69.13	46.23	23.16

Tableau A.3: Taux de production pour une taille de la file d'attente =6

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	24.59	24.72	24.84	24.82
API_m	23.72	23.70	23.80	23.88
ACO	21.51	22.85	22.98	23.13
SA	20.61	21.71	22.70	23.39
PS	23.12	23.40	23.78	24.07
GA	23.70	23.93	24.17	24.24
TS	21.78	23.19	23.80	24.07
EM	22.42	23.02	23.16	23.69

Tableau A.4: taux de production pour une taille de la file d'attente=1/5min

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	99.99	99.99	99.99	99.99
API_m	99.99	99.99	99.99	99.99
ACO	90.52	99.99	99.99	99.99
SA	94.73	99.99	99.99	99.99
PS	99.22	99.99	99.99	99.99
GA	98.44	99.99	99.99	99.99
TS	95.28	99.99	99.99	99.99
EM	98.52	99.99	99.99	99.99

Tableau A.5: Taux de production pour une taille de la file d'attente =1/20min

A.2 Le temps du cycle :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	94.16	83.01	91.02	95.97	124.60	168.13	168.46	171.13
API_m	93.27	82.08	87.50	95.92	116.78	169.78	169.31	170.69
ACO	90.90	83.70	89.00	98.80	155.20	166.60	165.40	163.50
SA	95.30	83.80	90.30	99.50	135.50	170.40	168.90	168.10
PS	89.60	81.20	91.70	96.10	125.90	173.50	173.20	169.40
GA	89.60	81.80	88.60	98.70	132.70	169.80	168.90	168.10
TS	91.40	80.30	86.20	98.00	135.90	171.70	174.80	173.90
EM	92.20	81.60	90.10	101.10	124.80	170.50	170.80	168.90

Tableau A.6: Temps de cycle pour une taille de la file d'attente =2

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	90.01	82.15	88.87	95.01	124.41	240.23	250.44	248.60
API_m	87.93	90.50	90.50	95.43	117.87	247.12	246.44	249.44
ACO	89.30	82.30	89.50	95.60	130.00	233.50	242.80	247.90
SA	92.60	81.90	87.60	94.60	120.20	249.50	249.30	249.10
PS	92.30	82.40	88.30	97.40	119.90	248.70	246.70	249.30
GA	91.10	82.40	91.40	98.60	118.00	250.90	245.60	249.90
TS	95.60	83.60	89.20	96.70	125.30	255.70	252.40	252.10
EM	93.50	81.40	89.50	98.80	120.70	243.80	252.50	249.60

Tableau A.7: Temps de cycle pour une taille de la file d'attente =4.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	92.40	84.06	88.84	97.43	125.97	323.50	320.26	327.75
API_m	92.48	83.42	87.54	96.61	124.46	327.08	325.66	330.46
ACO	91.10	82.60	89.30	97.00	121.40	326.00	324.00	320.60
SA	89.00	80.60	88.80	100.10	126.00	325.30	327.50	324.90
PS	91.00	82.50	91.10	102.20	133.60	332.30	326.00	331.20
GA	93.10	83.40	91.60	98.90	121.30	331.70	332.80	338.10
TS	97.10	78.80	89.60	98.60	125.60	329.20	328.80	330.30
EM	81.26	86.67	97.65	127.20	276.90	279.20	269.90	283.90

Tableau A.8: Temps de cycle pour une taille de la file d'attente =6.

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	124.60	124.41	125.97	128.87
API_m	170.69	246.77	329.68	406.21
ACO	163.50	247.90	320.60	382.40
SA	168.10	249.10	324.90	405.20
PS	169.40	249.30	331.20	406.90
GA	168.10	249.90	338.10	417.00
TS	173.90	252.10	330.30	428.50
EM	168.90	249.60	328.10	407.10

Tableau A.9: Temps de cycle pour un taux de production=1/5min

Taux d'arrivée des pièces (1/min)	2	4	6	8
MA	124.60	124.41	125.97	128.87
API_m	116.78	130.70	124.46	124.22
ACO	155.20	130.00	121.40	119.80
SA	135.50	120.20	126.00	124.50
PS	125.90	119.90	133.60	117.50
GA	132.70	118.00	121.30	119.70
TS	135.90	125.30	125.60	124.00
EM	124.80	120.70	122.20	122.70

Tableau A.10: Temps de cycle pour un taux de production=1/20min

A.3 Les en-cours :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	3.91	4.03	4.48	5.22	7.20	9.00	9.00	9.25
API_m	3.90	4.04	4.45	5.21	7.01	8.84	8.85	8.81
ACO	3.90	40.04	4.45	5.21	7.29	8.40	8.44	8.46
SA	3.91	4.03	4.46	5.22	7.11	7.95	7.95	7.95
PS	3.88	4.03	4.45	5.25	7.35	8.67	8.67	8.64
GA	3.90	4.04	4.46	5.23	7.45	8.90	8.85	8.85
TS	3.89	4.03	4.47	5.26	7.98	8.40	8.40	8.41
EM	3.89	4.04	4.44	5.28	7.99	8.64	8.66	8.62

Tableau A.11: Les en-cours pour une taille de la file d'attente =2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	7.88	8.03	8.46	9.26	11.15	23.00	20.00	21.00
API_m	7.90	8.03	8.44	9.24	11.41	19.14	19.18	19.13
ACO	7.89	8.02	8.45	9.20	11.31	18.46	18.56	18.56
SA	7.89	8.03	8.44	9.20	11.15	18.45	18.52	18.60
PS	7.88	8.04	8.47	9.26	11.16	19.27	19.28	19.24
GA	7.89	8.04	8.46	9.26	11.26	19.68	19.72	19.73
TS	7.89	8.03	8.48	9.28	11.19	19.46	19.45	19.53
EM	7.89	8.03	8.45	9.25	11.14	18.83	19.20	18.89

Tableau A.12: Les en cours pour une taille de la file d'attente =6.

A.4 Taux d'utilisation des machines :

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	14.49	17.53	19.78	24.94	33.29	32.26	32.43	32.48
API_m	14.43	17.42	20.33	25.00	33.58	30.42	30.41	29.43
ACO	14.53	17.56	20.66	24.74	28.78	26.77	26.93	26.67
SA	14.33	17.39	19.91	24.89	30.49	23.34	23.27	23.26
PS	14.71	17.42	20.16	24.44	32.51	27.61	27.44	27.72
GA	14.62	17.67	19.70	25.54	31.94	28.60	28.36	28.72
TS	14.78	17.78	20.45	24.59	30.13	24.10	24.08	24.24
EM	14.77	17.71	20.16	24.90	32.47	26.12	26.22	26.27

Tableau A.13: Le taux d'utilisation des machines FV1 et FV2 pour une taille de la file=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	23.23	25.19	30.34	34.72	40.45	40.37	40.36	40.28
API_m	23.32	25.35	29.57	34.65	40.05	40.96	40.95	41.41
ACO	23.19	25.15	29.52	35.00	38.52	37.44	37.52	37.59
SA	23.45	25.40	30.16	34.79	39.79	39.18	39.18	39.23
PS	22.92	25.35	29.81	35.42	40.87	41.82	41.87	41.71
GA	23.05	25.00	30.45	33.88	40.99	42.47	42.57	42.36
TS	22.84	24.85	29.40	36.22	40.76	41.63	41.66	41.92
EM	22.83	24.95	29.81	34.80	40.32	41.14	41.23	41.31

Tableau A.14: Le taux d'utilisation des machines FH1 et FH2 pour une taille de la file=2

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	48.47	54.62	64.26	76.15	93.51	92.22	92.39	92.66
API_m	48.52	54.71	63.82	76.11	93.28	90.95	90.93	90.45
ACO	48.44	54.59	63.99	76.31	85.73	81.97	82.26	82.06
SA	48.59	54.74	64.16	76.20	89.42	80.49	80.41	80.47
PS	48.29	54.71	63.96	76.55	93.20	89.20	88.84	88.95
GA	48.36	54.51	64.32	75.67	92.72	90.98	90.85	90.96
TS	48.24	54.42	63.72	76.43	90.36	84.73	84.75	85.27
EM	48.24	54.48	63.96	76.19	92.39	86.16	86.56	86.75

Tableau A.15: Le taux d'utilisation des machines T1 et T2 pour une taille de la file=2

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
MA	1.309	1.993	1.709	1.956	2.276	2.260	2.270	2.266
API_m	1.314	1.428	1.666	1.951	2.253	2.306	2.306	2.332
ACO	1.307	1.417	1.663	1.970	2.169	2.108	2.113	2.117
PS	1.322	1.431	1.699	1.959	2.240	2.209	2.209	2.212
SA	1.292	1.428	1.679	1.995	2.300	2.357	2.359	2.351
GA	1.299	1.408	1.711	1.907	2.307	2.393	2.399	2.387
TS	1.287	1.400	1.656	1.983	2.295	2.348	2.350	2.364
EM	1.287	1.405	1.679	1.959	2.269	2.319	2.323	2.329

Tableau A.16: Le taux d'utilisation de la machine TP pour une taille de la file=2.

A.4 Taux d'utilisation des machines (comparaison entre l'API et l'API_m)

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	1.287	1.422	1.716	1.975	2.129	1.700	1.690	1.689
API_m	1.292	1.431	1.686	1.951	2.320	2.346	2.390	2.393

Tableau A.17: Le taux d'utilisation de la machine TP pour une taille de la file=2.

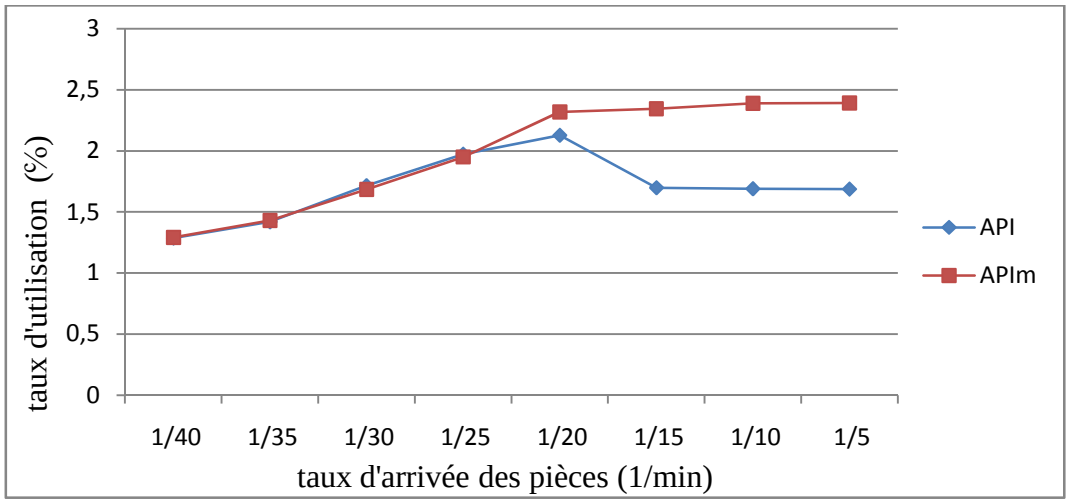


Figure 3.13: Le taux d'utilisation de la machine TP pour une taille de la file d'attente=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	14.78	17.49	19.70	24.69	28.02	17.20	16.87	16.87
API_m	14.71	18.49	20.08	24.99	31.31	29.53	28.60	28.46

Tableau A.18: Le taux d'utilisation des machines FV1 et FV2 pour une taille de la file=2.

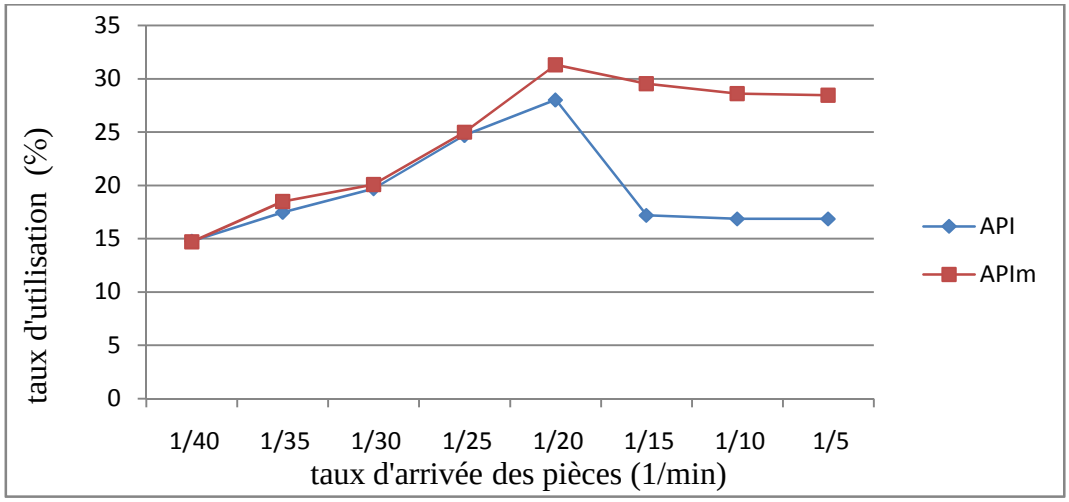


Figure 3.10: Le taux d'utilisation des machines FV1 et FV2 pour une taille de la file d'attente=2.

Taux d'arrivée des pièces (1/min)	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
API	22.84	25.25	30.45	35.07	37.82	30.13	29.95	29.94
API_m	22.92	23.85	29.92	34.65	41.21	41.64	42.42	42.47

Tableau A.18: Le taux d'utilisation des machines FH1 et FH2 pour une taille de la file=2.

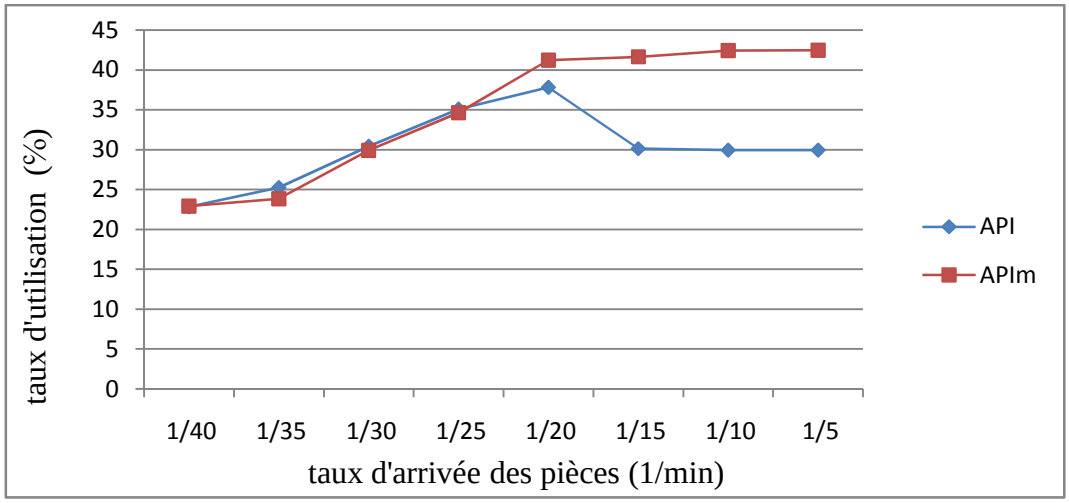


Figure 3.11: Le taux d'utilisation des machines FH1 et FH2 pour une taille de la file d'attente=2.

ANNEXE B

Liste des abréviations :

FMS : Flexible Manufacturing System (Système Flexible de Production)
AGV : Auto Guided Vehicle
GA: Genetic Algorithm
MA: Memetic Algorithm
API: Pachycondyla Apicalis
API_m: Pachycondyla Apicalis modifié
SA: Simulated Annealing
ACO: Ant Colony Optimization
EM: Electromagnetism like method
TS: Tabu search
PSO : Perticle Swarms Optimization
SC: Station de Chargement
SD: Station de Déchargement
FV : Fraiseuse Verticale
FH : Fraiseuse Horizontale
T: Tour
TP: Toupie
FIFO: First IN First Out
EPL: Equal Probability Loading
EDD: Earliest Due Date
MST: Minimum Slack Time
SPT: Shortest Processing Time

Références bibliographiques

[Adamou, 1997] Adamou, M., (1997). 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.*

[Akrout et Masmoudi, 2009] Akrout M., Masmoodi F., (2009). Fonction Ordonnancement au Sein d'un Système de Gestion de Production « étude d'un cas », *Lebanese Science Journal,* Vol. 10, No. 1, pp. 107-117.

[Amodeo, 1999] Amodeo, L., (1999). Contribution à la simplification et à la commande des réseaux de Petri stochastiques. Application aux systèmes de production, *Thèse de doctorat en Productique soutenue à l'INP, Grenoble (France).*

[Ayari, 2010] Ayari, N., (2010). Métaheuristiques parallèles hybrides pour l'optimisation combinatoire : problème de règles de Golomb, *Mémoire de Master, Université de Jendouba, Tunisie.*

[Back et *al.*, 2000] Back, T., Fogel, D.B., et Michalewicz, T., (2000). A history of evolutionary computation, Chapitre 6, *Institute of physics publishing.*

[Bachelet, 1999] Bachelet, V., (1999). Métaheuristiques parallèles hybrides : application au QAP. *Habilitation à Diriger des Recherche, USTL LIFL France.*

[Ben-Arieh, 1988] Ben-Arieh, D., (1988). Knowledge-based routing and control system for FMS. In. Kumara, S.T., Soyster, A.L., Kashyap, R.L., (Eds), Artificial intelligence: manufacturing theory and practice, Norcross, pp. 631-646.

[Bellman, 1954] Bellman, R., (1954). The theory of dynamic Programming. *Bulletin of the American Mathematical Society.* 60(6), 503-15.

[Birbil et Fang, 2003] Birbil, S.I., et Fang, S., (2003). An Electromagnetism-like Mechanism for global Optimization, *Journal of global Optimization*, vol. 25: 263-282.

[Bourdeaud'huy et Korbaa 2006] Bourdeaud'huy, T., Korbaa, O., (2006). 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.*

[Buzzcot et Yao, 1986] Buzzacot, J. A., Yao, D. D., (1986). Flexible Manufacturing systems: a review of analytic models. Management Science, 32(7), pp. 890-905.

[Caumond, 2006] Caumond, A., (2006). Le problème de jobshop avec contraintes: modélisation et optimisation, *Thèse de doctorat, Université Blaise Pascal-Clermont Ferrand II.*

[Caumond et *al.*, 2006a] Caumond, A., Lacomme, P., Moukrim, A., Tchernev, N., (2006). An MILP for scheduling problems in an FMS with one vehicle. *European journal of operational research.*

[Caumond et *al.*, 2006b] Caumond, A., Lacomme P., et Tcehrnev, N., (2006). A memetic algorithm for the jobshop with time lags. *Computers and operations research.*

[Chaari, 2010] Chaari, T., (2010). Un algorithme génétique pour l'ordonnancement robuste : Application au problème du flowshop hybride, *Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.*

[Charon et *al.*, 1996] Charon, I., Germa A., Hurdy, O., (1996). Méthodes d'optimisation combinatoire, éditions Masson, Paris.

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

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

[Colorni et *al.*, 1991a] Colorni, A., Dorigo, M., et Maniezzo, V., (1991). Distributed Optimization by Ant Colonies. *Proceedings of ECAL91-First European Conférence on Artificial Life,* pp. 134-42.

[Colorni et *al.*, 1991b] Colorni, A., Dorigo, M., et Maniezzo, V., (1991). Positive feedback as a search strategy. *Technical Report 91-016, Ecole Polytechnique de Milan (Polytechnico di Milan).*

[Colorni et *al.*, 1992] Colorni, A., Dorigo, M., et Maniezzo, V., (1992). An investigation of some Properties of an Ant Algorithm, *Proceedings of the parallel problem solving from nature conference (PPSN 92),* pp.509-20.

[Cotta et al., 2005] Cotta, C., Talbi, E-D., et Alba, E., (2005). Parallel hybrid Metaheuristics: A new Class of Algorithms, Wiley-Interscience. pp.347-370.

[Darwin, 1859] Darwin, C., (1859). On the Origin of Species. Jhon Murray, London.

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

[De Jong et *al.*, 2000] De Jong, K., Fogel, D.B., et Schwefel, H.P., (2000). Evolutionary computation 1: basic algorithms and operators, *Institute of physics publishing.*

[Deneubourg et *al.*, 1983] Deneubourg, J.L., Pasteels, J.M., et Verhaeghe, J.C., (1983). Probabilistic behavior in ants: a strategy of errors? *Journal of theoretical biology, pp. 259-71.*

[Dorigo et Stutzle, 2004] Dorigo, M., et Stutzle, T., (2004). Ant colony optimization. MIT Press.

[Dorigo et *al.*, 1996] Dorigo, M., Maniezzo, V., et Colorni, A., (1996). The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on systems, Man, and Cybernetics-Part B,* 26(1), pp.1-13.

[Dréo, 2004] Dréo, J., (2004). Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical, *Thèse de doctorat, Université Paris12.*

[Dupas, 2004] Dupas, R., (2004). Amélioration de performance des systèmes de production: apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles, *Habilitation à diriger des recherches, Université d'Atrois.*

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

[Eberhart et Kennedy, 1995] Eberhart, R.C., et Kennedy, J., (1995). New optimizer using particle swarm theory, *Proceedings of the 6th International Symposium on Micro Machine and Human Science, pp.39-43, Nagoya, Japan.*

[Esquirol et Lopez, 1999] Esquirol, P., Lopez, P., (1999). L'ordonnancement, Série: Production et techniques quantitatives appliquées à la gestion, collection Gestion, Economica.

[Fresneau, 1985] Fresneau, D., (1985). Individual Foraging and Path Fidelity in a Ponerine Ant. Insectes sociaux, Paris, 32(2):109-116.

[Fresneau, 1994] Fresneau, D., (1994). Biologie et comportement social d'une fourmi ponérine néotropicale (Pachycondyla apicalis), *Thèse d'état, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et comparée, France.*

[Giard, 1988] Giard, V., (1988). Gestion de la production. Paris : Economica.

[Glover, 1986] Glover, F., (1986). Future paths of integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533-549.

[Goldberg 1989] Goldberg, D.E., (1989). Genetic algorithms in search, Optimization and machine learning, Addison-Wesley publishing company.

[Goldberg, 1994] Goldberg, E.E., (1994). Algorithmes génétiques, Editions Addison-Wesley, Paris.

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

[Holldobler et Wilson, 1990] Holldobler, B., et Wilson, E., (1990). The Ants, Springer Verlag, Berlin, Germany.

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

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

[Jourdan, 2010] Jourdan, L., (2010). Métaheuristiques Coopératives: du déterministe au stochastique, *Habilitation à Diriger des Recherches, Université de Lille I.*

[Kazerooni et *al*., 1997] Kazerooni, A., Chan F.T., & Abhary, K., (1997). A fuzzy integrated decision-making support system for scheduling of FMS using simulation, Computer Integrated Manufacturing Systems, 10(1), pp. 27-34.

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

[Kirkpatrick et *al.*, 1983] Kirkpatrick, S., Gellat, C.D., et Vecchi, M.P., (1983). Optimisation by Simulated Annealing. Science, 220 :671-680.

[Lemamou, 2009] Lemamou, E.A., (2009). 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.*

[Lemlouma, 2001] Lemlouma, T., (2001). Une Etude d'Approches Heuristique pour l'Ordonnancement des Jobs dans un Flow-Shop, RIST Vol. 11 n° 2.

[MacCarthy et Liu, 1993] MacCarthy, B.L., Liu J., (1993). A new classification scheme for flexible manufacturing systems. *International journal of production research*, 31(2), 299-309.

[Mamalis et *al*., 1995] Mamalis, A.G., Malagardis, I., & Pachos, E., (1995). On-line Scheduling in Metal Removel Processing Using Variable Routing and Control Strategies, Computer Integrated Manufacturing Systems, 8, pp. 35-40.

[Mesghouni, 1999] Mesghouni, K., (1999). Application des algorithmes évolutionnistes dans les problems d'optimisation en ordonnancement de la production, *Thèse de doctorat, Université des sciences et technologies de Lille1*.

[Metropolis et *al*., 1953] Metropolis, N., Rosenbluth, A., Rosenbluth, M., et Teller, E., (1953). Equation of state calculations by fast computing machines, *Journal of chemical physics*, 21:1087-1092.

[Monmarché, 2000] Monmarché, N., (2000). Algorithmes de fourmis artificielles: applications à la classification et à l'optimisation, T*hèse de doctorat, Université François Rabelais Tours.*

[Moscato, 1989] Moscato, P., (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Technical Report C3P 826, Cal-teech Concurrent Computation Program.*

[Newman, 1988] Newman, P. P., (1988). Scheduling in CIM systems. In Kusiak A. (Ed.) Artificial Intelligence in Industry: Implications for CIM, New York: Springer-Verlag, pp. 361-402.

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

[Noël, 2007] Noël, S., (2007). Métaheuristiques hybrides pour la résolution du problème d'ordonnancement de voitures dans une chaîne d'assemblage automobile, *Mémoire présentée comme exigence partielle de la maitrise en informatique, Université du Québec.*

[Parunak, 1985] Parunak, H.V.D., (1985). Manufacturing experience with the contract net. *In Proceedings of the Fith Workshop on Distributed Artificial Intelligence.*

[Peng et Chen, 1998] Peng C., and Chen F.F., (1998). 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.

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

[Rebreyend, 1999] Rebreyend, P., (1999). Algorithmes génétiques hybrides en optimisation combinatoire, *Thèse de doctorat, l'école Normale Supérieure de Lyon.*

[Sari, 2003] Sari, Z., (2003). Modélisation, Analyse et Evaluation des Performances d'un AS/RS à Convoyeur Gravitationnel, *Thèse de doctorat d'état, Université Abou Bakr Belkaid, Tlemcen.*

[Sassine, 1998] Sassine, C., (1998). Intégration des politiques de maintenance dans les systèmes de production manufacturiers. *Thèse de doctorat soutenue à l'INP de Grenoble (France).*

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

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

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

[Souier et al., 2010] Souier, M., Hassam A., Sari Z., (2010). Metaheuristics for Real Time Routing Selection in FMS, Book chapter in: Lyes Benyoucef and Bernard Grabot (ED.), *Artificial intelligence techniques for networked manufacturing enterprises management*, Springer-Verlag, ch. pp.221-247.ISNB 978-1-84996-118-9.

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

[Wolpert et Macready, 1997] Wolpert, D. H., et Macready, W.G., (1997). No free lunc theorems for optimization. *IEEE Transactions on Evolutionnary Computation, 1(1). pp. 67-82.*

Ordonnancement en temps réel d'un FMS par métaheuristique hybride : l'algorithme mémétique.

Y. Houbad, M. Souier, A. Hassam, Z.Sari

Laboratoire de Productique Tlemcen
Faculté de technologie, Université Abou Bakr Belkaid, BP 230, Tlemcen 13000, Algérie.
houbadyamina@yahoo.fr, m_souier@mail.univ-tlemcen.dz,a_hassam@mail.univ-tlemcen.dz,z _sari@mail.univ-tlemcen.dz

RÉSUMÉ. *Les problèmes d'ordonnancement sont souvent des problèmes d'optimisation combinatoire de type NP-difficile. Leur résolution nécessite des méthodes dédiées à leur degré de complexité, c'est pour cette raison que plusieurs heuristiques et métaheuristiques ont été conçues.*

Notre étude est consacrée à l'adaptation de l'algorithme mémétique pour résoudre le problème de sélection de routages alternatifs en temps réel dans un FMS (flexible manufacturing system).

Le choix de cette métaheuristique est dû au fait que les métaheuristiques hybrides fournissent une compensation des faiblesses pouvant se trouver dans d'autres algorithmes, tel que l'algorithme que nous avons utilisé (l'algorithme mémétique) qui améliore la vitesse de convergence de l'algorithme génétique en lui ajoutant une recherche locale. Ensuite on va faire une comparaison entre cet algorithme et l'algorithme génétique et d'autres métaheuristiques déjà utilisées pour résoudre ce même problème afin d'avoir une idée sur les méthodes les plus performantes pour ce problème et opter pour la plus efficace.

ABSTRACT. *The Scheduling problems are often problems of combinatorial optimization of NP-hard. Their resolution requires methods dedicated to their degree of complexity, for this reason that several heuristics and metaheuristics have been designed. Our study is devoted to the adaptation of the memetic algorithm to solve the problem of selection of alternative routings in real time of an FMS (flexible manufacturing system). The choice of this metaheuristic is because the hybrid metaheuristics provide compensation for weaknesses that can be found in other algorithms, such as the algorithm that we used (the memetic algorithm) which improves the speed of convergence of the genetic algorithm by adding a local search. Then we will make a comparison between this algorithm and the genetic algorithm and other metaheuristics previously used to solve the same problem to get an idea on the most efficient methods for this problem and choose the most effective.*

MOTS-CLÉS: *ordonnancement, les systèmes flexibles de production , métaheuristiques, métaheuristiques hybrides, algorithme mémétique, simulation.*

KEYWORDS: *scheduling, flexible manufacturing system, metaheuristics, hybrid metaheuristics, memetic algorithm, simulation.*

1. Introduction

Les systèmes flexibles de production sont définis par MacCarthy et Liu (1993) : « un système flexible de production est 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 » (Caumond, 2006). Donc, les systèmes flexibles de production (FMS) peuvent changer rapidement de produits et de séquences de produits sans perdre leurs productivités, à condition qu'il soit possible d'obtenir le bon produit, palette, support, ou outil, à la bonne place au bon moment.

Les problèmes liées à la technologie des FMS sont relativement complexes comparés aux systèmes de production traditionnels. Les problèmes d'ordonnancement d'atelier sont des problèmes d'optimisation combinatoire qui nécessitent d'effectuer un nombre important de calculs pour obtenir un ordonnancement admissible (ou réalisable) qui optimise le (ou les) critère(s) retenu(s) en tenant compte des contraintes. Le développement de la théorie de complexité a permis de classifier ces problèmes selon leurs difficultés (Graham et *al*., 1979). Donc, il n'existe pas de méthode exacte capable de les résoudre, c'est une raison pour laquelle plusieurs heuristiques et métaheuristiques ont été conçues. Une heuristique est un algorithme d'optimisation qui a pour but de trouver une solution réalisable de la fonction objective, mais sans garantir d'optimalité. Parmi ces heuristiques certaines sont adaptables à un grand nombre de problèmes différents sans changements majeurs dans l'algorithme, on parle alors des métaheuristiques.

Les métaheuristiques sont souvent inspirées de la biologie (algorithmes évolutionnaires…), de l'éthologie (colonies de fourmis, essaims particulaires,…), et de la physique (recuit simulé,…), elles sont généralement conçues au départ pour des problèmes discrets, mais peuvent s'adapter à d'autres types de problèmes.

Pour aller plus loin dans la recherche de solutions, il faut avant tout pouvoir détecter de nouvelles solutions. Les algorithmes à base de population présentent un intérêt particulier : le parallélisme intrinsèque. En ajoutant de nouveaux composants à ces algorithmes, on peut alors construire des algorithmes « hybrides ». Cela peut être fait en combinant deux ou plusieurs métaheuristiques qui travaillent d'une manière coopérative ou une métaheuristique avec une méthode exacte. L'une des faiblesses d'un algorithme génétique comme la vitesse de convergence trop lente peut, par exemple, être compensée par l'ajout d'une méthode de recherche locale. C'est le cas des algorithmes mémétiques de Moscato (Moscato, 1986).

Dans ce travail nous nous sommes intéressés à l'adaptation de l'algorithme mémétique pour la résolution du problème de sélection de routages alternatifs dans un FMS. La raison pour laquelle nous avons choisit l'algorithme mémétique c'est qu'il présente une amélioration de l'algorithme génétique qui a été le plus

performant parmi six métaheuristiques utilisées pour résoudre ce même problème (l'algorithme génétique, les colonies de fourmis, les essaims particulaires, l'électromagnétisme, la recherche tabou, et le recuit simulé) selon l'étude faite par (Souier et al. 2010); de là notre idée était d'utiliser un algorithme qui peut compenser les faiblesses trouvées dans l'algorithme génétique, c'est donc l'algorithme mémétique qui est une combinaison entre un algorithme génétique et une recherche locale. Pour tester l'efficacité de cet algorithme nous avons choisit comme critères de performance le taux de production, le temps de cycle, et les en-cours. Ensuite nous avons établi une comparaison entre les résultats obtenus par l'algorithme mémétique et ceux obtenus par l'algorithme génétique, par les colonies de fourmis et par la règles DMM modifiée.

Cet article s'articule autour de quatre sections. La première est un état de l'art qui regroupe les travaux les plus importants dans la sélection de routages alternatifs. Dans la deuxième nous définissons le contexte de travail présenté. Dans la troisième, nous définissons le mode de fonctionnement des algorithmes génétiques et de l'algorithme mémétique adapté. La dernière section est consacrée aux résultats trouvés et aux interprétations.

2. Etat de l'art :

Plusieurs travaux ont été réalisés dans le cadre de l'ordonnancement d'un FMS depuis le début des années 80 lorsque les ateliers flexibles ont commencé à gagner l'acceptation par les pays industrialisés et ce domaine continue à attirer les intérêts des secteurs académiques et industriels.

La flexibilité des routages représente la possibilité d'utiliser des machines, ou des opérations différentes, pour réaliser des produits sous la même configuration du système. On trouve dans la littérature de nombreux travaux qui se sont intéressés à l'influence de la flexibilité de routage sur les performances d'un FMS avec et sans pannes de machines. On peut citer les travaux de (Tsubone et al., 1999) qui ont étudié par simulation l'impact de la flexibilité des machines et de routages en variant différentes conditions de l'atelier sur le temps de cycle moyen des pièces, on trouve aussi les travaux de (Mahmoodi et al., 1999) qui ont étudié l'effet des règles de priorité et des niveaux de flexibilité de routage sur les différentes performances d'un FMS.

Citons aussi les travaux de (Saygin et al.,1999) qui ont proposé une plateforme qui intègre le processus de planification flexible et l'ordonnancement prédictif`, ils ont présenté un concept nommé Dissimilarity Maximization Method (DMM) pour minimiser la congestion dans un FMS, l'idée de cette règle est de maximiser les dissimilitudes entre les routages occupés, dans leur algorithme chaque routage ne contient qu'une seule pièce à la fois. L'efficacité de cette règle dans la résolution des problèmes de sélection de routage en temps réel a été démontré dans (Saygin *et al.*, 2001) et (Gomri et al., 2007) où elle a surpassé deux autres règles de sélection de

routages alternatifs FIFO/FA (First-In First-Out/ First Available) et Equal Probability Loading (EPL) si chaque machine utilise la règle FIFO comme règle de séquencement. Dans l'étude de (Hassam et al., 2010) sur la règle DMM, dans laquelle ils ont proposé la règle DMM modifiée qui est une modification de la règle DMM qui vise à garder le même principe qui dépend de la maximisation des coefficients de dissimilitude pour la sélection des différents routages alternatifs mais en affectant plusieurs pièces à un seul routage.

On trouve aussi dans les travaux de (Souier et al., 2010) l'adaptation de plusieurs métaheuristiques (les colonies de fourmis, les algorithmes génétiques , les essaims particulaires, le recuit simulé, la recherche tabou et l'électromagnétisme) pour résoudre le problème de sélection de routages alternatifs dans un FMS avec un nombre important de pièces mais en temps différé, et dans (Souier et al., 2010) ils ont réussi à appliquer ces mêmes métaheuristiques sur le même problème en temps réel. Dans (Souier et al., 2010) ils ont adapté des métaheuristiques pour résoudre le problème en temps réel sans et avec pannes des machines avec ré-ordonnancement des pièces de la station de chargement.

3. Présentation du modèle étudié

Le système flexible de production qu'on étudie a déjà été traité dans la littérature, il est composé de sept machines et de deux stations une de chargement et l'autre de déchargement, le système va traiter six types de pièces. Chaque machine du système comprend une file d'attente d'entrée et une file d'attente de sortie, la station de chargement comprend aussi une file d'attente d'entrée. La figure (1) représente la configuration du système, et le tableau 1 présente les routages alternatifs possibles pour chaque type de pièce avec le temps de traitement pour chaque machine.

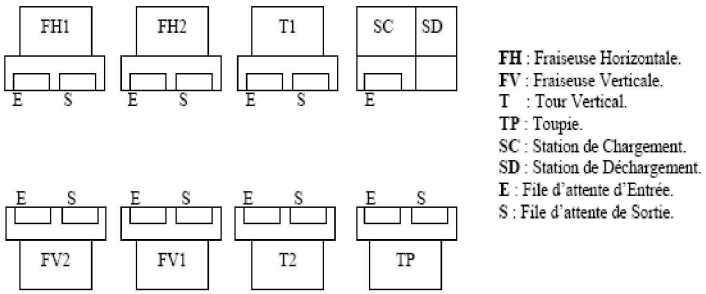


Figure 1. *Configuration du modèle FMS étudié.*

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

– Les routages alternatifs pour chaque type de pièce sont connus avant le début de la production.

–Le temps de traitement est déterminé et il comprend le temps de changement des outils et le temps d'exécution de la machine.

–chaque machine peut traiter une seule pièce à la fois.

Type des pièces	Taux d'arrivée	Routages et temps de traitement (min)
A	17%	SC-T1(30)-FV1(20)-SD
		SC-T1(30)-FV2(20)-SD
		SC-T2(30)-FV1(20)-SD
		SC-T2(30)-FV2(20)-SD
B	17%	SC-T1(20)-TP(1)-FV1(15)-SD
		SC-T1(20)-TP(1)-FV2(15)-SD
		SC-T2(20)-TP(1)-FV1(15)-SD
		SC-T2(20)-TP(1)-FV2(15)-SD
C	17%	SC-T1(40)-TP(1)-FV1(25)-SD
		SC-T1(40)-TP(1)-FV2(25)-SD
		SC-T2(40)-TP(1)-FV1(25)-SD
		SC-T2(40)-TP(1)-FV2(25)-SD
D	21%	SC-T1(40)-TP(1)-T1(20)-FH1(35)-SD
		SC-T1(40)-TP(1)-T1(20)-FH2(35)-SD
		SC-T1(40)-TP(1)-T2(20)-FH1(35)-SD
		SC-T1(40)-TP(1)-T2(20)-FH2(35)-SD
		SC-T2(40)-TP(1)-T1(20)-FH1(35)-SD
		SC-T2(40)-TP(1)-T1(20)-FH2(35)-SD
		SC-T2(40)-TP(1)-T2(20)-FH1(35)-SD
		SC-T2(40)-TP(1)-T2(20)-FH2(35)-SD
E	20%	SC-T1(25)-TP(1)-T1(35)-FH1(50)-SD
		SC-T1(25)-TP(1)-T1(35)-FH2(50)-SD
		SC-T1(25)-TP(1)-T2(35)-FH1(50)-SD
		SC-T1(25)-TP(1)-T2(35)-FH2(50)-SD
		SC-T2(25)-TP(1)-T1(35)-FH1(50)-SD
		SC-T2(25)-TP(1)-T1(35)-FH2(50)-SD
		SC-T2(25)-TP(1)-T2(35)-FH1(50)-SD
		SC-T2(25)-TP(1)-T2(35)-FH2(50)-SD
F	8%	SC-FH1(40)-SD
		SC-FH2(40)-SD

Tableau 1. *Routages alternatifs et temps de traitement des pièces.*

4. Les métaheuristiques

4.1. *L'algorithme génétique*

L'algorithme génétique présenté ici est celui adapté par (Souier, 2009) pour résoudre ce même problème.

n est la capacité des files d'attente.

Chaque chromosome artificiel représente les routages choisis des premières pièces par un code. Après chaque évaluation 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.

Le croisement est fait en prenant deux génotypes, puis on choisi un endroit le long de la chaine, on coupe chacun d'eux à cet endroit et on relie la partie gauche d'une chaine à la partie droite et vice versa.

S'il y'a une place libre dans la station de chargement alors

Génération d'une population aléatoire

Tant que critère d'arrêt non atteint faire

Pour chaque individu

Evaluation de la fitness de cet individu (le produit des charges de routages)

Si la fonction objective est améliorée alors mettre à jour la meilleure solution

Fin pour

Sélection des individus pour la reproduction (opérateur de sélection)

Application de l'opérateur de croisement (on obtient un ensemble de nouveaux individus)

Application de l'opérateur de mutation sur les nouveaux individus

Constitution de la nouvelle génération

Fin tant que

Finsi

4.2. *L'algorithme mémétique*

Moscato (Moscato, 1986) introduit pour la première fois les algorithmes mémétiques. On rencontre aussi le nom d'algorithme génétique hybrides ou celui de (genetic local search). Quelque soit le nom qu'on lui donne, l'idée principale de

cette technique est de rendre plus agressif ou à vrai dire plus performant un algorithme génétique par l'ajout d'une recherche locale en plus de la mutation.

L'idée de Moscato est d'ajouter une recherche locale à l'algorithme génétique, cette recherche locale sera appliquée à tout nouvel individu obtenu par le croisement.

Cette simple modification peut entrainer de profonds changements dans l'algorithme. Après avoir créé un nouvel individu à partir de deux parents sélectionnés, on applique une recherche locale et sous certaines conditions on applique un opérateur de mutation à cet individu.

L'algorithme mémétique utilisé est le suivant :

S'il ya une place libre dans la station de chargement alors

Génération d'une population aléatoire

Tant que critère d'arrêt non atteint **faire**

Pour chaque individu

Evaluation de la fitness de cet individu (le produit des charges de routages)

Appliquer une recherche locale sur chaque individu de la population générée

Fin pour

Répéter

Sélection des individus pour la reproduction (opérateur de sélection)

Application de l'opérateur de croisement (on obtient un ensemble de nouveaux individus)

Appliquer la recherche locale sur chaque individu obtenu

Application de l'opérateur de mutation sur les nouveaux individus

Constitution de la nouvelle génération

Fin tant que

Finsi

La méthode de recherche locale que nous avons utilisée est une méthode simple qui est la méthode de descente donnée par l'algorithme suivant :

Pour chaque individu (qui représente une condition initiale x) pour cette méthode)

Tant que la condition d'arrêt n'est pas vérifiée faire

Modifier les routages de certaines pièces (donc trouver une autre solution voisine x´ de x)

Si la fonction objective a été améliorée (produit des charges des routages) : f(x´)>f(x) alors

Remplacer x par x´

Finsi

Fin tant que

Le principe de la méthode de la méthode de descente est simple, en partant d'une condition initiale x on choisit une autre solution x´ dans le voisinage de x. si cette solution est meilleurs que x on l'accepte comme nouvelle solution x et on recommence le processus jusqu'à ne plus avoir de meilleures solutions améliorantes dans le voisinage.

5. Résultats et interprétations

Dans cette section, et afin de montrer les améliorations apportées par l'algorithme mémétique proposé, nous allons présenter les résultats obtenus pour différentes valeurs de la capacité de la file d'attente de la station de chargement et du taux de création des pièces. Les résultats obtenus seront comparés avec ceux obtenus par l'algorithme génétique, les colonies de fourmis, et la règle DMM modifiée pour trois critères de performances (taux de sortie des pièces, temps de cycle, et les en-cours).

MA : algorithme mémétique

GA :algorithme génétique

ACO : algorithme des colonies de fourmis

DMM modifiée: modified dissimolarity maximization method

5.1. L*e taux de sortie des pièces*

Taux de création	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
DMMmodifiée	99.99	99.99	99.98	99.71	84.47	60.73	41.67	21.15
GA	99.99	99.99	99.99	99.99	98.44	71.08	47.25	23.7
ACO	99.99	99.99	99.99	99.99	90.52	64.54	43.2	21.51
MA	99.99	99.99	99.99	99.99	99.99	73.68	49.21	24.62

Tableau 2. *Taux de sortie des pièces pour une capacité de file d'attente=2.*

Taux de création	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
DMMmodifiée	99.99	99.99	99.99	99.98	90.61	67.79	45.42	22.58
GA	99.99	99.99	99.99	99.99	99.99	72.06	47.88	23.93
ACO	99.99	99.99	99.99	99.99	99.99	68.78	45.73	22.85
MA	99.99	99.99	99.99	99.99	99.99	74.12	49.45	24.72

Tableau 3. *Taux de sortie des pièces pour une capacité de file d'attente=4.*

Les tableaux 2 et 3 montrent que le taux de création des pièces est le même pour toutes les méthodes pour un taux de création inférieur à 1/25, au dessus de cette valeur l'algorithme mémétique a donné les meilleurs résultats, donc pour un système saturé l'algorithme mémétique domine les autres méthodes.

5.2. L*e temps de cycle*

Taux de création	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
DMMmodifiée	81.9	87.8	101.6	155.8	203.3	204.7	207.2	204.2
GA	89.6	81.8	88.6	98.7	132.7	169.8	168.9	168.1
ACO	90.9	83.7	89	98.8	155.2	166.6	165.4	163.5
MA	94.16	83.01	91.02	95.97	124.60	168.13	168.46	171.13

Tableau 4. *Temps de cycle pour une capacité de file d'attente=2.*

Taux de création	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
DMMmodifiée	82.02	86.63	98.07	130.0	309.8	310.4	309.9	311.8
GA	91.1	82.4	91.4	98.6	118	250.9	254.6	249.9
ACO	89.3	82.3	89.5	95.6	130	233.5	242.8	247.9
MA	90.01	82.15	88.87	95.01	124.41	240.23	250.44	248.60

Tableau 5. *Temps de cycle pour une capacité de file d'attente=4.*

Le tableau 4 et le tableau 5 montrent que pour une capacité de file d'attente égale à 2 et 4 (Système saturé) l'algorithme mémétique n'a pas pu améliorer le temps du cycle du système, ceci peut être interprété par l'augmentation du temps du cycle.

5.3. *Les en-cours*

Taux de création	1/40	1/35	1/30	1/25	1/20	1/15	1/10	1/5
DMMmodifiée	6.79	12.95	15.79	19.01	17.84	18.19	16.86	18.12
GA	3.9	4.04	4.46	5.23	7.45	8.9	8.85	8.85
ACO	3.9	4.04	4.45	5.21	7.29	8.4	8.44	8.46
MA	3.91	4.03	4.48	5.22	7.20	9.00	9.00	9.25

Tableau 6. *Les en-cours pour une capacité de file d'attente=2.*

Le tableau 6 montre que les résultats obtenus pour l'algorithme mimétique ne sont pas meilleurs que ceux obtenus par les deux autres métaheuristiques, ceci revient à l'augmentation du taux de production, c'est-à-dire, le nombre de pièces présentes dans le système pour l'algorithme mimétique sont plus importantes que ceux présentes pour le cas des autres méthodes. D'autre part l'algorithme mimétique reste plus performant que la DMMmodifiée.

6. Conclusion

Dans cet article nous nous sommes intéressés à l'adaptation d'une nouvelle métaheuristique pour la résolution de sélection de routages alternatifs dans un Job Shop, en comparant cette métaheuristique avec celle de l'algorithme génétique on remarque que l'introduction de la recherche locale a amélioré les résultats trouvés par l'algorithme génétique.

Toutes les métaheuristiques s'appuient sur un équilibre entre l'intensification de la recherche et la diversification de celle-ci. D'un coté, l'intensification permet de rechercher des solutions de plus grande qualité en s'appuyant sur les solutions déjà trouvées et de l'autre, la diversification met en place des stratégies qui permettent d'explorer un grand espace de solutions et d'échapper à des minima locaux.

Dans l'algorithme mémétique, l'intensification est produite de toute évidence par l'application nouvelle de la recherche locale. L'opérateur de mutation assure la diversification de la méthode.

L'hybridation des métaheuristiques avec des méthodes de recherche locale, des méthodes exactes, ou d'autres métaheuistiques peut augmenter leur efficacité.

7. Bibliographie

Ghomri L., Influence des contraintes et des perturbations sur les performances des règles de routages dans un FMS, *la conférence internationale Conception et Production Intégrées*, 2007.

Graham, R.L., Lawler, E.L., Rinooy Kan, optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5, pp.287-326.

Hassam A., Manipulation des routages alternatifs en temps réel dans les systèmes flexibles de production, Thèse de Magister, Université Abou Bekr Belkaid Tlemcen, 2006.

Mahmoodi F.MOSIER C.T., The effects of sheduling rules and routing flexibility on the performace of a random flexible manufacturing system, *The international journal of flexible manufacturing systems*, p. 271-289.

Moscato P., On evolution, search, optimization, genetic algorithms and material arts: Towards memetic algorithms. *Technical Report C3P 826, Caltech Concurrent Computation Program*, 1989.

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*, 2001, 18, p. 755-763.

SAYGIN C. KILICK S.E., Integrating flexible manufacturing systems with sheduling in flexible manufacturing system. *International journal of advanced manufacturing technology*, 15(4), p. 268-280.1999.

Sevaux M., Métaheuristiques stratégies pour l'optimisation de la production de biens et de services, Habilitation à diriger des recherches, Préparée au Laboratoire d'Automatique, de Mécanique d'informatique Industrielles et Humaines du CNRS (UMR CNRS 8530) dans l'´equipe Systèmes de Production, 2004.

Souier M., Métaheuristiques pour la manipulation de routages alternatifs en temps réel dans un jobshop, Thèse du magister, Université Abou Bekr Belkaid Tlemcen, 2009.

Souier M., Hassam. A, Sari Z. Evaluation of metaheuristics in manipulation of alternative routing, *International conference on electrical engineering design and technologies*, Hammamet Tunisia, 2008.

Souier M., Hassam A., Sari Z., Metaheuristics for real time routing selection in FMS, Book chapter in: Lyes Benyoucef and Bernard Grabot (ED.), *Artificial intelligence techniques for networked manufacturing enterprises management*, Springer-Verlag, ch. pp.221-247.ISNB 978-1-84996-118-9. 2010.

Tsubone H. Horikawa M.,Comparison between Machine flexibility and routing Flexibility, *The international journal of flaxible manufacturing systems*, 11, p. 83-101.

Résumé

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

Notre étude se situe dans le cadre d'adaptation des métaheuristiques pour la résolution d'un problème d'ordonnancement dans un système flexible de production (FMS). La première métaheuristique est l'algorithme mimétique, et la deuxième est l'algorithme API basé sur le comportement de fourragement d'une espèce de fourmis primitives dites les Pachycondyla apicalis. Dans ce deuxième algorithme, nous avons introduit une petite modification qui a conduit à des changements majeurs dans les résultats obtenus. L'algorithme obtenu est ensuite nommé API_m.

Les résultats obtenus par les deux algorithmes sont comparés à ceux obtenus par l'algorithme génétique (GA) et par l'algorithme des colonies de fourmis (ACO). Les résultats montrent que l'algorithme mimétique donne, en général, les meilleurs résultats comparé aux autres métaheuristiques. L'algorithme API_m donne de bons résultats comparé à l'ACO.

Abstract :

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

Our study is devoted to the adaptation of two optimization algorithms to solve the scheduling problem in a flexible manufacturing system (FMS). The first metaheuristic is the Mimetic Algorithm (MA), and the second is the API algorithm based on the foraging behavior model of primitive ants' population called Pachycondyla apicalis. In this second algorithm we have introduced a minor modification which gave great changes in obtained results. Modified algorithm is then called API_m. In addition, we performed a sensitivity analysis of the API_m algorithm.

Results obtained of both algorithms (MA and API_m) are compared to those obtained by the Genetic Algorithm (GA) and Ant Colony Optimization algorithm (ACO). Results show that, generally, mimetic algorithm gave the best results compared to other metaheuristics. API_m gave good results compared to ACO.

ملخص

إشكاليات الترتيبات في الغالب تصنف ضمن الإشكاليات غير متعددات الحدود الصعبة.

حلها يتطلب طرقا مخصصة لدرجة صعوبتها. لهذا الغرض تم تصميم العديد من طرق الاستدلالات و فوقيات الاستدلالات. دراستنا هذه تصنف في إطار تكييف فوقيات الاستدلالات لحل إشكالية الترتيبات في نظام عمل مرن.

فوقية الاستدلال الأولى هي خوارزمية المثلية و الثانية هي الخوارزمية API التي ترتكز على سلوكيات البحث عن الطعام لدى فصيلة من النمل البدائي تسمى "باشيكونديلا ابيكاليس". في هذه الخوارزمية أدخلنا تعديلا صغيرا احدث تغييرات معتبرة في النتائج المحصل عليها. الخوارزمية المعدلة تدعى API_m

النتائج المحصل عليها من كلتا الخوارزميتين قورنت مع تلك التي تم التوصل إليها باستعمال الخوارزميات الجينية و خوارزمية مستعمرات النمل.

النتائج تثبت أن الخوارزمية المثلية تعطي إجماليا أحسن النتائج مقارنة بالخوارزميات الأخرى. الخوارزمية API_m أعطت نتائج جيدة بالمقارنة مع خوارزمية مستعمرات النمل.

Mots clés : Métaheuristiques, Ordonnancement, Systèmes flexibles de production, Algorithme mimétique, Pachycondyla apicalis, Algorithme API.

Key words: Metaheuristics, Scheduling, Flexible Manufacturing Systems, Mimetic Algorithm, Pachycondyla apicalis, API algorithm.

الكلمات المفتاحية

فوقيات الاستدلال- الترتيبات- ورشات الإنتاج المرنة- خوارزمية المثلية- باشيكونديلا ابيكاليس- الخوارزمية API