Introduction générale

INTRODUCTION GENERALE

Les problèmes d'ordonnancement formulés en problèmes d'optimisation sont souvent classés NP-Difficiles, en particulier ceux liés aux systèmes de production. La résolution de tels problèmes nécessitent des méthodes dédiées, tandis que les méthodes exactes ne peuvent pas résoudre ce types de problèmes vu le temps de calcul énorme les méthodes approchées offrent la possibilité de trouver une solution réalisable en un temps raisonnable.

Parmi les méthodes approchées les plus utilisées pour la résolution des problèmes NP-Difficiles on trouve les métaheuristiques. Durant des années plusieurs métaheuuristiques ont été construites et appliquées pour lé résolution de tels problèmes.

Ce travail s'inscrit dans le cadre d'utilisation et d'hybridation de métaheuristiques pour la résolution de problèmes Flow Shop avec minimisation de makespan. Notre contribution consiste à l'adaptation, la programmation et l'hybridation de l'algorithme BCO (Algorithme de colonies d'abeilles) avec une méthode de recherche locale.

Le reste de ce document est organisé comme suit :

Le chapitre I est consacré un la présentation des notions générales liées à notre travail, à savoir, les données des systèmes de production, leur classification, les problèmes liés aux systèmes de production et leur complexité.

Le chapitre II a pour but d'introduire les différentes techniques de résolution des problèmes d'optimisation des problèmes d'ordonnancement, commençant par les méthodes exactes allant aux heuristiques et métaheuristiques, avec la présentation des principes et des algorithmes de certaines techniques.

Le chapitre II est consacré à notre contribution proprement dite, nous présentons l'adaptation de l'algorithme des colonies d'abeilles auquel nous intéressons ainsi que la méthode de recherche locale utilisée et l'hybridation des deux techniques. Nous présentons par la suite les différents résultats de simulations avec leur interprétation.

Nous finirons ce document par une conclusion générale qui résumé les différents phases de travail et ouvre la porte devant certaines perspectives.

CHAPITRE I

Généralités sur les problèmes d’ordonnancement

I.1) Introduction

L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de la production qui vise à améliorer l'efficacité d'une entreprise en termes de coûts de production et de délais de livraison. Les problèmes d'ordonnancement sont présents dans tous les secteurs d'activités de l'économie, depuis l'industrie manufacturière [1]jusqu'à l'informatique[2].

La théorie de l'ordonnancement traite des modèles mathématiques mais analyse également des situations réelles fortes complexes.L'ordonnancement est lié à plusieurssecteurs de recherches et d'activités très variés. En informatique, le choix des tâches à envoyer aux processeurs se modélise comme un problème d'ordonnancement. En production,l'ordonnancement consiste à déterminer les séquences d'opérations à réaliser sur les différentes machines de l'atelier. En gestion de projet, ordonnancer, c'est déterminer lesdates d'exécution des activités constituant le projet. Chacun de ces contextes nécessite ladétermination des caractéristiques propres des tâches et des ressources, le type des décisions à prendre, les modalités d'exécution des tâches par les ressources, qui déterminentdes contraintes sur les décisions et aussi les différents critères à optimiser.

Dans ce chapitre,on s'intéresse ici uniquement aux problèmesd'ordonnancement dans les entreprises manufacturières. Il constitue un rappel de quelques notions de base relatives aux problèmes d'ordonnancement. Il rappelle aussi quelques concepts sur la théorie de la complexité des problèmes d'ordonnancement

I.2) Généralités sur l'ordonnancement

Ordonnancer le fonctionnement d'un système industriel de production consiste à gérer l'allocation des ressources au cours du temps, tout en optimisant au mieux un ensemble de critères [3]. C'est aussi programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution[4].

Ordonnancer peut également consister à programmer l'exécution des opérations en leur allouant les ressources requises et en fixant leurs dates de début de fabrication. D'une manière plus simple, un problème d'ordonnancement consiste à affecter des tâches à des moyens defabrication au cours du temps pour effectuer un ensemble de travaux demanière à optimiser certain(s) critère(s), tout en respectant les contraintestechniques de fabrication.

L'ordonnancement se déroule en trois étapes qui sont:

- La planification, qui vise à déterminer les différentes opérations à réaliser, les datescorrespondantes, et les moyens matériels et humains à y affecter.
- l'exécution, qui consiste à mettre en œuvre les différentes opérations définies dans laphase de planification.
- le contrôle, qui consiste à effectuer une comparaison entre planification et exécution,soit au niveau des coûts, soit au niveau des dates de réalisation.

Ainsi, le résultat d'un ordonnancement est un calendrier précis de tâches à réaliser qui sedécompose en trois importantes caractéristiques :

- l'affectation, qui attribue les ressources nécessaires aux tâches,
- le séquencement, qui indique l'ordre de passage des tâches sur les ressources,
- le datage, qui indique les temps de début et de fin d'exécution des tâches sur lesressources.

La solution d'un problème d'ordonnancement général doit répondre à deux questions:

- quand?
- avec quels moyens ?

Une solution répondant à ces questions est appelée ordonnancement. Uneméthode permettant de construire un ordonnancement est appelée algorithmeou méthode de résolution. Un ordonnancement réalisable est unordonnancement qui respecte toutes les contraintes du problème. Nousutilisons le terme "ordonnancement", pour simplifier, pour représenter unordonnancement réalisable.

I.3) Les données d'un problème d'ordonnancement

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

Ainsi, étant donnés un ensemble de tâches et un ensemble de ressources, il s'agit de programmer les tâches et affecter les ressources de façon à optimiser un ou plusieurs objectifs (un objectif correspondant à un critère de performance), en respectant un ensemble de contraintes.

I.3.1) Les tâches

Une tâche est une entité élémentaire de travail localisée dans le temps par une date de début et une date de fin d'exécution et qui consomme des ressourcesavec des quantités déterminées. Un

coût (ou poids) est attribué à une tâche pourestimer sa priorité, son degré d'urgence, ou son coût d'immobilisation dans lesystème.

On distingue deux types de tâches :

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

On note en général $N= \{J_1, J_2,\ldots, J_n\}$ l'ensemble des tâches, chaque tâche est caractérisée par :

- La durée opératoire de la tâche i sur la machine j : (p_{ij}).
- La date de disponibilité de la tâche i : (r_i).
- La date de début d'exécution de la tâche i :(s_i).
- Ladate de fin d'exécution de la tâche i :(C_i).
- Ladate d'achèvement souhaitée de la tâche i :(d_i).
- Le facteur de priorité ou poids de la tâche i : (w_i).
- Le retard algébrique de la tâche i : ($L_i = C_i – d_i$).
- Le retard vrai de la tâche i : ($T_i = \max (C_i – d_i, 0)$).
- L'indicateur de retard de la tâche i : ($U_i = 1$, si $T_i>0$, $U_i = 0$, sinon.

I.3.2) Les ressources

Une ressource est un moyen technique ou humain, destiné à être utilisé pourla réalisation d'une tâche et disponible en quantité limitée [5]. On note en général $M= \{M_1, M_2,\ldots, M_m\}$ l'ensemble des ressources. Plusieurstypes de ressources sont à distinguer.

I.3.2.1) Les ressources renouvelables

Une ressource est dite renouvelable, si après avoir été utilisée par une tâcheou allouée à une tâche, elle redevient disponible pour les autres tâches en mêmequantité. La quantité disponible est renouvelée d'une tâche à une autre, elle peutêtre constante, comme elle peut varier d'une tâche à une autre. Exemples deressources renouvelables : les machines, les hommes, l'équipement, lesprocesseurs, les fichiers…

On distingue deux types de ressources renouvelables :

- les ressources disjonctives (ou non partageables) : qui ne peuventexécuter qu'une tâche à la fois (machine, robot) ;

- les ressources cumulatives (ou partageables) : qui peuvent êtreutilisées par plusieurs tâches simultanément (équipe d'ouvriers).

I.3.2.2) Les ressources non renouvelables :

Une ressource est non renouvelable (ou consommable), si après avoir étéutilisée par une tâche, elle n'est plus disponible pour les autres tâches. Laconsommation globale en cours du temps est limitée, on dit que la ressource estépuisée en l'utilisant. Exemples de ressources consommables : le capital (oubudget), le carburant, l'énergie dans une batterie, la matière première,…

Les ressources non renouvelables sont généralement produites dans un système indépendant de la machine sur laquelle les tâches sollicitant cetteressource sont exécutées. Toutefois, certains problèmes peuvent exister oùcertaines tâches produisent des ressources qui peuvent être consommées plus tardpar d'autres tâches.

Les ressources non renouvelables peuvent également être stockées dans unou plusieurs dépôts (entrepôts, magasins ou warehouses) ayant une capacité destockage. Les ressources peuvent éventuellement avoir un stock initial dans un ouplusieurs dépôts.

Durant la consommation des ressources, la condition générale qui doittoujours être satisfaite est que la quantité totale de la ressource demandée par lestâches ne doit pas dépasser la quantité totale disponible.

I.3.3) Les contraintes

Suivant la disponibilité des ressources et suivant l'évolution temporelle, deux types decontraintes peuvent être distingués[4] : contraintes de ressources etcontraintes temporelles.

- les contraintes de ressources : plusieurs types de contraintes peuvent être induits parla nature des ressources. A titre d'exemple, la capacité limitée d'une ressourceimplique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cetteressource.

Les contraintes relatives aux ressources peuvent être disjonctives, induisant unecontrainte de réalisation des tâches sur des intervalles temporels disjoints pour unemême ressource, ou cumulatives impliquant la limitation du nombre de tâches àréaliser en parallèle.

- les contraintes temporelles : elles représentent des restrictions sur les valeurs quepeuvent prendre certaines variables temporelles d'ordonnancement. Ces contraintespeuvent être :
- des contraintes de dates butoirs, certaines tâches doivent être achevées avantune date préalablement fixée, des contraintes de précédence, une tâche i doit précéder la tâche j,

- des contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteursnécessaires pour commencer l'exécution des tâches.

I.3.4) Les critères

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettantd'évaluer la qualité de l'ordonnancement établi.Les critères dépendant d'une application donnée sont très nombreux; plusieurs critèrespeuvent être retenus pour une même application. Le choix de la solution la plus satisfaisantedépend du ou des critères préalablement définis, pouvant être classés suivant deux types,réguliers et irréguliers.

Les différents critères ne sont pas indépendants; certains même sont équivalents. Deuxcritères sont équivalents si une solution optimale pour l'un est aussi optimale pour l'autre etinversement[4].

Les critères réguliers constituent des fonctions décroissantes des dates d'achèvementdes opérations. Quelques exemples sont cités ci-dessous:

- la minimisation des dates d'achèvement des actions,
- la minimisation du maximum des dates d'achèvement des actions,
- la minimisation de la moyenne des dates d'achèvement des actions,
- la minimisation des retards sur les dates d'achèvement des actions,
- la minimisation du maximum des retards sur les dates d'achèvement des actions.

Les critères irréguliers sont des critères non réguliers, c'est-à-dire qui ne sont pas desfonctions monotones des dates de fin d'exécution des opérations, tels que:

- la minimisation des encours,
- la minimisation du coût de stockage des matières premières,
- l'équilibrage des charges des machines,
- l'optimisation des changements d'outils.

La satisfaction de tous les critères à la fois est souvent délicate, car elle conduit souvent à dessituations contradictoires [6] et à la recherche de solutions à des problèmescomplexes d'optimisation.

Les critères lesplus utilisés font intervenir la durée totale, les retards et le coût des stocks desencours.

Minimiser la durée totale : La durée totale de l'ordonnancement notée Cmax est égale à la date d'achèvement de la tâche la plus tardive : $Cmax = max\,(Ci)$. Minimiser la durée totale, revient à minimiser Cmax c'est-à dire$min\,(max\,(Ci))$.

Minimiser les retards : On peut rencontrer plusieurs problèmes dans lesquels il faut respecter les délais di. Les critères retenus sont les retards vrais $T_{max} = \max(T_i)$ et les retards algébriques $L_{max} = \max(L_i)$. Parfois on retient le nombre de tâches en retard ΣU_i

Minimiser les encours : Les encours sont déterminés par le temps de présence des tâches dans le système : $F_i = C_i - r_i$. Le critère essentiel est de minimiser la somme de ces temps $\Sigma_{i=1}^{n} Fi$. Lorsqu'on tient compte des coûts de présence des tâches dans le système ou de leur importance, on attache un poids wi au temps de présence, le critère devient$\Sigma_{i=1}^{n} w_i Fi$. Puisque les dates de disponibilité ri sont fixées (constantes), on retient seulement le critère $\Sigma_{i=1}^{n} Ci$ (somme de la date de fin d'exécution) et le critère $\Sigma_{i=1}^{n} w_i Ci$ (somme pondérée des dates de fin d'exécution).

I.4) Classification des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont généralement classés en deux principaux modèles dépendamment du nombre d'opérations que requièrent les jobs: des modèles à une opération (machine unique et machines parallèles) et des modèles à plusieurs opérations (flow-shop, open shop et job shop).

I.4.1) Modèles à une opération

I.4.1.1) Modèle à machine unique

Dans un modèle à machine unique, l'ensemble des tâches à réaliser est exécuté par une seule machine. L'une des situations intéressantes où on peut rencontrer ce genre de configurations est le cas où on est devant un système de production comprenant une machine goulot qui influence l'ensemble du processus. Ce modèle est illustré dans la fig 1.1.

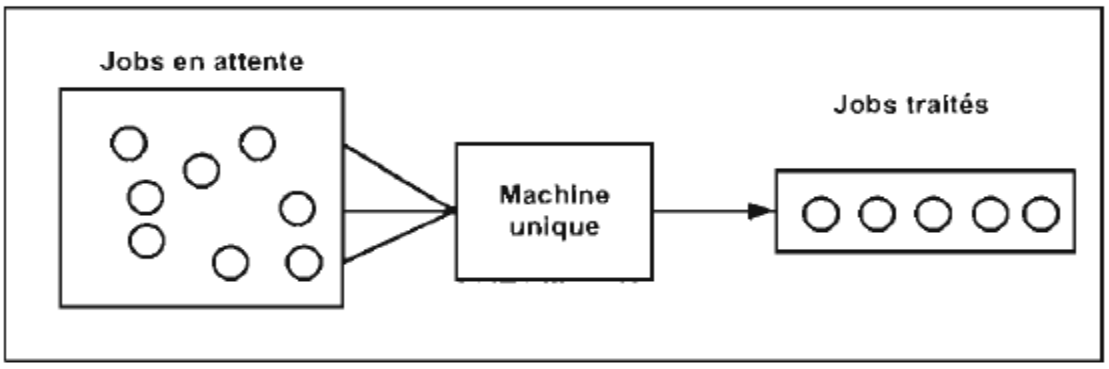

Fig 1.1 : Modèle à machine unique.

I.4.1.2) Modèle à machines parallèle

Le deuxième modèle est le modèle à machine parallèle. Ce modèle est utilisé surtout dans les secteurs industriels tels que: l'industrie alimentaire, les industries plastiques, les fonderies et en

particulier l'industrie textile. Le processus de déroulement de ce système de production est le suivant: à chaque fois qu'une machine i se libère, ont lui affecte un job j comme illustré à la Figure 1.2. Dans le cas d'un processus d'assemblage industriel, par exemple, si l'une des étapes d'assemblage nécessite beaucoup de temps, il serait très intéressant alors d'avoir plusieurs machines parallèles qui effectuent la même tâche. D'un autre côté, les machines parallèles sont classées suivant leur rapidité. Si toutes les machines de l'ensemble ont la même vitesse de traitement et effectuent les mêmes tâches, elles sont identiques. Si les machines ont des vitesses de traitement différentes mais linéaires alors elles sont dites uniformes. Dans le cas où les vitesses des machines sont indépendantes les unes des autres, on parle alors de modèle de machines parallèles non reliées ou indépendantes.

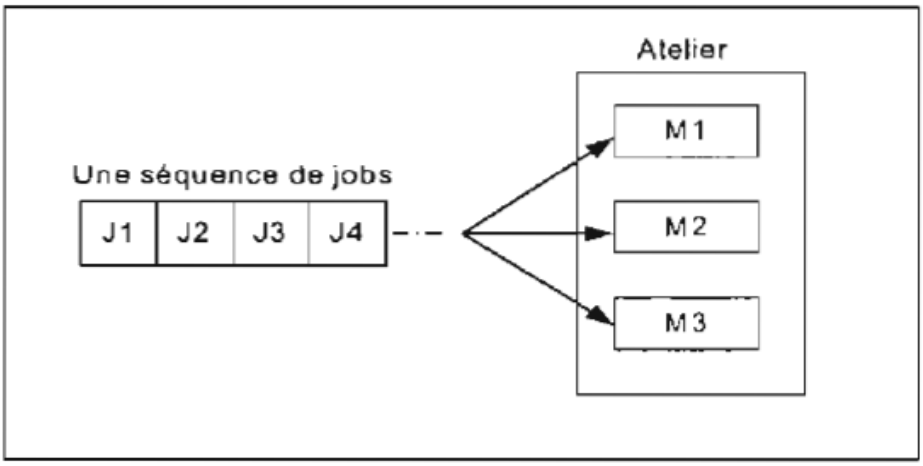

Fig 1.2 : Modèle à machines parallèle.

I.4.2) Modèles à plusieurs opérations

Le modèle à plusieurs opérations est constitué des cas où un job, pour se réaliser, doit passer par plusieurs machines, chacune de ces machines ayant ses spécificités. On distingue trois modèles selon l'ordre de passage des jobs sur les machines, à savoir les modèles de flow-shop, job-shop et open-shop.

I.4.2.1) Modèle flow-shop

Dans le modèle de flow-shop, les ordres de fabrication visitent les machines dans le même ordre, avec des durées opératoires pouvant être différentes. Chaque job va être s'exécuter sur les M machines en série et tous les jobs vont suivre le même ordre de passage sur ces machines. Ce type de modèle est aussi appelé modèle linéaire. La figure 1.3 illustre le cas d'un flow-shop avec quatre machines et quatre jobs. Les quatre jobs suivent le même ordre de traitement sur les quatre machines.

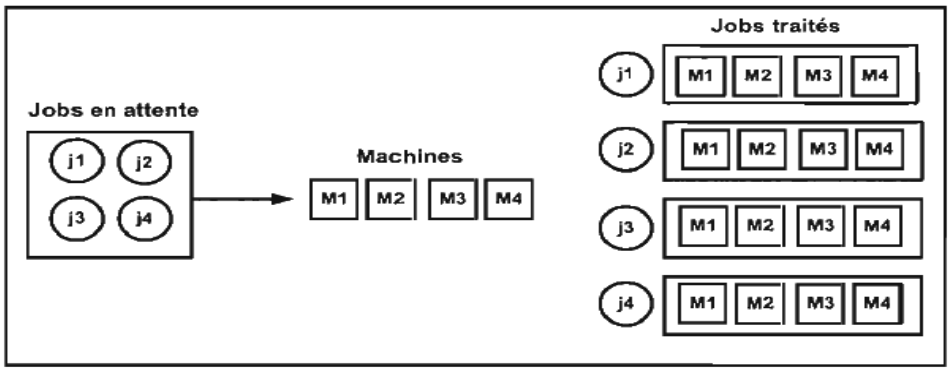

Fig 1.3 : Modèle flow-shop.

I.4.2.2) Modèle job-shop

Concernant le modèle de job-shop, chaque job à un ordre à suivre et chacun d'entre eux peut s'exécuter plusieurs fois sur la même machine; ce qui n'est pas le cas du flow-shop. Il s'agit dans ce cas de déterminer les dates de passage sur différentes ressources d'ordres de fabrication ayant des trajets différents dans l'atelier. Ces ordres de fabrication partageant des ressources communes, des conflits sont susceptibles de survenir, résultant des croisements de flux illustrés à la Figure 1.4. Dans cette figure, l'ordre de passage de chaque job est indiqué par un chemin de même couleur que celui du job. Par exemple, le job J_1 passe par toutes les machines, alors que le job J_3 ne passe que par la deuxième et la quatrième machine. Dans son expression la plus simple, le problème consiste à gérer ces conflits tout en respectant les contraintes données, et en optimisant les objectifs poursuivis. Les types de ressources et de contraintes prises en compte peuvent toutefois considérablement compliquer le problème. Plus on intégrera de contraintes, plus on se rapprochera d'un cas réel, mais moins on disposera de méthodes de résolution satisfaisantes.

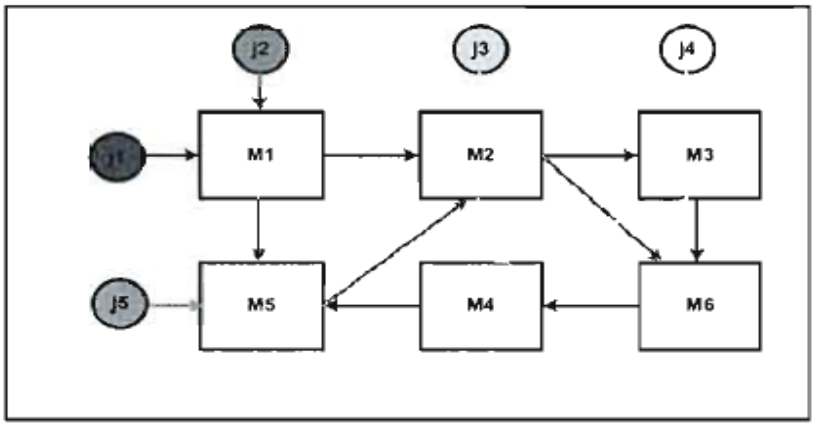

Fig 1.4 : Modèle job-shop.

I.4.2.3) Modèle open-shop

Dans le modèle d'open shop, l'ordre de passage des n jobs sur les m machines n'est pas connu à l'avance. Cet ordre est déterminé lors de la construction de la solution. Chaque job j peut avoir son propre ordre de passage sur toutes les machines. Le fait qu'il n'y ait pas d'ordre prédéterminé rend la résolution du problème d'ordonnancement de ce type plus complexe, mais offre cependant des degrés de liberté intéressants. À la Figure 1.5, nous avons un ensemble de quatre jobs et un ensemble de quatre machines. À droite de la figure nous pouvons remarquer que chaque job a suivi un ordre de passage différent sur les quatre machines.

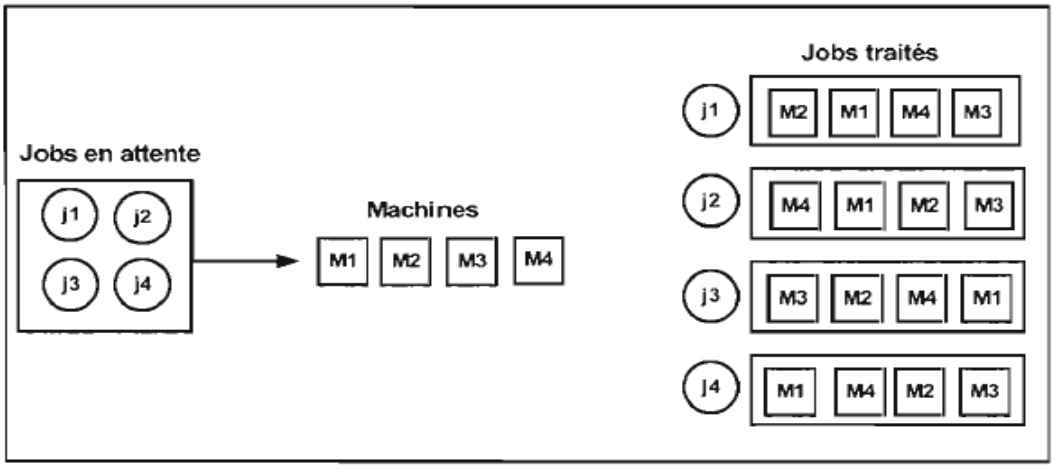

Fig 1.5 : Modèle open-shop.

I.6) Notion de complexité de problèmes

La théorie de la complexité a pour but d'apporter des informations sur la difficultéthéorique d'un problème à résoudre. Elle permet de classer "du point de vue mathématique"les problèmes selon leur difficulté [7].

La complexité analyse le temps nécessaire pour obtenir une solution (on peutégalement s'intéresser à la mémoire nécessaire, mais nous ne nousintéresserons pas ici à cet aspect de la complexité). On peut considérer la duréemoyenne ou la durée dans le pire des cas. Nous traitons essentiellement cedernier cas.

A première vue, comment définir un algorithme efficace ? Pour un problème donné,chercher un algorithme efficace, veut dire trouver un algorithme où le temps nécessaire à son exécution ne soit pas trop important.Un problème est dit facile si on peut le résoudre facilement, c'est-à-dire s'il ne fait pastrop de temps pour arriver à la solution. Donc, s'il existe un algorithme efficace pour unproblème donné, alors ce dernier est dit facile.Un problème pour lequel on ne connaît pas d'algorithme efficace, alors ce dernier est dit difficile.

Pour résoudre un problème d'ordonnancement, il ne suffit pas de prouver l'existenced'une solution, il faut également la construire, il est clair que construire la solution estplus difficile que de prouver son existence ce qui nous conduit donc à classer les problèmescomme étant difficiles ou faciles.

Les problèmes indécidables sont ceux pour lesquels aucun algorithme, quel qu'il soit, n'aété trouvé pour les résoudre.A l'opposé, les problèmes décidables sont ceux pour lesquels il existe au moins unalgorithme pour les résoudre [8].

I.6.1) La classe NP

La classe NP (Non déterministe Polynomial) est celle des problèmes d'existence dont une proposition de solution est Oui et qui est vérifiable polynomialement. Parmi les problèmes décidables, les plus simples à résoudre sont regroupés dans la classe NP.

I.6.2) La classe P

Un problème est dit polynomial s'il existe un algorithme de complexité polynomialepermettant de répondre à la question posée dans ce problème, quelle que soit la donnée de celui-ci.La classe P est l'ensemble de tous les problèmes de reconnaissance polynomiaux [9]. Pour le reste de la classe NP, on n'est pas sûr qu'il n'existe pas un algorithme polynomialpour résoudre chacun de ses problèmes. Ainsi, on sait que P est inclue dans NP mais on n'a paspu prouver que P n'est pas NP.

I.6.3) La classe NP-Complet

La classe NP-Complet regroupe les problèmes les plus difficiles de la classe NP. Ellecontient les problèmes de la classe NP tels que n'importe quel problème de la classe NP leur estpolynomialement réductible. Entre eux, les problèmes de la classe NP-Complet sont aussi difficiles.

I.6.4) La classe NP-Difficile

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

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

I.7) Problème d'optimisation

Un problème d'optimisation se définit comme la recherche, parmi un ensemble de solutionspossibles S (appelé aussi espace de décision ou espace de recherche), la (ou des) solution(s)x^*qui rend(ent) minimale (ou maximale) une fonction mesurant la qualité de cette solution.Cette fonction est appelée fonction objectif ou fonction coût. Si l'on pose $f : S \rightarrow \mathbb{R}$ la fonctionobjectif à minimiser (respectivement à maximiser) à valeurs dans$\mathbb{R}$, le problème revient alorsà trouver l'optimum $x^* \in S$ tel que $f(x^*)$ soit minimal (respectivement maximal).

Lorsque l'on veut résoudre un problème d'optimisation, on recherche la meilleure solutionpossible à ce problème, c'est-à-dire l'optimum global. Cependant, il peut exister des solutionsintermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreintde l'espace de recherche : on parle alors d'optimums locaux. Cette notion est illustrée dans la figure 1.6. La seule hypothèse faite sur S est qu'il s'agit d'un espace topologique, *i.e.*sur lequel est définie une notion de voisinage. Cette hypothèse est nécessaire pour définir lanotion de solutions locales du problème d'optimisation. On peut alors définir un optimum local(relativement au voisinage V) comme la solution $x*$ de S telle que $f(x*) \leq f(x); \forall x \in V(x*)$.

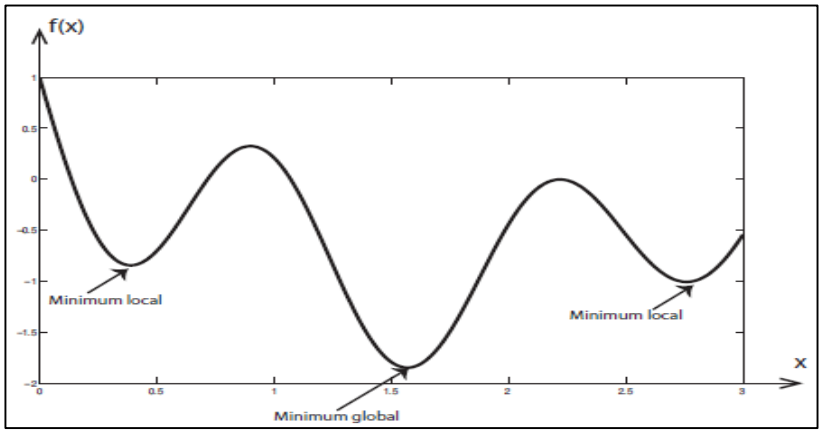

Figure 1.6: Différence entre un optimum global et des optima locaux.

I.8) Conclusion

L'ordonnancement est généralement décrit comme une fonction particulière de décision au sein d'un système de gestion du travail concernant la production de bien, d'ouvrages ou de services.

La majorité des problèmes d'ordonnancement sont NP-difficile, ça veut dire que, dans la pratique, la complexité croît exponentiellement avec le nombre de tâches et de ressources. Il n'est, donc, pas envisageable de résoudre de tels problèmes avec les méthodes exactes. C'est pour cela qu'ilfaut développer des nouveaux méthodes qui donnent des solutions certes sous-optimales, maisobtenues rapidement. dont l'objectif est de fournir des solutions aussi proches quepossible de la solution exacte en un temps raisonnable.

CHAPITRE II

Méthodes de résolution des problèmes d’optimisation

II.1) Introduction

Nous avons vu précédemment qu'il existe des problèmes d'ordonnancementde complexités différentes. Les problèmes appartenant à la classe P ont desalgorithmes polynomiaux permettant de les résoudre.Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes derésolution sont largement utilisées pourappréhender les problèmes NP-difficiles.

Dans ce chapitre, nous exposons en générale les méthodes de résolution des problèmes d'optimisation qui sont classées en deux grandes catégories : les méthodesexactes et les méthodes approchées. On explique brièvement les méthodes les plus connues dans chaque catégorie.

II.2) Les méthodes d'optimisation

Comme on l'a rappelé dans le paragraphe sur la complexité, la plupart desproblèmes d'ordonnancement sont NP-difficiles. On ne peut donc pas espérertrouver un ordonnancement optimal en un temps raisonnable pour desproblèmes de taille industrielle. Les méthodes utilisées sont donc des méthodesapprochées.

Pour des problèmes de petite taille, nous pouvons obtenir une solution exacte.Une méthode exacte peut servir pour résoudre des sous-problèmes d'unproblème de grande taille de manière optimale.

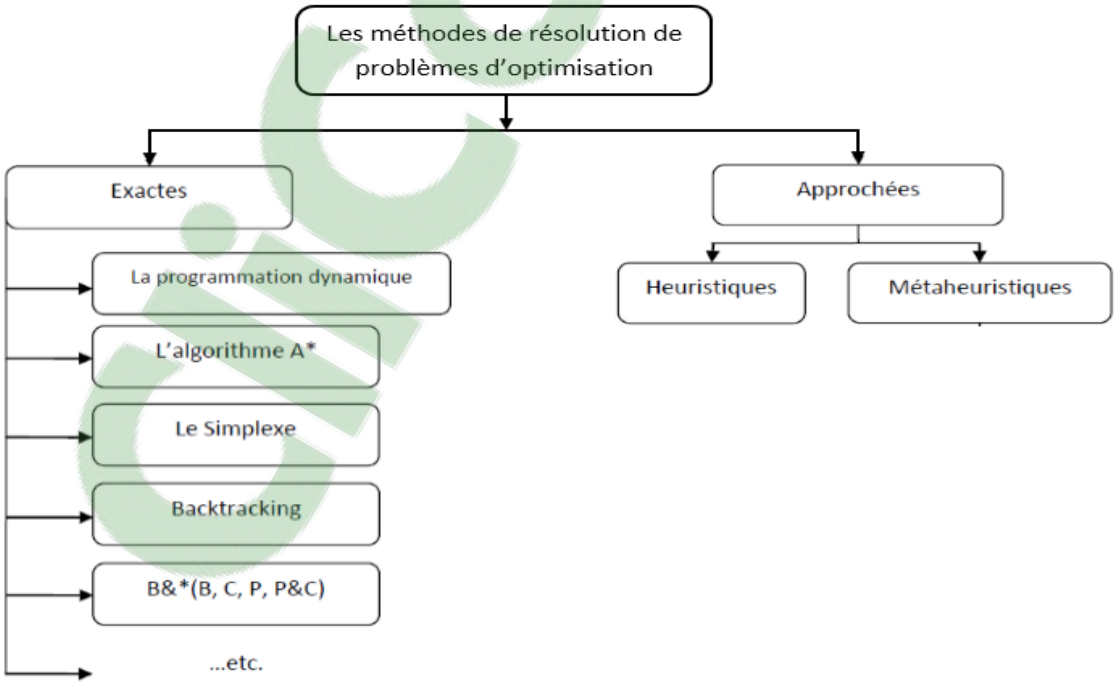

Fig 2.1 : Classification de méthodes de résolution de problèmes d'optimisation

II.2.1) Les méthodes exactes

Les méthodes exactes utilisent surtout deux approches de résolution trèsconnues: la programmation dynamique et les procédures par séparation etévaluation. Ces méthodes sont souvent utilisées pourrésoudre les problèmes combinatoires de manière exacte, en ordonnancementtout particulièrement. Ce sont des méthodes d'énumération implicite:

L'énumération explicite construit toutes les solutions réalisables et retient uneparmi les meilleures.L'énumération implicite consiste à explorer l'ensemble detoutes les solutionsréalisables en éliminant des sous-ensembles de solutionsmoins intéressantssans avoir à les construire. Nous pouvons citertrois approches particulièrement célèbres : La programmation dynamique introduite par Bellman dans les années 50^4 , la méthode par séparation et évaluation (Branch and Bound en anglais : notée B&B) et l'algorithme de retour arrière (Backtracking).

II.2.2) Les méthodes approchées

Une méthode approchée est une méthode d'optimisation qui a pour but de trouver une solution réalisable de la fonction objectif en un temps raisonnable, mais sans garantie d'optimalité. L'avantage principal de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, D'un autre côté les algorithmes d'optimisation tels que les algorithmes de recuit simulé, les algorithmes tabous et lesalgorithmes génétiques ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires.

Les méthodes approchées englobent deux classes : les heuristiques et les métaeuristiques. La particularité qui différencie les méthodes métaheuristiques des méthodes heuristiques c'est que les métaheuristiques sont applicables sur de nombreux problèmes. Tandis que, les heuristiques sont spécifiques à un problème donné.

II.2.2.1) Les heuristiques

Les méthodes heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures. Plusieurs définitions des heuristiques ont été proposées par plusieurs chercheurs dans la littérature, parmi lesquelles:

Définition 2.1. « Une heuristique (règle heuristique, méthode heuristique) est une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre type de dispositif qui limite considérablement la recherche de solutions dans des espaces problématiques importants. Les

heuristiques ne garantissent pas des solutions optimales. En fait, elles ne garantissent pas une solution du tout. Tout ce qui peut être dit d'une heuristique utile, c'est qu'elle propose des solutions qui sont assez bonnes la plupart du temps. »[10].

Définition 2.2. « Une méthode heuristique (ou simplement une heuristique) est une méthode qui aide à découvrir la solution d'un problème en faisant des conjectures plausibles mais faillible de ce qui est la meilleure chose à faire. »[11].

Définition 2.3. « Les heuristiques sont des ensembles de règles empiriques ou des stratégies qui fonctionnent, en effet, comme des règles d'estimation. »[12].

II.2.2.2) Les métaheuristiques

Les métaheuristiques d'optimisation sont des algorithmes généraux d'optimisation applicables à une grande variété de problèmes. Elles sont apparues à partir des années 80, dans le but de résoudre au mieux des problèmes d'optimisation. Les métaheuristiques s'efforcent de résoudre tout type de problème d'optimisation. Elles sont caractérisées par leur caractère stochastique, ainsi que par leur origine discrète.

Elles sont inspirées par des analogies avec la physique (recuit simulé, recuit micro-canonique), avec la biologie (algorithmes évolutionnaires) ou encore l'éthologie (colonies de fourmis, essaims particulaires). Cependant, elles ont l'inconvénient d'avoir plusieurs paramètres à régler. Il est à souligner que les métaheuristiques se prêtent à toutes sortes d'extensions, notamment en optimisation mono-objectif et multi-objectif.

Les métaheuristiques utilisent des recherches stratégiques afin d'explorer plus efficacement l'espace de recherche, et souvent se focalisent sur les régions prometteuses. Ces méthodes commencent par un ensemble de solutions initiales ou une population initiale, et après, elles examinent étape par étape une séquence de solutions pour atteindre, ou de s'approcher de la solution optimale du problème. Les métaheuristiques ont plusieurs avantages par rapport aux algorithmes traditionnels. Les deux avantages les plus importants sont la simplicité et la flexibilité. Les métaheuristiques sont souvent simples à implémenter, pourtant, elles sont capables de résoudre des problèmes complexes avec la capacité de s'adapter à plusieurs problèmes d'optimisation du monde réel, à partir du domaine de la recherche opérationnelle, d'ingénierie vers l'intelligence artificielle.

De nos jours les gestionnaires et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs très divers. Le problème à résoudre peut souvent s'exprimer sous la forme générale d'un problème d'optimisation, dans lequel

on définit une ou plusieurs fonctions objectif que l'on chercheà minimiser ou à maximiser par rapport à tous les paramètres concernés[13].

Le mot métaheuristique est dérivé de la composition de deux mots grecs :

- heuristique qui vient du verbe heuriskein et qui signifie 'trouver'.
- méta qui est un suffixe signifiant 'au -delà', 'dans un niveau supérieur'.

II.2.2.2.1) Les propriétés fondamentales des métaheuristiques

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale.

- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer dessolutions (presque) optimales.

- Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure derecherche locale à des processus d'apprentissage complexes.

- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité

- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régionsde l'espace de recherche.

- Les concepts de base des métaheuristiques peuvent être décrit de manière abstraite, sans faire appel à unproblème spécifique.

- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problèmetraité, mais ces heuristiques sont contrôlées par une stratégie de niveau supérieur.

- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pourmieux guider la suite du processus de recherche.

II.2.2.2.2) Classification des métaheuristiques

On peut regrouper 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.

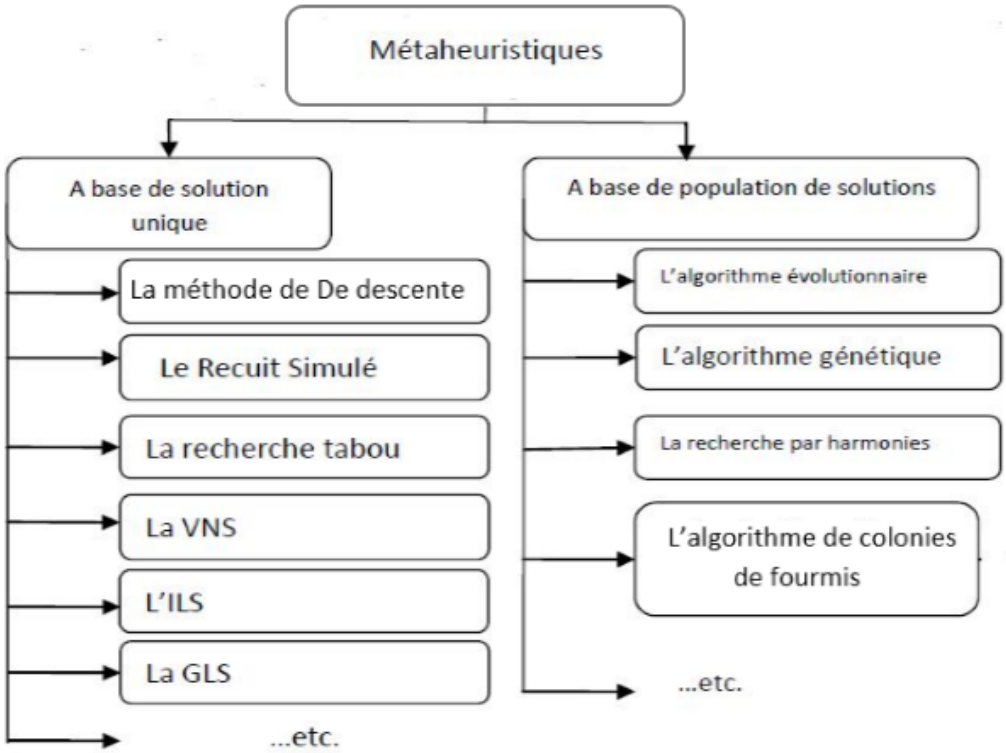

Fig 2.2: Classification des métaheuristiques.

II.2.2.2.3) Notion de paysage

La notion de paysage permet de décrire visuellement l'ensemble des valeurs que peut prendre unefonction objectif pour une paramétrisation donnée. Cette description permet notammentde voir s'il existe potentiellement de nombreux optima locaux. La description d'un paysage se fait enanalogie avec la notion géographique du paysage. Ainsi un paysage peut être décrit comme une vallée,une plaine, un bassin, ... Cela correspond à la forme général. Ensuite, il est possible d'apporter une ,information sur le nombre d'optima locaux existants. Ainsi, un paysage rugueux ou chaotique correspondraplutôt à un paysage possédant un nombre important d'optima locaux. Au contraire, un paysage toutà fait lisse amènera à priori vers un nombre restreint d'optima locaux. La figure 2.3 propose quelquesexemples de forme de paysage.

L'étude du paysage d'une fonction objectif permet de guider le choix de la méthode de résolution àappliquer.

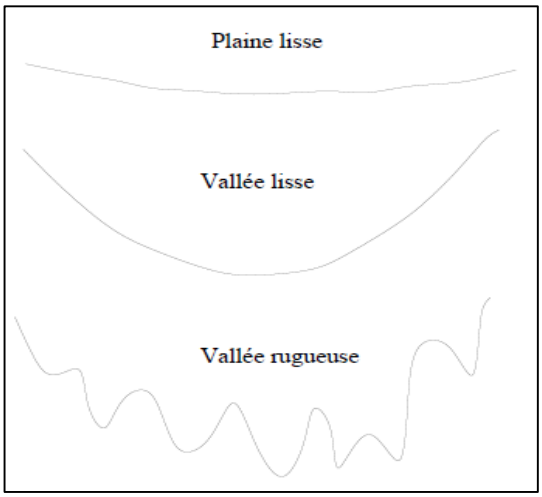

Fig 2.3 : Exemples de forme de paysage.

Le principe de diversification d'une méthode d'optimisation donnée correspond à sa capacité de parcouriraisément l'espace de recherche pour obtenir des solutions très différentes les unes des autres. Dansun paysage donné, cela se traduit par la possibilité de faire des déplacements plutôt horizontaux commecela est montré dans la figure 2.4. Cependant, comme cela est visible sur cette figure, la diversificationseule n'amène pas forcément à des optima locaux mais à des solutions se trouvant potentiellement dansla zone d'un optimum local non encore trouvé jusqu'à maintenant.

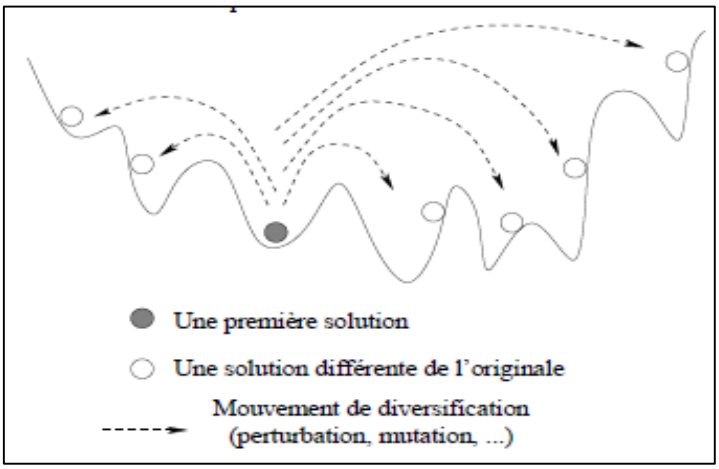

Fig 2.4 : Processus de diversification d'une solution dans un paysage donné.

Autant le principe de diversification essaye de déplacer les solutions dans d'autres zones de l'espacede recherche, autant le processus d'intensification vise à forcer une solution donnée à tendre versl'optimum local de la zone à laquelle elle est attachée.

Lafigure 2.5donne un exemple de processus d'intensification pour des solutions de départ données.Comme on peut le voir, une phase d'intensification correspond plutôt à un mouvement

vertical dans lepaysage pour accéder à un optimum local (peut-être l'optimum global). La force mais aussi la faiblessede l'intensification est de rester cantonnée à la zone dans laquelle la solution se trouve sans possibilitéde pouvoir sortir de celle-ci. Donc si les solutions que l'on veut intensifier ne sont pas dans la zone del'optimum global, il n'y a aucune chance que l'intensification seule permette d'y accéder.

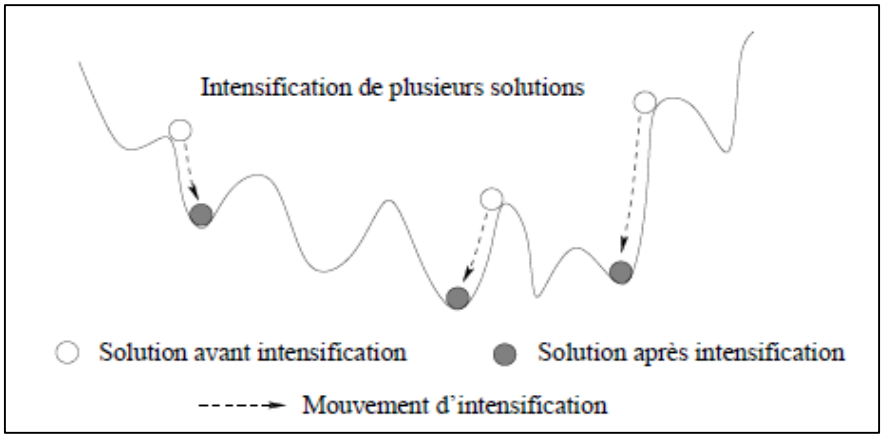

Fig 2.5 : Processus d'intensification de solutions dans un paysage donné.

Ne pas préserver cet équilibre conduit à une convergence trop rapide versdes minima locaux (manque de diversification) ou à une exploration trop longue(manque d'intensification).

Dans les sections qui vont suivre, nous avons essayé, de présenter chaque métaheuristique de façon uniforme. D'abord, nous rappelons, quand c'est possible, les papiers où la méthode a été introduite la première fois. Une descriptiontextuelle de l'algorithme est proposée avec ses particularités. Parfois, il existe des articles plus récents que les articles d'origine qui présentent soit une vue synthétique soit un tutorial détaillé et que nous citons. Ensuite, nous donnons une présentation algorithmique de chaque métaheuristique, avant de montrerquels sont les facteurs d'intensification et de diversification de chacune de ces méthodes.

II.3) Les métaheuristiques à solution unique

Dans cette section, nous présentons les métaheuristiques à base de solution unique, aussi appelées méthodes de trajectoire. Contrairement aux métaheuristiques à base de population, lesmétaheuristiques à solution unique commencent avec une seule solution initiale et s'en éloignentprogressivement, en construisant une trajectoire dans l'espace de recherche. Les méthodes detrajectoire englobent essentiellement la méthode de descente, le recuit simulé, la recherche tabou, la recherche à voisinage variable, la méthode GRASP, la recherche locale itérée, la recherche locale guidée et leurs variantes.

II.3.1) Les méthodes de descente (DM: Descent method)

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

L'algorithme 2.1 présente le squelette d'une méthode de descente simple (Simple descent). Apartir d'une solution initiale x, on choisit une solution x' dans le voisinage $N(x)$de x. Si cette solution est meilleure que $x, (f(x') < f(x))$ alors on accepte cettesolution comme nouvelle solution x et on recommence le processus jusqu'à cequ'il n'y ait plus aucune solution améliorante dans le voisinage de x.

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x)$
4: **if** $f(x') < f(x)$ **then**
5: $x' \leftarrow x$
6: **end if**
7: **until** $f(y) \geq f(x),\ \forall y \in \mathcal{N}(x)$

Algorithme 2.1 : l'algorithme de la méthode de descente

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

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x) \ / \ f(x') \leq f(x''), \forall x'' \in \mathcal{N}(x)$
4: **if** $f(x') < f(x)$ **then**
5: $x' \leftarrow x$
6: **end if**
7: **until** $f(x') \geq f(x),\ \forall x' \in \mathcal{N}(x)$

Algorithme 2.2 : l'algorithme de la méthode deplus grande descente

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

Un moyen très simple de diversifier la recherche peut consister à re-exécuterun des algorithmes en prenant un autre point de départ. Comme l'exécution de ces méthodes est souvent très rapide, on peut alors inclure cette répétition au sein d'une boucle générale. On obtient alors un algorithme de type "Multi-start descent" décrit par l'algorithme 2.3.

1: *initialise* : find an initial solution x, $k \leftarrow 1$, $f(B) \leftarrow +\infty$
2: **repeat**
3: *Starting point* : choose an initial solution x_0 at random
4: $x \leftarrow$ result of Simple Descent or Deepest Descent
5: **if** $f(x) < f(B)$ **then**
6: $B \leftarrow x$
7: **end if**
8: $k \leftarrow k + 1$
9: **until** stopping criterion satisfied

Algorithme 2.3 : l'algorithme de la méthode dedescenteavec relance

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

II.3.2) Le recuit simulé (SA: Simulated Annealing)

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

Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température estmaintenu jusqu'à obtention d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires du métal non satisfaisants, mais conduisant à la longue à des états meilleurs.

Dans la méthode SA, les mécanismes d'intensification et de diversification sont contrôléspar la température. La température t ne fait que décroitre pendant le processus de recherche,de sorte que la recherche tend à s'intensifier vers la fin de l'algorithme. L'idée est de diminuerpetit à petit la chance d'accepter des solutions qui dégradent la fonction objectif. l'algorithme 2.4 présente les principales caractéristiques d'un recuit simulé.

1: *initialise* : find an initial solution x, fix an annealing schedule $\mathcal{T}$, set initial temperature t
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x)$
4: determine $\Delta C = f(x') - f(x)$
5: draw $p \sim \mathcal{U}(0,1)$
6: **if** $\Delta C < 0$ or $e^{-\Delta C/t} > p$ **then**
7: $x' \leftarrow x$
8: **end if**
9: update temperature t according to $\mathcal{T}$
10: **until** stopping criterion satisfied

Algorithme 2.4 : l'algorithme du recuit simulé

II.3.3) La recherche tabou (TS : Tabu Search)

La recherche tabou est une métaheuristique originalement développée parGlover, 1986 et indépendamment par Hansen, 1986 [14] . Elle estbasée sur des idées simples, mais elle est néanmoins très efficace. Cette méthodecombine une procédure de recherche locale avec un certain nombre de règles et demécanismes permettant à celle-ci de surmonter l'obstacle des optima locaux, touten évitant de cycler. Elle a été appliquée avec succès pour résoudre de nombreuxproblèmes difficiles d'optimisation combinatoire : problèmes de routage devéhicule, problèmes d'ordonnancement, problèmes de coloration de graphes, etc.

Dans une première phase, la méthode de recherche tabou peut être vuecomme une généralisation des méthodes d'amélioration locales. En effet, en partantd'une solution quelconque x appartenant à l'ensemble de solutions S, on se déplacevers une solution x' située dans le voisinage $N(x)$. Donc l'algorithme exploreitérativement l'espace de solutions S.

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}^*(x)$
4: *update memory* : tabu list, frequency-based memory, aspiration level, ...
5: *move* $x \leftarrow x'$
6: **until** stopping criterion satisfied

Algorithme 2.5 : l'algorithme de recherche tabou classique

Afin de choisir le meilleur voisin x' dans $N(x)$, l'algorithme évalue lafonction objectif f en chaque point x', et retient le voisin qui améliore la valeur dela fonction objectif, ou au pire celui qui la dégrade le moins.

II.3.4) La recherche à voisinages variables (VNS : Variable neighborhood search)

La recherche àvoisinages variables est une méthode très simple, basée sur la performance des méthodes de descente. Introduite par Mladenović et Hansen[15], la méthode propose simplement d'utiliser plusieurs voisinagessuccessifs quand on se trouve bloqué dans un minimum local.

Avant tout, il est nécessaire de définir un ensemble de k_{max}voisinages, dénotéspar$N_{k=1\ldots k_{max}}$(et de préf´erence tels que $N_k \subset N_{k+1}$). On choisit une solution de départ x par heuristique. Ensuite, à partir d'une solution initiale x' choisie dansle premier voisinage $N(x)$ de x, on applique une méthode de descente (ou une autre méthode de recherche locale) jusqu'à arriver dans un minimum local (ou que la recherche locale s'arrête). Si la solution trouvée x'' est meilleure que x alorson recentre la recherche en repartant du premier voisinage, sinon on passe auvoisinage suivant (qui a priori est plus grand). La recherche s'arrête quand tousles voisinages ne sont plus capables d'améliorer la solution.

Le point crucial dans une VNS, c'est bien évidement la constitution des voisinagesde plus en plus grands et inclus les uns dans les autres. Une bonne structure de voisinage conduit généralement à de bons résultats ou au moins à une recherche efficace.

La méthode est décrite en détail par Hansen etMladenović et un squelette général de cette méthode est proposé dans l'algorithme2.6.

1: *initialise* : find an initial solution x, k $\leftarrow 1$
2: **repeat**
3: *shake* : generate a point x' at random from the neighbourhood $\mathcal{N}_k(x)$
4: *local search* : apply a local search procedure starting from the solution x' to find a solution x''
5: **if** x'' is better than x **then**
6: $x \leftarrow x''$ and $k \leftarrow 1$ (centre the search around x'' and search again with a small neighbourhood)
7: **else**
8: $k \leftarrow k + 1$ (enlarge the neighbourhood)
9: **end if**
10: **until** $k = k_{max}$

Algorithme 2.6 : l'algorithme de recherche à voisinage variable

II.3.5)La procédure de recherche gloutonne aléatoire adaptative (GRASP:Greedy Randomized Adaptative Search Procedure)

Introduite en 1989 par Feo et Resende [16] et présentée dans sa forme plus définitive en 1995[17], la méthode GRASP combine une heuristique gloutonne et une recherche aléatoire. A chaque itération,on construit une solution comme dans une heuristique gloutonne (en se servantd'une liste d'attributs comme liste de priorité). Cette solution est améliorée par l'intermédiaire d'une méthode de descente. En se basant sur la qualité généralede la solution ainsi obtenue, on met à jour l'ordre de la liste des attributs et le processus est itéré jusqu'à satisfaction d'un critère d'arrêt. Un des avantages de cette méthode est la simplicité avec laquelle on peut comprendre le processusd'optimisation. La mise en œuvre elle aussi n'est pas trop compliquée. Des implémentations réussies et un tutorial récent présentant cette méthode sont présentées par Festa [18].

1: *construct* an ordered list of solution attributes S
2: **repeat**
3: **repeat**
4: take $s \in S$ in a greedy way
5: $x = x \cup s$
6: **until** solution x complete
7: *local search* : find a locally optimal solution x' starting from x
8: update the benefits of the solution attributes in S
9: **until** stopping criterion satisfied

Algorithme 2.7 : l'algorithme de la méthode GRASP

II.3.6) La recherche locale itérée (ILS: iterated local search)

La méthode de recherche locale itérée est une variante très simple des méthodes de descente qui pallient au problème de l'arrêt de ces dernières dans des optima locaux. Donner la paternité de cette méthode à un ou plusieurs auteurs seraitabusif.

Dans cette méthode, on génère une solution initiale qui servira de point de départ. Ensuite, on va répéter deux phases : une phase de perturbation aléatoiredans laquelle la solution courante va être perturbée (parfois en tenant compte d'un historique maintenu à jour) et une seconde phase de recherche locale (ou tout simplement de méthode de descente) qui va améliorer cette solution jusqu'à buter sur un optimum local. Dans cette boucle, on est aussi à même d'accepterou non la nouvelle solution selon que l'on souhaite donner un caractère plus ou moins agressif à la méthode.

1: *initialise* : find an initial solution x
2: **repeat**
3: *random perturbation* : find a solution x' "close" to x
4: *local search* : starting from x', find a solution x'' locally optimal
5: **if** x'' is accepted (e.g. better than x) **then**
6: $x \leftarrow x''$
7: **end if**
8: **until** stopping criterion satisfied

Algorithme 2.8 : l'algorithme de la recherche locale itérée

II.3.7) La recherche locale guidée (GLS:guided local search)

Présentée pour la première fois dans un rapport de recherche de Voudouriset Tsang[19], la recherche locale guidée est une variante assez élaborée d'une méthode de descente classique. La méthode de base est simple. Elle consiste à modifier la fonction à optimiser en ajoutant des pénalités. La recherche locale est appliquée alors sur cette fonction modifiée. La solution trouvée sert à calculer les nouvelles pénalités. Pour cela, on calcule l'utilité de chacun des attributs de la solution et on augmente les pénalités associées aux attributs de valeur maximale.

1: *initialise* : find an initial solution x, set penalties to 0
2: **repeat**
3: *cost function* : augment the original cost function of x with penalties
4: *local search* : apply a local search procedure to find a solution x' locally optimal (based on the augmented cost function)
5: *update penalties* : by computing utility expressions
6: $x \leftarrow x'$
7: **until** stopping criterion satisfied

Algorithme 2.9 : l'algorithme de la recherche locale guidée

II.4) Les métaheuristiques à population de solutions

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

II.4.1) Les algorithmes génétiques (GA :Genetic Algorithm)

Proposé dans les années 1975 par Holland[21], les algorithmes génétiques doivent leur popularité à Goldberg[22]. Avant la parution de son livre qui est une des références les plus citées dans le domaine de l'informatique, on a puvoir un certain nombre d'autres présentations, citons Goldberg [23], Holland[24], Schwefel[25]. Le sujet connaît une très grande popularité. Il existe aujourd'huiplusieurs milliers de références sur le sujet et le nombre de conférences dédiéesau domaine (que ce soit sur les techniques elles-mêmes ou sur les applications)ne fait qu'augmenter.

De manière générale, les algorithmes génétiques utilisent un même principe.Une population d'individus (correspondants à des solutions) évoluent en même temps comme dans l'évolution naturelle en biologie. Pour chacun des individus, on mesure sa faculté d'adaptation au milieu extérieur par le fitness.

Les algorithmes génétiques s'appuient alors sur trois fonctionnalités :

-La Sélection : Pour déterminer quels individus sont plus enclins à se reproduire, une sélection est opérée. Il existe plusieurs techniques de sélection, les principales utilisées sont la sélection par tirage à la roulette (roulette-wheel selection), la sélection par tournoi (tournament selection), la sélection par rang (ranking selection), etc[26].

- Le Croisement : L'opérateur de croisement combine les caractéristiques d'un ensemble d'individus parents (généralement deux) préalablement sélectionnés, et génère de nouveaux individus enfants. Là encore, il existe de nombreux opérateurs de croisement, par exemple le croisement en un point, le croisement en n-points ($n \geq 2$) et le croisement uniforme 5 (voir figure 2.6).

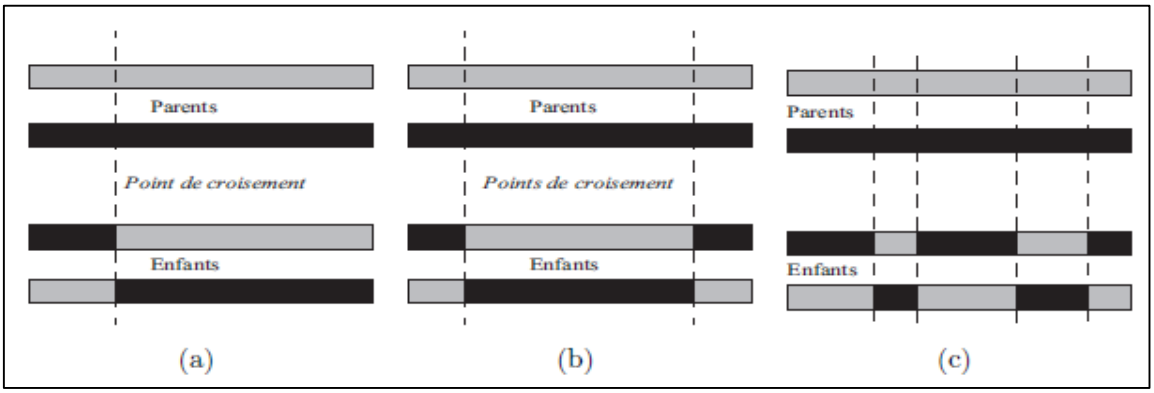

Fig 2.6 : (a) croisement simple en un point, (b) croisement en deux points, (c) croisement uniforme.

-La Mutation et le remplacement: Les descendants sont mutés, c'est-à-dire que l'on modifie aléatoirement une partie de leur génotype, selon l'opérateur de mutation.Le remplacement (ou sélection des survivants), comme son nom l'indique, remplace certains des parents par certains des descendants. Le plus simple est de prendre les meilleurs individus de la population, en fonction de leurs performances respectives, afin de former une nouvelle population (typiquement de la même taille qu'au début de l'itération).

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluation d'une solution. Le codage phénotypique ou codage direct correspond en général à une représentation de la solution très proche de la réalité. L'évaluation d'une solution représentée ainsi est en général immédiate.

L'algorithme2.10 propose une version qualifiée en Anglais par population replacement (ou algorithme générationnel)

```
1: initialise : generate an initial population P of solutions with size |P| = n
2: repeat
3:    P' ← ∅
4:    repeat
5:       selection : choose 2 solutions x and x' from P with probability proportio-
         nal to their fitness
6:       crossover : combine parent solutions x and x' to form child solutions y and
         y' with high probability
7:       mutate y and y' with small probability
8:       add y and y' to P'
9:    until |P'| = n
10:   P ← P'
11: until stopping criterion satisfied
```

Algorithme 2.10 : l'algorithme génétique simple par remplacement de la population

II.4.2) Algorithme de colonies de fourmis : (ACO : Ant Colony Optimization)

Les algorithmes de colonies de fourmis ont été proposés par Colorni,Dorigo et Maniezzo en 1992 [27] et appliquéesla première fois au problème du voyageur de commerce. Ce sont desalgorithmes itératifs à population où tous les individus partagent un savoir commun qui leur permet d'orienter leurs futurs choix et d'indiqueraux autres individus des choix à suivre ou à éviter. Le principe de cettemétaheuristique repose sur le comportement particulier des fourmis, ellesutilisent pour communiquer une substance chimique volatile particulière appelée phéromone grâce à une glande située dans leur abdomen. Enquittant leur nid pour explorer leur environnement à la recherche dela nourriture (Food), les fourmis arrivent à élaborer des chemins qui s'avèrentfréquemment être les plus courts pour aller du nid vers une source de nourriture. Chaque fourmi dépose alors une quantité de phéromonessur ces pistes qui deviendront un moyen de communication avec leurscongénères, les fourmis choisissent ainsi avec une probabilité élevée leschemins contenant les plus fortes concentrations de phéromones à l'aidedes récepteurs situés dans leurs antennes.

La figure 2.7illustre et confirme ce constat, une expérience a été faite par Gauss et Deneubourg en 1989 appelée expérience du pont à doublebranche, les fourmis qui retournent au nid rapidement, après avoir visitéla source de nourriture, sont celles qui ont choisi la branche courteet les fourmis empruntant cette branche faisant plus d'aller retour, etpar conséquent la quantité de phéromones déposée sur la plus courtebranche est relativement supérieure que celle présente sur la plus longuebranche. Puisque les fourmis sont attirées plus vers les pistes de plus

grande concentration en phéromones, alors la branche courte sera la plusempruntée par la majorité des fourmis.

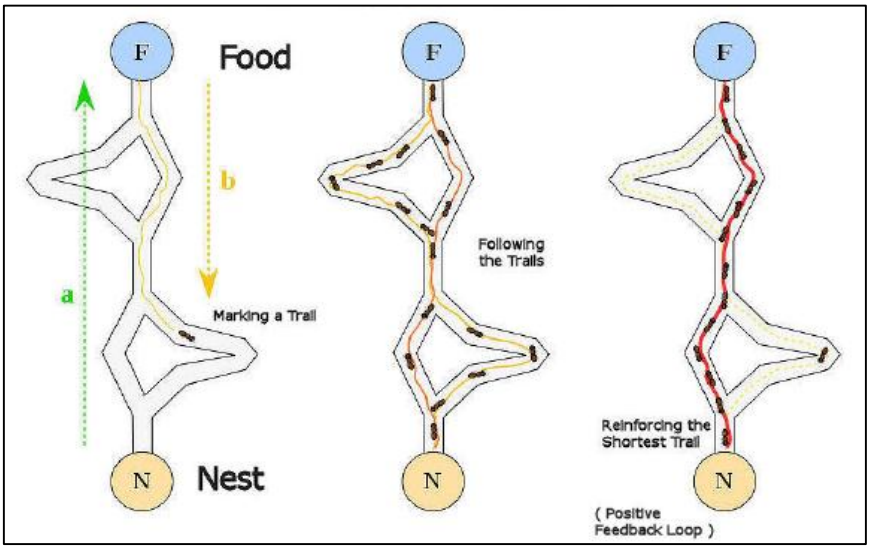

Fig 2.7 : Optimisation du chemin par les fourmis au cours des itérations.

L'algorithme (2.12) représente une version simplifiée de l'ACO.

1: *initialise* : create an initial population of ants
2: **repeat**
3: **for** each ant **do**
4: *construct a solution* based on the construction procedure, biased by the pheromone trails
5: *update the pheromone trails* based on the quality of the solutions found
6: **end for**
7: **until** stopping criterion satisfied

Algorithme 2.11 : l'algorithme de colonies de fourmis

II.5) Algorithme de colonie d'abeilles: (BCO : Bee Colony Optimization)

II.5.1) Historique

Au cours de la dernière décennie, les algorithmes d'abeilles inspirés de la nature, sont devenus un outil prometteur et puissant. Malheureusement, on ne parvient pas à connaître la date exacte de la première apparition des algorithmes d'abeilles. Ce qui est sûr pour nous c'est qu'ils ont été développés en quelques années de façon indépendante par différent groupes de chercheurs.

D'après la bibliographie, il semble que l'algorithme HONEY-BEE a été réalisé pour la première fois vers 2004 par Craig A.tovey à GEORGIA TECH en collaboration avec SUNIL NAKRANI. A la fin de 2004 et au début de 2005, Xin-She Yang à l'Université de CAMBRIDGE

a développé le VIRTUAL BEE ALGORITHM (VBA) pour résoudre des problèmes d'optimisation numérique, cet algorithme permet d'optimiser à la fois les fonctions et les problèmes discrets, cependant ils n'ont donné comme exemples que les fonctions à deux paramètres. Un peu plus tard en 2005, Haddad, Afshar et leurs collègues ont présenté un algorithme dit Honey-Bee Mating Optimization (HBMO) qui a ensuite été appliqué à la modélisation de réservoirs et de clustering.

En 2006, B.Basturk et D.jarabogo en Turquie, ont développé un algorithme appelé Artificial Bee Colony (ABC) pour l'optimisation de fonction numérique.

Nous remarquons ici que la méthode des abeilles est plus ou moins récente, et qu'avec le temps de nouvelles versions apparaissent, ce qui rend cette méthode de plus en plus populaire et maîtrisable par les chercheurs.

II.5.2) Comportement des abeilles

Comme les fourmis, les abeilles sont des insectes sociaux. Elles sont obligées de vivre en colonie très organisée, formée d'ouvrières, de faux-bourdon et d'une seule reine, et où chacune a un travail bien précis à faire. Les abeilles se nourrissent essentiellement de pollen et de miel. Elles vont butiner les fleurs pour prendre le nectar.

Au cours de sa courte vie (environ 45 jours), l'ouvrière fait plusieurs métiers : elle nettoie les cellules, nourrit les larves, elle range le pollen et le nectar dans les alvéoles, elle ventile la ruche en agitant rapidement ses ailes, elle construit les rayons avec la cire qu'elle produit, elle garde le trou de vol pour chasser les intrus, elle devient butineuse, porteuse d'eau et récolte du pollen et du nectar jusqu'à la fin de sa vie.

L'abeille est capable, par la danse ou par la production de substances chimiques appelées « phéromone », de communiquer aux autre abeilles l'endroit où elle a découvert de la nourriture. Elle danse en rond (Figure 1) quand elle a trouvé du pollen à faible distance (moins de 25 mètres). Elle utilise une danse très compliquée dite la danse frétillante (Figure 2), ou danse en huit, si la nourriture se trouve à moins de 10 kilomètres. La direction de la nourriture est exprimée par rapport à la position du soleil. La distance est exprimée par le nombre et la vitesse des tours effectués par l'abeille sur elle-même. Afin de survivre à l'hiver, les abeilles doivent recueillir et stocker environ 15 à 50 Kg de nectar.

Les faux bourdons ne servent que pour la reproduction. Ils sont incapables de se nourrir eux-mêmes (les ouvrières les nourrissent) et ils n'ont pas de dard pour protéger la ruche.Il n'y a qu'une seule reine dans la colonie. Quelques jours après sa naissance, elle s'envole pour la seule

fois de son existence pour être fécondée par quelques faux-bourdons. Elle occupera le reste de ses jours (4 à 5 ans) à pondre jusqu'à 2000 œufs par jour.

Les abeilles adultes (âgées de 20 à 40 jours) deviennent habituellement des butineuses. Les abeilles butineuses jouent en général l'un des trois rôles suivants : butineuses actives, butineuses éclaireuses et butineuses inactives.

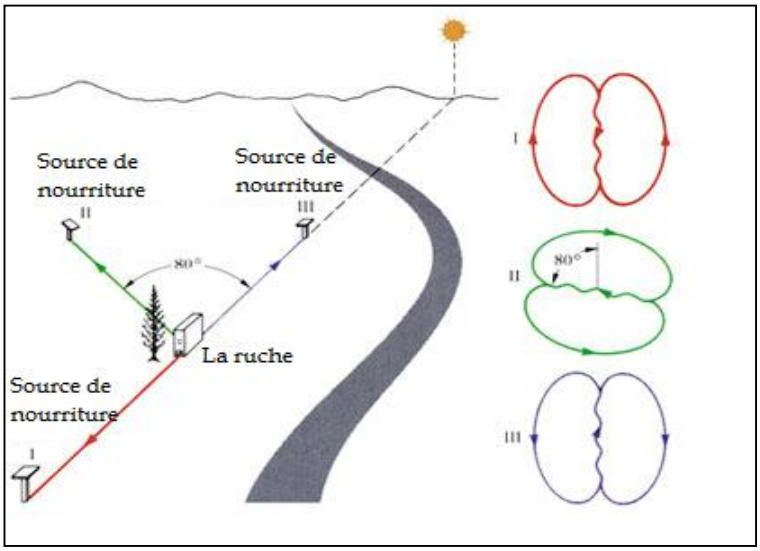

Fig 2.8: La danse en rond qu'effectue l'abeille en fonction de la direction de la source de nourriture.

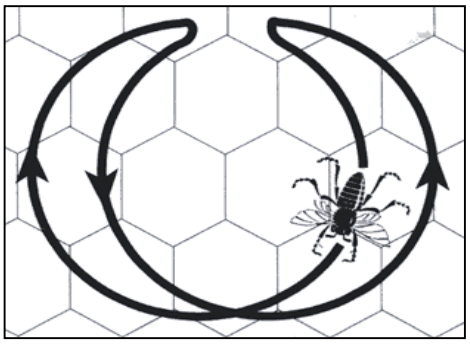

Fig 2.9 : La danse frétillante, appelée aussi en huit

II.5.3) Algorithme d'optimisation par colonie d'abeilles

Dans cet algorithme, l'emplacement de la source de nourriture représente la solution possible au problème, et la quantité du nectar de cette source correspond à une valeur objectif dite fitness.

Les butineuses sont attribués aux différentes sources de nourriture de façon à maximiser l'apport total de nectar. La colonie doit optimiser l'efficacité globale de la collecte. La répartition des abeilles est donc en fonction de nombreux facteurs tels que la quantité du nectar et la distance

entre la source de nourriture et la ruche. Ce problème et similaire à la répartition des serveurs d'hébergement web, qui était en fait un des premiers problèmes résolus en utilisant les algorithmes d'abeilles par Nakrani et Tovey en 2004.

Le nombre des butineuses actives ou inactives représente le nombre de solution dans cette populations.

Dans la première étape, l'algorithme génère une population initiale de SN solutions distribuées de façon aléatoire. Chaque solution$x_i (i = 1, 2, \dots, SN)$ qui est initialisée par les éclaireuses, et représente un vecteur de solution au problème d'optimisation. Les variables que contient chaque vecteur doivent être optimisées.

Après l'initialisation, la population des solutions est soumise à des cycles répétés $C = 1, 2, \dots, C_{max}$, ces cycles représentent des processus de recherches faits par les butineuses actives, inactives et les éclaireuses.

Les butineuses actives recherchent dans le voisinage de la source précédentex_i de nouvelles sources$v_i$ayant plus de nectar, Elles calculent ensuite leur fitness. Afin de produire une nouvelle source de nourriture à partir de l'ancienne, on utilise l'expression ci contre :

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

Où $k \in \{1, 2, \dots, BN\}$(BN est le nombes des butineuses actives)et $j \in \{1, 2, \dots, SN\}$ sont des indices choisis au hasard. Bien que k est déterminé aléatoirement, il doit être différent de$i$. ϕ_{ij} est un nombre aléatoire appartenant à l'intervalle [-1, 1], il contrôle la production d'une source de nourriture dans le voisinage dex_{ij}.

Après la découverte de chaque nouvelle source de nourriturev_{ij}, un mécanisme de sélection gourmande est adopté, c'est-à-dire que cette source est évaluée par les abeilles artificielles, sa performance est comparée à celle dex_{ij}. si le nectar de cette source est égale ou meilleur que celui de la source précédente, celle-ci est remplacée par la nouvelle. Dans le cas contraire l'ancienne est conservée.

Pour un problème de minimisation, La fitness est calculée suivant cette formule :

$$fit_i(\overrightarrow{x_i}) = \begin{cases} \dfrac{1}{1 + f_i(\overrightarrow{x_i})} si f_i(\overrightarrow{x_i}) \geq 0 \\ 1 + abs\big(f_i(\overrightarrow{x_i})\big) si f_i(\overrightarrow{x_i}) > 0 \end{cases}$$

Telle que $f_i(\overrightarrow{x_i})$est la valeur de la fonction objectif de la solution$\overrightarrow{x_i}$.

A ce stade, les butineuses inactives et les éclaireuses qui sont entrain d'attendre au sein de la ruche. A la fin du processus de recherche, les butineuses actives partagent les informations sur le nectar des sources de nourriture ainsi que leurs localisations avec les autre abeilles via la danse frétillante. Ces dernières évaluent ces informations tirées de toutes les butineuses actives, et choisissent les sources de nourriture en fonction de la valeur de probabilitéP_i associée a cette source, et calculée par la formule suivante :

$$P_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n}$$

Où fit_i est la fitness de la solution$i$, qui est proportionnelle à la quantité du nectar de la source de nourriture de la position$i$).

La source de nourriture dont le nectar est abandonné par les abeilles, les éclaireuses la remplacent par une nouvelle source. Si durant un nombre de cycle prédéterminé appelé « limite » une position ne peut être améliorée, alors cette source de nourriture est supposée être abandonnée.

Toutes ces étapes sont résumées dans l'algorithme suivant :

1. *Initialiser* la population avec **S+W** solutions aléatoires.
2. Evaluer la fitness de la population.
3. **Tant que** (*le critère d'arrêt n'est pas satisfait*) **faire** :
4. Recruter des abeilles : Rechercher de nouvelles solutions.
5. Evaluer la fitness de la population.
6. **Si** (*un membre de la population ne s'est pas amélioré*) **faire**
7. Enregistrer la solution et remplacer la par une solution aléatoire.
8. Trouver **S** solutions aléatoires et remplacer les **S** membres de la population qui ont la mauvaise fitness.
9. **Fin Tant que**
10. **Retourner** *la meilleure solution*.

Algorithme 2.12 : l'algorithme de colonies d'abeilles

II.5.4)Domaines d'application

Nombreux sont les domaines d'application des algorithmes d'abeilles, citons quelques unes :

- L'optimisation de fonction.
- La résolution du problème du voyageur de commerce (TSP) qui a était faite par Lucic et

Teodorovic et qui a donné de très bons résultats.

- L'apprentissage des réseaux de neurones tels que MLP, RBF, SNN, LVQ.
- Conception électroniques et mécanique.
- Optimisation de filtre digital.
- Clustering de données.
- Contrôle de robot.
- L'ordonnancement de tâches.
- La prédiction de structure de protéine.

La majorité des problèmes qui ont étaient résolus par cette méthode, ont donné de très bon résultats concernant la valeur de la fonction objectif, et le temps d'exécution qui a été acceptable.

II.5.5) Avantages

- Très efficace dans la recherche des solutions optimales.
- Surmonte le problème de l'optimum local.
- Facile à implémenter.

II.5.6) Inconvénients

- L'utilisation de plusieurs paramètres réglables.
- Sensible à des problèmes extrêmement difficiles.
- L'optimisation de fonction.

2.6) Conclusion

Dans ce chapitre nous avons essayé de présenter les principes et les algorithmes de différentes méthodes de résolution de problèmes d'optimisation commençant par les méthodes exactes aux méthodes approchées. Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace mémoire requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution. Nous avons pu constater que les méthodes approchées peuvent être partagées en deux classes: des méthodes heuristiques et des méthodes métaheuristiques. Une méthode heuristique est applicable sur un problème donné. Tandis qu'une méthode métaheuristique est plus générique et elle peut être appliquée sur plusieurs problèmes d'optimisation. En outre, nous avons constaté que les méthodes métaheuristiques peuvent être partagées en deux sous classes: des méthodes à base d'une solution unique et des méthodes à base de population de solutions. Les méthodes de la première sous classe (i.e. les méthodes à base d'une solution unique) se basent sur la recherche locale pour trouver la solution

du problème à traiter. Elles sont souvent piégées par l'optimum local d'un voisinage donné. Par contre, les méthodes de la deuxième classe (i.e. les méthodes à base de population de solutions) se basent sur une recherche globale ce qui leur permet d'échapper au problème de la convergence vers l'optimum global et augmente leur possibilité de fournir des solutions de bonnes qualités. Dans la dernière partie du chapitre nous avons présenté l'algorithme de colonies d'abeilles qui fait l'objet de notre étude.

CHAPITRE III

Résolution du problème flow-shop à l'aide de BCO hybride

III.1) Introduction

Le but de ce chapitre est de présenter notre contribution qui consiste à adapter l'algorithme de colonies d'abeilles inspiré de la danse d'abeilles, utilisée pour la recherche de nourriture, pour la résolution d'un problème d'ordonnancement de type Flow Shop et par la suite hybrider cette technique avec une méthode de recherche locale basique technique avec une méthode de recherche locale basique.

Pour pouvoir valider notre adaptation, nous avons appliqué les algorithmes adaptés (l'algorithme de colonies d'abeilles et l'algorithme de colonies d'abeilles hybride) pour la résolution de trois classes de problèmes Flow Shop avec différents exemples.

Dans les différentes sections de ce chapitre nous présentons l'adaptation des deux algorithmes et les résultats de simulation obtenus.

III.2) Problématique

Notre but dans la résolution d'un problème flow-shop avec minimisation de makespan est de trouver la séquence minimale (c'est-à-dire qui permet le meilleur ordonnancement de tâches dans le système).

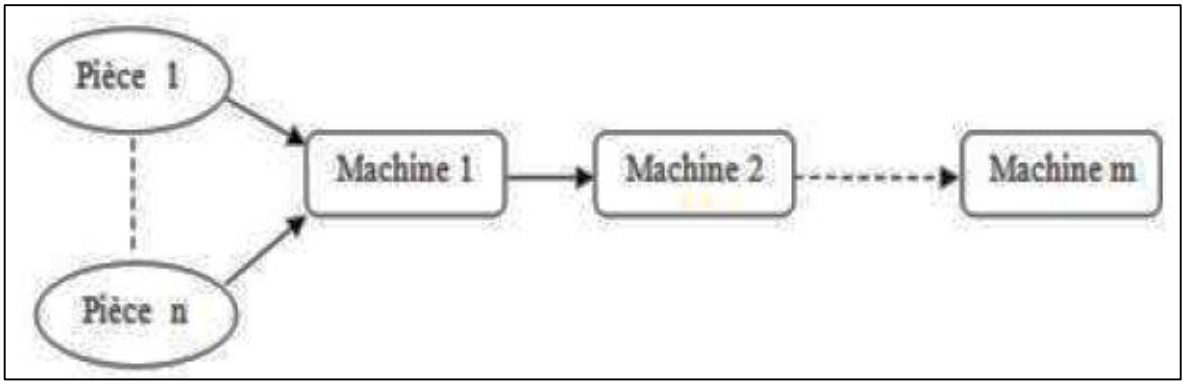

Fig 3.1 : Exemple d'un atelier flow-shop.

Les contraintes imposées pour notre problème sont les suivantes:

- Tous les jobs sont disponibles à l'instant t=0
- Chaque machine ne peut traiter qu'un seul job à la fois
- Les temps opératoires de tous les jobs sont connus auparavant
- Un job ne peut être traité par une machine que lorsque celle-ci fini le traitement du job précédent

III.3) Adaptation de l'algorithme des colonies d'abeilles (BCO)

La première étape pour l'adaptation d'une métaheuristique est la représentation des solutions. Cette représentation dépend du problème à résoudre et de la métaheuristique elle même.

Dans un problème d'ordonnancement flow-shop avec minimisation de makespan, une solution représente le séquencement de jobs sur les différentes machines, c'est-à-dire l'ordre de passage des différents jobs sur les différentes machines. La figure 3.2 représente une solution possible d'un problème de 10 tâches :

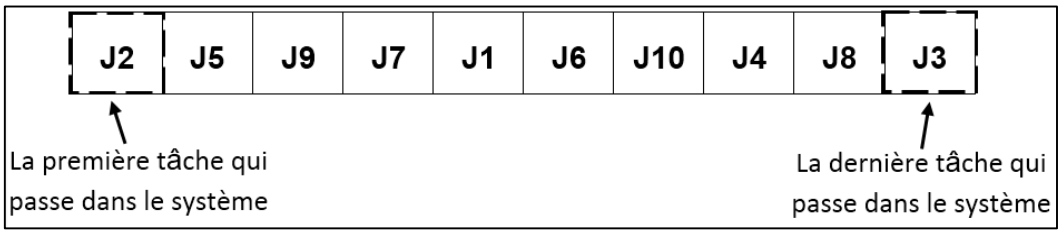

Fig 3.2 : Exemple d'un solution possible de problème.

L'algorithme 3.1 représente l'algorithme des colonies d'abeilles adapté.

1. *Initialisation* : Génération d'une population initiale aléatoire de P solutions.
2. *Evaluation* : calculer le fitness pour chaque solution de la population initiale.
3. Décomposer la population en S+W solutions.
4. Fixer un nombre d'itérations.
5. Fixer un nombre de butineuses.
6. Fixer un nombre de solutions à affecter à chaque butineuse.
7. **Pour** chaque butineuse **faire :**
8. Appliquer la recherche pour un nouvelle source.
9. Evaluer la fitness de la population.
10. Remplacer les solutions non améliorées par des solutions aléatoirement.
11. **Fin pour.**
12. Classer les solutions de la meilleure vers la mauvaise.
13. Classer les S mauvaises solutions par des solution aléatoires.
14. Retourner Cmax.

Algorithme 3.1 : l'algorithme des colonies d'abeilles adapté.

III.4) L'algorithme des colonies d'abeilles hybride

L'hybridation consiste à combiner deux ou plusieurs métaheuristiques (ou autres techniques) de manière complémentaire, elle permet l'amélioration des opérateurs d'intensification et de diversification d'une métaheuristique. Nous proposons dans notre approche l'hybridation de l'algorithme BCO avec la recherche par voisinage qui est une méthode de recherche locale basique.

L'hybridation consiste à insérer la recherche par voisinage au niveau de chaque butineuse pour la recherche de nourriture, ceci permettra à chaque butineuse de trouver des solutions de meilleurs qualité.

L'algorithme 3.2 représente la méthode de la recherche par voisinage.

1. Trouver un solution initiale x.
2. Fixer un nombre de répétition.
3. Chercher x' dans le voisinage de x.
4. **Si** $x' < xx \longrightarrow$ '.
5. **Fin si.**
6. Répéter.

Algorithme 3.2 : l'algorithmede recherche par voisinage.

L'algorithme 3.3 représente l'algorithme de la BCO hybride.

15. *Initialisation* : Génération d'une population initiale aléatoire de P solutions.
16. *Evaluation* : calculer le fitness pour chaque solution de la population initiale.
17. Décomposer la population en S+W solutions.
18. Fixer un nombre d'itérations.
19. Fixer un nombre de butineuses.
20. Fixer un nombre de solutions à affecter à chaque butineuse.
21. **Pour** chaque butineuse **faire :**
22. Fixer un nombre de répétition.
23. Pour chaque solution chercher une solution voisine.
24. **Si**la solution est améliorée $\longrightarrow$ éxraser l'ancienne solution.
25. **Fin si.**
26. Répéter.
27. Evaluer la fitness de la population.
28. Remplacer les solutions non améliorées par des solutions aléatoirement.
29. **Fin pour.**
30. Classer les solutions de la meilleure vers la mauvaise.
31. Classer les S mauvaises solutions par des solution aléatoires.
32. Retourner Cmax.

Algorithme 3.3 : l'algorithmede de la BCO hybride.

III.5) Résultats et discussions

Cette section est consacrée à la présentation des résultats obtenus par les deux algorithmes adaptés. Nous proposons une étude de sensibilité des deux techniques para rapport à leurs paramètres car la puissance d'une métaheuristique dépend potentiellement du choix de ses paramètres.

III.5.1) Etude de sensibilité de l'algorithme BCO

L'analyse de sensibilité a été appliquée sur une classe de systèmes de tailles (10 jobs×10 machines) avec un temps de traitement variant de 0 à 30.

III.5.1.1) Etude de sensibilité par rapport au nombre de butineuses

Cette étude de sensibilité consiste à fixer tous les paramètre de la métaheuristique (initialement S=W , nombre de solutions examinées = 7) et varier le nombre de butineuses. Les résultats de simulation sont donnés par le tableau (3.1).

	Nombre de butineuses	Valeur de Cmax
Exemple 1	2	348
	3	356
	4	351
	5	355
	6	**337**
	7	345
	8	350
Exemple 2	2	358
	3	340
	4	352
	5	357
	6	**336**
	7	344
	8	343

Exemple 3	2	356
	3	363
	4	352
	5	353
	6	**346**
	7	346
	8	358

Tab 3.1:Etude de sensibilité par rapport au nombre de butineuses.

Le tableau (3.1) montre les résultats de simulation pour une étude de sensibilité par rapport au nombre de butineuses. Les résultats montrent que les Cmax minimums sont obtenus pour un nombre de butineuses égale à 6.

III.5.1.2) Etude de sensibilité par rapport au nombre de solutions examinées

Cette étude de sensibilité consiste à fixer le nombre de butineuses à 6, S =W et varier le nombre de solutions examinées. Les résultats de simulation sont donnés par le tableau 3.2.

	Nombre de solutions examinées	Valeur de Cmax
Exemple 1	4	358
	5	358
	6	337
	7	**338**
	8	350
Exemple 2	4	356
	5	348
	6	336
	7	**329**
	8	345
Exemple 3	4	340
	5	358
	6	346
	7	**340**
	8	342

Tab 3.2:Etude de sensibilité par rapport au nombre de solutions examinées.

Le tableau (3.2) représente les résultats de l'étude de sensibilité de l'algorithme BCO par rapport au nombre de solutions examinées. Les résultats montrent que les meilleurs résultats sont obtenus pour un nombre de solutions égale à 7. Nous remarquons que plus le nombre de solutions augmente plus la qualité des solutions obtenus s'améliore, ceci peut être expliqué par le fait qu'il y a plus de possibilités d'avoir de bonne solutions.

III.5.1.3) Etude de sensibilité par rapport à la taille de S et W

Cette étude de sensibilité consiste à fixer tous les paramètre de la métaheuristique (nombre de butineuses =6, nombre de solutions examinées = 7) et varier la taille de S et W. Les résultats de simulation sont donnés par le tableau 3.3.

	Les valeurs de S et W	Valeur de Cmax
Exemple 1	S = 2.W	358
	S = W	**338**
	S = 0,5. W	354
Exemple 2	S = 2.W	349
	S = W	**329**
	S = 0,5. W	354
Exemple 3	S = 2.W	347
	S = W	**340**
	S = 0,5. W	352

Tab 3.3:Etude de sensibilité par rapport à la taille de S et W

Le tableau (3.3) représente les résultats de l'étude de sensibilité de l'algorithme BCO par rapport aux deux sous populations S et W. il est clair que les meilleurs résultats sont obtenus pour S=W, ceci peut être expliqué par le fait que dans ce cas il y a un certain équilibre entre les nouvelle solutions divrsifiées et cele gardées durant les différentes opérations de l'algorithme.

III.5.2) Etude de sensibilité de l'algorithme BCO hybride (BCOH)

Cette étude de sensibilité consiste à étudier la sensibilité de l'algorithme BCO hybride suite à la variation de nombre de recherche effectuées par la recherche par voisinage, en d'autres termes, étude de sensibilité par rapport au nombre de voisinages considérés.Nous fixons le nombre de butineuses à 7, le nombre de solutions à examiner à 6, S = W et nous faisons varier le nombre de voisinages à examiner.Les résultats de simulation sont données par le tableau 3.4 :

	Nombre de voisinages à examiner.	Valeur de Cmax
Exemple 1	2	339
	3	333
	4	**331**
	5	332
	6	332
Exemple 2	2	350
	3	343
	4	**340**
	5	342
	6	342
Exemple 3	2	346
	3	341
	4	**338**
	5	347
	6	338

Tab 3.4:Etude de sensibilité de BCOH par rapport au nombre de voisinages à examiner

Le tableau (3.4) représente les résultats de l'étude de sensibilité de l'algorithme BCOH par rapport au nombre de voisinages. Nous remarquons que les meilleurs résultats sont obtenus pour un nombre de voisinages égale à 4, ceci est dû au nombre de solutions examinées au voisinage de chaque solution, mais si le nombre de voisinage augmente la qualité des solutions peut diminuer à cause de l'éloignement de la solution de départ qui peut être de bonne qualité.

III.5.3) Comparaison entre l'algorithme BCO et l'algorithme BCO hybride

Dans cette section nous présentons la comparaison entre les résultats de simulation obtenus par les deux techniques adaptées en fixant pour chacune les meilleurs paramètres obtenus.

Les résultats de simulation sont donnés pour quatre classes de problèmes (10jobs×10machines, 20jobs×20machines, 50jobs×30machines et 100jobs×30machines) chacune contenant trois différents exemples en termes de temps de traitement.

	10jobs×10machines		20jobs×20machines		50jobs×30machines		100jobs×30machines	
	BCO	BCOH	BCO	BCOH	BCO	BCOH	BCO	BCOH
Exemple 1	338	331	788	736	1731	1697	2706	2670
Exemple 2	329	340	837	828	1738	1709	2686	2651
Exemple 3	340	338	802	763	1697	1616	2718	2662

Tab 3.5: Comparaison entre les résultats de simulation de BCO et BCOH

Le tableau (3.5) représente les résultats obtenus par les deux algorithmes considérés (BCO et BCOH). Nous remarquons que l'algorithme BCOH est plus performant que le BCO en terme de qualité de solutions obtenus, ce qui est normal pour une métaheuristique hybride (les métaheuristique lorsqu'elle est hybridée elle dépasse sa version de base pour un choix judicieux des paramètres).

III.6) Conclusion

Le but de ce chapitre était de présenter notre adaptation de l'algorithme de colonies d'abeille et son hybridation avec une méthode de recherche locale (recherche par voisinage) ainsi que l'étude de sensibilité des deux techniques et les résultats de simulations obtenus pour différentes classes de problèmes.

Les résultats obtenus montrent que l'hybridation de l'algorithme BCOH permet l'obtention de meilleurs résultats par rapport à l'algorithme de base BCO.

CONCLUSION GENERALE

CONCLUSION GENERALE

Ce document s'intéresse à la résolution de problèmes d'ordonnancement de grande complexité par les méthodes approchées, en particulier les métaheuristiques. Nous nous sommes intéressé à l'algorithme de colonies d'abeilles (BCO) qui est une métaheuristique inspirée du comportement des abeilles (la danse des abeilles) pour la recherche de nourriture.

Le problème traité dans ce travail est le problème d'ordonnancement de tâches dans un atelier Flow Shop en vue de minimiser le makespan.

Notre contribution consiste à améliorer le mécanisme d'intensification de l'algorithme BCO en l'hybridant avec une méthode de recherche locale (recherche par voisinage).

Pour valider notre adaptation, nous avons appliqué l'algorithme de base (BCO) et celui que nous proposons (BCOH) sur plusieurs classes de problèmes avec différentes instances et pour différents exemples avec différents temps opératoires avec une étude de sensibilité sur les paramètres des deux algorithmes.

Comme futur travaux, nous proposons de remplacer la méthode de recherche locale par voisinage par une métaheuristique de recherche locale.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1]: M. Pinedo, « Scheduling: Theory, Algorithms and systems ». Prentice-Hall, Englewood Clis, New Jersey, 1955.

[2] : J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt et J. Weglarz,« Scheduling computer and manufacturing process ». Springer, Berlin, 1996.

[3]: F.A. Rodammer et K. Preston White, « A recent survey of production scheduling ». IEEE Transaction on Systems, Man and Cybernetics, pp. 6-18, 1999.

[4] : J. Carlier et P. Chrétienne, « Problèmes d'ordonnancement, Modélisation,Complexité, Algorithmes ». Edition Masson, Paris, 1988.

[5] : P. Esquerol et P. Lopez, 1999,«L'ordonnancement», Economica.

[6] : B. Roy et D. Bouysson, « Aide multi-critères à la décision : Méthodes et cas ».Collection Gestion Série : Production et technologie quantitatives appliquées à la gestion.Edition Economica, Paris, 1993.

[7] : C. Esswein. « Un apport de flexibilité séquentielle pour l'ordonnancementrobuste ». thèse de doctorat, Université François Rabelais,Tours, France, Décembre 2003

[8] : M. SAKAROVICH, Optimisation combinatoire, HERMANN, Paris, France, 1984.

[9] : Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet . mouloud nasser .

[10]:E. A. Feigenbaum, j. Feldman, (Edirors). Computers and thought. McGraw-Hill Inc. pp.6. New York, 1963.

[11]:E. A. Feigenbaum and J. Feldman. (Edirors). Computers and thought. McGraw-Hill Inc. pp.192. New York, 1963.

[12]:R. L. Solso. Cognitive psychology. Harcourt Brace Jovanovich, Inc., New York. pp. 436, 1979.

[13]:C. Hamzacebi and F. Kutay. "A heuristic approach for finding theglobal minimum : Adaptive random search technique. Applied Mathematicsand Computation", 173 :1323–1333, 2006.

[14]: Fred Glover and Manuel Laguna, 1997, Tabu search, KluwerAcademic Publishers.

[15]:M. Mladenovic and P.Hansen. Variable neighborhood search. Computers and Operations Research,24 :1097–1100, 1997.

[16]:T.A. Feo etM.G.C. Resende. A probabilistic heuristic for a computationallydifficult set covering problem. Operations Research Letters, 8 :67–71, 1989.

[17]: A. Feo et M.G.C. Resende. Greedy randomized adaptive search procedures.Journal of Global Optimization, 6 :109–133, 1995.

[18]:P. Festa. Greedy randomized adaptative search procedures. AIRO News, 7 (4) :7–11, 2003.

[19] : C. Voudouris et E. Tsang. Guided local search. Technical Report TR CSM-247, University of Essex, UK, 1995.s

[20]: C. Darwin. On the Origin of Species by Means of Natural Selection, or thePreservation of Favoured Races in the Struggle for Life. J. Murray, June 1859. ISBN0486450066.

[21]: J.H. Holland. Adaptation in natural and artificial systems. Technical report,University of Michigan, Ann Arbor, 1975.

[22]: D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning.AddisonWesley, 1989.

[23]: D. Goldberg. Genetic algorithms with sharing for multimodal function optimization.In Proceedings of the Second International Conference on GeneticAlgorithms, pages 41–49, 1987..

[24]:J.H. Holland, editor. Adaptation in natural and artificial systems : An introductoryanalysis with applications to biology, control, and artificial intelligence. MITPress/Bradford books, Cambridge, MA, 1992. 2nd edition.

[25]:H.-P. Schwefel. Evolution strategies : A family of non-linear optimizationtechniques based on imitating some principles of organic evolution. Annalsof Operations Research, 1 :165–167, 1984.

[26]:D. E. Goldberg & K. Deb. A comparative analysis of selection schemesused in genetic algorithms. In Foundations of Genetic Algorithms, pp. 69–93. MorganKaufmann, 1991.

[27] : Colorni, M. Dorigo, et V. Maniezzo. An investigation of some proprietiesof an "ant algorithm". In Manner and Manderick, pages 509–520,1992b.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

Résumé

Dans ce travail nous nous intéressons à l'adaptation et l'hybridation de l'algorithme de colonies d'abeilles (BCO) pour la résolution de problème d'ordonnancement Flow Shop. L'idée consiste à insérer une méthode de recherche locale au niveau de l'algorithme de base pour améliorer l'intensification et la diversification de la technique de recherche de l'algorithme.

Les résultats de simulation montrent que l'algorithme BCO hybride est plus performant que le BCO de base pour différentes classes de systèmes proposées.

Mots clés

Ordonnancement, métaheurisyiques hybrides, algorithme de colonies d'abeilles, Flow Shop.

Abstract

In this work we are interested to the adaptation and the hybridysation of the Bee Colony Optimisation Algorithm (BCO) to solve the Flow Shop Scheduling Problem. The main idea is to introduce a local search technique in the basic algorithm to improve the intensification and diversification of the algorithm.

Computational results show that the BCO algorithm is more powerful than the basic BCO for different problem classes.

Key words

Scheduling, hybrid metaheuristics, Bee Colony Algorithm, Flow Shop.

ملخص

نتطرق في هذا العمل لتطبيق و تهجين خوارزمية مستعمرة النحل لحل إشكالية الترتيبات أحادي المسار. تتمثل الفكرة في إدراج طريقة بحث موضعية على مستوى الخوارزمية الأصلية لتحسين البحث المكثف و المنتشر لتقنية البحث الخاصة بالخوارزمية.

نتائج المحاكاة المحصل عليها بينت أن خوارزمية مستعمرة النحل المهجنة هي أكثر فاعلية من الخوارزمية الأصلية لدى تطبيقها على عدة طبقات من الأنظمة المقترحة.

الكلمات المفتاحية

الترتيبات، فوقيات الاستدلال المهجنة، خوارزمية مستعمرات النحل، أحادي المسار.