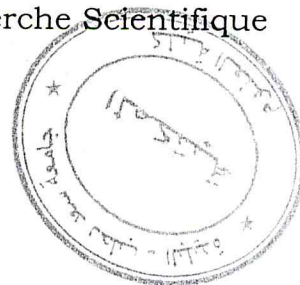


République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université Saad Dahlab, Blida
USDB.

Faculté des Sciences.
Département de Mathématiques.

Mémoire

pour l'obtention du diplôme de Master en
Recherche Opérationnelle

Thème :

*Approche méta- heuristique pour la résolution des problèmes
d'ordonnancement industriel de type flow shop hybride*

Présenté par : GACEM Zohra

Promoteur : GAHAM Mehdi

Organisme d'accueil : *Centre de Développement de Technologies Avancées*

Soutenu le : 05 novembre 2011, devant le jury composé de :

Président : MESSAOUDENE Lyes

USDB

Examineur : SAKRI Redha

U KHMISS Miliana

Rapporteurs : GAHAM Mehdi

CDTA

MOUSSAOUI Hichem

USDB

- 2010/2011-

MA-510-10-1

RESUME:

Ce mémoire a pour objectif d'appliquer une approche méta heuristique pour la résolution des problèmes d'ordonnancement industriel de type flow shop hybride.

Les problèmes d'ordonnancement des ateliers constituent pour les entreprises une des difficultés majeures de leur système de gestion et de conduite de la production, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers.

Le problème proposé s'intéresse précisément à l'ordonnancement d'atelier de Production de type flow shop hybride avec machines parallèles identiques en minimisant la durée totale de l'ordonnancement.

L'objet de ce mémoire est donc de proposer et tester une méthode approchée qui est l'algorithme génétique pour la résolution du problème considéré.

Une phase expérimentale a été menée pour vérifier l'efficacité de l'algorithme génétique pour l'ordonnancement déterministe, sans tenir compte des incertitudes.

Mots clés : Flow shop hybride, Algorithme génétique, LEKIN,



Remerciements

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 serais en effet ingrate si je n'exprime pas ma reconnaissance et ma gratitude à tous ceux, de près ou de loin, qui ont facilité ma tâche et m'ont permis de mener à bien ce travail.

Mes vifs remerciements à Monsieur BOUZOUIA Brahim, Directeur du Centre de Développement des Technologies Avancée, de m'avoir donné l'opportunité de m'inscrire en Master 2.

Je tiens tout particulièrement à exprimer toute ma gratitude et mes vifs remerciements à Monsieur GAHAM Mahdi qui m'a accordé sa confiance en me proposant ce sujet de mémoire et pour son aide avec son expérience et savoir faire. Je les prie de croire à ma respectueuse estime et ma sincère reconnaissance pour ses conseils qui m'ont été très précieux et indispensables

Je ne saurais oublier de remercier Monsieur MOSSAOUI Hichem, de l'université Saad Dahlab de Blida qui a accepté d'être mon co- encadreur.

J'adresse mes remerciements à Monsieur CHERGUI Mahmoud , Vice Recteur chargé de la pédagogie, Monsieur BENZINA Mohamed ancien doyen de la faculté des sciences, Monsieur TAMI Mohamed, chef de département de mathématiques de m'avoir accueilli au sein de l'institut de mathématique de l'université Saad Dahlab de Blida et m'ont permis d'accomplir ce mémoire dans les meilleurs conditions.

Mes remerciements vont également à tous les membres de jury pour avoir accepté d'évaluer ce mémoire : Monsieur MESSAOUDENE Lyes comme président de jury, Monsieur SAKRI Redha comme examinateur.

Un grand Merci à mes parents qui m'ont donné la chance de poursuivre mes études et qui m'ont appris à surpasser les moments difficiles.

Tables des matières

Liste des figures.....	1
Liste des tableaux.....	2
Introduction générale.....	3
CHAPITRE 1 Ordonnancement d'atelier	
1 Introduction.....	6
2 l'ordonnancement de production	6
2.1 Concepts de base de l'ordonnancement.....	7
2.1.1 Tâches.....	7
2.1.2 Ressources.....	7
2.1.3 Contraintes.....	8
2.1.4 Objectifs.....	9
3 Ordonnancement dans les différents types d'ateliers.....	10
3.1 Machine unique.....	10
3.2 Machines parallèles.....	10
3.3 Atelier à cheminement unique (Flow shop)	11
3.4 Atelier flow shop hybride.....	12
3.5 Atelier à cheminement multiple (Job shop)	13
3.6 Atelier à cheminement libre(Open shop).....	14
3.7 Atelier flexible.....	14
4 Complexité des problèmes d'ordonnancement.....	14
5 Problématique d'ordonnancement flow shop hybride.....	15
5.1 Présentation.....	15
5.2 État de l'art du problème d'ordonnancement du flow shop hybride	16
5.3 Modélisation.....	18
5.4 Représentation des solutions.....	18
6 Méthodes de résolution	20
6.1 Programmation dynamique.....	20
6.2 Programmation linéaire	21
6.3 Les procédures par séparation et évaluation.....	21
6.4 Les heuristiques spécifiques	21
6.5 Les méta- heuristiques.....	22
6.6 Comparaison entre méthodes exactes et méthodes approchées	22
7 Conclusion.....	22

CHAPITRE 2 Les Méta –heuristiques

1	Introduction.....	24
2	Méta- heuristiques : Etat de l'art.....	24
2.1	Classification des méta- heuristiques.....	26
2.2	Approche constructive.....	27
2.3	Approche de recherche locale.....	27
2.3.1	Recuit simulé.....	28
2.3.2	Méthodes d'acceptation à seuil ou méthode de descente.....	28
2.3.3	Méthode Tabou.....	29
2.4	Approche évolutive.....	30
2.4.1	Algorithme génétique.....	30
2.4.2	Algorithme de la fourmi	30
2.4.3	Méthode de recherche distribuée.....	31
2.4.4	La méthode essaim particulière PSO.....	32
2.5	Quelques propriétés associées aux méthodes approchées.....	32
2.6	Autres méthodes de résolution	32
3	Quelle méta-heuristique utiliser ?.....	33
4	Analyse.....	34
5	Conclusion.....	35

CHAPITRE 3 Algorithme génétique pour l'ordonnancement de type flow shop hybride

1	Introduction.....	37
2	Concept de base d'un algorithme génétique	38
3	Terminologie.....	39
4	Les étapes de l'algorithme génétique.....	39
4.1	Codage.....	41
4.2	Génération de la population initiale.....	42
4.3	Evaluation : fitness.....	42
4.4	Sélection.....	42
4.5	Croisement.....	43
4.6	Mutation.....	44
4.7	Insertion.....	45
4.8	Critère d'arrêt.....	45
5	Affectation et séquence ment.....	46

6 Application de l'algorithme génétique au problème d'ordonnancement de type flow shop hybride.....	46
6.1 Les paramètres de l'algorithme génétique.....	46
6.2 L'organigramme de l'algorithme génétique.....	47
6.3 Codage utilisé pour notre algorithme.....	49
6.4 Fonction d'évaluation : fitness.....	51
6.5 Génération d'une population initiale.....	52
6.6 Opérateur de sélection.....	52
6.7 Opérateur d'élitisme.....	53
6.8 Opérateur de croisement.....	53
6.9 Opérateur de mutation.....	55
6.10 Critère d'arrêt.....	56
7 Conclusion.....	56

CHAPITRE 4 : Evaluation et validation de l'algorithme génétique

1 Introduction.....	58
2 Validation de l'algorithme génétique pour l'ordonnancement flow shop hybride	58
2.1 Description des benchmarks.....	58
2.2 Expérimentation et résultat.....	59
3 Conclusion.....	71

Conclusion générale et perspectives.....	72
---	-----------

Bibliographie

Annexes

LISTE DES FIGURES

Figure 1 - Machine unique.....	10
Figure 2 - Machines parallèles.....	11
Figure 3 - Atelier à cheminement unique (flow shop)	12
Figure 4 - Atelier à cheminement multiple.....	14
Figure 5 - Flow shop hybride à k étages.....	15
Figure 6 - Exemple de diagramme de Gantt.....	19
Figure 7 - Procédure de descente	29
Figure 8 - Recueil des ressources par des fourmis.....	31
Figure 9 - Fonctionnement général de l'algorithme génétique.....	40
Figure 10 - Croisement en 1- point de deux chromosomes.....	43
Figure 11 - Croisement en 2- points de deux chromosomes.....	44
Figure 12 - Mutation : opérateur d'insertion.....	45
Figure 13 - L'organigramme de l'algorithme génétique.....	47
Figure 14 - Présentation de la partie machine sélection (MS).....	49
Figure 15 - Présentation de la partie opération sélection (OS).....	50
Figure 16 - présentation de la solution à l'aide de diagramme de Gantt.....	51
Figure 17 - Croisement en deux points.....	54
Figure 18 - Croisement uniforme.....	54
Figure 19 - Croisement de type POX.....	55
Figure 20 - Comparaison entre l'AG et les règles de priorités de LEKIN.....	61
Figure 21 - Comparaison entre l'AG et l'heuristique SBLS.....	62
Figure 22 - Comparaison entre l'AG et SB ROUTINE.....	62
Figure 23 - Performance des trois méthodes.....	69
Figure 24 - Performance des trois méthodes instances faciles.....	70
Figure 25 - Performance des trois méthodes instances faciles	70

LISTE DES TABLEAUX

Tableau 1 - Temps opératoire pour l'exemple d'un flow shop.....	19
Tableau 2 - Comparaison générale des principales méta- heuristiques.....	33
Tableau 3 - Exemple de problème d'ordonnancement de type flow shop hybride.....	49
Tableau 4 - Différents types de configurations.....	59
Tableau 5 - Comparaison entre l'algorithme génétique et LEKIN.....	61
Tableau 6 - Comparaison entre AG, B&B et AIS pour l'instance de type j10c5a.....	63
(instance faciles)	
Tableau 7 - Comparaison entre AG, B&B et AIS pour l'instance de type j10c5b.....	63
(instance faciles)	
Tableau 8 - Comparaison entre AG, B&B et AIS pour l'instance de type j10c5c.....	64
(instance difficiles)	
Tableau 9 - Comparaison entre AG, B&B et AIS pour l'instance de type j10c10b.....	64
(instances faciles)	
Tableau 10 -Comparaison entre AG, B&B et AIS pour l'instance de type j10c10c.....	65
(instances faciles)	
Tableau 11 -Comparaison entre AG, B&B et AIS pour l'instance de type j15c5a.....	65
(instances faciles)	
Tableau 12 -Comparaison entre AG, B&B et AIS pour l'instance de type 15c5b.....	66
(instance faciles)	
Tableau 13 -Comparaison entre AG, B&B et AIS pour l'instance de type j15c5c.....	66
(instances difficiles)	
Tableau 14 - Comparaison entre AG, B&B et AIS pour l'instance de type j15c5d.....	64
(instances difficiles)	
Tableau 15 - Comparaison entre AG, B&B et AIS pour l'instance de type j15c10a.....	67
(instances faciles)	
Tableau 16 - Comparaison entre AG, B&B et AIS pour l'instance de type j15c10b.....	68
(instances faciles)	
Tableau 17 - Performance des trois méthodes.....	68

INTRODUCTION GENERALE

Introduction générale

La théorie de l'ordonnancement est une branche de la Recherche Opérationnelle, elle occupe un rôle important dans nombreux secteurs de l'économie, notamment en gestion de production des entreprises et des services. L'enjeu économique considérable de ces problèmes ainsi que leur champ d'application qui n'ont pas cessé d'augmenter et d'évoluer, ont suscité depuis des décennies une recherche intensive.

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 critères donnés, tout en respectant les contraintes de fabrication et d'organisation.

En général les méthodes de résolutions utilisées pour ce type de problèmes prennent en considération deux facteurs : La qualité des solutions et le temps de résolution. Ainsi elles peuvent être classées en deux catégories : Les méthodes exactes qui garantissent l'optimalité mais avec un temps de calcul qui risque d'augmenter avec la taille du problème, et les méthodes approchées qui peuvent perdre en optimalité pour gagner en temps d'exécution. Parmi ces dernières les méta-heuristiques tiennent une place très importante du fait qu'elles permettent en général la résolution d'instances très complexes de ces problèmes en des temps de calcul raisonnables.

Nous pouvons distingués parmi les problèmes d'ordonnancement d'ateliers les plus rencontrés ceux à machine unique, à machines parallèles, le flow shop, le flow shop hybride, le job shop, l'open shop, etc.

Dans l'étude théorique de ce mémoire, nous nous focalisons sur l'ordonnancement d'atelier de type flow shop hybride avec machines parallèles identiques en minimisant la durée totale de l'ordonnancement ou le Makespan (C_{max}). L'objet de ce mémoire est donc d'adapter et de tester une méta-heuristique évoluée, qui est l'algorithme génétique pour la résolution du problème considéré.

Ainsi le mémoire est organisé de la manière suivante :

Dans le premier chapitre, nous présentons une introduction aux problèmes d'ordonnancement d'ateliers, les concepts de base de l'ordonnancement et les différents types d'ateliers.

Dans le second chapitre, nous abordons les méthodes méta-heuristiques en présentant les concepts de base et certaines de ces approches.

Le troisième chapitre est consacré à l'adaptation d'un algorithme génétique au flow shop hybride

Quant au quatrième chapitre, il est consacré à la validation expérimentale de l'approche par un ensemble de tests sur des benchmark de la littérature. Une étude comparative critique est ensuite présentée.

CHAPITRE 1

Ordonnancement d'atelier

Chapitre 1

Ordonnancement d'atelier

Sommaire

1	Introduction.....	6
2	l'ordonnancement de production	6
2.1	Concepts de base de l'ordonnancement.....	7
2.1.1	Tâches.....	7
2.1.2	Ressources.....	7
2.1.3	Contraintes.....	8
2.1.4	Objectifs.....	9
3	Ordonnancement dans les différents types d'ateliers.....	10
3.1	Machine unique.....	10
3.2	Machines parallèles.....	10
3.3	Atelier à cheminement unique (Flow shop)	11
3.4	Atelier flow shop hybride.....	12
3.5	Atelier à cheminement multiple (Job shop)	13
3.6	Atelier à cheminement libre(Open shop).....	14
3.7	Atelier flexible.....	14
4	Complexité des problèmes d'ordonnancement.....	14
5	Problématique d'ordonnancement flow shop hybride.....	15
5.1	Présentation.....	16
5.2	État de l'art du problème d'ordonnancement du flow shop hybride ...	17
5.3	Modélisation.....	18
5.4	Représentation des solutions.....	18
6	Méthodes de résolution	20
6.1	Les procédures par séparation et évaluation.....	20
6.2	Programmation dynamique.....	21
6.3	Programmation linéaire	21
6.4	Les heuristiques spécifiques	21
6.5	Les méta- heuristiques.....	22
6.6	Comparaison entre méthodes exactes et méthodes approchées	22
7	Conclusion.....	22

1 Introduction

L'ordonnancement d'atelier consiste à organiser dans le temps le fonctionnement d'un atelier pour utiliser au mieux les ressources humaines et matérielles disponibles dans le but de produire les quantités désirées dans le temps imparti [3, 7].

Les données d'un problème d'ordonnancement sont les tâches et leurs caractéristiques, les contraintes potentielles, les ressources, et la fonction objectif [4].

Une solution à un tel problème consiste à trouver une planification des tâches sur les ressources en optimisant les objectifs et en respectant les contraintes.

Dans le cadre de notre étude, nous nous intéressons principalement aux problèmes d'ordonnancement d'atelier. Dans ce cas, une tâche est généralement désignée par le terme opération. Il s'agit d'une entité élémentaire d'un travail (job en anglais). Quant aux ressources, elles sont représentées par des machines. La modélisation d'un tel problème dépend fortement des différents paramètres tels que les contraintes et les objectifs.

Ce chapitre est subdivisionné en cinq sections. La première section consiste à présenter l'ordonnancement de production. La deuxième section présente quelques généralités sur les différents types d'ateliers. Dans la troisième section nous abordons la complexité des problèmes d'ordonnancement d'atelier. Dans la quatrième section, nous présentons la problématique d'ordonnancement flow shop hybride et la dernière section sera consacrée aux méthodes de résolution.

2 Ordonnancement de production

L'ordonnancement est la principale fonction de base utilisée pour le pilotage des activités opérationnelles de systèmes physiques de production

Ordonnancer un ensemble de tâches, c'est programmer leur exécution en leur allouant les ressources requises et en fixant leurs dates de début [4].

Un problème d'ordonnancement est composé d'un ensemble de tâches soumises à certaines contraintes dont l'exécution de ces tâches nécessite des ressources.

Résoudre un problème d'ordonnancement consiste à programmer dans le temps l'exécution des tâches, en leur allouant les ressources requises et en respectant les contraintes imposées, dans le but d'atteindre certains objectifs.

2.1 Concepts de base de l'ordonnancement

Un problème d'ordonnancement est défini par les tâches à réaliser, les contraintes potentielles, les ressources disponibles ainsi que la fonction économique à optimiser [4].

2.1.1 Tâches

Une tâche i est une entité élémentaire de travail localisée dans le temps par une date de début t_i ou de fin c_i , dont la réalisation est caractérisée par une durée p_i (on a $c_i = t_i + p_i$.) En outre, la tâche i utilise une (ou plusieurs) ressource k avec une intensité a_{ik} souvent supposée constante pendant l'exécution de la tâche, [7].

Dans certains problèmes, les tâches peuvent être exécutées par morceaux. L'entrelacement des différents morceaux permettant de laisser le moins possible les ressources inactives. Dans d'autres problèmes au contraire on ne peut pas interrompre une tâche une fois commencée. On parle respectivement de problème préemptifs et non préemptifs.

2.1.2 Ressources

Une ressource k est un moyen technique ou humain requis pour la réalisation d'une tâche et disponible en quantité limitée, sa capacité A_k , [7].

On distingue plusieurs types de ressources :

Ressources renouvelables

Si après avoir été utilisée par une ou plusieurs tâches, elles sont à nouveau disponibles en même quantité (les hommes, les machines, l'espace, etc.) . La quantité de ressources utilisée à chaque instant est limitée [7].

Ressources consommables

Elles sont utilisées une seule fois (budget, matière première), la consommation globale (ou cumul) au cours du temps est limitée.

Ressources disjonctives ou non partageables

Il s'agit des ressources qui ne peuvent être utilisées que par une tâche à la fois (machine, robot).

Ressources cumulatives ou partageables

Elles peuvent être utilisées par plusieurs tâches simultanément (Equipe d'ouvriers).

2.1.3 Contraintes

Deux types de contraintes sont distinguées en ordonnancement à savoir les contraintes temporelles et les contraintes de ressources.

Les contraintes temporelles

Elles intègrent en général :

**Les contraintes de temps alloué*, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (délai de livraison par exemple) ou à la durée totale d'un projet.

**Les contraintes d'antériorité et plus généralement les contraintes de cohérence technologique*, qui décrivent le positionnement relatif de certaines tâches par rapport à d'autres (exemple contraintes de gammes dans le cas des problèmes d'ateliers).

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

Contraintes de ressources

Les contraintes de ressources traduisent le fait que celles-ci sont disponibles en quantité limitée (leur capacité).

On distingue deux types de contraintes de ressources, liées à la nature disjonctive ou cumulative des ressources.

Pour les ressources disjonctives, ces contraintes de ressources permettent de spécifier que deux tâches ne peuvent pas être exécutées en même temps sur la même ressource.

Pour les ressources cumulatives, ces contraintes de ressources permettent d'assurer que la somme des besoins simultanés des tâches pour une ressource est inférieure à la capacité de la ressource.

2.1.4 Objectifs

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.

Pour évaluer la qualité d'un ordonnancement, des mesures connues sous l'appellation de Critères, dites aussi objectifs, sont utilisées. Ces critères sont généralement liés aux temps, aux ressources ou aux coûts. Il s'agit, le plus souvent, de considérer le maximum ou la somme (qui peut être une somme totale ou pondérée) sur tous les travaux d'une certaine mesure ou d'une combinaison de plusieurs mesures. Parmi les critères les plus utilisés, nous retrouvons :

- **la durée totale de l'ordonnancement** (« makespan ») définie par $C_{\max} = \max C_j$, C_j étant la date de fin d'exécution du travail j qui représente la date d'achèvement du travail le plus tardif. En minimisant ce critère, nous avons une utilisation plus efficace des Ressources.

- **la somme des dates de fin des travaux** (« total completion time ») $\sum_{1 \leq j \leq n} C_j$

- **le plus grand retard algébrique** défini par $L_{\max} = \max L_j = L$ tel que $L_j = C_j - d_j$ est le retard algébrique (« lateness » en anglais) et d_j désigne la date de fin souhaitée ou encore la date d'échéance (« due date » en anglais)

le plus grand retard vrai $T_{\max} = \max T_j$, où $T_j = \max(0, C_j - d_j)$ est le retard vrai $1 \leq j \leq n$

(« tardiness » en anglais) du travail j

- **la somme des retards** $\sum_{1 \leq j \leq n} T_j$

- **le nombre de travaux en retard** $\sum_{1 \leq j \leq n} U_j$, U_j étant l'indicateur de retard qui est égal à 1

Si $T_j > 0$ et 0 sinon

la somme des avances $\sum_{1 \leq j \leq n} E_j$, tel que $E_j = \max(0, d_j - C_j)$ représente l'avance du travail $1 \leq j \leq n$

- **la somme des encours** $\sum_{1 \leq j \leq n} F_j$, définie par le temps de séjour des travaux dans l'atelier $1 \leq j \leq n$

$F_j = C_j - r_j$, r_j étant la date de disponibilité du travail j .

3 Ordonnancement dans les différents types d'ateliers

Dans le cas d'un problème d'ordonnancement d'atelier, les tâches à réaliser correspondent aux opérations élémentaires de fabrication sur les ressources correspondant aux machines sur lesquelles les opérations sont exécutées.

3.1 Machine unique

Dans ce cas, chaque travail comprend une seule opération, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine. Dans ce cas l'ordonnancement se résume au séquençement des opérations.

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. L'étude peut alors être restreinte à l'étude de cette machine, [7].

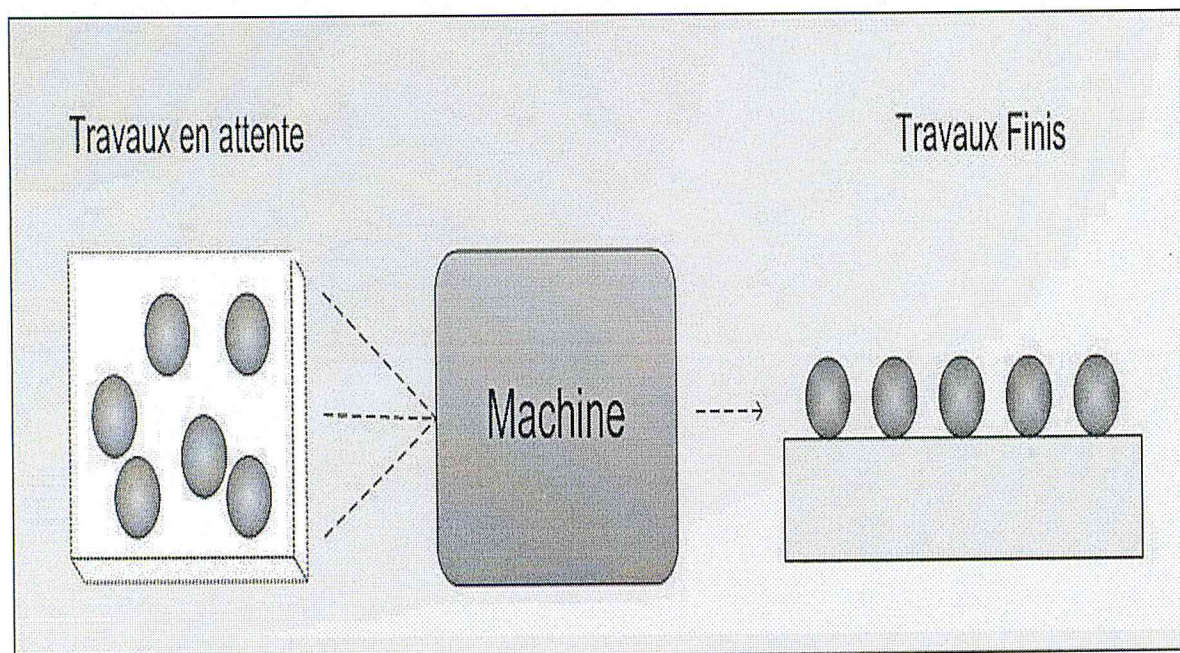


Fig .1 - Machine unique

3.2 Machines parallèles

Plusieurs machines sont pour l'exécution d'un travail qui nécessite qu'une machine. Ce problème est une généralisation du problème à une machine, il est souvent rencontré dans le contexte du flow shop hybride.

La résolution passe par le traitement des décisions d'affectation s'effectue en deux étapes:

- 1- Décider sur quelle machine effectuer chaque opération
- 2- Déterminer la séquence d'opérations sur chaque machine

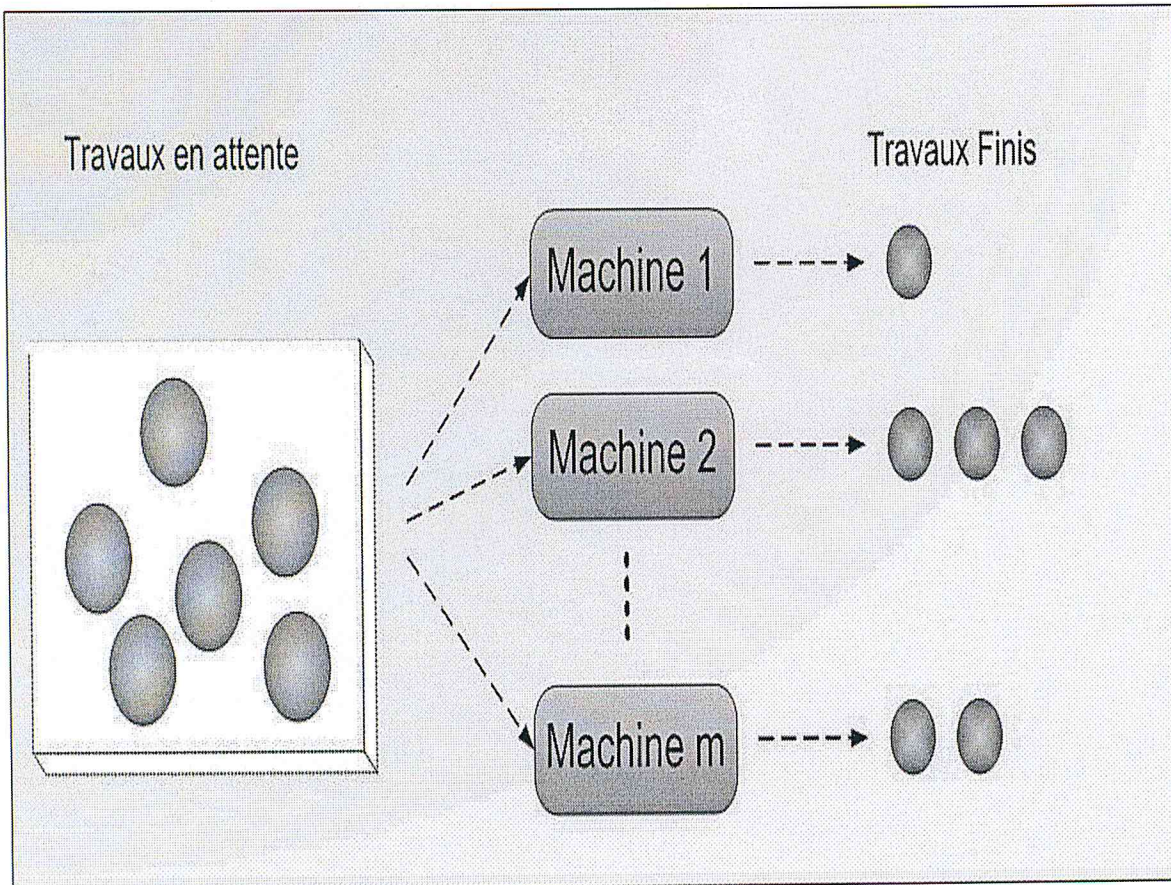


Fig.2 - Machines parallèles

3.3 Atelier à cheminement unique (Flow shop)

Tout travail visite chaque machine de l'atelier et l'ordre de passage d'un travail sur les différentes machines est le même pour tous les travaux (flot unidirectionnel). Un atelier à cheminement unique est un atelier où le processus d'élaboration de produits est dit « linéaire », c'est-à-dire lorsque les étapes de transformation sont identiques pour tous les produits fabriqués. Selon les types de produits élaborés, on distingue la production continue et la production discrète. La production continue est caractérisée par la fluidité de son processus et l'élimination du stockage. C'est le cas notamment dans les raffineries, les cimenteries, les papeteries... La production discrète de masse s'applique principalement aux produits de grande consommation fabriqués à la chaîne (exemple automobile, la majorité du domaine du textile, machines-outils...).

Dans les deux cas, les machines peuvent être dédiées à une opération précise, et sont implantées en fonction de leur séquence d'intervention dans la gamme de production.

L'un des objectifs principaux dans le cas d'atelier à cheminement unique est de trouver une séquence des tâches en main qui respecte un ensemble de contraintes et qui minimise le temps total de production. Parmi les caractéristiques d'un problème de cette catégorie :

- il existe au minimum $n!$ différentes solutions où n est le nombre de travaux à réaliser. Notons que $n! = n \times (n-1) \times (n-2) \times \dots \times 1$;
- le problème est NP- difficile à l'exception des versions avec deux machines et certains cas particuliers avec trois machines ;
- une grande productivité mais une faible flexibilité.

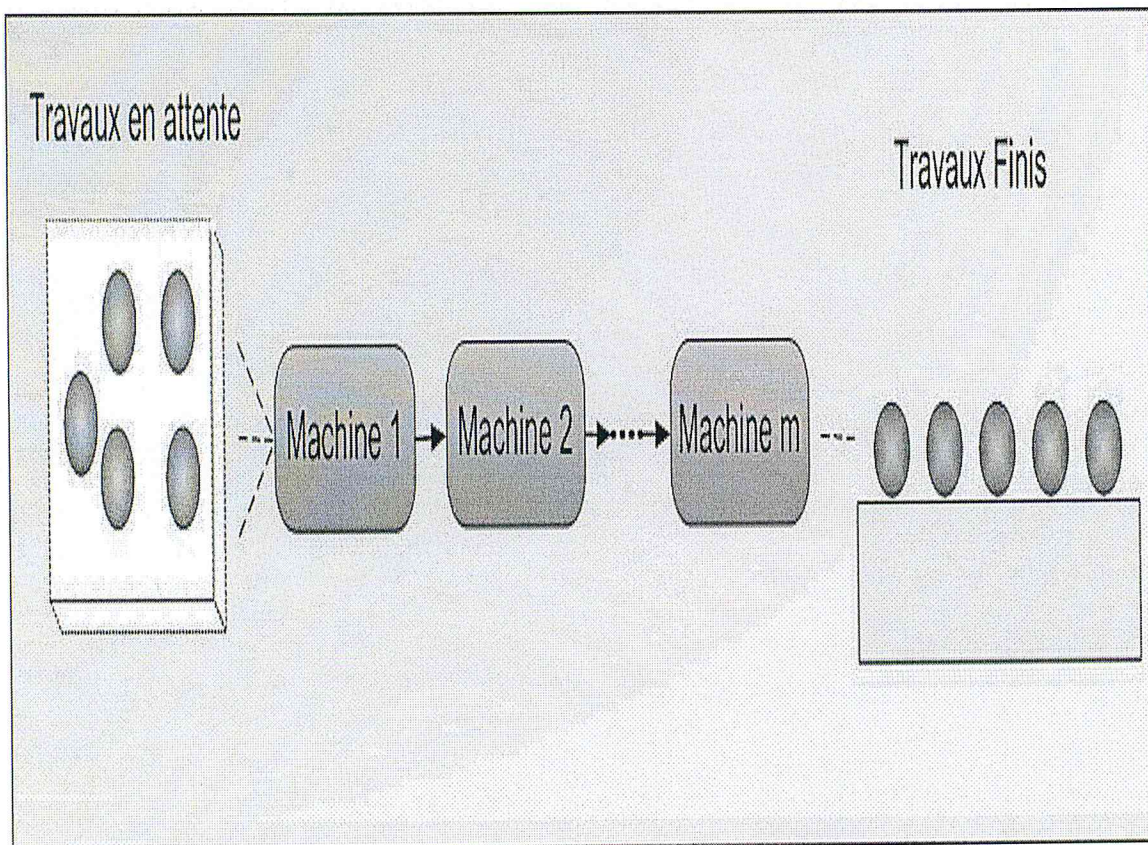


Fig . 3 - Atelier à cheminement unique (Flow shop)

3.4 Atelier flow shop hybride

Un atelier flow-shop hybride, également appelé flow-shop multiprocesseur, représente un problème dans lequel, d'une part, le cheminement de l'atelier est le même pour tous les travaux (le flot des travaux à travers l'atelier est unidirectionnel) et d'autre part, les machines sont en exemplaire multiples et regroupées en étages (dites aussi centres ou cellules) disposées en séries de production, [7].

Les machines sur chaque étage peuvent être identiques, uniformes, ou non reliées.

Dans la littérature, le flow-shop hybride est référé par FSH.

3.5 Atelier à cheminement multiple (job shop)

Les ateliers à cheminements multiples (ACM) sont des unités manufacturières traitant une variété de produits individuels dont la production requiert divers types de machines dans des séquences variées. L'une des caractéristiques d'un atelier à cheminement multiple est que la demande pour un produit particulier est généralement d'un volume petit ou moyen. Une autre caractéristique est la variabilité dans les opérations et un mix produit constamment changeant. Ainsi, il est nécessaire que le système soit de nature flexible. Dans un sens général, la flexibilité est la capacité d'un système de répondre aux variations dans l'environnement.

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

La figure 4 montre un exemple d'un atelier à cheminements multiples avec quatre travaux et six machines.

Le cas du job shop se rencontre dès qu'un atelier fabrique des pièces présentant des caractéristiques différentes mais nécessitant néanmoins des traitements dont les séquences sont connus à l'avance.

Chaque travail passe sur les machines dans un ordre fixé mais à la différence du flow shop, cet ordre peut être différent pour chaque travail (flot).

Parmi les caractéristiques d'un problème d'ordonnancement dans un atelier à cheminements multiples :

- * Le nombre de solutions possibles est de l'ordre de $(n!)^m$, où n est le nombre de tâches à effectuer et m le nombre de machines. Notons qu'une tâche veut dire la même chose qu'un travail.

- * Le problème est NP-difficile et est considéré parmi les problèmes les plus difficiles à traiter.

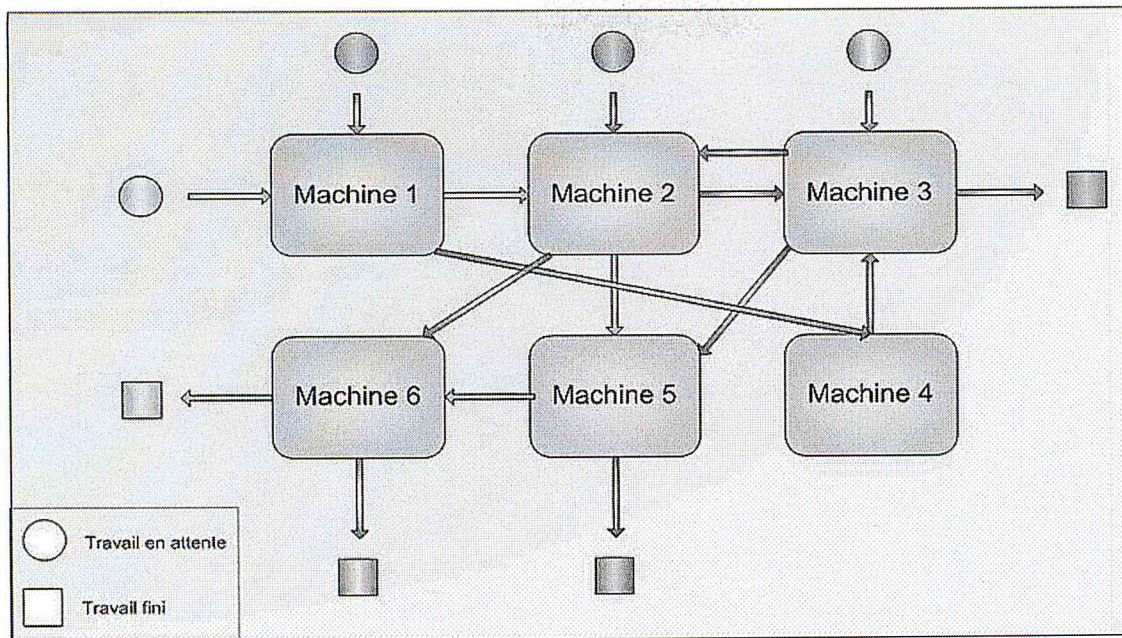


Fig.4 – Atelier à cheminement multiple

3.6 Atelier à cheminements libres (open shop)

Dans le cas d'atelier à cheminement libre (open shop), chaque produit à traiter doit subir un ensemble d'opérations sur un ensemble de machines, *mais dans un ordre totalement libre*.

Il n'y a pas de contraintes de précédence entre deux opérations d'un même travail.

3.7 Atelier flexible

Ces ateliers sont caractérisés par un niveau d'automatisation élevé, cherchant par là un compromis entre flexibilité et productivité. Ils sont à la base des ateliers à cheminements multiples où les principales tâches (stockage, traitement de pièces, manutention...) sont automatisées.

4 Complexité des problèmes d'ordonnancement

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté théorique intrinsèque des problèmes en informatique ou d'un problème par rapport à un autre et de l'analyse de la complexité des programmes et des algorithmes. Concrètement, on cherche à savoir si le problème étudié est plutôt « facile » ou plutôt « difficile » à résoudre en se basant sur une estimation (théorique) des temps de calcul et des besoins en mémoire informatique.

Les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire. Pour résoudre un problème d'ordonnancement, nous devons toujours chercher à établir sa *complexité*, car cela *détermine la nature de l'algorithme à mettre en oeuvre*. Si le problème étudié appartient à la *classe P*, nous savons d'avance qu'un *algorithme polynomial existe* pour le résoudre. Dans le cas contraire, si le problème est *NP-difficile*, deux approches sont possibles. La première est de proposer un *algorithme approché*, donc une heuristique, qui

calcule en temps polynomial une solution s'approchant au mieux de la solution optimale. Une alternative est de proposer un algorithme qui calcule la solution optimale du problème, mais pour lequel la complexité dans le pire des cas est exponentielle. Dans ce cas, le défi est de concevoir un algorithme qui peut résoudre en temps acceptable les problèmes de la plus grande taille possible.

5 Problématique d'ordonnancement flow shop hybride

5.1 Présentation

Une organisation de type flow shop hybride constitue notre cas d'étude. Il s'agit d'ordonnancer un ensemble de n jobs types qui sont représentatifs des scénarios de production possibles $J_1, J_2, \dots, J_n$ dans un atelier de production composé de plusieurs étages k .

Chaque job J_j est constitué de k opérations types O_{jt} : O_{jt} est l'opération du job j traitée par l'étage t , [8].

L'étage t contient un ensemble de $M_t \geq 1$ machines parallèles identiques, ce qui signifie que les jobs peuvent être exécutés indifféremment sur l'une ou l'autre des machines d'un même étage. Chaque job doit passer sur une des machines dans chaque étage (voir figure 5)

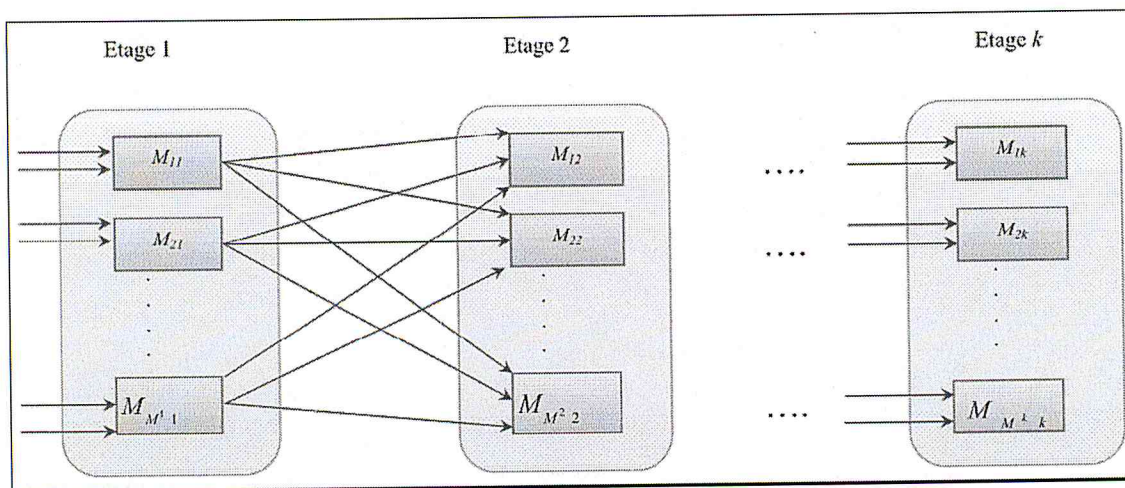


Fig. 5 - Flow shop hybride à k étages

Une fois que l'exécution d'un job a débuté sur une machine, celle-ci ne peut pas être interrompue. Ceci revient à dire que l'ordonnancement est non préemptif. Le job ne peut pas être exécuté dans des différentes machines. Aucun job ne peut commencer sur cette machine avant la fin du job en cours.

Pour chacun des jobs, l'ordre de passage des opérations dans les k étages est le même :

$O_{j1}, O_{j2}, \dots, O_{jt}$. La machine M_{it} avec $i = 1, 2, \dots, M_t$ et $t = 1, 2, \dots, k$ ne peut exécuter qu'un seul job à la fois.

Le temps opératoire du job j à l'étage t est noté P_{jt} (les temps de montage, de transport, de décharge sont contenus dans le temps opératoire du job).

5.2 Etat de l'art du problème d'ordonnancement flow shop hybride

Dans notre mémoire, nous nous basons dans notre étude au problème du flow shop hybride statique et déterministe.

Nous restreindrons notre étude au problème du flow shop hybride statique et déterministe. Les hypothèses suivantes sont retenues :

- Les travaux sont indépendants les uns des autres
- Tous les travaux sont connus et disponibles dès le début de l'ordonnancement ;
- Les machines sont disponibles du début à la fin de l'ordonnancement
- Les machines sur chaque étage sont supposées identiques
- Chaque machine ne peut réaliser à un instant donné qu'une seule opération à la fois
- Les durées opératoires sont déterministes et connues à l'avance
- Les temps de montages et de démontages sont inclus dans les temps opératoires ;
- Les temps de transport sont négligeables
- Les opérations peuvent attendre dans des stocks de capacité illimitée entre les étages
- La préemption des opérations n'est pas autorisée. Ainsi une opération ne peut être interrompue une fois que son exécution a débuté sur une machine.

Nous adaptons un algorithme génétique pour la résolution du problème d'ordonnancement du flow shop hybride dans un environnement déterministe. Cet algorithme consiste à minimiser seulement le makespan, sans prendre en compte les perturbations sur les temps d'exécution des jobs sur les machines. L'objectif de l'adaptation de cet algorithme est de vérifier son efficacité dans le cas déterministe du flow shop hybride.

Cette méthode a été implémentée l'année dernière pour la résolution des problèmes d'ordonnancement job shop flexible.

Les problèmes d'ordonnancement type flow shop hybride (FSH) sont connus pour être des problèmes très complexes et fortement combinatoires rencontrés dans bons nombres d'applications réelles. Le FSH est la plupart du temps classé NP-Difficile. Pour des instances qui ne contiennent que deux étages, même dans le cas où l'un des étages ne contient que deux machines et le deuxième une machines, le problème reste classé NP-Difficile. Dans ce qui suit un état de l'art succinct sur les approches de résolution de ce problème inspiré de [8] et [13] est présenté.

Les problèmes d'ordonnancement type flow shop hybride sont largement étudiés dans littérature depuis les années 1970. La complexité de ce problème a fait que beaucoup de travaux ont été consacré à sa résolution. Les différentes approches souvent employées sont les approches exactes, heuristiques et méta-heuristiques.

Les approches exactes

L'approche par séparation et évaluation est de loin la technique la plus utilisée pour la résolution optimale du FSH. Cette méthode est effective pour la résolution optimale de petites instances du problème comme pour la résolution de grande instance mais toujours dans le cadre de variantes simplifiées (deux étages, deux machines). Des variantes de cette approche ont été développées pour des instances allant de 15 jobs à parfois 1000 jobs.

L'approche par séparation et évaluation a été adaptée au problème général du FSH avec nombre d'étages et nombre de machines illimité mais elle reste applicable qu'à de petites instances. La résolution d'instances plus difficiles qui contiennent deux étages à trois machines avec 8 jobs reste très coûteuse en temps de calcul. Les approches exactes n'arrivent généralement pas à résoudre de larges instances du FSH, et elles restent très complexes à mettre en œuvre pour des applications réelles.

Les heuristiques

Les heuristiques les plus simples adoptées pour la résolution du FSH sont les règles de priorité et un nombre important de travaux ont été consacrés à leur étude et comparaison. Étant facile de mise en œuvre et adaptées au cas complexes et dynamiques, ces approches sont très appliquées dans les cas réels. Différentes autres approches heuristiques ont été développées pour le FSH et leur énumération sort du cadre de cet exposé, mais ces approches sont la plupart du temps conçues pour des cas particuliers et ne sont pas généralisables.

Les méta-heuristiques

Les approches méta-heuristiques ont beaucoup été explorées pour la résolution du FSH. Leurs principaux avantages sont la facilité d'adaptation, d'implémentation et leur capacité à trouver de bonne solution en des temps de calcul raisonnables. La recherche tabou est parmi les premières méthodes à avoir été utilisée pour la résolution du FSH. Depuis un certain nombre d'approches ont été adapté à ce problème, parmi celle-ci les approches

par population tel que les algorithmes génétiques restent très utilisés. Nous pouvons aussi citer le recuit simulé, les systèmes immunitaires artificiels, l'algorithme de fourmis...etc.

5.3 Modélisation

Un problème d'ordonnancement flow-shop hybride est défini comme étant un ensemble $J = \{1, 2, \dots, n\}$ de n travaux à ordonnancer sur k étages consécutifs de production dans le même ordre, en commençant à l'étage 1 jusqu'au dernier étage. Chaque étage i ($i = 1, 2, \dots, k$) est composé d'un ensemble $M_i = \{M_{i1}, M_{i2}, \dots, M_{i, m_i}\}$ de m_i ($m_i \geq 1$) machines identiques en parallèle, tel qu'au moins l'un de ces étages est composé de plus d'une machine. Chaque travail j est composé d'une suite d'opérations O_{ij} ($i = 1, 2, \dots, k$), une opération O_{ij} n'est exécutée que sur une machine à choisir parmi M_i . La durée opératoire de l'opération O_{ij} sur une machine $k \in M_i$ est notée p_{ijk} . L'ordre de passage des opérations pour chaque travail j est le même et l'opération $O_{i+1,j}$ ne peut commencer que si l'opération précédente O_{ij} est terminée, [8,11].

Résoudre un tel problème revient à trouver un ordonnancement réalisable pour exécuter les n travaux sur les machines tout en respectant les contraintes de précédence entre les opérations d'un même travail. Le problème consiste à affecter une machine à chaque opération (problème d'affectation) et à déterminer l'ordre dans lequel les opérations doivent être séquencées sur les machines retenues (problème de séquence ment) afin de minimiser un critère donné. Dans la théorie de l'ordonnancement, sont utilisés souvent les termes suivants : « séquence » et « ordonnancement » pour présenter ou décrire un problème d'ordonnancement. Ces termes sont souvent interchangeable. Pour mieux distinguer les nuances entre ces deux termes, il faut comprendre le sens de chaque mot. Ainsi, l'exécution d'une séquence de travaux se traduit par l'ordre de passage des travaux sur les machines. Dans une séquence, on n'a pas à préciser le temps de début ou de fin de chaque opération effectuée. Pour exécuter une séquence de travaux, on a besoin d'un calendrier contenant des informations concernant l'ordonnancement des travaux. Un ordonnancement donné permet de déterminer le temps de début et de fin d'exécution de chaque travail. Pour tout critère régulier, un séquence ment ou ordre donné permet de déterminer les dates de début et de fin de chaque travail.

5.4 Représentation des solutions

En ordonnancement, il est utile de disposer d'une représentation graphique plus précise des solutions établies. Ce but est atteint par l'utilisation de diagramme de Gantt. Ce diagramme constitue un formalisme graphique qui a été mis au point par Henry Gantt en 1910. Il s'agit d'un outil qui est couramment utilisé pour représenter la solution d'un problème d'ordonnancement.

Il permet de visualiser l'utilisation des machines, l'avancement de l'exécution des opérations sur celles-ci ainsi que les dates de début et de fin de chaque opération. La Figure 6 représente le diagramme de Gantt associé à l'ordonnancement de trois travaux (J1, J2, J3) dans un atelier flow-shop à deux machines. Les durées opératoires sont données dans le Tableau 1.

	J1	J2	J3
m1	4	2	4
m2	2	6	4

Tableau .1 - Temps opératoires pour l'exemple d'un flow-shop

À partir de ce diagramme on peut déterminer par exemple la valeur de la date d'achèvement du travail le plus tardif.

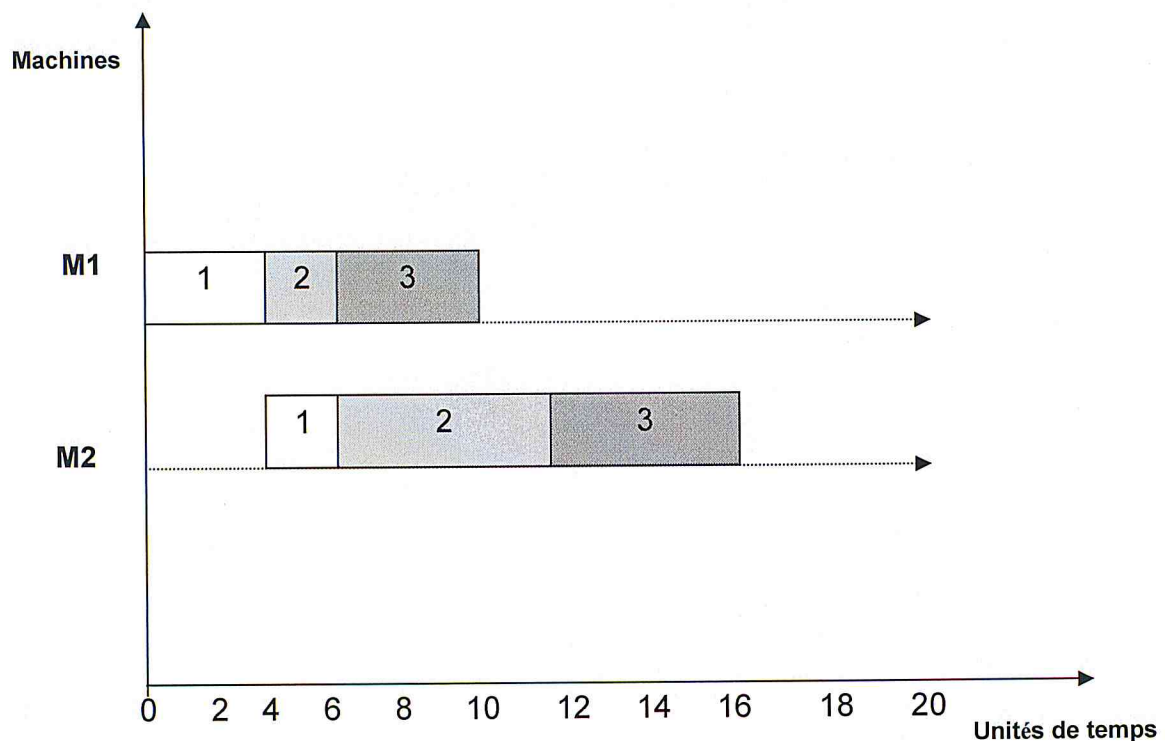


Figure. 6 - Exemple d'un diagramme de Gantt.

6 Méthodes de résolution d'optimisation combinatoire et d'ordonnancement

D'une manière très générale, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche de relaxation, approche de voisinage et l'approche d'évolution.

Ces méthodes font partie de deux groupes de nature différente.

Le premier groupe comprend les méthodes exactes d'arborescence qui garantissent la complétude de la résolution. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre.

Une méthode utilisant un critère d'optimisation est exacte si elle garantit l'optimalité des solutions trouvées, sinon elle sera dite approchée ou heuristiques, [7].

Le second groupe comprend les méthodes approchées dont le but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème.

Les méta-heuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire, [5].

La résolution des problèmes d'ordonnancement, passe par l'exploration de méthodes s'appuyant sur des techniques de résolution diverses dont :

6.1 La programmation dynamique

La programmation dynamique a été établie par Bellman (1954). On essaie alors de résoudre le problème général par récurrence sur ces paramètres entiers. Pour les problèmes d'ordonnancement, cette technique est applicable uniquement si le problème est décomposable en étape et si le critère d'optimisation présente certaines propriétés particulières par exemple une forme additive ou multiplicative, [7].

La programmation dynamique utilise le principe d'optimalité que l'on peut énoncer de la façon suivante : dans une séquence optimale de décisions, quelle que soit la première décision prise, les décisions suivantes forment une sous-suite qui est optimale, compte tenu des résultats de la première.

6.2 Programmation Linéaire

Un programme linéaire modélise un problème d'optimisation dans lequel critères et contraintes sont des fonctions linéaires des variables. Pour un programme linéaire en variables continues, les deux types d'algorithmes les plus importants sont la méthode du simplexe et la méthode des points intérieurs.

6.3 Les procédures par séparation et évaluation "Branch-and-Bound"

Les procédures d'optimisation par séparation et évaluation (PSE), (en anglais « Branch and Bound »), sont les méthodes génériques les plus utilisées pour la résolution de problèmes d'optimisation combinatoire. Elles explorent, par énumération implicite, l'ensemble de toutes les solutions. Le principe de la PSE repose sur une technique de séparation et d'évaluation. La séparation permet de décomposer le problème en un ensemble de sous-problèmes de taille réduite. Les séparations successives, effectuées durant la procédure, permettent de construire un arbre de recherche formé d'un ensemble de nœuds, chacun d'eux représentant un sous problème. L'évaluation associe une borne du critère d'optimisation (une borne inférieure dans le cas d'une minimisation ou une borne supérieure dans le cas d'une maximisation) à chaque nœud de l'arborescence de recherche. Elle a pour but de déterminer un minorant de l'optimum de l'ensemble des solutions réalisables associé au nœud en question ou, au contraire, de prouver mathématiquement que cet ensemble ne contient pas de solutions intéressantes pour la résolution du problème. Lorsqu'un tel nœud est identifié dans l'arbre de recherche, il est donc inutile d'effectuer la séparation de son espace de solutions. Étant donnée une arborescence en cours de construction, il faut donc choisir le prochain sommet à séparer.

6.4 Les heuristiques spécifiques

Afin d'améliorer le comportement d'un algorithme dans son exploration de l'espace des solutions d'un problème donné, le recours à une méthode *heuristique* (du verbe grec *heuriskein*, qui signifie « trouver ») permet de guider le processus dans sa recherche des solutions optimales.

Une **heuristique** est définie comme une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre sorte de système qui limite drastiquement la recherche des solutions dans l'espace des configurations possibles. Elle peut résoudre un problème donné, mais n'offre pas la garantie de le faire.

Dans la pratique, certaines heuristiques sont connues et ciblées sur un problème particulier.

Il s'agit de procédures qui, pour la résolution d'un problème, permettent de déterminer, parmi plusieurs méthodes de recherche, celle qui semble la plus prometteuse pour atteindre le but visé. Elles peuvent être exprimées sous formes de critères, de principes ou de méthodes, et

formalisent l'intuition que l'on a de la voie à suivre. Elles représentent un compromis entre deux exigences : le besoin de rendre de tels critères simples, et le désir de les voir établir une distinction entre les bons et les mauvais choix.

6.5 Les méta-heuristiques

Parmi les méta heuristiques, on parle des méthodes dites de recherche locale (méthode tabou, recuit simulé, méthodes d'acceptation à seuil . . .) et des méthodes d'approche évolutive (méthode de recherche dispersée, algorithme de la fourmi, algorithmes génétiques et mémétiques). Durant ces dernières années, plusieurs méta heuristiques ont prouvé leur efficacité pour la résolution de problèmes combinatoires comme les problèmes d'ordonnancement.

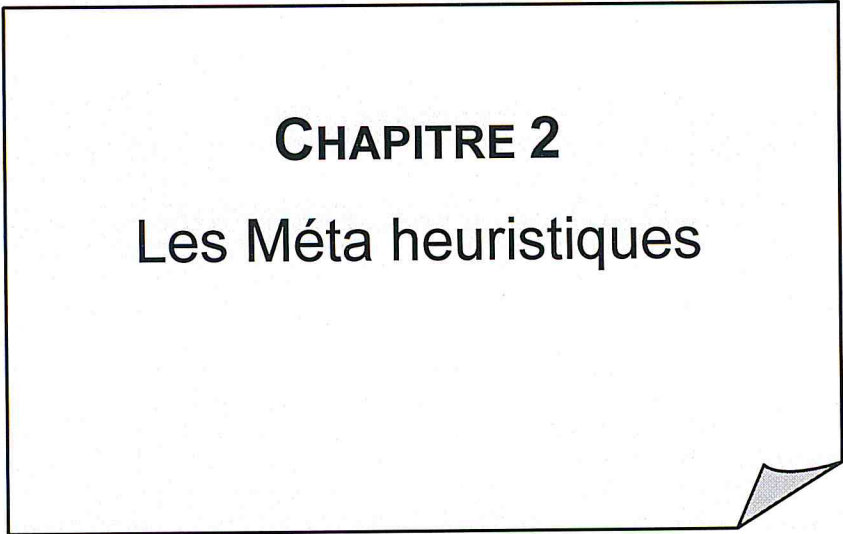
6.6 Comparaison entre méthodes exactes et méthodes approchées

On représente ci-dessous quelques différences entre les méthodes exactes et les méthodes approchées.

Méthodes approchées	Méthodes exactes
Efficacité pour les problèmes de grande taille	Efficacité pour des cas particuliers des problèmes de taille raisonnable
Espace mémoire raisonnable	Espace mémoire considérable
Durée de résolution très petite par rapport aux méthodes exactes	Durée de temps de résolution est généralement considérable pour les problèmes de grande taille
Aucune garantie d'optimalité	Garantie d'optimalité
Pratiquement simple à implanter et à comprendre	Pratiquement difficile à implanter et à comprendre

7 Conclusion

Nous avons présenté dans la première partie de ce chapitre quelques généralités et notions de base de l'ordonnancement ainsi qu'une typologie du problème d'ordonnancement selon le type d'atelier. Nous avons présenté aussi la problématique d'ordonnancement flow shop hybride. Nous avons aussi présenté quelques brèves descriptions de méthodes de résolutions telles que les méthodes générales d'optimisation combinatoire et la programmation dynamique, la programmation linéaire, heuristiques et méta heuristiques. Dans le chapitre suivant, nous présenterons les méta-heuristiques.



CHAPITRE 2

Les Méta heuristiques

Chapitre 2

Les Méta –heuristiques

Sommaire

1	Introduction.....	24
2	Méta- heuristiques : Etat de l'art.....	24
2.1	Classification des méta heuristiques.....	26
2.2	Approche constructive.....	27
2.3	Approche de recherche locale.....	27
2.3.1	Recuit simulé.....	28
2.3.2	Méthodes d'acceptation à seuil ou méthode de descente.....	28
2.3.3	Méthode Tabou.....	29
2.4	Approche évolutive.....	30
2.4.1	Algorithme génétique.....	30
2.4.2	Algorithme de la fourmi	30
2.4.3	Méthode de recherche distribuée.....	31
2.4.4	La méthode essaim particulière PSO.....	32
2.5	Quelques propriétés associées aux méthodes approchées.....	32
2.6	Autres méthodes de résolution.....	32
3	Quelle méta-heuristique utiliser?.....	33
4	Analyse.....	34
5	Conclusion.....	35

1 Introduction

Dans ce mémoire, nous nous intéressons à la résolution du problème d'ordonnancement de type flow shop hybride (FSH). Il s'agit d'une extension du flow-shop classique, il constitue un modèle théorique pour résoudre des problèmes issus directement du monde industrie (pour plus de détails voir P. Lopez et F. Roubellat (2001).

Les différents problèmes d'ordonnancement de type FSH sont classés parmi les problèmes NP- difficiles. Par conséquent et comme beaucoup de problèmes dits difficiles, les méthodes mathématiques restent limitées. Le recours aux méthodes approchées est plus fréquent pour la résolution de ces problèmes, notamment les méta heuristiques.

Elles sont souvent inspirées de processus biologiques ou physiques naturels qui doivent être adaptés selon le problème à résoudre. L'avantage de ce type de méthodes est de construire un grand nombre de solutions possibles en un temps raisonnable, et une réutilisation des informations collectées durant la recherche via l'évaluation des solutions générées.

Une méta heuristique est une stratégie principale qui guide d'autres heuristiques. Leurs principes communs et essentiels sont exprimés par les concepts classiques d'intensification et de diversification, d'une part, et par les notions de stratégies de recherche et d'adaptation et/ou d'auto- adaptation des mécanismes de recherche appliqués. Elles utilisent des processus de génération itératifs qui guident une heuristique sous-jacente en nuancant l'intensification et la diversification de l'espace de recherche. De plus, les méta heuristiques induisent un aspect de multiplicité pour certains cas (comme pour le cas des algorithmes génétiques), par l'utilisation dans la recherche d'une population de solutions.

Dans ce chapitre, nous présentons les différentes méta heuristiques pour la résolution des problèmes d'ordonnancement flow shop hybride.

2 Méta- heuristiques : Etat de l'art

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. L'optimisation combinatoire est le domaine des mathématiques discrètes qui traite la résolution du problème suivant :

Soit X un ensemble de solutions admissibles, soit f une fonction permettant d'évaluer chaque solution admissible. Il s'agit de déterminer une solution $S^* \in X$ qui **minimise** f tel que :

$$f(S^*) = \min f(S)$$

Le développement de la théorie de complexité a permis de classer ces problèmes selon leurs difficultés (Rinnooy Kan, 1976; Graham et al., 1979). Dans la plupart des cas, ils ont été

démontrés NP- difficiles. Par conséquent, il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les problèmes d'ordonnancement sans que le temps de résolution n'augmente de façon exponentielle avec la taille de celui-ci. Dans la littérature, nous retrouvons plusieurs approches de résolution des problèmes d'ordonnancement qui font appel aux techniques de l'optimisation combinatoire. Si une méthode peut fournir une solution optimale, il s'agit alors de méthode optimale ou exacte. Dans ce cas, pour trouver un optimum, une recherche exhaustive explicite ou implicite est donc nécessaire. De manière générale, ce type d'approches permet de résoudre les problèmes d'ordonnancement de petite taille, mais il reste difficile de fournir une résolution exacte en un temps raisonnable lorsque les problèmes sont de taille importante (ou industrielle). D'où la nécessité d'adopter des méthodes de résolution approchée proposant des solutions acceptables (c'est-à-dire non nécessairement optimales) en un temps de calcul réduit. Nous allons passer en revue les méthodes les plus connues en les répertoriant en méthodes approchées, [6,8].

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. Ainsi, les performances de ce type d'approches dépendent de la qualité de la solution obtenue et du temps de calcul nécessaire pour sa réalisation. Plusieurs méthodes approchées ont été développées pour la résolution des problèmes d'ordonnancement, [4,7]. Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle heuristique. Leur principe de base repose généralement sur l'utilisation de règles empiriques simples qui exploitent les propriétés structurelles d'une solution et tente de retrouver une solution admissible par des critères de décision déduits de la connaissance du problème. Cependant, lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de méta heuristique. Les méta heuristiques sont généralement des algorithmes stochastiques, qui progressent vers un optimum par échantillonnage d'une fonction objectif.

Ainsi, elles sont le plus souvent, guidées par l'exploration aléatoire de l'espace de recherche.

La **méta-heuristique**, elle, se place à un niveau plus général encore, et intervient dans toutes les situations où l'ingénieur ne connaît pas d'heuristique efficace pour résoudre un problème donné, ou lorsqu'il estime qu'il ne dispose pas du temps nécessaire pour en déterminer une.

En 1996, I.H. Osman et G. Laporte définissaient la méta-heuristique comme « un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage

sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales ».

En 2006, le réseau Metaheuristics (metaheuristics.org) définit les méta-heuristiques comme « un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliqués à une grande variété de problèmes. On peut voir la méta-heuristique comme une « boîte à outils » algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier ».

Elle a donc pour objectif de pouvoir être programmée et testée rapidement sur un problème.

Comme l'heuristique, la méta-heuristique n'offre généralement pas de garantie d'optimalité, bien qu'on ait pu démontrer la convergence de certaines d'entre elles. Non déterministe, elle incorpore souvent un principe stochastique pour surmonter l'explosion combinatoire. Elle fait parfois usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche. Dans ce qui suit, nous présentons quelques unes de ces méthodes approchées.

2.1 Classification des méta heuristiques

Les méta heuristiques ne sont pas, a priori, spécifiques à la résolution de tel ou tel problème, leur classifications reste assez arbitraire , on peu distinguer :

a - Les approches **trajectoires**

Ces algorithmes partent d'une solution initiale (obtenue d'une façon exacte, ou par tirage aléatoire) et s'en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l'espace des solutions. :

- la méthode de descente ou recherche locale
- Le recuit simulé
- La méthode Tabou
- La recherche par voisinage variable

Le terme de recherche locale est de plus en plus utilisé pour qualifier ces méthodes

b - Les approches **population** (ou évolutionnaires)

Elles consistent à travailler avec un ensemble de solutions simultanément, que l'on fait évoluer graduellement. L'utilisation de plusieurs solutions simultanément permet naturellement d'améliorer l'exploration de l'espace de configurations. Dans cette deuxième catégorie, on a :

- Les algorithmes génétiques
- Les algorithmes par colonie de fourmi
- L'optimisation par essaim particulaires
- Les algorithmes à estimation de distribution

Il faut distinguer les méta heuristiques qui ont la faculté de mémoriser des informations à mesure que leur recherche avance, de celle qui fonctionnent sans mémoire, en aveugle, et qui peuvent revenir sur des solutions qu'elles ont déjà examinées. On distingue la mémoire à court terme (celles des derniers mouvements effectués), et la mémoire à long terme (qui concerne des paramètres plus généraux).

2.2 Approche constructive

Il s'agit de construire de manière itérative une solution en complétant étape par étape un ordonnancement partiel. De nombreux algorithmes constructifs existent, à savoir :

- les algorithmes gloutons. Les décisions faites durant le processus de recherche ne sont jamais remises en cause. Les algorithmes de liste sont des exemples d'algorithmes gloutons. Ils utilisent des règles de priorité (exemples : priorité au travail ayant le plus court temps opératoire, priorité au travail ayant la plus proche date au plus tard,...) pour ordonner la liste des travaux. Une autre manière de construire des algorithmes gloutons consiste à choisir des décisions qui correspondent localement à des optima partiels locaux.

2.3 Approche de recherche locale

Les approches par voisinage, appelées également les méthodes de recherche locale, sont des algorithmes itératifs initialisés par une solution réalisable (obtenue soit de manière aléatoire, soit à l'aide d'une autre heuristique, par exemple une heuristique constructive). Ils cherchent à améliorer, à chaque itération, la solution courante par des modifications locales. La recherche consiste donc à se déplacer successivement dans l'ensemble des solutions en passant, à chaque étape, d'une solution à une autre voisine, jusqu'à la vérification d'une condition d'arrêt, [4,7].

Les modalités des déplacements conduisent à de nombreuses méthodes telles que :

2.3.1 Recuit simulé

La méthode du recuit simulé (SA pour « Simulated Annealing ») tire son origine de la physique statistique. Elle a été introduite par analogie avec des phénomènes de la thermodynamique (Kirkpatrick et al., 1983). Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux en s'appuyant sur les travaux de Metropolis et al. (1953). Le recuit est une opération consistant à laisser refroidir lentement un métal d'une manière contrôlée pour améliorer ses qualités et obtenir une structure régulière, comme dans les cristaux et l'acier. L'idée physique est qu'un refroidissement trop brutal peut bloquer le métal dans un état peu favorable présentant ainsi des défauts : c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Un refroidissement lent permettra aux molécules de s'agencer au mieux dans une configuration stable, équivalent à un optimum global. Ainsi, la température est un paramètre indispensable pour modifier l'état du matériau. Le recuit est donc une stratégie de contrôle de la température pour avoir un système physique dans un état de basse énergie. De manière analogue, la méthode de recuit simulé est une procédure itérative qui consiste à engendrer, à chaque itération, une nouvelle solution dans le voisinage de la solution courante. Si la nouvelle solution est meilleure, elle est acceptée, sinon elle est acceptée selon le critère de Metropolis, qui utilise une probabilité d'acceptation dépendante de la variable de température du matériau.

2.3.2 Méthodes d'acceptation à seuil ou méthode de descente

La méthode de descente (« Hill Climbing » en anglais) qui consiste à partir d'une solution courante, de choisir à chaque itération, une nouvelle solution dans le voisinage. Cette solution est acceptée si elle est meilleure que la solution courante et devient ainsi la nouvelle solution courante. Ce processus recommence jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage. Dans le cas contraire, si aucune solution parmi les solutions voisines n'est strictement meilleure que la solution courante, alors celle-ci est un optimum local et on arrête l'algorithme.

Le principe de la méthode de descente (dite aussi *basic local search*) consiste à partir d'une solution s et à choisir une solution s' dans un voisinage de s , telle que s' améliore la recherche (généralement telle que $f(s') < f(s)$).

On peut décider soit d'examiner toutes les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

La méthode de recherche locale la plus élémentaire est la *méthode de descente*. On peut la schématiser comme suit :

Procédure de descente simple (solution initiale s)

Répéter :

Choisir s' dans $N(s)$

Si $f(s') < f(s)$ alors $s \leftarrow s'$

Jusqu'à ce que $f(s') \geq f(s)$, $s' \notin S$

Fin

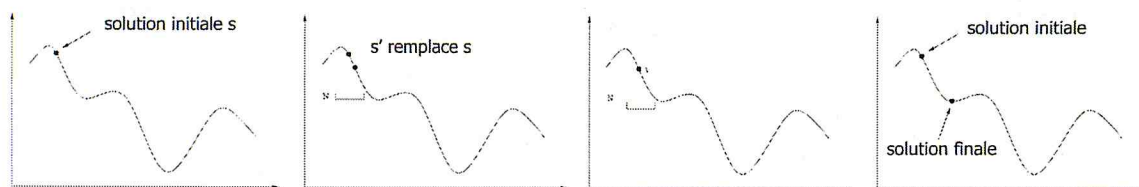


Fig. 7 – Procédure de descente

On peut varier cette méthode en choisissant à chaque fois la solution s' dans $N(s)$ qui améliore le plus la valeur de f . C'est la *méthode de plus grande descente*.

2.3.3 Méthode Tabou

La méthode de recherche tabou (TS pour « Tabu Search ») est une méthode itérative d'optimisation combinatoire. C'est une procédure de recherche locale qui utilise un mécanisme de mémoire. Elle a été présentée par Glover(1987). L'idée principale est de pouvoir mémoriser l'historique de l'exploration de solutions afin de conserver les informations sur les solutions déjà visitées. Comme la méthode de la descente, nous partons d'une solution initiale, puis à chaque itération, la recherche tabou va examiner les voisinages. La meilleure solution devient la nouvelle solution courante. Le processus recommence jusqu'à ce qu'une condition d'arrêt soit satisfaite. Pour empêcher le blocage de la recherche sur un optimum local (en interdisant de revenir sur des solutions précédemment visitées de l'espace d'état), la recherche tabou est dotée d'une mémoire de mouvements et d'un mécanisme d'échappement (appelé aussi mécanisme d'aspiration). La mémoire de mouvements constitue la liste tabou de mouvements interdits qui va servir à empêcher l'accès aux dernières solutions visitées.

2.4 Approche évolutive

Contrairement aux méthodes par construction et par voisinage qui font intervenir une solution unique, les méthodes évolutives, dites aussi à base de population, considèrent un ensemble de solutions, appelé population. Ils font évoluer une population de solutions à chaque étape du processus de recherche permettant d'identifier et d'explorer les caractéristiques que les solutions ont en commun. Parmi les méthodes de cette catégorie, on peut citer les algorithmes évolutionnaires, les algorithmes de colonies de fourmis, les méthodes par essaims particulaires et les algorithmes à estimation de distribution. 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.

2.4.1 Algorithme génétique

Les algorithmes évolutionnaires (« Evolutionary Computation » en anglais) sont des techniques d'optimisation itératives et stochastiques inspirées de la théorie de l'évolution. Cette théorie a été élaborée par Charles Darwin et fondée sur le modèle biologique. développé initialement par Holland (1975) 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.

2.4.2 Algorithme de la fourmi

Les algorithmes de colonies de fourmis ont été introduits par Colnari et al. (1991). L'origine de cette approche repose sur une métaphore où le comportement des fourmis réelles lors de la quête de recherche de nourriture est reproduit. En effet, bien qu'ayant individuellement des capacités cognitives très limitées, les fourmis sont capables de sélectionner collectivement le plus court chemin pour aller de leur nid à une source de nourriture. Pour cela, elles utilisent l'environnement comme support de communication grâce au dépôt et au suivi de pistes de *phéromone* (une substance chimique volatile). En imitant ce comportement, une famille d'algorithmes d'optimisation a été proposée (Dorigo et Stützle, 2004). L'idée de base consiste à travailler sur une population de solutions, chacune correspondant à une fourmi en utilisant une structure de donnée commune (partagée par toutes les fourmis) qui contient des informations sur les quantités de phéromones accumulées dans l'espace des solutions pour guider leurs recherches.

Principe de l'algorithme

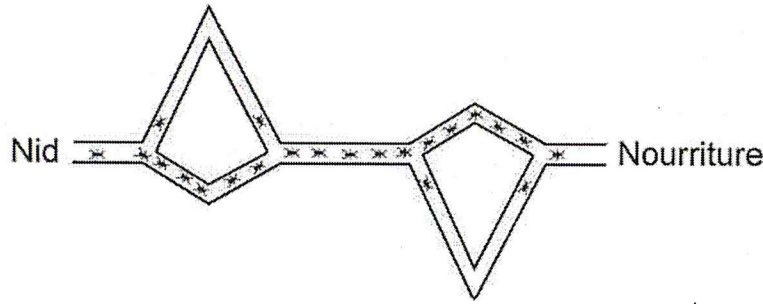


Fig. 8 - Recueil des ressources par des fourmis

Soit A un ensemble de k fourmis :

Répéter :

Pour $i=1$ à k faire

Construire Trajet(i)

Fin Pour

Mettre à Jour Pheromones ()

Jusqu'à ce que le critère de terminaison soit satisfait.

2.4.3 Méthode de recherche distribuée

Algorithmes à estimation de distribution (EDA pour « Estimation of distribution algorithm » ou encore « Probabilistic Model Building Genetic Algorithms »). Il s'agit d'approches basées sur des modèles probabilistes et inspirées des algorithmes génétiques. Ils travaillent sur une population de solutions qu'ils cherchent à améliorer en générant d'autres populations. La construction des nouvelles populations se fait grâce à l'estimation d'une distribution de probabilité (issue de la fonction objectif), associée à chaque individu de la population de l'itération précédente pour décrire la qualité des solutions. La distribution doit faire apparaître les zones prometteuses de manière à y concentrer la recherche. La méthode commence avec un échantillonnage aléatoire des solutions. Une partie de ces solutions va permettre de faire une première estimation de la distribution de la fonction de coût. A partir de cette estimation, un nouvel échantillon de solutions est créé de manière probabiliste de façon à concentrer l'échantillon dans les zones prometteuses identifiées par l'estimation de distribution. Ce processus est réitéré afin d'affiner l'estimation de distribution et de trouver de meilleures solutions.

2.4.4 La méthode par essais particuliers (PSO)

La méthode par essais particuliers (PSO pour « Particle Swarm Optimization ») est une technique d'optimisation stochastique développée par Eberhart et Kennedy (1995). elle s'inspire des déplacements collectifs observés chez certains animaux sociaux tels que les poissons et les oiseaux. En effet, ces groupes d'animaux montre 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'approche 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 un vecteur de changement de position (appelé vitesse). 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.

2.5 Quelques propriétés associées aux méthodes approchées

La plupart des méthodes approchées ne sont pas déterministes, c'est-à-dire que deux exécutions successives utilisant les mêmes données peuvent retourner deux résultats différents. Ils reposent généralement sur un ensemble de concepts à savoir la mémoire, qui est un support d'apprentissage permettant de sauvegarder les informations recueillies par l'algorithme, à mesure que la recherche avance afin d'en faire usage ultérieurement ; l'intensification, qui permet de rechercher des solutions a priori de plus grande qualité en exploitant les informations rassemblées par le système durant la recherche, la diversification, qui permet d'explorer de nouvelles régions de l'espace de recherche non encore visitées. Les associations multiples de ces concepts peuvent conduire à une grande variété d'autres méthodes.

2.6 Autres méthodes de résolution

Il existe bien d'autres méthodes pour résoudre les problèmes d'ordonnancement. Parmi celles ci, on retrouve les approches hybrides qui combinent au moins deux procédures de recherche différentes, y compris les méthodes exactes et/ou approchées (sus-décrites). L'hybridation est une technique qui essaie de tirer profit des points forts de chaque approche de résolution utilisée pour améliorer le comportement global de la méthode hybride. Elle semble constituer une technique intéressante qui permet d'élargir le spectre d'application d'une méthode de recherche et/ou d'augmenter ses performances. Cependant, le choix des approches à combiner est primordial pour obtenir une bonne coopération entre les constituants de

l'approche hybride. Ainsi il est important de savoir dégager les points forts et également les points faibles de ces dernières. Nous citons à titre d'exemple le cas des algorithmes mémétiques, connus aussi sous le nom d'algorithmes génétiques hybrides, qui combinent une recherche locale (qui assure l'intensification de la recherche) et un algorithme génétique (qui renforce la diversification). Plusieurs types d'hybridations sont possibles.

Les algorithmes génétiques n'ont pas fourni de résultats compétitifs, mais les méthodes hybrides comptent parmi les techniques les plus prometteuses malgré leur coût. Les résultats montrent également l'importance du choix du voisinage.

3 Quelle méta-heuristique utiliser ?

Le premier problème pratique qui se pose à un utilisateur confronté à une application concrète est d'effectuer un choix parmi les différentes méta-heuristiques disponibles. Ce choix est d'autant plus difficile qu'il n'existe pas de comparaison générale et fiable des différentes méta-heuristiques. Cependant, il est possible de caractériser les méta-heuristiques selon quelques critères généraux, ce qui pourrait faciliter ce choix.

Le tableau de synthèse ci-dessous met en relation cinq critères avec cinq méta-heuristiques parmi les plus représentatives. Les indications sont présentées à titre purement indicatif et correspondent aux expériences et aux résultats publiés dans la revue intelligence artificielle. Il doit être clair que le choix d'une méta-heuristique appropriée ne constitue qu'une condition nécessaire. La qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée, [5].

	AI	RS	TABOU	AG	AH
Simplicité	0	-	-	-	--
Facilité d'adaptation	0	0	-	-	-
connaissance	0	0	+	+	+
Qualité	0	+	++	+	+++++
Rapidité	0	-	-	--	--

Tableau. 2 - Comparaison générale des principales méta-heuristiques

Dans ce tableau, les cinq méta-heuristiques comparées sont les suivantes :

- AI : amélioration itérative (descente) avec relance,
- RS : recuit simulé,
- tabou : méthode tabou,
- AG : algorithme génétique adapté,
- AH : algorithme hybride.

Les critères de comparaisons retenus sont les suivants :

- simplicité de la méta-heuristique, *i.e.* simplicité de la méthode elle-même,
- facilité d'adaptation au problème,
- possibilité d'intégrer des connaissances spécifiques du problème,
- qualité des meilleures solutions trouvées,
- rapidité, *i.e.*, le temps de calcul nécessaire pour trouver une telle solution (sur une machine séquentielle).

La méthode d'amélioration itérative est utilisée comme point de référence pour l'ensemble des méthodes : les signes -, 0, + indiquent des performances respectivement inférieures, égales, supérieures à celles obtenues par l'amélioration itérative.

Le critère de *qualité* utilisé dans le tableau correspond à la meilleure qualité qu'il est possible d'obtenir par une exécution prolongée. Le critère de *rapidité* représente le temps de calcul typiquement nécessaire pour obtenir une telle solution.

4 Analyse

Les méta-heuristiques présentent de nombreux avantages mais aussi un certain nombre de limitations. Nous résumons ci-dessous les différentes caractéristiques de ces méthodes :

- généralité et application possible à une large classe de problèmes,
- efficacité pour de nombreux problèmes,
- possibilité de compromis entre qualité des solutions et temps de calcul,
- optimum non garanti,
- nécessité d'adaptation de la méthode au problème traité,
- nécessité de réglage des paramètres,
- difficulté de prévoir la performance.

Tout d'abord, rappelons qu'il s'agit d'heuristiques dont l'objectif n'est pas le même que celui des méthodes exactes. En particulier, ces méthodes ne garantissent pas de trouver une solution optimale ni de prouver l'optimalité de la solution.

Dans ces conditions, méthodes exactes et méta-heuristiques sont plutôt complémentaires que véritablement concurrentes.

5 Conclusion

Dans ce chapitre nous avons introduit les différentes méta- heuristiques.

Dans le cadre de la problématique d'ordonnancement type flow shop hybride que nous traitons dans ce mémoire, l'approche de résolution proposée est de nature approchée.

Le prochain chapitre présente l'adaptation d'une méta-heuristique qui est l'algorithme génétique proposé pour la résolution du problème flow shop hybride.

Chapitre 2

Les Méta –heuristiques

Sommaire

1	Introduction.....	24
2	Méta- heuristiques : Etat de l'art.....	24
2.1	Classification des méta heuristiques.....	26
2.2	Approche constructive.....	27
2.3	Approche de recherche locale.....	27
2.3.1	Recuit simulé.....	28
2.3.2	Méthodes d'acceptation à seuil ou méthode de descente.....	28
2.3.3	Méthode Tabou.....	29
2.4	Approche évolutive.....	30
2.4.1	Algorithme génétique.....	30
2.4.2	Algorithme de la fourmi	30
2.4.3	Méthode de recherche distribuée.....	31
2.4.4	La méthode essaim particulière PSO.....	32
2.5	Quelques propriétés associées aux méthodes approchées.....	32
2.6	Autres méthodes de résolution.....	32
3	Quelle méta-heuristique utiliser?.....	33
4	Analyse.....	34
5	Conclusion.....	35

1 Introduction

Dans ce mémoire, nous nous intéressons à la résolution du problème d'ordonnancement de type flow shop hybride (FSH). Il s'agit d'une extension du flow-shop classique, il constitue un modèle théorique pour résoudre des problèmes issus directement du monde industrie (pour plus de détails voir P. Lopez et F. Roubellat (2001).

Les différents problèmes d'ordonnancement de type FSH sont classés parmi les problèmes NP- difficiles. Par conséquent et comme beaucoup de problèmes dits difficiles, les méthodes mathématiques restent limitées. Le recours aux méthodes approchées est plus fréquent pour la résolution de ces problèmes, notamment les méta heuristiques.

Elles sont souvent inspirées de processus biologiques ou physiques naturels qui doivent être adaptés selon le problème à résoudre. L'avantage de ce type de méthodes est de construire un grand nombre de solutions possibles en un temps raisonnable, et une réutilisation des informations collectées durant la recherche via l'évaluation des solutions générées.

Une méta heuristique est une stratégie principale qui guide d'autres heuristiques. Leurs principes communs et essentiels sont exprimés par les concepts classiques d'intensification et de diversification, d'une part, et par les notions de stratégies de recherche et d'adaptation et/ou d'auto- adaptation des mécanismes de recherche appliqués. Elles utilisent des processus de génération itératifs qui guident une heuristique sous-jacente en nuancant l'intensification et la diversification de l'espace de recherche. De plus, les méta heuristiques induisent un aspect de multiplicité pour certains cas (comme pour le cas des algorithmes génétiques), par l'utilisation dans la recherche d'une population de solutions.

Dans ce chapitre, nous présentons les différentes méta heuristiques pour la résolution des problèmes d'ordonnancement flow shop hybride.

2 Méta- heuristiques : Etat de l'art

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. L'optimisation combinatoire est le domaine des mathématiques discrètes qui traite la résolution du problème suivant :

Soit X un ensemble de solutions admissibles, soit f une fonction permettant d'évaluer chaque solution admissible. Il s'agit de déterminer une solution $S^* \in X$ qui **minimise** f tel que :

$$f(S^*) = \min f(S)$$

Le développement de la théorie de complexité a permis de classer ces problèmes selon leurs difficultés (Rinnooy Kan, 1976; Graham et al., 1979). Dans la plupart des cas, ils ont été

démontrés NP- difficiles. Par conséquent, il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les problèmes d'ordonnancement sans que le temps de résolution n'augmente de façon exponentielle avec la taille de celui-ci. Dans la littérature, nous retrouvons plusieurs approches de résolution des problèmes d'ordonnancement qui font appel aux techniques de l'optimisation combinatoire. Si une méthode peut fournir une solution optimale, il s'agit alors de méthode optimale ou exacte. Dans ce cas, pour trouver un optimum, une recherche exhaustive explicite ou implicite est donc nécessaire. De manière générale, ce type d'approches permet de résoudre les problèmes d'ordonnancement de petite taille, mais il reste difficile de fournir une résolution exacte en un temps raisonnable lorsque les problèmes sont de taille importante (ou industrielle). D'où la nécessité d'adopter des méthodes de résolution approchée proposant des solutions acceptables (c'est-à-dire non nécessairement optimales) en un temps de calcul réduit. Nous allons passer en revue les méthodes les plus connues en les répertoriant en méthodes approchées, [6,8].

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. Ainsi, les performances de ce type d'approches dépendent de la qualité de la solution obtenue et du temps de calcul nécessaire pour sa réalisation. Plusieurs méthodes approchées ont été développées pour la résolution des problèmes d'ordonnancement, [4,7]. Lorsque ces méthodes sont conçues de manière simple, rapide et ciblée sur un problème particulier, on les appelle heuristique. Leur principe de base repose généralement sur l'utilisation de règles empiriques simples qui exploitent les propriétés structurelles d'une solution et tente de retrouver une solution admissible par des critères de décision déduits de la connaissance du problème. Cependant, lorsque celles-ci sont générales, adaptables et applicables à plusieurs catégories de problèmes d'optimisation combinatoire, elles portent le nom de méta heuristique. Les méta heuristiques sont généralement des algorithmes stochastiques, qui progressent vers un optimum par échantillonnage d'une fonction objectif.

Ainsi, elles sont le plus souvent, guidées par l'exploration aléatoire de l'espace de recherche.

La **méta-heuristique**, elle, se place à un niveau plus général encore, et intervient dans toutes les situations où l'ingénieur ne connaît pas d'heuristique efficace pour résoudre un problème donné, ou lorsqu'il estime qu'il ne dispose pas du temps nécessaire pour en déterminer une.

En 1996, I.H. Osman et G. Laporte définissaient la méta-heuristique comme « un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage

sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales ».

En 2006, le réseau Metaheuristics (metaheuristics.org) définit les méta-heuristiques comme « un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliqués à une grande variété de problèmes. On peut voir la méta-heuristique comme une « boîte à outils » algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier ».

Elle a donc pour objectif de pouvoir être programmée et testée rapidement sur un problème. Comme l'heuristique, la méta-heuristique n'offre généralement pas de garantie d'optimalité, bien qu'on ait pu démontrer la convergence de certaines d'entre elles. Non déterministe, elle incorpore souvent un principe stochastique pour surmonter l'explosion combinatoire. Elle fait parfois usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche. Dans ce qui suit, nous présentons quelques unes de ces méthodes approchées.

2.1 Classification des méta heuristiques

Les méta heuristiques ne sont pas, a priori, spécifiques à la résolution de tel ou tel problème, leur classifications reste assez arbitraire , on peu distinguer :

a - Les approches **trajectoires**

Ces algorithmes partent d'une solution initiale (obtenue d'une façon exacte, ou par tirage aléatoire) et s'en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l'espace des solutions. :

- la méthode de descente ou recherche locale
- Le recuit simulé
- La méthode Tabou
- La recherche par voisinage variable

Le terme de recherche locale est de plus en plus utilisé pour qualifier ces méthodes

b - Les approches **population** (ou évolutionnaires)

Elles consistent à travailler avec un ensemble de solutions simultanément, que l'on fait évoluer graduellement. L'utilisation de plusieurs solutions simultanément permet naturellement d'améliorer l'exploration de l'espace de configurations. Dans cette deuxième catégorie, on a :

- Les algorithmes génétiques
- Les algorithmes par colonie de fourmi
- L'optimisation par essaim particulaires
- Les algorithmes à estimation de distribution

Il faut distinguer les méta heuristiques qui ont la faculté de mémoriser des informations à mesure que leur recherche avance, de celle qui fonctionnent sans mémoire, en aveugle, et qui peuvent revenir sur des solutions qu'elles ont déjà examinées. On distingue la mémoire à court terme (celles des derniers mouvements effectués), et la mémoire à long terme (qui concerne des paramètres plus généraux).

2.2 Approche constructive

Il s'agit de construire de manière itérative une solution en complétant étape par étape un ordonnancement partiel. De nombreux algorithmes constructifs existent, à savoir :

- les algorithmes gloutons. Les décisions faites durant le processus de recherche ne sont jamais remises en cause. Les algorithmes de liste sont des exemples d'algorithmes gloutons. Ils utilisent des règles de priorité (exemples : priorité au travail ayant le plus court temps opératoire, priorité au travail ayant la plus proche date au plus tard,...) pour ordonnancer la liste des travaux. Une autre manière de construire des algorithmes gloutons consiste à choisir des décisions qui correspondent localement à des optima partiels locaux.

2.3 Approche de recherche locale

Les approches par voisinage, appelées également les méthodes de recherche locale, sont des algorithmes itératifs initialisés par une solution réalisable (obtenue soit de manière aléatoire, soit à l'aide d'une autre heuristique, par exemple une heuristique constructive). Ils cherchent à améliorer, à chaque itération, la solution courante par des modifications locales. La recherche consiste donc à se déplacer successivement dans l'ensemble des solutions en passant, à chaque étape, d'une solution à une autre voisine, jusqu'à la vérification d'une condition d'arrêt, [4,7].

Les modalités des déplacements conduisent à de nombreuses méthodes telles que :

2.3.1 Recuit simulé

La méthode du recuit simulé (SA pour « Simulated Annealing ») tire son origine de la physique statistique. Elle a été introduite par analogie avec des phénomènes de la thermodynamique (Kirkpatrick et al., 1983). Son principe de fonctionnement repose sur une imitation du phénomène de recuit en science des matériaux en s'appuyant sur les travaux de Metropolis et al. (1953). Le recuit est une opération consistant à laisser refroidir lentement un métal d'une manière contrôlée pour améliorer ses qualités et obtenir une structure régulière, comme dans les cristaux et l'acier. L'idée physique est qu'un refroidissement trop brutal peut bloquer le métal dans un état peu favorable présentant ainsi des défauts : c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Un refroidissement lent permettra aux molécules de s'agencer au mieux dans une configuration stable, équivalent à un optimum global. Ainsi, la température est un paramètre indispensable pour modifier l'état du matériau. Le recuit est donc une stratégie de contrôle de la température pour avoir un système physique dans un état de basse énergie. De manière analogue, la méthode de recuit simulé est une procédure itérative qui consiste à engendrer, à chaque itération, une nouvelle solution dans le voisinage de la solution courante. Si la nouvelle solution est meilleure, elle est acceptée, sinon elle est acceptée selon le critère de Metropolis, qui utilise une probabilité d'acceptation dépendante de la variable de température du matériau.

2.3.2 Méthodes d'acceptation à seuil ou méthode de descente

La méthode de descente (« Hill Climbing » en anglais) qui consiste à partir d'une solution courante, de choisir à chaque itération, une nouvelle solution dans le voisinage. Cette solution est acceptée si elle est meilleure que la solution courante et devient ainsi la nouvelle solution courante. Ce processus recommence jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage. Dans le cas contraire, si aucune solution parmi les solutions voisines n'est strictement meilleure que la solution courante, alors celle-ci est un optimum local et on arrête l'algorithme.

Le principe de la méthode de descente (dite aussi *basic local search*) consiste à partir d'une solution s et à choisir une solution s' dans un voisinage de s , telle que s' améliore la recherche (généralement telle que $f(s') < f(s)$).

On peut décider soit d'examiner *toutes* les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

La méthode de recherche locale la plus élémentaire est la *méthode de descente*. On peut la schématiser comme suit :

Procédure de descente simple (solution initiale s)

Répéter :

Choisir s' dans $N(s)$

Si $f(s') < f(s)$ alors $s \leftarrow s'$

Jusqu'à ce que $f(s') \geq f(s)$, $s' \notin S$

Fin

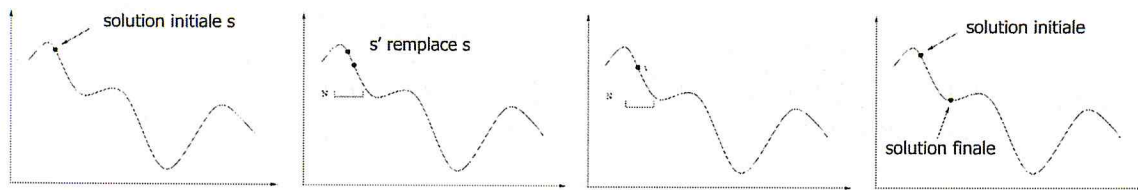


Fig. 7 – Procédure de descente

On peut varier cette méthode en choisissant à chaque fois la solution s' dans $N(s)$ qui améliore le plus la valeur de f . C'est la *méthode de plus grande descente*.

2.3.3 Méthode Tabou

La méthode de recherche tabou (TS pour « Tabu Search ») est une méthode itérative d'optimisation combinatoire. C'est une procédure de recherche locale qui utilise un mécanisme de mémoire. Elle a été présentée par Glover(1987). L'idée principale est de pouvoir mémoriser l'historique de l'exploration de solutions afin de conserver les informations sur les solutions déjà visitées. Comme la méthode de la descente, nous partons d'une solution initiale, puis à chaque itération, la recherche tabou va examiner les voisinages. La meilleure solution devient la nouvelle solution courante. Le processus recommence jusqu'à ce qu'une condition d'arrêt soit satisfaite. Pour empêcher le blocage de la recherche sur un optimum local (en interdisant de revenir sur des solutions précédemment visitées de l'espace d'état), la recherche tabou est dotée d'une mémoire de mouvements et d'un mécanisme d'échappement (appelé aussi mécanisme d'aspiration). La mémoire de mouvements constitue la liste tabou de mouvements interdits qui va servir à empêcher l'accès aux dernières solutions visitées.

2.4 Approche évolutive

Contrairement aux méthodes par construction et par voisinage qui font intervenir une solution unique, les méthodes évolutives, dites aussi à base de population, considèrent un ensemble de solutions, appelé population. Ils font évoluer une population de solutions à chaque étape du processus de recherche permettant d'identifier et d'explorer les caractéristiques que les solutions ont en commun. Parmi les méthodes de cette catégorie, on peut citer les algorithmes évolutionnaires, les algorithmes de colonies de fourmis, les méthodes par essaims particulaires et les algorithmes à estimation de distribution. 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.

2.4.1 Algorithme génétique

Les algorithmes évolutionnaires (« Evolutionary Computation » en anglais) sont des techniques d'optimisation itératives et stochastiques inspirées de la théorie de l'évolution. Cette théorie a été élaborée par Charles Darwin et fondée sur le modèle biologique. développé initialement par Holland (1975) 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.

2.4.2 Algorithme de la fourmi

Les algorithmes de colonies de fourmis ont été introduits par Colomni et al. (1991). L'origine de cette approche repose sur une métaphore où le comportement des fourmis réelles lors de la quête de recherche de nourriture est reproduit. En effet, bien qu'ayant individuellement des capacités cognitives très limitées, les fourmis sont capables de sélectionner collectivement le plus court chemin pour aller de leur nid à une source de nourriture. Pour cela, elles utilisent l'environnement comme support de communication grâce au dépôt et au suivi de pistes de *phéromone* (une substance chimique volatile). En imitant ce comportement, une famille d'algorithmes d'optimisation a été proposée (Dorigo et Stützle, 2004). L'idée de base consiste à travailler sur une population de solutions, chacune correspondant à une fourmi en utilisant une structure de donnée commune (partagée par toutes les fourmis) qui contient des informations sur les quantités de phéromones accumulées dans l'espace des solutions pour guider leurs recherches.

Principe de l'algorithme

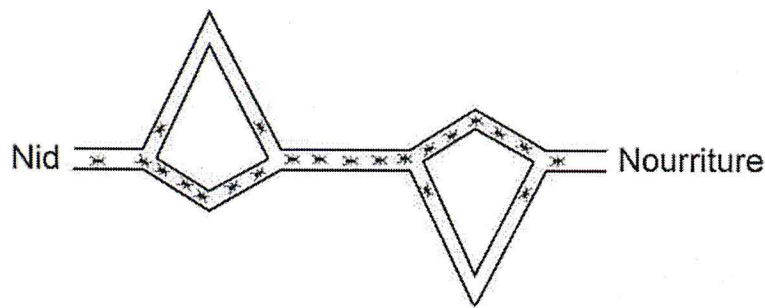


Fig. 8 - Recueil des ressources par des fourmis

Soit A un ensemble de k fourmis :

Répéter :

Pour $i=1$ à k faire

Construire Trajet(i)

Fin Pour

Mettre à Jour Pheromones ()

Jusqu'à ce que le critère de terminaison soit satisfait.

2.4.3 Méthode de recherche distribuée

Algorithmes à estimation de distribution (EDA pour « Estimation of distribution algorithm » ou encore « Probabilistic Model Building Genetic Algorithms »). Il s'agit d'approches basées sur des modèles probabilistes et inspirées des algorithmes génétiques. Ils travaillent sur une population de solutions qu'ils cherchent à améliorer en générant d'autres populations. La construction des nouvelles populations se fait grâce à l'estimation d'une distribution de probabilité (issue de la fonction objectif), associée à chaque individu de la population de l'itération précédente pour décrire la qualité des solutions. La distribution doit faire apparaître les zones prometteuses de manière à y concentrer la recherche. La méthode commence avec un échantillonnage aléatoire des solutions. Une partie de ces solutions va permettre de faire une première estimation de la distribution de la fonction de coût. A partir de cette estimation, un nouvel échantillon de solutions est créé de manière probabiliste de façon à concentrer l'échantillon dans les zones prometteuses identifiées par l'estimation de distribution. Ce processus est réitéré afin d'affiner l'estimation de distribution et de trouver de meilleures solutions.

2.4.4 La méthode par essais particuliers (PSO)

La méthode par essais particuliers (PSO pour « Particle Swarm Optimization ») est une technique d'optimisation stochastique développée par Eberhart et Kennedy (1995). elle s'inspire des déplacements collectifs observés chez certains animaux sociaux tels que les poissons et les oiseaux. En effet, ces groupes d'animaux montre 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'approche 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 un vecteur de changement de position (appelé vitesse). 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.

2.5 Quelques propriétés associées aux méthodes approchées

La plupart des méthodes approchées ne sont pas déterministes, c'est-à-dire que deux exécutions successives utilisant les mêmes données peuvent retourner deux résultats différents. Ils reposent généralement sur un ensemble de concepts à savoir la mémoire, qui est un support d'apprentissage permettant de sauvegarder les informations recueillies par l'algorithme, à mesure que la recherche avance afin d'en faire usage ultérieurement ; l'intensification, qui permet de rechercher des solutions a priori de plus grande qualité en exploitant les informations rassemblées par le système durant la recherche, la diversification, qui permet d'explorer de nouvelles régions de l'espace de recherche non encore visitées.

Les associations multiples de ces concepts peuvent conduire à une grande variété d'autres méthodes.

2.6 Autres méthodes de résolution

Il existe bien d'autres méthodes pour résoudre les problèmes d'ordonnancement. Parmi celles ci, on retrouve les approches hybrides qui combinent au moins deux procédures de recherche différentes, y compris les méthodes exactes et/ou approchées (sus-décrites). L'hybridation est une technique qui essaie de tirer profit des points forts de chaque approche de résolution utilisée pour améliorer le comportement global de la méthode hybride. Elle semble constituer une technique intéressante qui permet d'élargir le spectre d'application d'une méthode de recherche et/ou d'augmenter ses performances. Cependant, le choix des approches à combiner est primordial pour obtenir une bonne coopération entre les constituants de

l'approche hybride. Ainsi il est important de savoir dégager les points forts et également les points faibles de ces dernières. Nous citons à titre d'exemple le cas des algorithmes mémétiques, connus aussi sous le nom d'algorithmes génétiques hybrides, qui combinent une recherche locale (qui assure l'intensification de la recherche) et un algorithme génétique (qui renforce la diversification). Plusieurs types d'hybridations sont possibles.

Les algorithmes génétiques n'ont pas fourni de résultats compétitifs, mais les méthodes hybrides comptent parmi les techniques les plus prometteuses malgré leur coût. Les résultats montrent également l'importance du choix du voisinage.

3 Quelle méta-heuristique utiliser ?

Le premier problème pratique qui se pose à un utilisateur confronté à une application concrète est d'effectuer un choix parmi les différentes méta-heuristiques disponibles. Ce choix est d'autant plus difficile qu'il n'existe pas de comparaison générale et fiable des différentes méta-heuristiques. Cependant, il est possible de caractériser les méta-heuristiques selon quelques critères généraux, ce qui pourrait faciliter ce choix.

Le tableau de synthèse ci-dessous met en relation cinq critères avec cinq méta-heuristiques parmi les plus représentatives. Les indications sont présentées à titre purement indicatif et correspondent aux expériences et aux résultats publiés dans la revue intelligence artificielle. Il doit être clair que le choix d'une méta-heuristique appropriée ne constitue qu'une condition nécessaire. La qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée, [5].

	AI	RS	TABOU	AG	AH
Simplicité	0	-	-	-	--
Facilité d'adaptation	0	0	-	-	-
connaissance	0	0	+	+	+
Qualité	0	+	++	+	+++++
Rapidité	0	-	-	--	--

Tableau. 2 - Comparaison générale des principales méta-heuristiques

Dans ce tableau, les cinq méta-heuristiques comparées sont les suivantes :

- AI : amélioration itérative (descente) avec relance,
- RS : recuit simulé,
- tabou : méthode tabou,
- AG : algorithme génétique adapté,
- AH : algorithme hybride.

Les critères de comparaisons retenus sont les suivants :

- simplicité de la méta-heuristique, *i.e.* simplicité de la méthode elle-même,
- facilité d'adaptation au problème,
- possibilité d'intégrer des connaissances spécifiques du problème,
- qualité des meilleures solutions trouvées,
- rapidité, *i.e.*, le temps de calcul nécessaire pour trouver une telle solution (sur une machine séquentielle).

La méthode d'amélioration itérative est utilisée comme point de référence pour l'ensemble des méthodes : les signes -, 0, + indiquent des performances respectivement inférieures, égales, supérieures à celles obtenues par l'amélioration itérative.

Le critère de *qualité* utilisé dans le tableau correspond à la meilleure qualité qu'il est possible d'obtenir par une exécution prolongée. Le critère de *rapidité* représente le temps de calcul typiquement nécessaire pour obtenir une telle solution.

4 Analyse

Les méta-heuristiques présentent de nombreux avantages mais aussi un certain nombre de limitations. Nous résumons ci-dessous les différentes caractéristiques de ces méthodes :

- généralité et application possible à une large classe de problèmes,
- efficacité pour de nombreux problèmes,
- possibilité de compromis entre qualité des solutions et temps de calcul,
- optimum non garanti,
- nécessité d'adaptation de la méthode au problème traité,
- nécessité de réglage des paramètres,
- difficulté de prévoir la performance.

Tout d'abord, rappelons qu'il s'agit d'heuristiques dont l'objectif n'est pas le même que celui des méthodes exactes. En particulier, ces méthodes ne garantissent pas de trouver une solution optimale ni de prouver l'optimalité de la solution.

Dans ces conditions, méthodes exactes et méta-heuristiques sont plutôt complémentaires que véritablement concurrentes.

5 Conclusion

Dans ce chapitre nous avons introduit les différentes méta- heuristiques.

Dans le cadre de la problématique d'ordonnancement type flow shop hybride que nous traitons dans ce mémoire, l'approche de résolution proposée est de nature approchée.

Le prochain chapitre présente l'adaptation d'une méta-heuristique qui est l'algorithme génétique proposé pour la résolution du problème flow shop hybride.

CHAPITRE 3

Algorithme génétique pour l'ordonnancement de type flow shop hybride

Chapitre 3

Algorithme génétique pour l'ordonnancement de type flow shop hybride

Sommaire

1	Introduction.....	37
2	Concept de base d'un algorithme génétique	38
3	Terminologie.....	39
4	Les étapes de l'algorithme génétique.....	39
4.1	Codage.....	41
4.2	Génération de la population initiale.....	42
4.3	Evaluation : fitness.....	42
4.4	Sélection.....	42
4.5	Croisement.....	43
4.6	Mutation.....	44
4.7	Insertion.....	45
4.8	Critère d'arrêt.....	45
5	Affectation et séquence ment.....	46
6	Application de l'algorithme génétique au problème d'ordonnancement de type flow shop hybride.....	46
6.1	Les paramètres de l'algorithme génétique.....	46
6.2	L'organigramme de l'algorithme génétique.....	47
6.3	Codage utilisé pour notre algorithme.....	49
6.4	Fonction d'évaluation : fitness.....	51
6.5	Génération d'une population initiale.....	52
6.6	Opérateur de sélection.....	52
6.7	Opérateur d'élitisme.....	53
6.8	Opérateur de croisement.....	53
6.9	Opérateur de mutation.....	55
6.10	Critère d'arrêt.....	56
7	Conclusion.....	56

1 Introduction

Les algorithmes génétiques (AG) sont certainement la branche des algorithmes évolutionnaires la plus connue et la plus utilisée. Ils ont été créés par analogie avec des phénomènes naturels. Dans ce cas, il s'agit de simuler l'évolution naturelle d'organismes (individus), génération après génération, en respectant des phénomènes d'hérédité et une loi de survie. Dans une population d'individus, ce sont en général les plus forts, c'est à dire les mieux adaptés au milieu, qui survivent et donnent une descendance. Par ailleurs, on suppose que les qualités et les défauts peuvent être hérités des parents de manière stochastique. De tels algorithmes furent développés des 1950 par des biologistes qui utilisaient des ordinateurs pour simuler l'évolution des organismes. Vers la fin des années 60, J. Holland (1975) et son équipe, ont adapté ces algorithmes pour la recherche de solutions de problèmes d'optimisation, en développant une analogie entre un individu dans une population et une solution d'un problème dans un ensemble de solutions.

La recherche d'une solution en temps de calcul raisonnable et de bonne qualité revient à utiliser les méta-heuristiques. Ces techniques constituent un ensemble de règles et de mécanismes généraux ayant comme fonction de contrôler et de guider la recherche d'une bonne solution. Elles sont appliquées avec succès pour des problèmes très variés, tels que les problèmes du voyageur de commerce, d'ordonnancement, de localisation, de transport, de partition et de coloration de graphes. Parmi les techniques les plus employées, on trouve les algorithmes génétiques qui présentent des solutions de qualités intéressantes pour la résolution des problèmes combinatoires complexes.

Dans ce chapitre, nous nous intéressons à un problème connu dans l'industrie, le problème du flow shop hybride. L'objectif est donc de proposer un algorithme génétique permettant d'obtenir une solution optimale. Cet algorithme génétique a été adapté l'année dernière au problème d'ordonnancement job shop flexible. Nous adaptons un algorithme génétique pour la résolution du problème d'ordonnancement du flow shop hybride dans un environnement déterministe.

Cet algorithme consiste à minimiser seulement le makespan, sans prendre en compte les perturbations sur les temps d'exécution des jobs sur les machines. L'objectif l'adaptation de cet algorithme est de vérifier son efficacité dans le cas déterministe.

2 Concept de base d'un algorithme génétique

Les algorithmes génétiques font partie de la famille des algorithmes évolutionnaires, inspirés du concept de sélection naturelle élaboré par **Darwin** basé sur la **théorie de l'évolution**. Ils s'inspirent de l'évolution naturelle des espèces. Avec ce type de méthodes, il ne s'agit pas de trouver une solution analytique exacte mais de trouver une bonne solution satisfaisante dans un temps de calcul raisonnable. La première description du processus des algorithmes génétiques a été donnée par J. Holland en 1975, puis Goldberg (1989) les a utilisés pour résoudre des problèmes concrets d'optimisation, [10].

Le but de ces 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* (on utilisera indifféremment individu ou chromosome). Ces derniers sont constitués d'un ensemble d'éléments, appelés "gènes", qui peuvent prendre plusieurs valeurs, appelées "allèles". Un *chromosome* est une *représentation ou un codage d'une solution du problème donné*. 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. Les algorithmes génétiques génèrent de nouveaux individus, de telle sorte qu'ils soient plus performants que leurs prédécesseurs. Le processus d'amélioration des individus s'effectue par utilisation d'opérateurs génétiques, qui sont la **sélection, le croisement et la mutation**.

Pour mettre en oeuvre un algorithme génétique, il est nécessaire de disposer :

- Une représentation génétique du problème, c'est-à-dire un *codage de solutions utilisé sous la forme de chromosomes*.
- Un mécanisme de *générations de la population initiale*. Ce mécanisme est indispensable de construire une population d'individus non homogène.
- Une fonction qui permet *d'évaluer l'adaptation* d'un chromosome à son environnement, ce qui offre la possibilité de comparer des individus. Cette fonction est construite à partir du critère que l'on désire optimiser. L'application de cette fonction à un élément de la population donne sa fitness.
- Un mode de *sélection des chromosomes à reproduire*. Cette sélection est basée sur la *reproduction* et sur le *codage génétique*, qui stocke les informations décrivant l'individu sous forme de gènes.
- L'opérateur de *croisement et de mutation* permettant de diversifier la population au cours des générations et d'explorer l'espace d'état.

- Des paramètres qu'utilise l'algorithme : **taille de la population**, **probabilité de croisement** et de **mutation**, **nombre total de générations**.

3 Terminologie

Nous allons maintenant donner un certain nombre de définitions de vocabulaire nécessaire à une bonne compréhension. En effet, la communauté scientifique utilisant les algorithmes génétiques (AG) utilise nombre d'analogies issues du monde biologique pour illustrer des idées.

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.

Dans le cas général, un gène correspond à un seul symbole (0 ou 1 dans le cas binaire).

Une mutation changera donc systématiquement l'expression du gène mute.

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 du problème étudié.

Définition 3 (Individu) : On appellera individu une des solutions potentielles. Dans la plupart des cas un individu sera représenté par un seul chromosome, dans ce cas, par abus de langage, on utilisera indifféremment individu et chromosome.

Définition 4 (Population) : On appellera population l'ensemble des solutions potentielles qu'utilise l'AG.

Définition 5 (Génération) : On appellera génération l'ensemble des opérations qui permettent de passer d'une population Pop_i à une population Pop_j . Ces opérations seront généralement : sélection des individus de la population courante, application des opérateurs génétiques, évaluation des individus de la nouvelle population.

On pourra aussi appeler $i^{\text{ème}}$ génération l'ensemble des individus après i itérations de l'algorithme.

4 Les étapes de l'algorithme génétique

L'algorithme génétique commence par une génération d'une population initiale de $Psize$ individus, pour lesquels nous calculons leurs fitness et nous sélectionnons les individus par une méthode de sélection. Ces individus seront manipulés par un opérateur de croisement qui les choisit selon une probabilité P_{cross} . Leurs résultats peuvent être mutés par un opérateur de mutation avec une probabilité de mutation P_{mut} . Les phases de sélection et de recombinaison (croisement et mutation) permettent de générer une nouvelle population d'individus, qui ont de bonne chance d'être plus forts que ceux de la génération précédente. Les individus issus de la phase de recombinaison seront insérés par une méthode d'insertion dans la nouvelle population, dont nous évaluons la valeur de la fonction objectif de chacun de ses individus.

De génération en génération, la force des individus de la population augmente et un test d'arrêt sera effectué pour décider quand arrêter l'algorithme. La figure 6 présente un schéma de fonctionnement général de l'algorithme génétique.

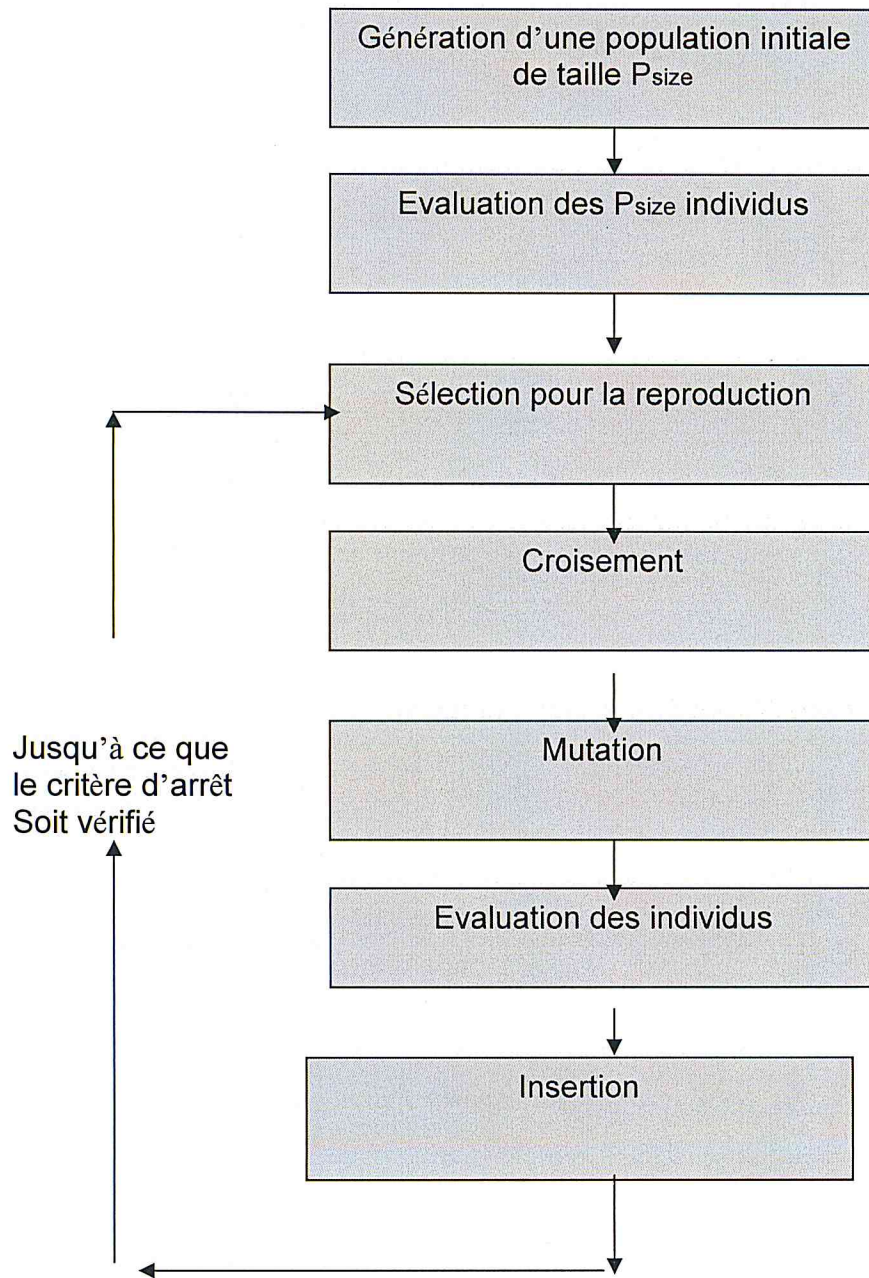


Fig .9 - Fonctionnement général de l'algorithme génétique

4.1 Codage

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

Une population est donc un ensemble de chromosomes. Chaque chromosome code un point de l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome.

Deux types de codages ont été présentés dans la littérature : le *codage direct* et le *codage indirect* Guillaume et al., (2001). Si le *chromosome représente implicitement la solution*, le **codage est direct**. Par contre, si le chromosome *contient un ensemble d'informations spécifiques à un problème donné et la solution du problème est obtenue après une transformation*, le **codage est indirect**.

Un chromosome est représenté sous forme de chaînes de bits 0-1. Ce codage binaire a permis de résoudre beaucoup de problèmes, mais il s'est avéré que, pour **des problèmes d'ordonnancement**, il est plus pratique d'utiliser un **codage entier** des chromosomes. **Ce codage permet d'augmenter l'efficacité de l'algorithme génétique.**

Dans notre cas d'études, Un flow shop hybride est composé d'étages. Chaque étage comporte des machines parallèles identiques, proportionnelles ou identiques. Les produits doivent être traités sur une seule machine de chaque étage. L'ordre des étages est le même pour tous les produits.

Nous avons recensé dans la littérature trois approches pour coder une solution au problème de minimisation de la durée totale de l'ordonnancement.

Le premier codage appelle CD1-AFF (Codage Direct pour l'AFFECTation). Il s'agit de coder une solution sous la forme d'un vecteur dont chaque élément représente un travail et donne le numéro de la machine qui l'exécute. Ce codage offre l'avantage de fournir une unique évaluation pour un individu.

Exemple : Considérons un problème de cinq travaux à exécuter dans un flow shop hybride à *un étage* et *trois machines parallèles*. Un codage possible d'une solution au problème est le suivant : (1, 2, 3, 2, 1). Cela signifie qu'il faut affecter les travaux 1 et 5 sur la machine 1, les travaux 2 et 4 sur la machine 2 et le travail 3 sur la machine 3.

Le deuxième est une représentation matricielle. Il permet de supprimer, dans le cas des machines parallèles identiques, le caractère lié à la numérotation des machines. Ce type de codage est d'autant plus efficace que des contraintes supplémentaires sur les travaux existent. Un codage d'une solution est donc une matrice M telle que :

$M(i,j) = 1$ si les travaux i et j sont sur la même machine

$M(i,j) = 0$ sinon

Le troisième codage de type indirect, appelé (Codage Indirect pour l'affectation), consiste à établir une liste de priorités pour l'ensemble des travaux fournis, par exemple, par un vecteur permutation. Cette solution nécessite l'emploi d'un générateur de solutions. Il aura pour objet de déterminer une solution à partir d'un codage donné.

4.2 Génération de la population initiale

Plusieurs mécanismes de génération de la population initiale sont utilisés dans la littérature. Le problème principal dans cette étape est le *choix de la taille de la population*. Si la taille de la population est trop grande, le temps de calcul augmente et demande un espace mémoire important. Par contre, une population de taille très petite, la solution obtenue n'est pas satisfaisante. Il faut donc trouver le bon compromis.

4.3 Evaluation : fitness

Une fonction d'évaluation est utilisée pour mesurer les performances de chaque individu, qui correspond à une solution donnée du problème à résoudre. Cette fonction permet d'évaluer la capacité d'un individu à survivre en lui affectant un *poids appelé fitness*. La force de chaque chromosome de la population est calculée afin que les plus forts soient retenus dans la phase de sélection, puis modifiés dans la phase de croisement et mutation.

4.4 Sélection

La sélection permet d'identifier les individus susceptibles d'être croisés dans une population.

Il existe plusieurs techniques de sélection. Nous présentons ici les quatre les plus utilisées parmi elles :

- **sélection par rang** : consiste à ranger les individus de la population dans un ordre croissant ou décroissant, selon l'objectif (fonction fitness)

- **sélection par roulette** : elle consiste à créer une roue de loterie biaisée pour laquelle chaque individu de la population occupe une sélection de la roue proportionnelle à sa valeur d'évaluation
- **sélection aléatoire** : cette sélection se fait *aléatoirement*, uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité uniforme $1/Psize$ d'être sélectionné, où $Psize$ est le nombre total d'individus dans la population
- **sélection par tournoi** : le tournoi le plus simple consiste à choisir aléatoirement un nombre k d'individus dans la population et à sélectionner celui qui a la meilleure performance. Les individus qui participent à un tournoi sont remis ou sont retirés de la population, selon le choix de l'utilisateur. Avec le tournoi binaire, sur deux individus en compétition, le meilleur gagne avec une probabilité $p \in [0, 5; 1]$

4.5 Croisement

Le croisement permet d'enrichir la population en manipulant les composantes des chromosomes.

Un croisement est envisagé avec deux parents et génère un ou deux enfants. Il est appliqué avec une probabilité P_{cross} , appelée probabilité de croisement. Après la sélection de deux individus, nous générons un nombre aléatoire $\alpha \in [0, 1]$. Si $\alpha \leq P_{cross}$, appliquons l'opérateur de croisement sur les deux parents.

Selon la littérature, plusieurs opérateurs de croisement sont proposés Nearchou, (2004), nous citons ici les plus utilisés :

- **Croisement en 1-point** : consiste à diviser chacun des deux parents en deux parties à la même position, choisie au hasard et à recopier la partie inférieure du parent à l'enfant et à compléter les gènes manquants de l'enfant à partir de l'autre parent en maintenant l'ordre des gènes. La figure 7 présente un exemple illustratif de ce type de croisement.

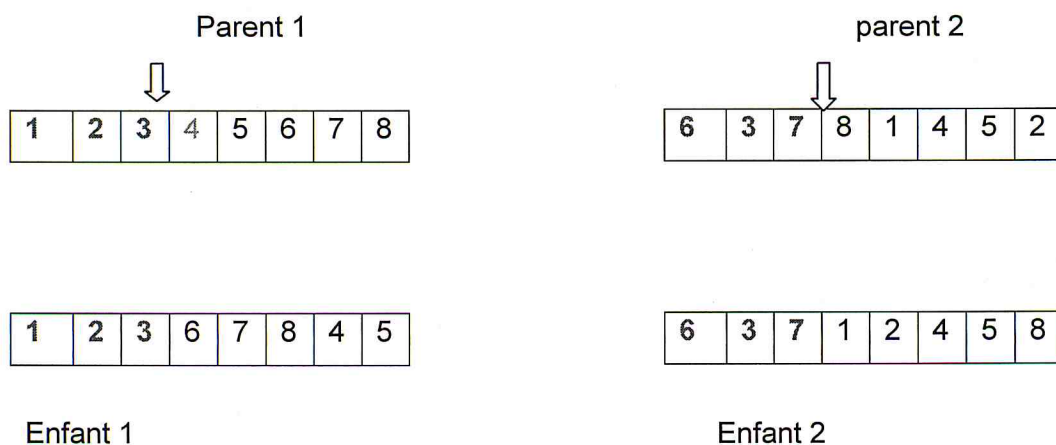


Fig. 10 – Croisement en 1-point de deux chromosomes

–Croisement en 2-points :

Ce type de croisement est utilisé en choisissant aléatoirement 2 points de coupure pour dissocier chaque parent en 3 fragments. Les 2 fragments en extrémités pour le Parent 1 (respectivement Parent 2) sont copiés à l'Enfant1 (respectivement Enfant 2). On complète la partie restante de l'Enfant 1 par les éléments du Parent2 et la partie restante de l'Enfant 2 par les éléments du Parent 1 en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis. La figure présente un exemple illustratif de ce type de croisement.

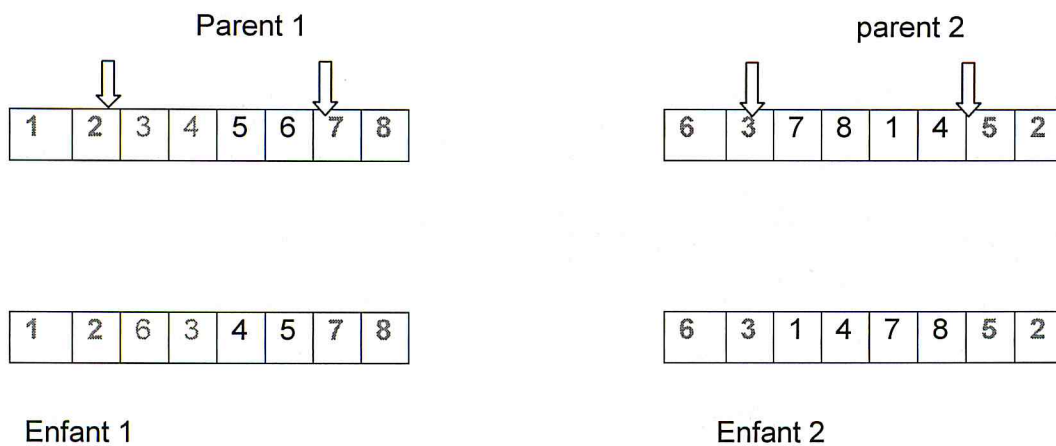


Fig . 11 - Croisement en 2-points de deux chromosomes

4.6 Mutation

La mutation apporte l'aléa nécessaire à une exploration efficace de l'espace. Elle permet de quitter les extremas locaux. Cet opérateur de mutation est utilisé avec une probabilité P_{mut} . Si β est généré aléatoirement, appartient à $[0, P_{mut}]$, nous appliquons l'opérateur de mutation sur cet individu. Comme pour les croisements, de nombreuses méthodes de mutation ont été présentées dans la littérature (Nearchou, 2004). Nous citons les plus connues :

- Opérateur d'inversion simple : cet opérateur consiste à choisir aléatoirement deux points de coupure et inverser les positions des gènes situées au milieu
- Opérateur d'insertion : cet opérateur consiste à sélectionner au hasard un gène et une position dans le chromosome à muter, puis à insérer le gène sélectionné dans la position choisie ;

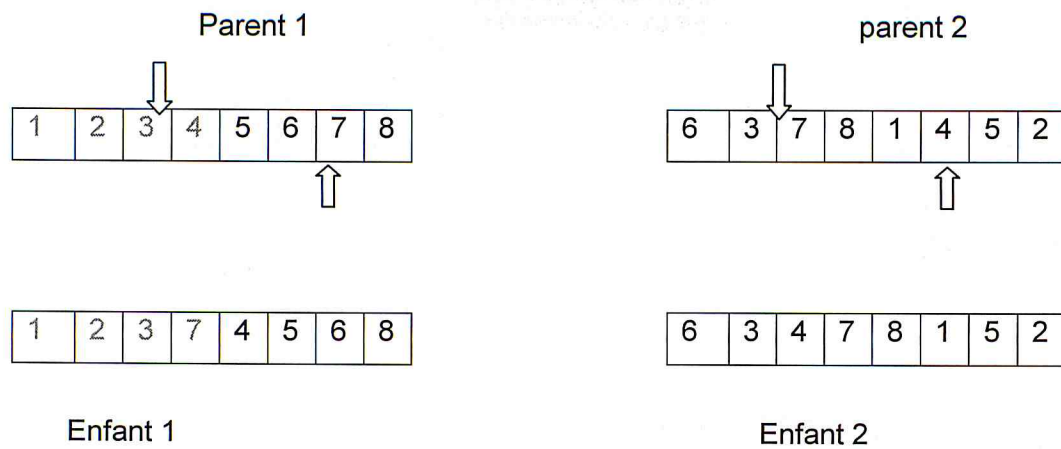


Fig. 12 – Mutation : opérateur d'insertion

– Opérateur d'échange réciproque : c'est un opérateur qui permet de sélectionner deux gènes et de les changer.

4.7 Insertion

Après l'étape de mutation, on utilise une méthode d'insertion pour générer une nouvelle population. Plusieurs stratégies ont été présentées dans la littérature :

– **première stratégie**, consiste à choisir les Psize individus à partir des Psize enfants déjà créés par les opérateurs de croisement et mutation. Dans ce cas, les parents sont remplacés par les enfants mutés.

– **une autre stratégie** consiste à choisir les Psize individus à partir des Psize parents de la population précédente et de Psize nouveaux enfants. A chaque itération, les individus ayant les meilleures fitness seront sélectionnés afin de créer des individus fils qui remplaceront les plus mauvais parents. Le reste survit et sera copié dans la nouvelle génération.

– **l'élitisme**, consiste à copier quelques meilleurs individus dans la nouvelle population. L'objectif est d'éviter que les meilleurs individus soient perdus après les opérateurs de croisement et de mutation. Cette méthode permet de conserver, à une itération donnée, le meilleur individu trouvé dans toutes les populations générées antérieurement.

4.8 Critère d'arrêt

Le test d'arrêt joue un rôle très important dans le jugement de la qualité des individus.

Les critères d'arrêt sont de deux types :

- arrêt après un nombre fixé a priori de générations ,
- arrêt lorsque la population cesse d'évoluer ou n'évolue plus suffisamment.

Les algorithmes génétiques ont prouvé leurs performances, dans la littérature, en tant qu'outils de résolution des problèmes réels. A cet effet, notre objectif est l'adaptation d'un algorithme génétique pour l'ordonnancement de type flow shop hybride.

5 Affectation et séquence ment des travaux

Pour résoudre un tel problème, il faut déterminer l'affectation de chaque opération à l'une des machines éligibles (vue la nature de l'atelier FSH un ensemble de machines est associé à chaque opération mais une seule machine doit l'exécuter). Il reste ensuite à trouver la meilleure séquence d'opérations s'exécutant sur chaque machine. Ainsi, l'algorithme que nous proposons consiste à structurer notre problème *en deux phases : une première pour l'affectation* et une *autre pour le séquence ment*.

Toute opération candidate à un placement se voit attribuer une machine selon une technique spécifique du générateur d'ordonnancement. En effet, si le choix des machines est multiple, l'affectation des opérations peut se faire par des règles de priorité statiques ou dynamiques. Concernant les premières règles, *l'affectation est réalisée avant le processus de séquence ment* et reste donc invariante le long du processus. Quant aux règles dynamiques, elles utilisent une connaissance instantanée du système. Elles déterminent donc les priorités des opérations durant le processus d'ordonnancement.

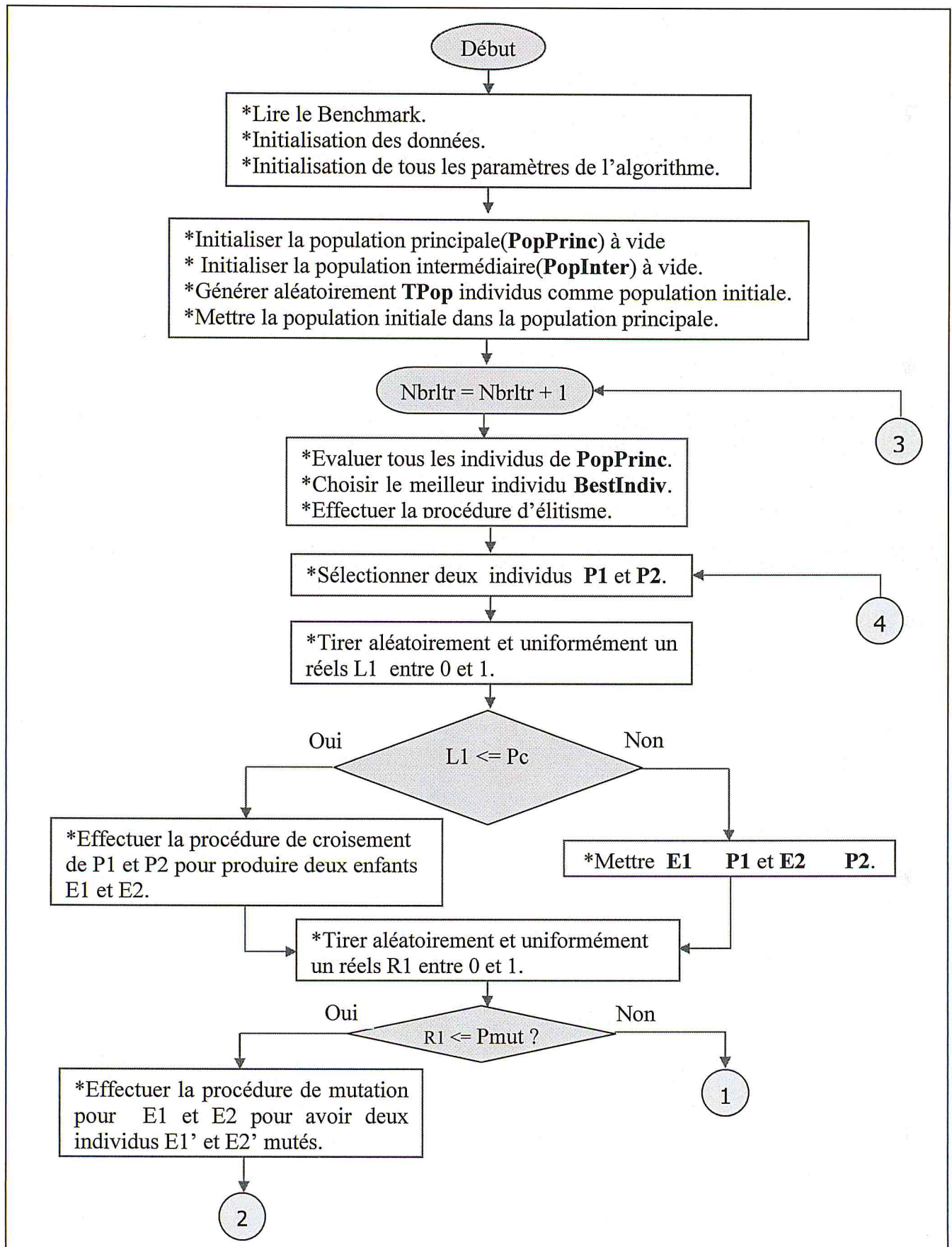
6 Application de l'algorithme génétique aux problèmes d'ordonnancement de type flow shop hybride

Notre application porte sur l'algorithme génétique, cependant, nous avons utilisé un codage et des opérateurs améliorés, basé sur des connaissances spécifiques du problème. L'algorithme implémenté est détaillé par l'organigramme dans la (Figure 13).

6.1 Les paramètres de l'algorithme génétique

1. Initialisation population principale **PopPrinc.**
2. Population intermédiaire **PopInter.**
3. Taille de population **TPop.**
4. Initialisation meilleur individu **BestIndiv.**
5. Probabilité de croisement **Pc.**
6. Probabilité de mutation **Pmut.**
7. Elitisme **Elit.**
8. Type de croisement.
9. Nombre Itérations **Nbrltr.**

6.2 L'organigramme de l'algorithme génétique



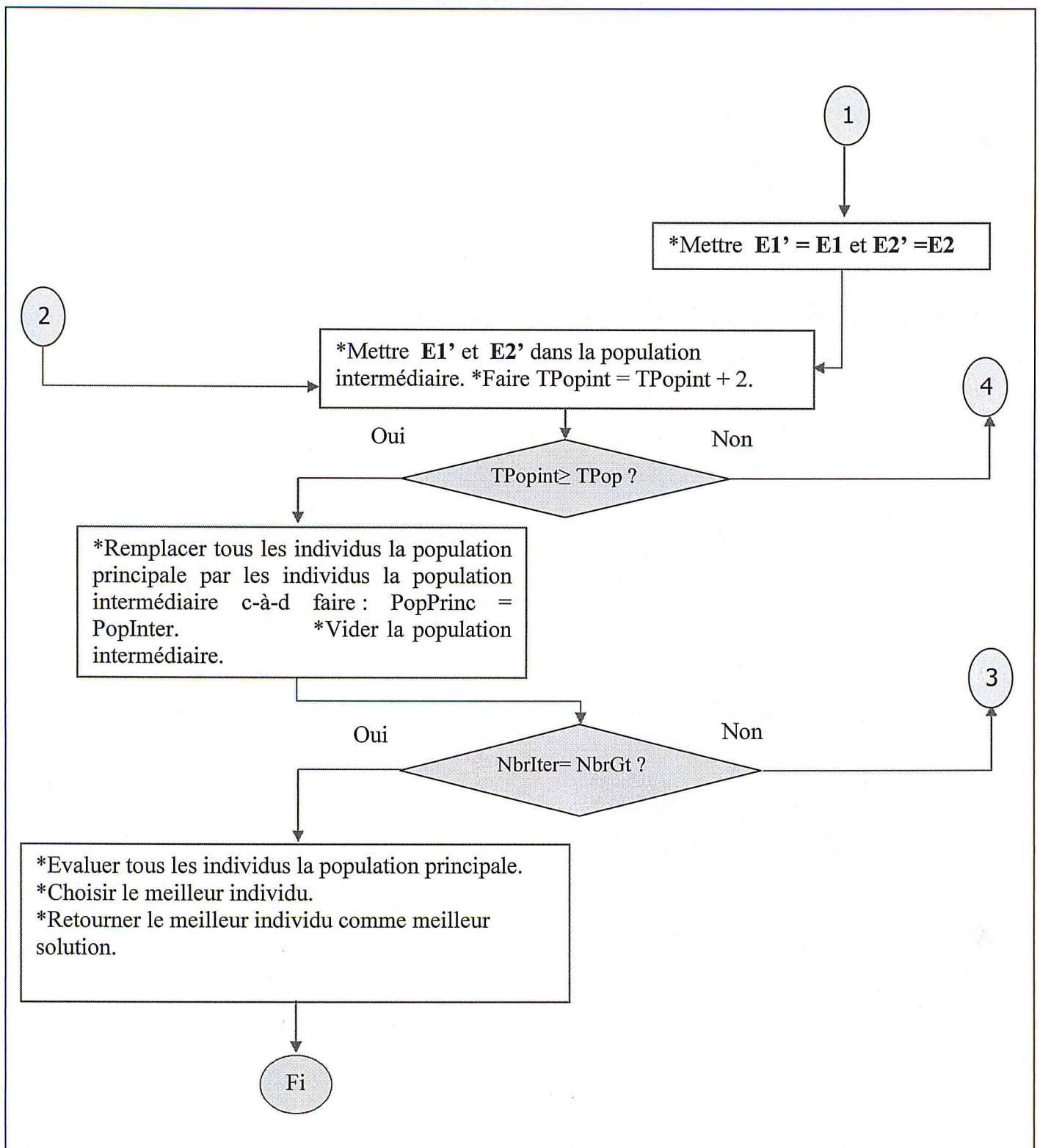


Fig. 13 – l'organigramme de l'algorithme génétique

6.3 Codage utilisé pour notre algorithme

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

Dans notre travail chaque un individu est codé sur deux parties. Une partie pour coder les opérations: «Opération sélection **OS**», et une autre partie pour le codage des choix de machines pour chaque opération: «Machine sélection **MS**».

Exemple : Le tableau suivant donne un exemple d'un problème de type flow Shop hybride.

	Etage	1	2	3
job	machines opérations	M1	M2/ M3	M4
J1	O ₁₁	2	-	-
	O ₁₂	-	4	-
	O ₁₃	-	-	3
J2	O ₂₁	7	-	
	O ₂₂	-	5	-
	O ₂₃	-	-	4

Tableau. 3 - Exemple d'un problème d'ordonnancement de type flow shop hybride.

❖ Partie machine sélection(MS) :

Elle est constituée d'une série ordonnée (selon les jobs) de nombres entiers qui représentent le **choix de machine pour les opérations**, tel que chaque opération à son choix de machine. **La taille** de la partie machine sélection (**MS**) est égale à la **somme des opérations de toutes les tâches** du problème.

Par exemple la figure suivante représente la partie MS d'une solution candidate au problème précédent:

Machine sélection (MS)

1	2	1	1	2	1
O ₁₁	O ₁₂	O ₁₃	O ₂₁	O ₂₂	O ₂₃

Figure.14 - Présentation de la partie Machine sélection (MS)

Le choix machine pour l'opération O11= 1

Le choix machine pour l'opération O12= 2

Le choix machine pour l'opération O13= 1

Le choix machine pour l'opération O21= 1

Le choix machine pour l'opération O22= 2

Le choix machine pour l'opération O23= 1



❖ Partie opération sélection(OS) :

Dans cette partie, nous avons utilisé un codage basé sur les opérations (Opération Based).

Codage basé sur les opérations : un codage basé sur les opérations représente un ordonnancement comme une suite d'opérations. Chaque opération est codée par un gène dans la partie OS du chromosome tel que toutes les opérations d'une tâche portent le même symbole (le numéro de la tâche). L'interprétation de ce symbole se fait suivant l'ordre de son occurrence dans la liste.

Opération sélection (OS)

2	1	2	1	2	1
O21	O11	O22	O12	O23	O13

Figure. 15 - Présentation de la partie Machine opération (OS)

Lecture et évaluation d'un ordonnancement :

La figure illustre un exemple de décodage qui se fait de la façon suivante :

- 1- Placement successif des tâches selon leur ordre d'apparition la fonction OS.
- 2- L'information dans la partie MS permet de déterminer pour chaque opération la machine et le temps d'exécution.
- 3- Tout en satisfaisant les contraintes de disjonction, précédence).

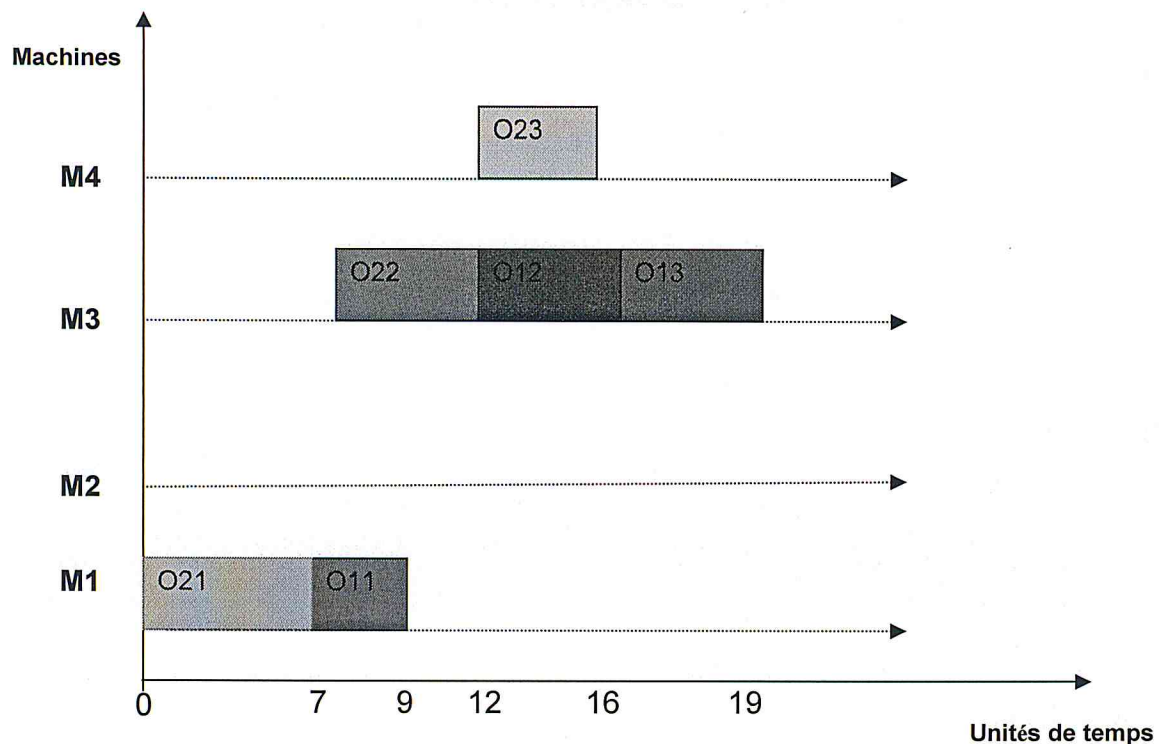


Figure.16 - Présentation d'une solution à l'aide du diagramme de Gantt

Exemple de décodage de O12 :

O12 est placée après O21, O11, O22, elle a le choix machine 2, donc elle est exécutée sur la machine 3, en un temps de 7.

La valeur du choix machine ne correspond pas au numéro de la machine sélectionnée mais au choix.

Exemple :

Pour l'opération O12, le choix machine est de 2, ce qui signifie que l'opération est exécutée sur la machine 3.

Avec ce type de codage, La taille de la partie (**OS**) est égale à la somme des opérations de toutes les tâches du problème. Donc la taille du chromosome (individu) est égale à la taille de la partie **OS** plus la taille de la partie **MS**.

6.4 Fonction d'évaluation : fitness

Une fonction d'évaluation est utilisée pour mesurer les performances de chaque individu, qui correspond à une solution donnée du problème à résoudre. Cette fonction permet d'évaluer la capacité d'un individu à survivre en lui affectant un poids appelé fitness. La force de chaque chromosome de la population est calculée afin que les plus forts soient

retenus dans la phase de sélection, puis modifiés dans la phase de croisement et mutation. La fonction d'évaluation dépend d'un problème à un autre.

La fonction d'évaluation consiste à **minimiser le makespan (noté C_{max})**.

$$C_{max} = \max \{C_{jt}\}$$

$$j=1, \dots, n$$

$$t=1, \dots, k$$

C_{jt} : désigne le temps d'achèvement du job j dans l'étage t .

Ce critère, s'appuie le calcul du *fitness* des individus.

L'évaluation du *fitness* des chromosomes se fait selon trois étapes :

- La construction de l'ordonnancement correspondant au chromosome.
- Le calcul du makespan de cet ordonnancement.
- L'attribution au chromosome de son makespan comme *fitness*.

6.5 Génération d'une population initiale

Dans cette étape, une population initiale doit être générée, où chaque chromosome représente une solution réalisable du problème. La procédure que nous avons utilisée pour générer la population initiale des individus, adaptée à notre problème est une *génération aléatoire*. Ce type d'individu est le meilleur choix, car il permet l'hétérogénéité de la population.

6.6 Opérateur de sélection

La sélection permet d'identifier les individus susceptibles d'être croisés dans une population. Les étapes suivantes montrent la stratégie de la sélection d'individus utilisés dans notre algorithme.

➤ **Etape 1 :**

- Sélectionner aléatoirement deux individus ***Ind1*** et ***Ind2***.
- Choisir le meilleur individu entre ***Ind1*** et ***Ind2*** selon la *fitness*.
- Mettre ***P1*** (***P1*** : c'est le premier individu sélectionné) meilleur individu.

➤ **Etape 2 :**

- Sélectionner aléatoirement de la population deux individus ***Ind3*** et ***Ind4***.
- Mettre ***P2*** (***P2*** : c'est le deuxième individu sélectionné) meilleur individu.

➤ **Etape 3 :**

- Retourner **P1** et **P2** comme les deux individus sélectionnés.

6.7 Opérateur d'élitisme

L'élitisme est une opération qui consiste à copier un ou plusieurs des meilleurs individus dans la nouvelle génération pour ne pas perdre les meilleurs individus après les opérations de la reproduction de l'algorithme génétique.

6.8 Opérateur de croisement

Le croisement permet d'enrichir la population en manipulant les composantes des individus. Un croisement est envisagé avec deux parents et génère un ou deux enfants. Il est appliqué avec une probabilité **Pcross**, appelée probabilité de croisement. Après la sélection de deux individus, nous générons un nombre aléatoire **A** [0, 1]. Si **A** $\leq$ **Pcross**, nous appliquons l'opérateur de croisement (uni-point, multipoints) sur les deux parents.

Dans notre travail nous appliquons un type de croisement pour chaque partie d'individu. Pour la partie (OS) on a utilisé un croisement basé sur les opérations et pour la partie (MS) on a utilisé au choix deux types de croisement un croisement uniforme et un croisement en deux points.

❖ **Croisement de la partie MS**

On a utilisé deux types de croisement soit c'est un croisement **uniforme** ou un croisement en **deux points**. Le choix du type de croisement est déterminé par le paramètre de l'AG « **Type de croisement (0 ou 1)** ».

1. Croisement en deux points

Ce type de croisement est utilisé en choisissant aléatoirement 2 points de coupure pour dissocier chaque parent en 3 fragments. Les 2 fragments en extrémités pour le Parent1(**MS1**) (respectivement Parent2(**MS2**)) sont copiés à l'Enfant1 (respectivement Enfant2). On complète la partie restante de l'Enfant1 par les éléments du Parent2 et la partie restante de l'Enfant2 par les éléments du Parent1 en balayant de gauche à droite et en ne reprenant que les éléments non encore transmis. La figure 1.3 présente un exemple illustratif de ce type de croisement.

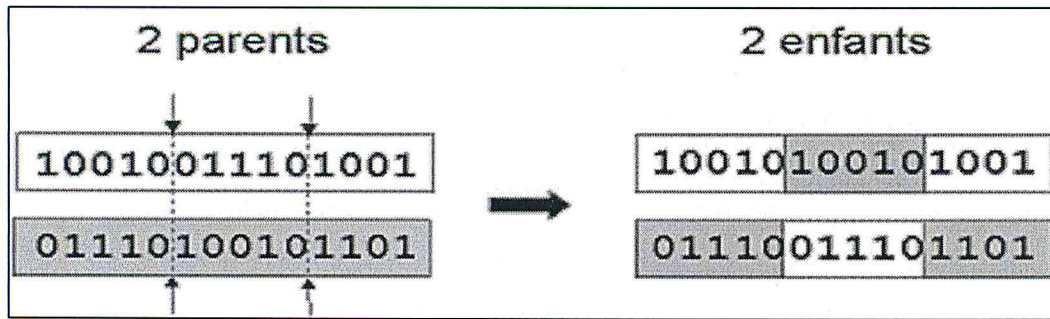


Figure. 17 - croisement avec deux points

2. Croisement uniforme

On peut citer aussi une autre méthode très utilisée dans le cas des problèmes modélisés par un codage binaire, il s'agit du **croisement uniforme**. La mise en œuvre de ce procédé est fort simple, elle consiste à définir de manière aléatoire un "masque", c'est-à-dire une chaîne de bits de même longueur que les chromosomes des parents sur lesquels il sera appliqué.

Ce masque est destiné à savoir, pour chaque locus, de quel parent le premier fils devra hériter du gène s'y trouvant; si face à un locus le masque présente un 0, le fils héritera le gène s'y trouvant du parent **MS1**, s'il présente un 1 il en héritera du parent **MS2**. La création du fils n° 2 se fait de manière symétrique : si pour un gène donné le masque indique que le fils **MS1** devra recevoir celui-ci du parent **MS1** alors le fils **MS2** le recevra du parent **MS2**, et si le fils **MS1** le reçoit du parent **MS2** alors le fils 2 le recevra du parent **MS1**.

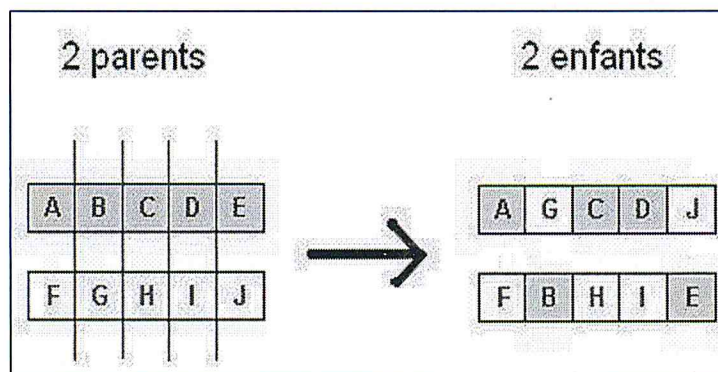


Figure.18 - croisement uniforme

❖ Croisement de la partie OS

Nous avons utilisé une stratégie POX qui est utilisé par [GZ et all, 08] son principe est montré par la figure suivante :

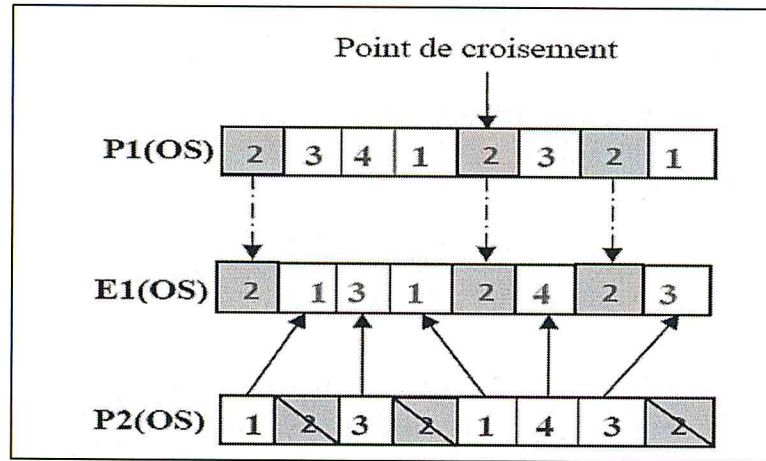


Figure.19 - croisement de type POX

A partir d'un parent :

- Détermination du point de croisement (une tâche est déterminée)
- Copier toutes les opérations de la tâche dans les mêmes emplacements dans l'enfant
- Remplir les emplacements vides dans l'enfant avec les opérations absentes du deuxième parent, dans leurs ordres d'apparition, en ignorant les opérations qui appartiennent à la tâche correspondant au point de croisement.

Remarque :

Nous avons montré le croisement de P1(OS) et P2(OS) pour avoir E1(OS). Alors pour avoir E2(OS), il suffit d'inverser les deux parents (P2(OS) et P1(OS)) et on a appliqué la même stratégie POX.

6.9 Opérateur de mutation

La mutation apporte l'aléa nécessaire à une exploration efficace de l'espace. Elle permet de quitter les extrêmes locaux. Cet opérateur de mutation est utilisé avec une probabilité **Pmut**. Si **B**, généré aléatoirement, appartient à $[0, Pmut]$, nous appliquons l'opérateur de mutation sur cet individu.

Ces opérateurs sont expliqués par les étapes suivantes :

1. Mutation

➤ Etape 1 :

- Un individu **E1** en entrée qui résulte de la procédure de croisement.

➤ Etape 2 :

- Muter la partie **MS** de l'individu **E1** : On tire aléatoirement une opération qui à plus d'un choix de machine et réaffecter cette opération à une autre machine.

➤ Etape 3 :

- Muter la partie **OS** de l'individu **E1** : On tire aléatoirement une opération et en lui change d'indice.

➤ Etape 4 :

- Concaténer la partie **MS** mutée et la partie **OS** mutée afin d'obtenir muté **E1'**.

6.10 Critère d'arrêt

Le critère d'arrêt utilisé est le nombre de générations égal à IterMax. Après IterMax générations, l'algorithme génétique s'arrête et donne le meilleur chromosome qui possède un Cmax minimum.

7 Conclusion

Dans ce chapitre, nous avons commencé par présenter les concepts de base de l'approche de résolution du problème flow shop hybride, qui est l'algorithme génétique.

Les algorithmes génétiques présentent un fort potentiel d'applications pratiques. Ils permettent d'explorer des domaines possédant de très nombreuses solutions.

Il n'y a pas de garantie quant à l'obtention de la solution optimale au problème posé en un temps fini.

Nous avons décrit les différentes étapes de l'algorithme génétique proposé pour la résolution du problème flow shop hybride. Les principales étapes de résolution de cette approche sont illustrées.

Dans le chapitre qui suit, nous allons étudier à travers les expérimentations, la performance de la méthode de résolution appliquée et la comparer avec d'autres méthodes.

CHAPITRE 4

Evaluation et Validation de l'algorithme génétique

Chapitre 4

Evaluation et Validation de l'algorithme génétique

Sommaire

1 Introduction.....	58
2 Validation de l'algorithme génétique pour l'ordonnancement flow shop hybride	58
2.1 Description des benchmarks.....	58
2.2 Expérimentation et résultat.....	59
3 Conclusion.....	71

1 Introduction

Dans le chapitre précédent, nous avons présenté un algorithme génétique pour la résolution du problème d'ordonnancement d'un flow shop hybride. Un algorithme génétique pour l'ordonnancement déterministe minimisant le makespan, sans tenir compte des perturbations sur les temps d'exécution des jobs dans les étages.

Dans ce chapitre, nous testons en premier lieu l'efficacité de l'algorithme génétique pour l'ordonnancement déterministe minimisant le Cmax en utilisant le langage Java sous l'environnement Netbeans.

Dans le but de vérifier les résultats obtenus, cet algorithme a été comparé à trois méthodes de résolution, une méthode LEKIN, une méthode de type Branch and Bound (B&B) et une autre méthode de système immunitaire artificiel (AIS) présentées dans la littérature.

En deuxième lieu, nous validons notre algorithme génétique pour le problème d'ordonnancement flow shop hybride.

Cette série d'expérimentation est effectuée d'une manière statique, sans intégrer les perturbations.

2 Validation de l'algorithme génétique pour l'ordonnancement flow shop hybride

Dans ce chapitre nous présentons l'efficacité de notre algorithme génétique pour l'ordonnancement déterministe dont l'objectif est de minimiser le makespan (Cmax).

2.1 Description des Benchmarks

L'algorithme génétique pour l'ordonnancement déterministe a été testé sur des Benchmarks (jeux-tests, instances) présentés par (Carlier et Néron). Le nombre total de ces benchmarks est 63, dont la taille est de 10 ou 15 jobs, et 5 ou 10 étages, suivant les cas. Les temps d'exécution sont générés aléatoirement dans l'intervalle [3, 15], quel que soit le nombre de machines dans les étages, [1, 12].

Pour un nombre de jobs et un nombre d'étages fixés, il existe plusieurs *configurations machines possibles* (*a*, *b*, *c* et *d*). Nous appelons une configuration de machine une liste de nombres représentant le nombre de machines dans chacun des étages expliqué dans le tableau 4.

Types	J10c5 et j15c5	J10c10	J15c10
A	33133	3333133333	13333333333
B	13333	1333333333	3333333331
C	33233	3333233333	/
D	33333	/	/

Tab. 4 – Différents types de configurations

La configuration de type a dispose de 3 machines dans chaque étage, sauf le premier, ou l'étage qui se situe au milieu (selon le nombre de jobs et d'étages) qui dispose seulement d'une seule machine. Par exemple l'instance j10c5a2 correspond à une configuration de 33133 (de gauche à droite, nous avons 3 machines à l'étage 1, 3 machines à l'étage 2, 1 à l'étage 3, 3 à l'étage 4 et 3 à l'étage 5).

La configuration de type b dispose d'une seule machine dans le premier étage et de 3 machines dans les autres étages (c'est à dire 13333 dans le cas de 5 étages), sauf le cas de 15 jobs et 10 étages, où une seule machine est disponible dans le dernier étage.

La configuration de type c dispose de 3 machines dans chaque étage, sauf l'étage qui se situe au milieu, qui dispose de 2 machines seulement par exemple l'instance j10c5c1 est une instance de 10 jobs et 5 étages, dans laquelle les deux premiers étages disposent de 3 machines, seulement 2 machines dans le troisième étage et les deux derniers étages disposent de 3 machines).

La configuration de type d dispose de 3 machines dans chaque étage.

En se basant sur (Carlier et Néron), certains des benchmarks sont groupés en tant qu'instances difficiles. Ces instances comprennent les configurations de types c et d pour les instances de 10 jobs et 5 étages et les instances de 15 jobs et 5 étages (j10c5× et j15c5×). Le reste des instances (de types a, b et j10c10c) sont identifiées en tant qu'instances faciles.

2.2 Expérimentations et résultats

Les tests de notre algorithme ont été fait avec Microsoft Visual JAVA Netbeans 6.7 et testé sous Windows XP professionnel.

Chapitre4 : Evaluation et Validation de l'algorithme génétique

Les expérimentations sont effectuées sur un ordinateur personnel de marque DELL VOSTRO A 180, Pentium DUAL CORE avec 2.2 GHZ et 1 Gø de RAM.

Nous avons effectué plusieurs séries d'expérimentations, les paramètres suivants ont été retenus :

On teste un ensemble de 20 instances

une taille de population égale à 300 ;

une probabilité de croisement égale à 0.8 ;

une probabilité de mutation égale à 0.1 ;

un critère d'arrêt qui est le nombre de générations, égal à 3000 (IterMax = 3000).

Les résultats de l'algorithme génétique ont été comparés à une méthode LEKIN qui est un système d'ordonnancement développé à Stern of business, université de New York.

N°	Instances	AG Cmax	Cmax MS	Cmax LPT	Cmax SPT	Cmax FCFS	Cmax Général SB Routine	Cmax SBLS
01	J10c5a2	88	98	98	112	105	88	92
02	J10c5b1	130	135	135	156	144	130	130
03	J10c5c5	80	96	96	100	89	78	92
04	J10c5d6	64	77	77	79	71	66	69
05	J10c5a5	122	131	131	146	131	122	132
06	J10c5b6	115	119	119	144	117	115	129
07	J10c5c3	73	80	80	85	80	73	85
08	J10c5d1	69	81	81	83	82	69	82
09	J15c5a4	158	171	171	177	177	156	167
10	J15c5b2	152	155	155	168	158	152	168
11	J15c5c5	82	98	98	87	89	80	82
12	J15c5d6	88	96	96	97	93	90	98
13	J10c10a1	139	153	153	181	174	139	148
14	J10c10b3	170	169	169	200	203	169	169
15	J10c10c4	127	149	149	143	134	131	129

16	J10c10d2	123	138	138	141	125	123	141
17	J15c10a3	199	198	198	253	215	198	224
18	J15c10b4	222	243	243	224	221	221	227
19	J15c10c1	199	198	198	253	215	198	233
20	J15c10b2	191	225	222	191	208	187	195

Tab .5- comparaison entre AG et LEKIN

Interprétation des résultats par les graphes suivants :

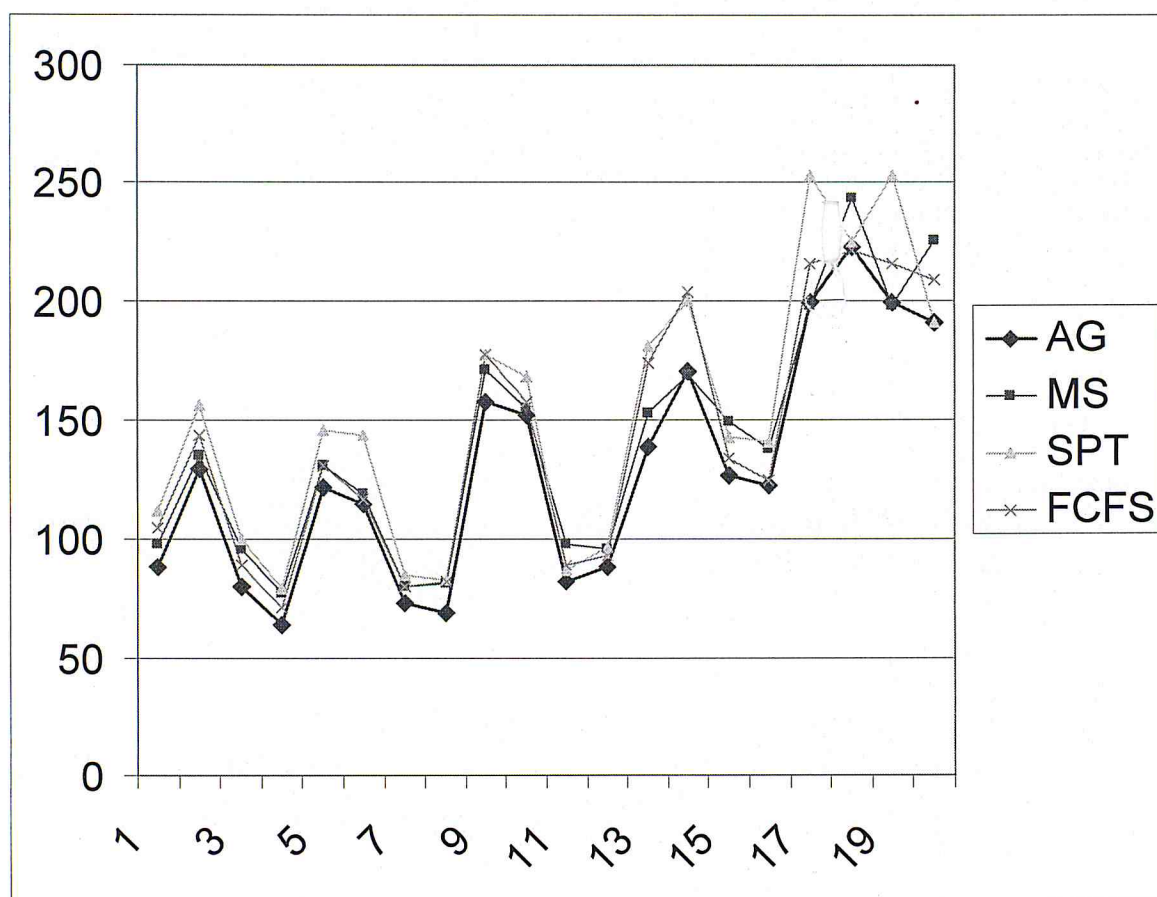


Fig .20- comparaison entre l'AG et les règles de priorités de LEKIN

Les résultats trouvés de l'algorithme génétique sont meilleurs que les résultats trouvés par les règles de priorités de LEKIN.

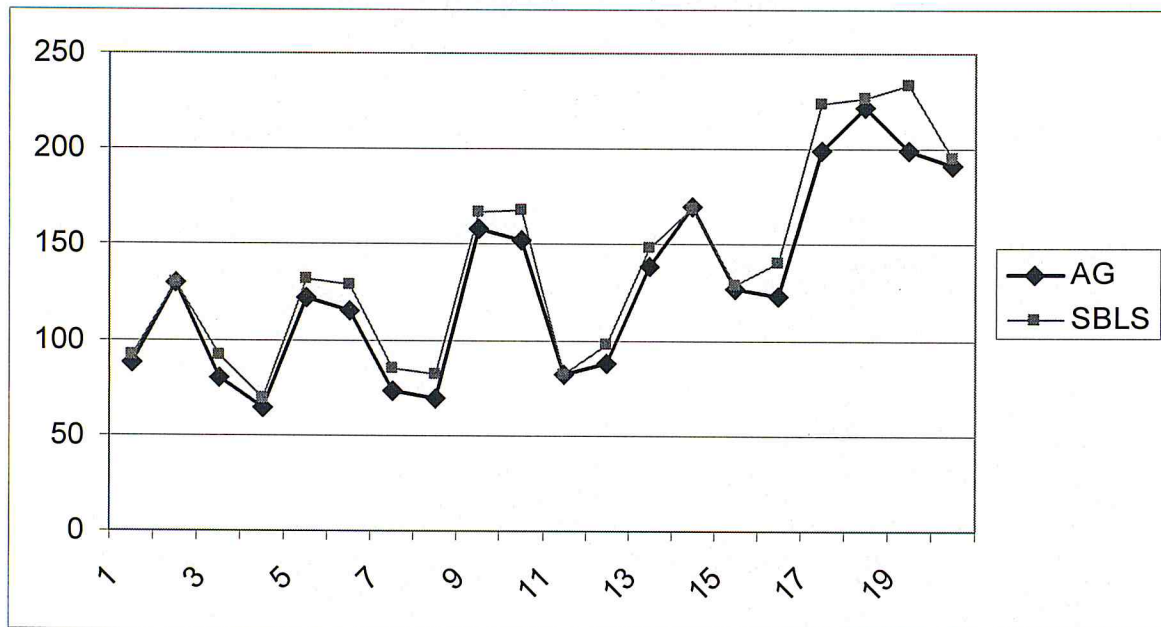


Fig .21- comparaison entre l'AG et l'heuristique SBLS

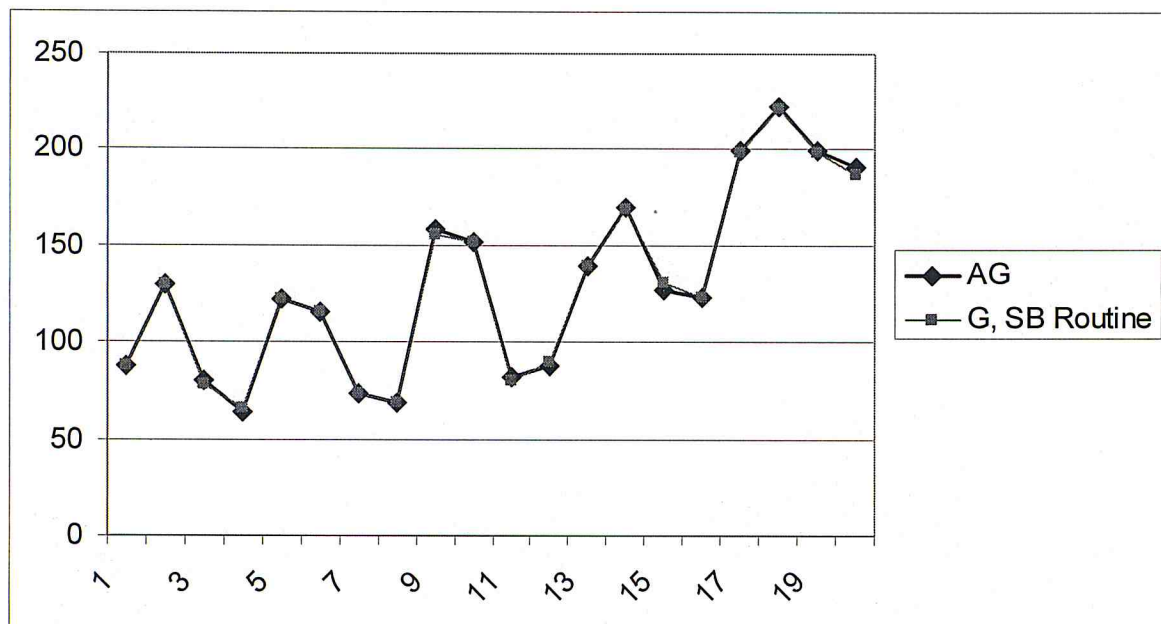


Fig .22- comparaison entre l'AG et SBROUTINE

Quant aux deux heuristiques leurs résultats sont proches des résultats trouvés par l'algorithme génétique.

Une autre comparaison de l'algorithme génétique avec deux méthodes de résolution qui ont été développées dans la littérature. La première méthode, notée B&B , est une procédure par séparation et évaluation (Branch & Bound) développé par Carlier et Néron (2000). La deuxième méthode est un algorithme basé sur le système immunitaire artificiel, noté AIS, développé par Engin et Doyen (2004).

Chapitre4 : Evaluation et Validation de l'algorithme génétique

Les résultats obtenus ont été comparés à une borne inférieure globale (LB) développée Par Carlier et Néron (2000). Le pourcentage de déviation est calculé par la formule suivante :

$$\% \text{Déviation} = \frac{\text{Cmax} - \text{LB}}{\text{LB}} * 100$$

Les tableaux suivant présentent une comparaison des pourcentages de déviation par rapport à la borne inférieur LB de notre algorithme génétique pour l'ordonnancement flow shop hybride avec l'algorithme B&B et AIS développés dans la littérature.

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence% AG	Différence% B&B	Différence% AIS
J10c5a2	88	88	88	88	0	0	0
J10c5a3	117	117	117	117	0	0	0
J10c5a4	121	121	121	121	0	0	0
J10c5a5	122	122	124	122	0	0	0
J10c5a6	110	110	110	110	0	0	0
Moyenne	111,60	111,60	112	111,60	0	0	0

**Tab .6- comparaison entre AG, B&B et AIS pour l'instance de type j10c5a
(instance faciles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence% AG	Différence % B&B	Différence % AIS
J10c5b1	130	130	131	130	0	0	0
J10c5b2	107	107	107	107	0	0	0
J10c5b3	110	109	109	109	0	0	0
J10c5b4	122	122	124	122	0	0	1,63
J10c5b5	153	153	153	153	0	0	0
J10c5b6	115	115	115	115	0	0	0
Moyenne	122 ,83	122,67	147,8	122,67	0	0	0,27

**Tab .7- comparaison entre AG, B&B et AIS pour l'instance de type j10c5b
(instance faciles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence% AIS
J10c5c1	71	68	68	68	4,41	0	0
J10c5c2	76	74	76	74	2,70	0	2,70
J10c5c3	73	71	72	71	2,81	0	1,40
J10c5c4	71	66	66	66	7,57	0	0
J10c5c5	80	78	78	78	2,56	0	0
J10c5c6	73	69	69	69	5,79	0	0
Moyenne	88,8	71	85,8	71	5,16	0	0,68

**Tab .8 - comparaison entre AG, B&B et AIS pour l'instance de type j10c5c
(instance difficiles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence% AG	Différence % B&B	Différence % AIS
J10c10b1	163	163	163	163	0	0	0
J10c10b2	158	157	157	157	0,63	0	0
J10c10b3	170	169	169	169	0,59	0	0
J10c10b4	159	159	159	159	0	0	0
J10c10b5	165	165	165	165	0	0	0
J10c10b6	168	165	165	165	1,83	0	0
Moyenne	163,83	163	163	163	0,50	0	0

**Tab .09- comparaison entre AG, B&B et AIS pour l'instance de type j10c10b
(instances faciles)**

Instances	Best Cmax AG	Best Cmax x B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence % AIS
J10c10c1	122	127	118	118	3,38	7,62	0
J10c10c2	124	116	117	116	6,89	0	0,86
J10c10c3	124	133	108	108	14,81	23,14	0
J10c10c4	127	135	112	112	13,39	20,53	0
J10c10c5	132	145	126	126	4,76	15,07	0
J10c10c6	111	112	102	102	8,82	9,80	0
Moyenne	123,33	128	113,83	113,66	8,67	12,69	0,14

Tab .10- comparaison entre AG, B&B et AIS pour l'instance de type j10c10c (instances faciles)

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence % AIS
J15c5a1	178	178	178	178	0	0	0
J15c5a2	165	165	165	165	0	0	0
J15c5a3	130	130	132	130	0	0	1,53
J15c5a4	158	156	156	156	0	0	0
J15c5a5	164	164	166	164	0	0	1,21
J15c5a6	178	178	178	178	0	0	0
Moyenne	162,16	163,83	162,5	161,83	1,47	1,63	0,45

Tab .11- comparaison entre AG, B&B et AIS pour l'instance de type j15c5a (instances faciles)

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence% AIS
J15c5b1	170	170	170	170	0	0	0
J15c5b2	152	152	152	152	0	0	0
J15c5b3	157	157	157	157	0	0	0
J15c5b4	147	147	149	147	0	0	1,36
J15c5b5	171	166	166	166	3,01	0	0
J15c5b6	175	175	176	175	0	0	0,51
Moyenne	162	161,16	161,66	161,16	0,50	0	0,31

**Tab .12- comparaison entre AG, B&B et AIS pour l'instance de type j15c5b
(instance faciles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence% AIS
J15c5c1	91	85	85	85	7,05	0	0
J15c5c2	95	90	90	90	5,55	0	0
J15c5c 3	93	87	87	87	6,89	0	0
J15c5c4	93	90	89	89	4,49	1,12	0
J15c5c5	82	84	73	73	12,32	15,06	0
J15c5c6	93	91	91	91	2,19	0	0
Moyenne	91,16	87,83	85,83	85,83	6,41	2,69	0

**Tab .13- comparaison entre AG, B&B et AIS pour l'instance de type j15c5c
(instances difficiles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence % AIS
J15c5d1	167	167	167	167	0	0	0
J15c5d2	92	85	86	85	8,23	0	1,17
J15c5d3	90	96	83	83	8,43	15,66	0
J15c5d4	90	101	84	84	7,14	20,23	0
J15c5d5	89	97	80	80	11,25	21,25	0
J15c5d6	88	87	79	79	11,39	10,12	0
Moyenne	102,66	105,5	96,5	96,33	7,74	11,21	0,19

**Tab .14- comparaison entre AG, B&B et AIS pour l'instance de type j15c5d
(instances difficiles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence% AG	Différence% B&B	Différence% AIS
J15c10a1	236	236	236	236	0	0	0
J15c10a2	209	200	200	200	4,5	0	0
J15c10a3	199	198	198	198	0,5	0	0
J15c10a4	229	225	228	225	1,77	0	1,33
J15c10a5	183	183	182	182	0,54	0,54	0
J15c10a6	200	200	200	200	0	0	0
Moyenne	209,33	207	207,33	206,83	1,21	0,09	0,22

**Tab .15- comparaison entre AG, B&B et AIS pour l'instance de type j15c10a
(instances faciles)**

Instances	Best Cmax AG	Best Cmax B&B	Best Cmax AIS	LB	Différence % AG	Différence % B&B	Différence% AIS
J15c10b1	224	222	222	222	0,90	0	0
J15c10b2	191	187	188	187	2,13	0	0,53
J15c10b3	222	222	224	222	0	0	0,90
J15c10b4	222	221	221	221	0,45	0	0
Moyenne	214,75	213	213,75	213	0,87	0	0,35

Tab .16- comparaison entre AG, B&B et AIS pour l'instance de type j15c10b (instances faciles)

Interprétation des résultats

La comparaison faite nous a permis de montrer que le pourcentage de déviation de notre algorithme génétique pour l'ordonnancement déterministe, dans la plupart des instances , égal et parfois meilleur que l'algorithme du B&B et l'algorithme AIS.

Ces derniers sont parfois meilleurs que notre algorithme génétique pour l'ordonnancement déterministe dans quelques instances.

Méthodes	Déviation Instances faciles	Déviation Instances difficiles
AG	1 ,65	6,43
B&B	1,80	4,66
AIS	0,21	0 ,29

Tab .17- Performance des trois méthodes

Suite à ces résultats trouvés nous interprétons les résultats de la façon suivante :

Pour les instances faciles : L'algorithme génétique fourni de meilleurs résultats par rapport à la méthode B&B par contre la méthode AIS est meilleure que l'algorithme génétique pour l'ordonnancement déterministe.

Pour les instances difficiles : Nous avons trouvé un pourcentage de déviation moyen de l'algorithme génétique égal à 6,43, la méthode AIS obtient de meilleurs résultats par rapport à la méthode B&B .

Le pourcentage de déviation pour les des instances faciles est inférieur à celui des instances difficiles, les instances faciles convergent rapidement vers la borne inférieure.

Le but des tests effectués n'est pas de donner de meilleurs résultats en terme de pourcentage de déviation par rapport à LB ou de montrer que l'algorithme génétique est meilleur que d'autres méthodes. Notre objectif est de montrer que l'algorithme génétique appliqué pour l'ordonnancement déterministe permet de donner des solutions fiables ou acceptables.

Les résultats comparatifs sont représentés dans les figures suivantes :

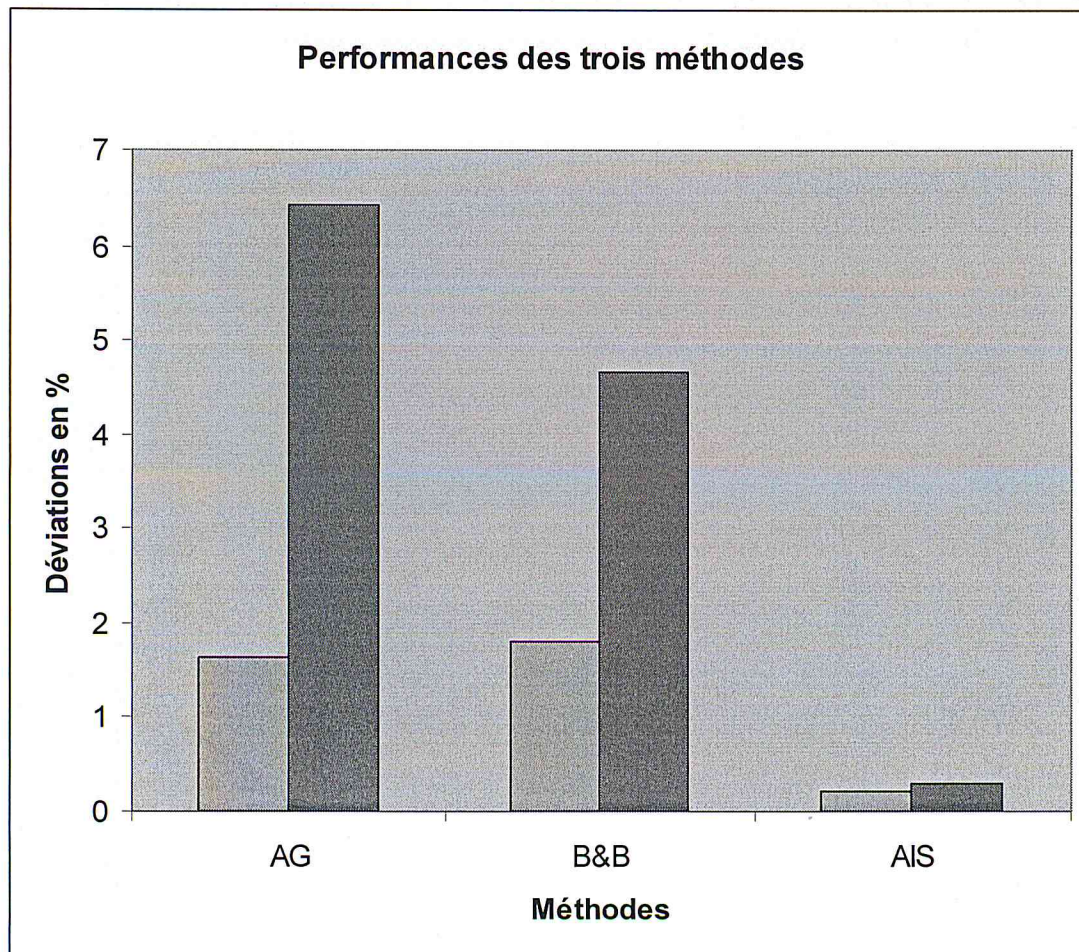


Fig .23- Performance des trois méthodes

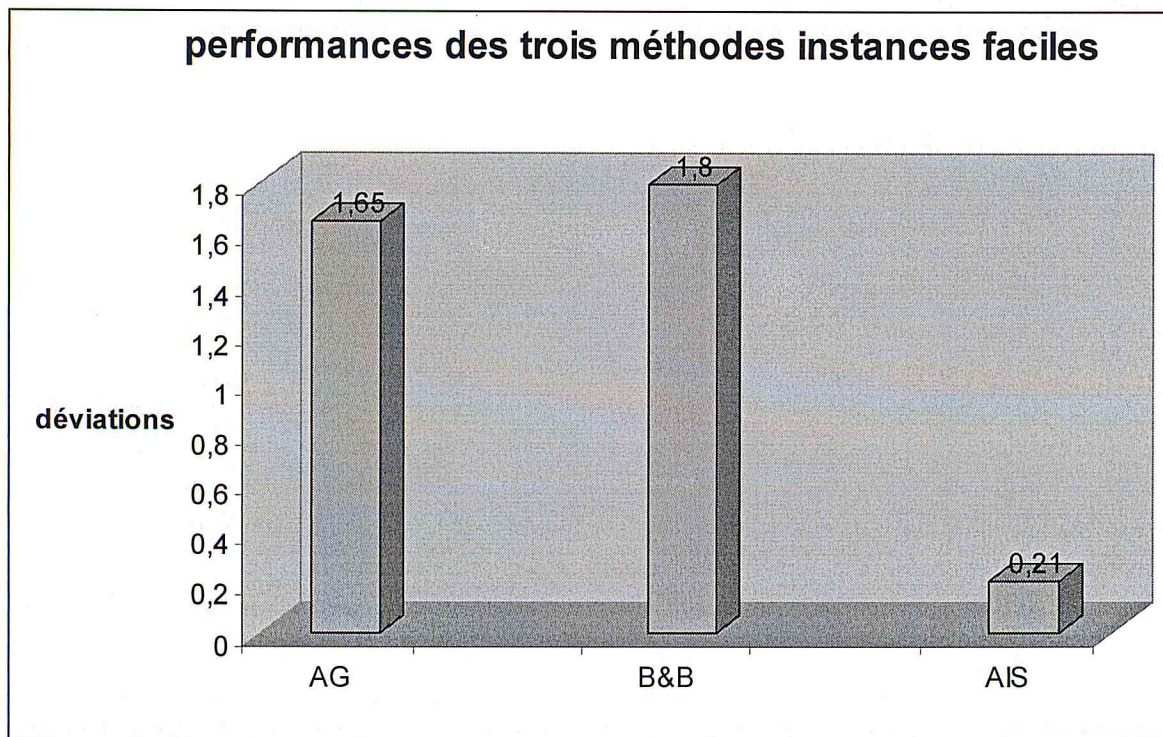


Fig .24- Performance des trois méthodes- instances faciles

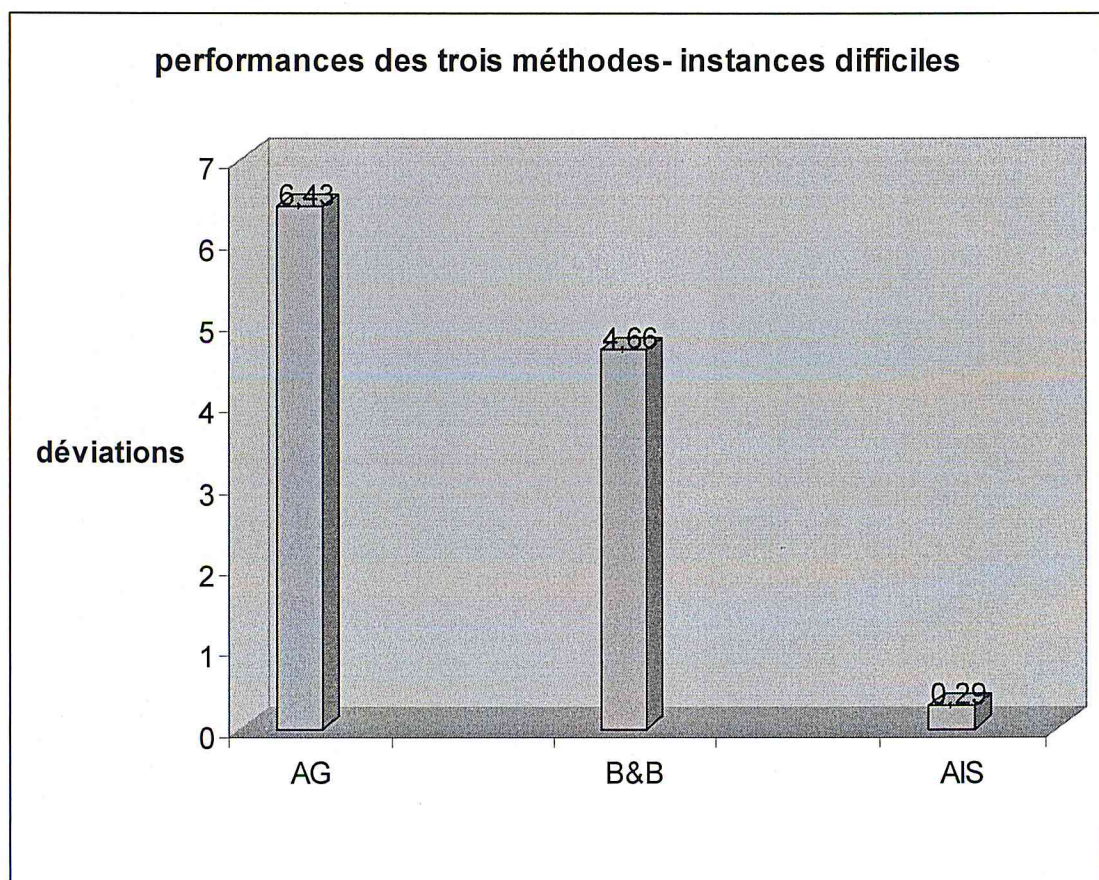


Fig .25- Performance des trois méthodes- instances difficiles

3 Conclusion

Dans ce chapitre, nous avons effectué une série d'expérimentation pour tester l'efficacité de l'algorithme génétique qui a pour objectif de minimiser le makespan (C_{max}).

Notre algorithme génétique a été comparé lors d'une première série d'expérimentation aux règles de priorité dans LEKIN. La deuxième série d'expérimentation a été effectuée en comparant l'algorithme génétique aux méthodes heuristiques dans LEKIN.

Ensuite nous avons comparé notre algorithme génétique à deux méthodes de résolution Branch and Bound et AIS présentées dans la littérature.

Les résultats trouvés montrent que notre algorithme génétique nous permet de résoudre le problème de manière effective.

CONCLUSION GENERALE

Conclusion générale et perspectives

Les problèmes d'ordonnancement d'ateliers de type flow shop hybride sont des problèmes d'ordonnancement extrêmement complexes. Ils sont classés parmi les problèmes combinatoires difficiles au sens fort. Cette complexité est due à l'explosion combinatoire du nombre de solution qui croît exponentiellement avec la taille du problème. De ce fait, l'utilisation de méthodes exactes en vue de l'obtention de solutions optimales semble non réaliste. Le recours à des méthodes approchées comme les heuristiques est donc devenue incontournable. Parmi ces méthodes, le paradigme des Méta-heuristiques s'impose comme une approche très prometteuse. En effet, en plus de leur adaptabilité aux différents problèmes combinatoires, les méta-heuristiques ont l'avantage de ne parcourir qu'une faible fraction de l'espace de solution pour parvenir à une solution acceptable. Ce qui réduit nettement les temps de calcul.

Nous nous sommes intéressés dans ce travail à l'adaptation d'un algorithme génétique au problème d'ordonnancement de type flow shop hybride.

Un cadrage théorique de notre thème a consisté à présenter ses trois volets : les problèmes d'ordonnancement en générale, le problème d'ordonnancement des ateliers de type Flow Shop hybride et les Méta-heuristiques. La deuxième partie du travail s'est attachée à l'adaptation proprement dite de l'approche. Des essais numériques ont été ensuite réalisés, afin d'évaluer et de comparer les résultats obtenus, et de valider notre application.

Les tests ont été effectués sur des benchmarks existant dans la littérature, dont l'objectif est la minimisation du makespan (C_{max}) et notre objectif était de montrer que l'algorithme génétique appliqué pour l'ordonnancement déterministe type Flow Shop Hybride peut donner des solutions fiables ou acceptables.

Ainsi, les résultats obtenus montre que l'algorithme est effectif pour la résolution de ce problème, et bien que pour certains cas la déviation des résultats par rapport aux valeurs optimales reste sensiblement palpable, l'algorithme est dans sa globalité efficace.

Cette étude ouvre différentes perspectives en relation directe avec cette thématique de recherche. Nous envisageons donc l'exploration de l'hybridation de l'approche adoptée avec d'autres méthodes de résolution notamment des méthodes de recherche locale, mais aussi l'extension de l'approche à d'autres variantes du problème.

**REFERENCES
BIBLIOGRAPHIQUES**

Bibliographie

- [1] A. Benhmida Sakli , Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible, thèse de doctorat, 2009.
- [2] A,Vignier, J-C, Billavi, C,Poroust, les problèmes d'ordonnancement de type flow shop hybride « état de l'art », revue française de l'automatique d'informatique et de recherche opérationnelle, tome 33,N° 2,p117-183, 1999.
- [3] C. Gagné, l'ordonnancement industriel : stratégies de résolution méta heuristiques et objectifs multiples, thèse Phd 2001.
- [4] Ghota, les problèmes d'ordonnancement, Recherche opérationnelle, opérations research, 1993.
- [5] J.Kao Hao,P.Galinier, M.Habib, méta- heuristiques pour l'optimisation combinatoire et l'affectation sous contraintes, revue intelligence artificielle,1999.
- [6] J. Teghem et M. Pirlot, optimisation approchée en recherche opérationnelle, Hermes science, 2002.
- [7] K . Alykyran, using ant colony optimisation to solve hybrid flow shop scheduling problems, Inte j Adv Manuf Technol, 2007.
- [8] P. Lopez et F. Roubellat, ordonnancement de la production, Hermes science Ltd 2001.
- [9] S. Khalouli, Méta heuristiques à base de modèle : application à l'ordonnancement d'atelier flow shop hybride monocritère et multi critère, thèse de doctorat,2010.
- [10] S.Khalouli, F.Ghedjati, A.Hamzaoui, une méta heuristique pour un ordonnancement juste à temps d'un atelier flow shop hybride.

[11] S. Khalouli, F.Ghedjati, A. Hamzaoui, une méta- heuristique pour un ordonnancement juste à temps d'un atelier flow shop hybride ROMDEF 2010.

[12] T. Chari, un algorithme génétique pour l'ordonnancement robuste : application au problème de flow shop hybride, thèse de doctorat, 2010.

[13] Rubén Ruiz, José Antonio Vázquez Rodríguez: The hybrid flow shop scheduling problem. European Journal of Operational Research 205(1): 1-18 (2010).

ANNEXES

DETAILS DES TESTS EFFECTUES

ANNEXE I

LEKIN

LEKIN est un système d'ordonnancement développé en 1995 à Stern, université de New York.

Il a été créé comme un outil pédagogique dont le but principal d'initier les élèves à la théorie d'ordonnancement et ses applications.

Lecture d'un benchmark

Nous donnons comme exemple une instance J10c5a5

5

10

12	14	6	3	3	6	11	6	8	3	3
3	12	13	12	6	15	15	6	11	8	3
14	13	6	13	11	11	10	11	5	8	1
14	14	6	4	14	7	8	10	10	13	3
10	10	10	7	7	14	7	7	4	15	3

Première ligne : indique le nombre de centres ou étages

Deuxième ligne : indique le nombre de jobs

Troisième à la cinquième ligne : indiquent les durées d'exécutions des jobs dans chaque machine

Pour la dernière colonne , elle indique le nombre de machine dans chaque étage

ANNEXE II

Tests effectués pour les 63 benchmarks

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J10c5a2	88	88	88	88	88	88
J10c5a3	117	117	117	117	117	117
J10c5a4	121	121	121	121	121	121
J10c5a5	122	122	122	122	122	122
J10c5a6	110	110	110	110	110	110

**Tab .6- comparaison entre AG, B&B et AIS pour l'instance de type j10c5a
(instance faciles)**

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J10c5b1	130	130	130	130	130	130
J10c5b2	107	107	107	107	107	107
J10c5b3	110	110	110	110	112	110
J10c5b4	122	127	122	122	122	122
J10c5b5	153	153	153	153	153	153
J10c5b6	115	115	115	115	115	115

Tab .7- Test de l' AG pour l'instance de type j10c5b

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J10c5c1	72	71	75	71	73	71
J10c5c2	78	79	76	78	81	76
J10c5c3	74	73	75	76	74	73
J10c5c4	71	72	72	71	69	71
J10c5c5	87	80	86	81	81	80
J10c5c6	75	73	77	78	76	73

Tab .8- Test de l' AG pour l'instance de type j10c5c

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J10c10b1	163	163	163	163	163	163
J10c10b2	158	158	158	158	170	158
J10c10b3	174	173	175	170	174	170
J10c10b4	170	163	170	159	161	159
J10c10b5	165	165	176	165	165	165
J10c10b6	168	170	170	168	168	168

Tab .9- Test de l' AG pour l'instance de type j10c10b

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J10c10c1	125	123	122	122	124	122
J10c10c2	125	124	125	124	125	124
J10c10c3	123	131	126	124	126	124
J10c10c4	138	133	132	127	129	127
J10c10c5	142	141	138	136	132	132
J10c10c6	113	114	116	111	115	11

Tab .10- Test de l' AG pour l'instance de type j10c10c

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c5a1	178	178	178	178	178	178
J15c5a2	165	165	165	165	165	165
J15c5a3	130	130	130	130	131	130
J15c5a4	158	158	158	159	158	158
J15c5a5	164	164	164	164	164	164
J15c5a6	178	178	178	178	178	178

Tab .11- Test de l' AG pour l'instance de type j15c5a

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c5b1	170	170	170	170	170	170
J15c5b2	152	152	152	152	152	152
J15c5b3	160	157	158	160	158	157
J15c5b4	150	150	147	150	150	147
J15c5b5	168	171	171	173	173	171
J15c5b6	175	175	175	175	175	175

Tab .12- Test de l' AG pour l'instance de type j15c5b

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c5c1	91	90	97	91	92	91
J15c5c2	102	102	97	95	99	95
J15c5c 3	91	98	99	96	93	93
J15c5c4	94	93	101	99	102	93
J15c5c5	83	85	87	82	84	82
J15c5c6	97	102	93	102	95	93

Tab .13- Test de l' AG pour l'instance de type j15c5c

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c5d1	167	167	167	170	170	167
J15c5d2	92	92	97	92	94	92
J15c5d3	91	91	95	90	88	90
J15c5d4	93	92	92	90	90	90
J15c5d5	90	92	92	89	89	89
J15c5d6	88	91	88	88	91	88

Tab .14- Test de l' AG pour l'instance de type j15c5d

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c10a1	236	236	236	248	238	236
J15c10a2	211	219	209	214	214	209
J15c10a3	200	199	212	205	205	199
J15c10a4	229	238	229	229	229	229
J15c10a5	183	194	193	183	192	183
J15c10a6	214	213	205	200	213	200

Tab .15- Test de l' AG pour l'instance de type j15c10a

Instances	Test1	Test2	Test3	Test4	Test5	Best Cmax
J15c5b1	241	238	229	224	238	224
J15c10b2	202	195	191	195	194	191
J15c10b3	222	242	228	232	228	222
J15c10b4	222	238	243	228	225	222

Tab .16- Test de l' AG pour l'instance de type j15c10b