N° d'ordre : 2451

THESE

Présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR

Spécialité : Automatique et Informatique Industrielle

Par

Khaled MESGHOUNI

Ingénieur électronicien de l'université de Constantine

APPLICATION DES ALGORITHMES EVOLUTIONNISTES DANS LES PROBLEMES D'OPTIMISATION EN ORDONNANCEMENT DE LA PRODUCTION

Soutenue le 5 janvier 1999 devant le jury d'examen :

Président : J.C. Gentina (Prof, Ecole Centrale de Lille, France)

Rapporteurs : C. Tahon (Prof, Université de Valenciennes; France)
: M.G. Singh (Prof, UMIST, Manchester, Royaume-Uni)

Examinateurs : R. Soenen (Prof, Université de Valenciennes, France)
: S. Maouche (Prof, Université de Lille I, France)
: S. Hayat (Dr. HDR, INRETS-ESTAS, France)

Directeurs de thèse : P. Borne (Prof, Ecole Centrale de Lille, France)
: S. Hammadi (Dr, Ecole Centrale de Lille, France)

Travail préparé au Laboratoire d'Automatique et d'Informatique Industrielle de Lille (LAIL UPRESA 8021)

A mes parents

A mon frère et mes Sœurs

A Soumeya

Table des Matières

Avant propos 4

Introduction générale 5

I Problèmes d'ordonnancement : Définitions et choix d'une problématique 8

I.1 Introduction 8

I.2 Définitions 10

I.2.1 Les opérations élémentaires (Les tâches) 10

I.2.2 Les contraintes 11

I.2.3 Les ressources 12

I.2.4 Les critères 12

I.3 Pourquoi s'intéresser aux problèmes d'ordonnancement ? 14

I.3.1 Complexité des algorithmes 15

I.3.2 Complexité de problèmes 16

I.4 Comment résoudre les problèmes d'ordonnancement ? 17

I.4.1 Méthodes exactes 17

I.4.2 Méthodes approchées : 19

I.4.3 Programmation avec contraintes 24

I.4.4 Algorithmes évolutionnistes 25

I.4.5 Classification des problèmes d'ordonnancement 25

I.5 Choix d'une problématique 27
I.5.1 Le projet 4 GRAISyHM 28
I.6 Conclusion 30

II Algorithmes génétiques **32**
II.1 Introduction 32
II.2 Que sont les algorithmes génétiques ? 33
II.2.1 Principe 34
II.2.2 L'opérateur de sélection 35
II.2.3 L'opérateur de croisement 36
II.2.4 L'opérateur de mutation 38
II.2.5 Notion de schéma 39
II.3 Adaptabilité des algorithmes génétiques 44
II.3.1 Les problèmes d'ordonnancement vus par les algorithmes génétiques 44
II.3.2 Le codage dans les problèmes d'ordonnancement 46
II.3.3 Le croisement pour les problèmes d'ordonnancement 51
II.3.4 La mutation pour les problèmes d'ordonnancement 57
II.4 Conclusion 57

III Problèmes d'ordonnancement et d'affectation : Approche par les algorithmes évolutionnistes **59**
III.1 Introduction 59
III.2 Identification du problème considéré 60
III.2.1 Données et hypothèses 60
III.2.2 Les contraintes 61
III.3 Première approche 62
III.3.1 Codage parallèle des machines (CPM) 63
III.3.2 Création de la population initiale 65
III.3.3 L'opérateur de sélection 67

III.3.4 L'opérateur de croisement pour le CPM 67
III.3.5 Les opérateurs de mutation pour le CPM 72
III.4 Deuxième approche 76
III.4.1 Codage parallèle des ordres de fabrication (CPOF) 77
III.4.2 Opérateurs de croisement pour le CPOF 78
III.4.3 Opérateur de mutation pour le CPOF 83
III.5 Calcul de la date de début d'exécution 85
III.5.1 Algorithme de calcul des dates pour le CPM 85
III.5.2 Algorithme de calcul des dates pour le CPOF 86
III.6 Calcul de la fonction d'évaluation 87
III.7 Conclusion 90

IV Application et résultats **91**
IV.1 Première approche 91
IV.1.1 Exemple de petite taille 92
IV.1.2 Exemple de taille importante 96
IV.2 Deuxième approche 103
IV.3 Durée d'exécution 108
IV.4 Conclusion 110

Conclusion et perspectives **111**

Annexe A **121**

Annexe B **123**

Avant propos

Cette thèse est l'aboutissement des travaux réalisés au Laboratoire d'Automatique et d'Informatique Industrielle de Lille (LAIL UPRESA C.N.R.S. 8021) dans l'équipe Analyse et Commande sous la direction conjointe du Professeur Pierre BORNE. (Directeur scientifique de l'Eole Centrale de Lille, Vice-président d'IEEE/SMC et vice-président de l'IMACS) et le Docteur Slim HAMMADI Mâitre de Conférence à l'Ecole Centrale de Lille.

Nous tenons à remercier très vivement le **professeur Pierre BORNE** pour les précieux conseils qu'il nous a prodigués tout au long de ce travail. Nous le remercions tout particulièrement d'avoir su nous transmettre "le virus" de la recherche et du travail bien fait.

Ce travail n'existerait pas sans l'aide du **Docteur Slim HAMMADI**. Il nous à été d'un grand soutien aussi bien scientifique que moral. Qu'il reçoive ici toute notre reconnaissance.

Nous sommes très reconnaissant au **Professeur Christian TAHON** (Professeur à l'Université de Valenciennes et du Hainaut-Cambrésis) ainsi qu'au **Professeur Madan.G SINGH** (Professeur à l'Université de Manchester, UMIST, Royaume Uni) pour l'honneur qu'ils nous font en acceptant malgrè leur lourdes tâches, d'examiner ce travail et d'être les rapporteurs de cette thèse.

Nous remercions vivement le **Professeur Jean-Claude GENTINA** (Directeur de l'Ecole Centrale de Lille) et le **Professeur Salah MAOUCHE** (Professeur à l'Université des Sciences et Technologies de Lille) ainsi que le **Professeur René SOENEN** (Professeur à l'Université de Valenciennes et du Hainaut-Cambrésis), pour l'honneur qu'ils nous font en examinant ce travail et en acceptant de participer à notre jury de thèse malgrè leur nombreuses occupations.

Nous tenons également à remercier le **Docteur HDR Saïd HAYAT** (chargé de recherche à l'Institut National de Recherche sur les Transports et leur Sécurité) pour ses conseils, ainsi que pour l'honneur qu'il nous fait en acceptant de participer à notre jury de thèse.

Nous tenons à adresser un merci particulier à tous les membres du LAIL pour leur sympathie et leur bonne humeur, leur disponibilité et leurs conseils avec une mention particulière à mes collègues de bureau et plus spécialement Philippe TCHANGANI et Patrick GALLAIS, Monsieur "dépannage PC" ainsi qu' à mon ami Aîssa.

Introduction générale

L'industrie est confrontée actuellement à une mondialisation des marchés, le développement dans les domaines de communication et du transport rend la concurrence très accrue, ceci conduit les entreprises à mettre en place une logique industrielle qui remet en cause certains fondements tels que l'organisation et la gestion des ateliers de fabrication. Cette gestion passe essentiellement par une réorganisation des tâches à exécuter afin d'optimiser un ou plusieurs critères, permettant une fabrication à moindre coût temporel et/ou financier. Ceci est désigné sous le nom de problème d'ordonnancement.

En effet, les problèmes d'ordonnancement appartiennent à la catégorie des problèmes NP- difficiles. Parmi les méthodes permettant la résolution de ce type de problème il y'a les méthodes exactes telles que le "branch and bound" et la programmation dynamique. Ces dernières ne sont malheureusement pas applicable aux problèmes de taille industrielle. Leur temps de calcul et leur complexité mathématique les excluent d'une éventuelle utilisation. Heureusement il existe une deuxième catégorie de méthodes dites approchées. A défaut d'obtenir l'optimum, il faudrait se contenter d'une solution approchée. L'efficacité de ces méthodes dépend de leur capacité à s'approcher le plus possible et en un temps raisonnable de la valeur optimale du critère à optimiser.

Le travail présenté dans ce mémoire traite de l'apport des algorithmes à stratégie d'évolution dans le domaine de l'optimisation des problèmes d'ordonnancement des ateliers du type job-shop flexible, avec contraintes de précédence et machines non reliées. Le critère retenu est la minimisation de la durée totale de l'exécution des tâches, soit l'optimisation du Cmax ou Makespan.

L'utilisation des algorithmes évolutionnistes dans de nombreux domaines a fait ses preuves, notamment dans les problèmes combinatoires à fortes contraintes tels que les problèmes d'ordonnancement. Dans ce travail, nous avons procédé à la mise au point de

deux nouveaux codages, ainsi que de différents opérateurs génétiques permettant, soit la correction des éventuelles "malformations" de certaines solutions ou encore l'accélération de la convergence.

Ce mémoire est organisé en 4 chapitres.

Dans le premier chapitre, nous donnons certaines définitions générales concernant les problèmes d'ordonnancement et nous présentons les différentes approches de résolution. Nous présentons en suite la problématique que nous avons retenue ainsi que les motivations de ce choix.

Le deuxième chapitre présente les algorithmes génétiques de base, leurs principes de fonctionnement et leurs particularités. Nous présentons aussi leur extension, les algorithmes à stratégie d'évolution permettant leur adaptation pour la résolution des problèmes d'optimisation combinatoire.

Le chapitre trois contient le développement que nous avons apporté à ces algorithmes pour permettre une utilisation efficace de ces derniers dans le cadre de la problématique retenue.

Dans le quatrième et dernier chapitre, nous illustrons les méthodes proposées au chapitre trois sur un ensemble de jeux de données afin de montrer l'efficacité et de vérifier les performances des approches proposées.

Enfin en conclusion, nous présentons les perspectives liées au travail exposé dans ce mémoire.

Chapitre I

Problèmes d'ordonnancement : Définitions et choix d'une problématique

I.1 Introduction

La nécessité d'évoluer dans un espace mondial, se traduit pour les entreprises par des exigences accrues, exigences sur la fixation et le respect des délais, et sur la qualité et la variété des produits. Or, chacune de ces nouvelles exigences entraîne au niveau de la production des contraintes importantes de plus en plus difficiles à appréhender avec les outils de gestion classiques. Multiplier les variantes, fabriquer en un temps plus court, limiter les investissements financiers, produire des produits compétitifs sur le plan de la qualité sont autant d'objectifs souvent contradictoires auxquels les entreprises devront faire face. L'évolution des entreprises passe par la mise en place d'une logique industrielle qui remet en cause certains fondements actuels, notamment les pratiques classiques d'organisation d'atelier. Nous nous intéresserons dans le cadre de cette thèse au problème d'ordonnancement d'atelier. En effet, il s'agit d'une problématique générale, commune à tout système de production, dès lors que l'on a des produits à réaliser sur des moyens de production.

Nous pouvons dans ce cas dire qu'il s'agit de déterminer l'ordre de passage des produits sur les machines de façon à respecter les contraintes techniques représentées par les gammes et à répondre à des objectifs choisis. Le problème d'ordonnancement se formalise comme suit : **quels choix effectuer afin de satisfaire au mieux un critère d'efficacité préalablement choisi ?** En effet, les problèmes d'ordonnancement se posent dans tous les domaines de l'économie, ils revêtent une importance particulière dans cet environnement dynamique dans lequel les objectifs de délais (fixation et respect de ces délais) ou de stock minimum constituent des préoccupations majeures.

De façon plus précise, **quelles sont les réponses théoriques qui sont données à ces problèmes ?** Généralement, on connaît de l'ordonnancement un cas particulier très simple, celui de l'ordonnancement de projet : il s'agit de déterminer les dates d'exécution au plus tard d'un ensemble de tâches définies à l'avance et dont on connaît la durée, afin de réaliser un projet dans un temps minimum. Si un certain nombre de problèmes industriels peuvent se poser sous la forme d'ordonnancement de projet (réalisation d'un chantier, de fabrications unitaires spécifiques, etc), la plupart n'entrent pas dans ce cadre. En effet, la problématique de l'ordonnancement d'atelier est dynamique : à tout moment de nouvelles commandes urgentes peuvent apparaître, des pannes peuvent survenir, ce qui remet en cause l'ordonnancement prévu. Enfin, la fonction économique à optimiser est beaucoup plus complexe : le temps minimisant la réalisation d'une commande peut être un critère, mais il en existe d'autres : minimisation des temps d'attente, minimisation des retards, équilibrage des charges de travail sur les machines, etc.

Dans ce chapitre nous définissons les notions générales concernant l'ordonnancement ainsi que des éléments de classification des problèmes traités. Ces éléments seront utilisés lors de la résolution du problème posé par le projet 4 : "méthodes de planification, optimisation et ordonnancement des systèmes complexes et incertains" du "GRAISyHM" (Groupement de Recherche en Automatisation Intégrée et Systèmes Hommes / Machines) dont nous donnerons une description détaillée au paragraphe I.5.1

I.2 Définitions

Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution, de manière à atteindre l'optimum d'un critère préalablement défini, tout en respectant les contraintes de la réalisation. Ainsi, les différentes données d'un problème d'ordonnancement sont les tâches, les contraintes, les ressources, et les critères retenus pour évaluer la solution [1].

I.2.1 Les opérations élémentaires (Les tâches)

Les tâches sont le dénominateur commun des problèmes d'ordonnancement ; leur définition ne peut se faire avec exactitude a priori, elles dépendent fortement du problème à traiter et de la finesse du découpage.

Si l'ensemble des tâches à exécuter au cours du temps est donné a priori c'est à dire si on connaît à l'avance quelles seront les tâches à exécuter et à partir de quelles dates (dites dates de disponibilité) le problème d'ordonnancement est dit *statique*. Par contre quand l'ensemble des tâches à exécuter évolue avec le temps, le problème est dit *dynamique*.

Nous pouvons distinguer selon les modes d'exécution trois types de tâches :

- **Tâches morcelables** : Dans certains cas, une tâche peut être exécutée par morceaux (dans un système multiprogrammé). On dit alors que la tâche est morcelable ou que la préemption est possible. Souvent la préemption rend les problèmes plus faciles à résoudre car on peut faire appel aux mathématiques du continu, comme la programmation linéaire.

- **Tâches non morcelables** : Contrairement au premier cas, la tâche ne peut être interrompue avant qu'elle soit complètement terminée, comme par exemple dans le cas des problèmes d'ateliers où la tâche est considérée comme un tout indivisible.

Une opération élémentaire est notée i, elle appartient à un produit désigné par OF_j, ainsi, l'opération i du produit OF_j est noter $\mathrm{O}_{i,OFj}$ et est caractériseé par :

$t_{i,OFj,Mk}$: date de début d'exécution de l'opération $O_{i,OFj}$ sur la machine M_k (représente une caractéristique d'époque),

$r_{i,OFj,Mk}$: date de disponibilité ou date de début au plus tôt de l'opération $O_{i,OFj}$ sur la machine M_k,

$p_{i,OFj,Mk}$: le temps d'exécution de l'opération $O_{i,OFj}$ sur la machine M_k (représente une caractéristique de durée),

$c_{i,OFj,Mk} = t_{i,OFj,Mk} + p_{i,OFj,Mk}$: la date de fin d'exécution de l'opération $O_{i,OFj}$ sur la machine M_k (représente une caractéristique de moyens),

$d_{i,OFj,Mk}$: Date de fin au plus tard de l'opération $O_{i,OFj}$ sur la machine M_k (deadline).

Une condition nécessaire de réalisabilité d'un ordonnancement sera par conséquent :

$$\forall i \in I \qquad r_{i,OFj,Mk} \leqslant t_{i,OFj,Mk} \leqslant c_{i,OFj,Mk} \leqslant d_{i,OFj,Mk} \tag{I.1}$$

où I est l'ensemble de toutes les tâches à ordonnancer.

I.2.2 Les contraintes

Les problèmes d'ordonnancement sont rendus difficiles à cause des contraintes à respecter nous pouvons distinguer plusieurs types d'entre elles [1][2]

- **Les contraintes potentielles** : elles lient les tâches entre elles c'est à dire que telle opération ne peut être exécutée avant telle autre, elles peuvent être temporelles ou de succession (temporelles quand il faut attendre la fin d'une tâche avant de lancer la suivante ; de succession par exemple les gros oeuvres suivent les fondations etc...)

- **Les contraintes disjonctives** : ce type de contraintes est lié aux ressources utilisées par les tâches, il apparaîtra lorsque deux tâches, utilisant la même machine, ne pourront pas s'exécuter simultanément.

- **Les contraintes cumulatives** : elles sont aussi liées aux ressources, elles apparaîssent par exemple lorsque trois processus sont disponibles pour l'exécution de

quatre tâches : on ne pourra exécuter plus de trois de ces tâches en même temps sans que l'on puisse savoir à l'avance laquelle sera retardée.

I.2.3 Les ressources

Essentiellement on distingue deux types de ressources pour l'exécution d'une tâche [1][3].

Les ressources renouvelables sont des ressources réutilisables, après avoir été allouées à une tâche elles redeviennent disponibles à la fin de cette dernière. Les ressources renouvelables usuelles sont principalement les machines, le personnel, etc.

Les ressources non renouvelables comme leur nom l'indique, ne sont plus disponibles après la fin de la tâche, elles sont en fait épuisées pour la réalisation de cette opération, par exemple les matières premières. Dans le cadre de cette thèse, nous considérons principalement les machines comme ressources de base nécessaires à la réalisation de la production. Nous ne ferons pas de distinction entre machine, poste de travail, poste de charge, ni entre les différents types de machines.

Une machine est définie comme un type de ressource qui n'est caractérisé que par son horaire de travail.

I.2.4 Les critères

Les facteurs importants dans l'évolution d'un problème d'ordonnancement sont l'utilisation efficace des ressources, et le respect du plus grand nombre possible de contraintes. Il convient donc de préciser les objectifs devant intervenir dans le choix d'un ordonnancement. La détermination de ces objectifs s'avère souvent extrêmement délicate. En effet, on se trouve généralement en présence d'un ensemble d'objectifs plus ou moins contradictoires et dont l'importance relative est difficile à apprécier. Parmi ces critères nous pouvons citer :

Le coût et la durée de la réalisation,

le respect du délai d'exécution,

la quantité de moyens nécessaires,

la quantité de travail en attente,

l'équilibrage dans le temps de la quantité de moyens utilisés,

le temps d'immobilisation des moyens,

la sécurité et la souplesse dans l'exécution (sensibilité aux aléas).

Ces objectifs peuvent correspondre à des exigences quantitatives (valeurs à atteindre ou à ne pas dépasser) et se présentent sous forme de contraintes à respecter, ou bien d'exigences qualitatives s'exprimant sous forme de critères à optimiser. Parmi les critères les plus utilisés pour évaluer la qualité d'un ordonnancement obtenu on distingue [1]:

Durée totale (Makespan) : la durée totale de l'ordonnancement notée $Cmax$ est égale à la date d'achèvement de la tâche la plus tardive : $C\max = \max_{i \in I} (C_i)$. C'est le critère le plus utilisé ; de plus, il est vraisemblable qu'en minimisant la durée totale, on ait une utilisation efficace des ressources.

Respect des dates au plus tard : Dans beaucoup de problèmes réels, il faut respecter au mieux les délais, en occurrence les dates de fin au plus tard $d_{i,OFj}$, ceci se fait en minimisant soit le plus grand retard soit la somme des retards.

Minimisation d'un coût : ce type de critère peut s'exprimer sous des formes très variées par exemple la minimisation des stocks d'encours.

Par ailleurs, on peut remarquer que certains objectifs ont un caractère externe (durée totale), car ils portent sur tous les produits à réaliser considérés comme un tout, alors que d'autres objectifs ont un caractère interne (souplesse, équilibrage dans le temps des moyens), parce qu'ils portent sur chaque produit pris comme un ensemble d'opérations élémentaires indépendantes.

Dans ce travail, nous nous intéressons essentiellement à des objectifs internes exprimés sous forme de critères à optimiser. Ces critères se basent, en fait, sur les caractéristiques externes d'un produit (époque, durée, moyens). Le critère que nous traitons dans le cadre de cette thèse est principalement la minimisation de la durée totale. Des précisions sur ce point sont apportées dans les paragraphes suivants.

I.3 Pourquoi s'intéresser aux problèmes d'ordonnancement ?

Les problèmes d'ordonnancement apparaissent dans tous les domaines de l'économie : l'informatique (tâches : jobs, ressources : processeur, mémoire, etc...), l'industrie (problèmes d'ateliers, gestion de production) [1].

Le problème d'ordonnancement des ateliers constitue sûrement pour les entreprises une des difficultés importantes de leur système de gestion et de conduite. En effet, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers, ainsi que les perturbations, aussi bien internes (panne de machine, absence, etc.) qu'externes (perturbation de la demande, etc.), qui viennent les modifier.

Une enquête menée aux Etats-Unis dans le secteur des industries mécaniques à fait apparaître un résultat surprenant : le temps réel de travail sur les machines représente moins de 5% du temps total de passage dans les ateliers, autrement dit plus de 95% du temps sont consacrés à des inspections, au contrôle, à la manutention et surtout à l'attente (près de 80%) [4]. Ces temps d'attente sont évidemment directement liés aux procédures d'ordonnancement. En dehors de quelques rares configurations spécifiques (machines identiques en parallèle), les problèmes d'ordonnancement d'ateliers n'auront pas de solution optimale reposant sur des algorithmes utilisables. Pour tout ces problèmes non résolus, on pourrait penser qu'une solution possible consisterait à décrire toutes les solutions et à sélectionner la meilleure au sens d'un critère fixé à l'avance : cette voie de recherche est celle des explorations arborescentes dont le principe est une énumération de solutions possibles selon des procédures permettant d'aller au plus vite vers la solution optimale. L'inconvénient de cette dernière est qu'elle ne donne aucune garantie, en début de calcul, et que cette recherche ne sera pas possible sans une analyse de toutes les solutions du problème. On se heurte ici à un problème combinatoire. Pour bien se rendre compte de la complexité des problèmes d'ordonnancement, prenons l'exemple suivant : supposons que l'on a 10 opérations différentes à ordonnancer sur une machine

unique, la combinatoire sous-jacente est donc de 10! (Un choix parmi 10 pour la première opération, puis un choix parmi 9 pour la suivante, etc.). Si l'on dispose d'un calculateur capable de faire un million de calcul par seconde, il nous faut 3,6 secondes par contre s'il faut ordonnancer non pas 10 opérations mais plutôt 20 toujours sur une seule machine, il nous faut plus de 750 siècles !. L'explosion combinatoire est donc l'écueil essentiel pour la résolution des problèmes d'ordonnancement, le nombre des solutions réalisables, croît exponentiellement avec le nombre d'opérations. La banalisation des moyens informatiques a permis de résoudre plus facilement et plus rapidement un certain nombre de problèmes, hélas, on cherche toujours à résoudre des problèmes de plus en plus combinatoires en Recherche Opérationnelle (emploi, du temps, tournées, ordonnancement). On s'est vite aperçu que, malgré la rapidité des ordinateurs, il y avait des limites qu'il fallait prendre en compte.

I.3.1 Complexité des algorithmes

Une première étape a été d'évaluer en fonction de la taille des données le nombre d'opérations des algorithmes. En effet, la connaissance du temps d'exécution et la place mémoire prise par l'algorithme pour résoudre un problème permet de calculer cette complexité.

Les complexités en temps et en mémoire d'un algorithme s'expriment en fonction du nombre N de données appelé aussi taille du problème, le nombre de produits, de machines, de tâches ou encore une combinaison de ces quantités en font partie.

Le calcul de la complexité en mémoire consiste à calculer le nombre de variables ou de cases mémoires utilisées.

La théorie de la complexité algorithmique propose, en se plaçant dans le cas le plus défavorable, de majorer le nombre d'opérations nécessaires pour une fonction de la taille du problème posé, noté N. Un algorithme sera de complexité $f(N)$, noté $Of((N))$, s'il existe une constante C et un entier N_0 tel que le nombre d'instructions élémentaires $I(N)$

(qui représente aussi le temps de calcul) vérifie l'équation suivante :

$$I(N) \leq Cf(N) \qquad pour\ tout\ N \geq N_0 \tag{I.2}$$

Si $f(N)$ est un polynôme, alors l'algorithme est dit polynomial et il est d'autant meilleur pour N grand que le degré du ploynôme est faible [5][6].

I.3.2 Complexité de problèmes

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

Les problèmes les plus difficiles :

ce sont les problèmes indécidables pour lesquels il n'existe pas d'algorithme pour leur résolution.

Les problèmes de classe P :

un problème de décision est dit facile ou appartenir à la classe P s'il existe au moins un algorithme polynomial pour le résoudre.

Les problèmes de la classe NP :

dans la classe NP, les problèmes sont dits NP-difficiles, ces derniers sont liés par la propriété suivante : si un seul problème NP-difficile est polynomial, alors la plus part des problèmes difficiles sont également polynomiaux. Aussi les chercheurs pensent-ils que si un problème est NP-difficille il n'existe pas d'algorithme polynomial le résolvant ; c'est la conjecture fondamentale de la théorie de la complexité [1].

Généralement lors de l'étude d'un problème d'ordonnancement, on commence par chercher à classer le problème. Si on parvient à montrer qu'il est polynomial, le problème sera résolu, par contre s'il s'avère plutôt NP-difficile, il restera à mettre au point des méthodes de résolution adaptées.

I.4 Comment résoudre les problèmes d'ordonnancement ?

Comme il a été établi, la plupart des problèmes d'ordonnancement sont NP-difficiles [7]. Pour les résoudre, il existe principalement deux méthodes : les méthodes exactes si la nature et la taille du problème le permettent et les méthodes approchées dans le cas ou une solution exacte n'est pas admissible. Vu la taille des problèmes à traiter actuellement, nous ne pouvons espérer trouver une solution exacte, d'où le recours aux méthodes approchées mettant en oeuvre des règles de priorité ou des heuristiques proposant des solutions acceptables en un temps raisonnable [8][9].

I.4.1 Méthodes exactes

On peut définir une méthode exacte comme une méthode qui fournit une solution optimale pour un problème d'optimisation. L'utilisation de ces méthodes s'avère particulièrement intéressante dans le cas des problèmes de petite taille. Nous pouvons citer trois approches particulièrement célèbres

La méthode par séparation et évaluation

Plus connue sous le nom de branch and bound, c'est une méthode basée sur une technique implicite énumérative. En effet, elle permet de trouver une solution optimale en construisant un arbre de recherche en examinant systématiquement les sous chemins qui sont susceptibles d'aboutir à une solution réalisable et exclurant les autres sous-arbres

de la recherche [10][11].

L'algorithme de Johnson

L'algorithme de Johnson est fondé sur le calcul itératif du minimum des durées des tâches associées aux travaux non encore placés. cet algorithme donne une solution optimale pour le problème à deux machines de type flow shop. Nous présentons ici l'algorithme de johnson 2 [1]

Début

$U = \oslash$; $V = \oslash$

Pour ($i = 1$ jusqu'à n) **faire** *(n est le nombre total d'opérations)*

Si $(a_i < b_i)$ **alors**

$U = U + \{i\}$

Sinon

$V = V + \{i\}$

Fin Pour

LU = *liste ordonnée par* P_i^1 *croissants au sens large des élements de* U

LV = *liste ordonnée par* p_i^2 *croissants au sens large des élements de* V

(*où* P_i^k *est le temps d'exécution de l'opération* i *sur la machine* k)

$L = (LU).(LV)$ (*concaténation des 2 lilstes* LU *et* LV)

(L représente la séquence optimale).

Fin.

La programmation dynamique

Elle est due à Bellman qui en a établi les principes en 1957. Le domaine d'application de cette méthode est assez général ; elle a été et reste particulièrement utilisée dans la conduite de stocks, la planification et d'une manière plus générale pour beaucoup de problèmes d'optimisation combinatoire. La programmation dynamique est fondée sur une approche récursive. Etant donné un problème initial qualifié d'ordre N, on lui substitue

la résolution d'un grand nombre de problèmes d'ordres $N-1$, $N-2$,..., 1, de telle sorte que la solution d'un problème d'ordre N s'exprime simplement en fonction des solutions de problèmes d'ordre $N-1$. Les problèmes résolus par la programmation dynamique se ramènent au choix d'une suite de décisions séquentielles [10][1][12].

I.4.2 Méthodes approchées :

Les problèmes d'ordonnancement étant en général NP-difficiles, on cherche à les simplifier pour réduire le volume des calculs. Bien entendu, cela se fait au prix d'une dégradation de la qualité de la solution. Il existe plusieurs familles de méthodes qui permettent d'approcher efficacement la solution optimale, on distingue :

Les méthodes par construction :

Elles consistent à complèter progressivement étape par étape une solution partielle. On distingue les méthodes par construction ordre de fabrication par ordre de fabrication (à chaque itération on place toutes les opérations d'un travail) ou opération par opération (à chaque itération on place une seule opération) ou machine par machine (on ordonnance les opérations affectées à une machine puis on passe à la machine suivante). Le choix des tâches candidates à un placement peut être aléatoire ou fixé par des règles de priorité [2].

Les méthodes par décomposition

Elles consistent à décomposer le problème en plusieurs sous-problèmes, cette décomposition peut être réalisée en plusieurs niveaux elle s'appelle alors décomposition hiérarchique, ou consister à découper le problème en sous ateliers avec le moins possible de déplacements de produit entre les sous-ateliers, elle est dite alors décomposition spatiale, ou encore cette décomposition consiste à découper le problème en sous-problèmes par la création de tâches que l'on ordonnance successivement en progressant le long de l'axe du temps, elle est dite alors décomposition temporelle [13][2].

Les méthodes par voisinages

En partant d'une solution réalisable (dont on dispose initialement) on passe à une autre solution réalisable voisine de façon, aléatoire ou non et meilleure si possible par rapport au critère utilisé. Cette méthode consiste à introduire un graphe de voisinage. Les sommets de ce graphe sont les solutions du problème ; si (x, y) appartient au graphe, on dit que y est un voisin de x. Ce graphe possède les deux propriétés suivantes :

- le calcul de y à partir de x demande un temps très court,
- chaque sommet x a un nombre polynomial de voisins y.

L'algorithme de voisinage se présente comme suit [1]

1. *Construire une solution initiale x*
2. *Déterminer parmi les solutions voisines y de x celle de $f(y)$ minimale.*
3. **Si** $f(y) < f(x)$ ***alors***

   remplacer x par y et aller en 2

   Sinon

   retenir x comme meilleure solution.

Il existe d'autres algorithmes utilisant le principe du voisinage, nous distinguons :

Le recuit simulé (simulated annealing) permet de déterminer les minima globaux de la fonction à optimiser. Il a été proposé en 1982 par Kirkpatrick [14] et est inspiré de la technologie expérimentale du recuit utilisée par les métallurgistes pour obtenir un état solide bien ordonné, d'énergie minimale (en évitant les structures métastables, caractéristiques des minima locaux d'énergie). L'algorithme est inspiré de celui de Métropolis [1][16]. Partant d'un état initial, on lui fait subir une perturbation, c'est à dire une modification d'amplitude limitée (dans les problèmes d'ordonnancement ceci consiste à

translater un composant, ou a échanger deux composants). Si cette perturbation a pour effet d'améliorer le critère, le nouvel état est retenu. Si, au contraire, elle provoque une détérioration ΔE du critère, le nouvel état est retenu avec une probabilité $p = exp(-\frac{\Delta E}{T})$, où T est un paramètre qui diminue lorsque le nombre d'itérations augmente. On recommence le processus soit avec le nouvel état, s'il est retenu, soit avec l'ancien état dans le cas contraire. Ce processus est arrête lorsque T devient inférieur à une valeure donnée notée ε, ou lorsqu'aucune amélioration du critère n'a été observée au cours des L dernières itérations. [15][17]. L'algorithme du recuit simulé se présente comme suit :

1. *Initialisation des différents paramètres de l'algorithme* (T, L, ε) ;
2. *Modification élémentaire (variation d'énergie* ΔE*)* ;
3. *Application de la règle d'acceptation de Metropolis* ;

   Si $\Delta E \leq 0$ ***alors*** *modification acceptée*

   Si $\Delta E > 0$ ***alors*** *modification acceptée avec la probabilité* $exp(-\frac{\Delta E}{T})$

   Si $(T > \varepsilon)$ ***alors***

   Diminution lente de T *et aller à* 2

   Si $((T \leq \varepsilon)$ *ou (système non figé))* ***alors***

   L'état actuel est la solution recherchée.

4. **Fin.**

La recherche tabou (tabu search) La technique tabou est une méthode itérative générale d'optimisation combinatoire qui à été introduite par Glover, elle apparait très performante sur un nombre considérable de problèmes d'optimisation combinatoire, en particulier les problèmes d'ordonnancement [18]. Elle se comporte comme toute méthode de descente, elle améliore à chaque étape la valeur de la fonction objectif. Lorsqu'on atteint par contre un optimum local, le deuxième principe de cette méthode entre en jeu ; en interdisant de revenir sur le même chemin. Ce principe consiste à garder en

mémoire les dernières solutions visitées et à interdire le retour vers celles-ci [19][20][21]. Nous allons décrire l'algorithme tabou qui permet de résoudre le problème suivant :

Soit X un ensemble de solutions réalisables, soit F une fonction permettant d'évaluer chaque solution réalisable. Il s'agit de determiner une solution $s^* \in X$ qui minimise F, c'est à dire telle que $F(s^*) = \min_{s \in X} F(s)$

Les différents éléments de cet algorithme sont :

X^a : espace des solutions admissibles,

F : la fonction objectif définie sur cet espace,

$N(s)$: le voisinage d'une solution $s \in X^a$,

$| T |$: le nombre de solutions taboues,

A : la fonction d'aspiration,

$nbmax$: le nombre maximun d'itérations entre deux améliorations de s^*,

N' : un sous-ensemble de $N(s)$,

F_{inf} : la borne inférieure de la fonction objectif.

L'algorithme se présente comme suit :

1. *Initialisation des différents paramètres de l'algorithme*

   Choisir une solution admissible initiale X^a

   $nbiter = 0$ *(itération courante)*

   $T = \oslash$ *(*T *étant la liste Taboue)*

   Si $s \in X$ ***alors***

   $s^* = s$

   $F^* = F(s)$

   $bon = 0$ *(itération ayant donné la meilleure solution)*

   Sinon

   $F^* = \infty$

   Fin sinon

2. *Processus itératif*

Tant que $(F(s) > F_{\text{inf}}$ et $((nbiter - bon) < nb\max))$ **faire**

nbiter $= nbiter + 1$

générer un ensemble $N' \subseteq N(s)$ *de solutions voisines de* s

choisir la meilleure solution $s' \in N'$ *telle que* $F(s') \leq A(F(s))$ *où* $s' \notin T$

mettre à jour la fonction d'aspiration A *et la liste* T *des solutions taboues*

$s = s'$

Si $F(s) < F^*$ ***alors***

$s^* = s$

bon $= nbiter$

Fin si

Fin tant que

3. **Fin.**

Les méthodes liées à l'intelligence artificielle

Sous l'expression "Intelligence Artificielle (IA)" on désigne l'ensemble des travaux centrés sur la prise en charge par des machines logicielles et/ou matérielles d'activités dont on estime qu'elles nécessitent de l'intelligence de la part des humains qui les assurent ordinairement [22]. Les techniques de l'IA ont été efficacement utilisées pour résoudre les problèmes d'ordonnancement [23][25]. Les systèmes de l'IA doivent constituer des outils adaptés permettant d'exprimer les stratégies du chef d'atelier et celles des bureaux d'études, et de modifier automatiquement celles-ci en fonction des objectifs et de l'état de l'atelier. Ces méthodes utilisent des techniques de représentation des connaissances, de raisonnement à base de règles, d'apprentissage ou d'analyse sous contraintes. Parmi ces méthodes basées sur l'IA nous pouvons citer :

SOJA : Système d'Ordonnancement Journalier d'Atelier [24] qui est un système d'ordonnancement qui procède en deux étapes :

Sélection des opérations à exécuter en fonction de critères fixés au début.

Ordonnancement des opérations sélectionnées et satisfaisant les contraintes, à l'aide d'un paragraphe potentiel tâche.

OPAL [25][26] qui est un système expert hybride dans le sens où il s'agit d'un compromis entre approches classiques et systèmes experts. On intègre dans ce système à la fois les méthodes théoriques d'analyse des contraintes temporelles (contraintes de date limite par exemple) et les connaissances beaucoup moins structurées de l'ingénieur et du chef d'atelier (connaissances empiriques) au lieu d'une pure base de système expert classique.

D'autres systèmes experts sont présentés dans [2].

I.4.3 Programmation avec contraintes

Depuis peu, une nouvelle technique permet de résoudre, entre autres, les problèmes d'ordonnancement [27][28][29], il s'agit de la programmation avec contraintes. Si les principes de cette méthode sont connus depuis longtemps, leur utilisation effective pour la résolution de problèmes fortement combinatoires s'est généralisée depuis l'apparition des langages de programmation avec contraintes à part entière : Charme de Bull en 1989, Prolog III en 1990. Les problèmes d'ordonnancement s'expriment naturellement en termes de contraintes : contraintes temporelles, contraintes de précédences, de successions et d'exclusion mutuelle , et l'aspect déclaratif sous-jacent à ce type de programmation est particulièrement séduisant. Il apparaît cependant à l'usage que l'ordre d'écriture des contraintes influe grandement sur le temps de résolution. Dans le chapitre III, nous allons montrer comment nous avons utilisé cette approche avec la collaboration de l'équipe Génie Industriel et Logiciel "EGIL" du laboratoire d'Automatique et de Mécanique Industrielles et Humaines "LAMIH" de l'université de Valenciennes dans le cadre du projet 4 de "GRAISyHM" (voir paragraphe I.5.1), pour mettre au point un système hybride : Programmation logique avec contraintes et algorithmes évolutionnistes pour résoudre le problème d'ordonnancement du type job-shop flexible.

I.4.4 Algorithmes évolutionnistes

Ils sont dits aussi algorithmes à stratégie d'évolution ou encore sous leur forme classique algorithmes génétiques. Un algorithme évolutionniste modélise le processus d'apprentissage collectif d'une population d'individus pour s'adapter à un environnement. Chaque individu va non seulement représenter un point dans l'espace des solutions du problème, mais il va aussi contenir la connaissance actuelle de l'individu par rapport à l'environnement. Les algorithmes génétiques ont été efficacement utilisés pour résoudre des problèmes combinatoires tels que les problèmes d'ordonnancement [30][31][32][33][34][35][36][37]. Dans notre travail, nous utiliserons ces algorithmes pour résoudre conjointement les problèmes d'ordonnancement et d'affectation dans les ateliers du type job-shop flexible en minimisant le temps total d'ordonnancement ($Cmax$). Dans le chapitre II nous présenterons le principe des algorithmes génétiques. les modalités de leur utilisation pour résoudre notre problème sont présentées dans le chapitre III.

I.4.5 Classification des problèmes d'ordonnancement

Avant de faire un choix sur la problématique, on doit d'abord classer les différents types de problèmes d'ordonnancement, cette classification se fait généralement en fonction du type de machines utilisées ou selon l'organisation de l'atelier.

Problème à machines parallèles

L'atelier possède des machines parallèles, ces dernières sont des machines interchangeables, c'est à dire des machines qui peuvent effectuer le même ensemble d'opérations, Bien entendu, le temps nécessaire pour exécuter une tâche n'est pas nécessairement le même sur toutes les machines. On peut distinguer trois types de parallélismes [1].

Machines identiques : dans ce cas la durée opératoire d'une tâche est la même sur toutes les machines,

Machines uniformes : les durées opératoires, lorsque l'on passe d'une machine à

une autre, sont dans des proportions qui ne dépendent pas de l'opération à exécuter, par exemple, si une opération est exécutée deux fois plus rapidement sur la machine $M1$ que sur la machine $M2$, cela reste vrai pour toute autre opération exécutable par ces deux machines parallèles, en d'autre terme $P_{i,OFj,Mk}$ est inversement proportionnel à la vitesse v_k de la machine k,

Machines non reliées : Contrairement au cas précédent, il n'y a plus de proportionnalité indépendante des opérations entre les temps de fabrication. Ce problème est classé parmi les plus difficiles puisque le problème est déjà NP-difficile dans le cas de deux machines identiques. le problème d'ordonnancement d'atelier traité dans le cadre de cette thèse étant du type job-shop flexible, ceci suppose avoir plusieurs machines pouvons traiter la même opération, ce qui nous conduit donc à ne considèrer dans notre travail que les machines non reliées.

Problème de type flow-shop

Un flow-shop ou plutôt fabrication en ligne, est un ensemble de machines visitées dans le même ordre par tous les produits fabriqués par cet ensemble. De ce fait, les machines sont habituellement rangées en ligne, dans l'ordre dans lequel elles sont visitées. Les problèmes qui se posent pour ce type de systèmes de production sont variés. On peut vouloir, par exemple, réduire le temps de séjour des produits dans le système.

Problème de type Job-shop

Un job-shop est un atelier dont les ressources sont visitées dans un ordre qui dépend du type de produit fabriqué. Bien que de nombreux travaux portent sur ce sujet, la plupart se concentrent sur le critère makespan, c'est à dire sur la minimisation de la différence entre l'instant où le dernier produit de l'ensemble à fabriquer sort du système et l'instant ou le premier produit entre dans le dit système. Plusieurs tentatives pour résoudre ce problème qu'elles soient exactes ou approchées ont malheureusement échoué, jusqu'en 1989, Carlier et Pinson [11] ont réussi à trouver une solution au problème à une machine

en utilisant la méthode de branch and bound avec des conditions qui limitent l'espace de recherche en utilisant une sélection immédiate de la branche à explorer.

I.5 Choix d'une problématique

Dans le contexte actuel, les entreprises doivent assurer une production en quantité, et en qualité à un coût minimum et dans des délais raisonnables, alors que la demande est de plus en plus variée, exigeante et souvent incertaine. Pour l'entreprise industrielle, les enjeux de la gestion de production sont vitaux. De nombreux travaux ont portés sur la planification prévisionnelle de la production, conduisant aux approches de type MRP (Material Requirement Planning). Ces approches permettent, de façon plus au moins précise, de définir un programme de fabrication qui assure un certain équilibre entre les charges de travail planifiées et la capacité des ressources disponibles. Mais il apparaît au-delà de cette planification, qu'il faut coordonner le travail dans les ateliers qui doivent réaliser ce plan de production, afin de respecter les délais annoncés et d'utiliser au mieux les ressources disponibles, malgré les multiples aléas qui surviennent. C'est ce problème qui est pris en charge par l'ordonnancement. En effet, les réponses théoriques apportées au cas général sont limitées. La majeure partie des systèmes industriels relèvent de configuration du job-shop.

Un très grand nombre de logiciels utilisant de façon plus ou moins simplifiée, certaines méthodes basées sur des heuristiques ou utilisant l'intelligence artificielle sont implantés en milieu industriel. Malgré les solutions données par ces derniers, le niveau de satisfaction des contraintes laisse à désirer dans beaucoup de cas réels à cause de la forte spécificité des systèmes de production pour lesquels ces systèmes ont été réalisés. Un système d'aide à l'ordonnancement doit s'adapter aisément :

- à la diversité des configurations et des contraintes de production des ateliers,

- à des modifications de la production et/ou des différents critères à privilégier,

- à des modifications de l'unité de production.

I.5.1 Le projet 4 GRAISyHM

Le travail que nous avons effectué dans le cadre de cette thèse fait partie du projet 4 intitulé "méthodes de planification, optimisation et ordonnancement des systèmes complexes et incertains"d'un plus grand projet nommé GRAISyHM, "Groupement de Recherche en Automatisation Intégrée et Systèmes Hommes / Machines". Ce projet est soutenu par le conseil régional du Nord Pas de Calais, et implique plusieurs laboratoires de la région. L'objectif de ce projet est de concevoir un système hybride d'aide à l'ordonnancement constituant un environnement intégré permettant d'élaborer un ensemble d'ordonnancement admissibles, et de sélectionner parmi cet ensemble ceux qui correspondent aux souhaits du responsable de l'atelier selon divers critères (qualité des produits réalisés, minimisation des temps de préparation, occupation suffisante des machines, ...).

Le système envisagé est caractérisé par :

- la coopération de nouvelles méthodes de résolution utilisées pour résoudre des problèmes d'optimisation et d'apprentissage : algorithmes génétiques, réseaux de neurones, programmation sous contraintes et logique floue,

- l'utilisation de plusieurs modèles pour la description des connaissances, permettant de prendre en compte les connaissances beaucoup moins structurées de l'ingénieur en simulation ou du chef d'atelier : la logique floue, l'approche multicritère, les systèmes à base de connaissances.

L'architecture fonctionnelle du système proposé peut être représentée par le schéma de la figure ci dessous.

système est structuré en trois niveaux :

le niveau "métiers" correspondant aux divers intervenants : les experts qui participent à l'élaboration, la création et la gestion des bases de connaissance, au choix des

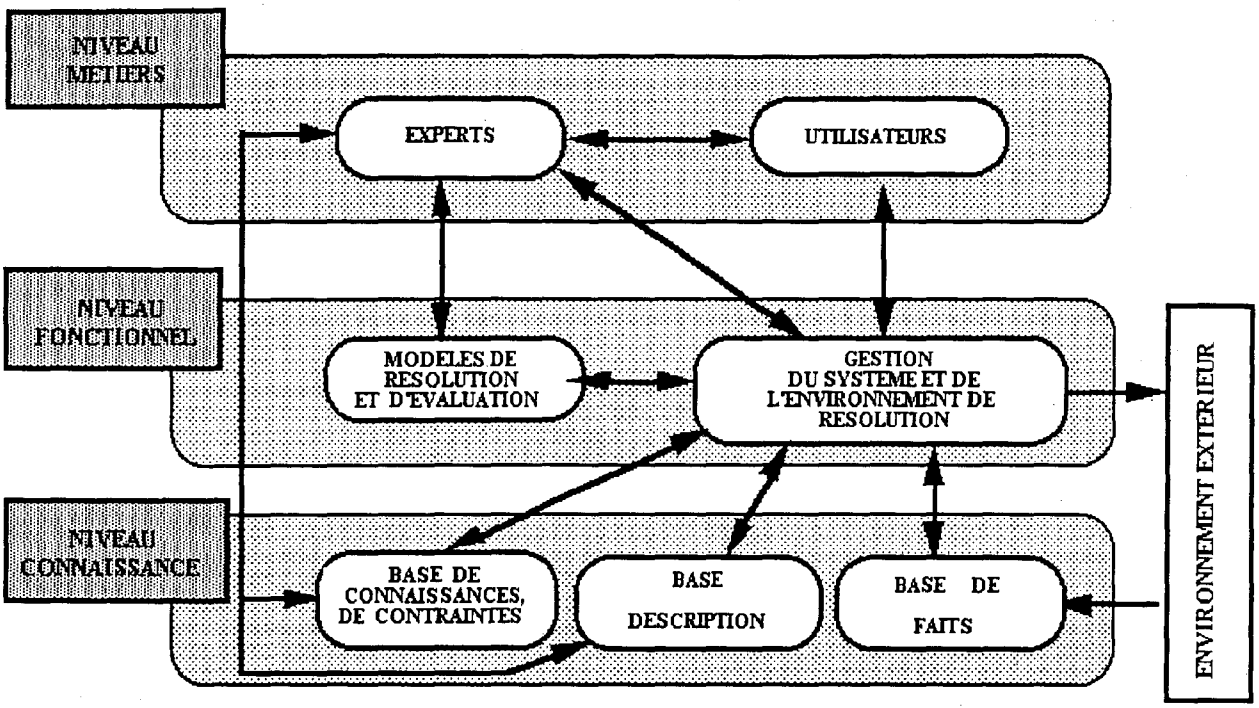

Figure I-1: Figure : I.1 Système d'aide à l'ordonnancement

modèles, des systèmes de gestion de la connaissance, et les utilisateurs du système d'aide à l'ordonnancement.

le niveau "fonctionnel "constitué par :

- un module "Modèles de résolution et d'évaluation" : regroupant l'ensemble des méthodes de résolution et d'exploitation des connaissances et d'évaluation des solutions.

- un module "gestion du système et de l'environnement de résolution" : il contrôle l'activation des mécanismes précédents, les accès et l'exploitation des différentes bases du niveau connaissance.

le niveau "connaissance" comportant :

- une base de "règles et de contraintes" spécifiques à l'application exprimant les modèles des connaissances "non procédurales" sur le domaine (ordonnancement) et le système applicatif,

- une base "description" : c'est le modèle descriptif représentant les différentes données relatives aux éléments caractéristiques du domaine et le système applicatif,

- une base de faits : c'est une base de données dynamique qui permet d'enregistrer les événements significatifs de l'évolution du processus d'élaboration de la décision (base de faits "décision") et du système applicatif (base de faits "applications").

Notre intervention dans le cadre de ce projet se situe au niveau fonctionnel et plus précisément dans le module "modèles de résolution et d'évaluation". En effet, le travail que nous avons effectué dans le cadre de notre thèse consistait à trouver un modèle de résolution et d'évaluation, utilisant les algorithmes à stratégie d'évolution pour résoudre conjointement les problèmes d'ordonnancement et d'affectation dans les ateliers du type job-shop flexible.

En effet, le job-shop flexible est un problème général car il représente la configuration réelle de la plus part des problèmes existant dans l'industrie. Ainsi dans les chapitres 3 et 4 nous détaillerons l'utilisation des algorithmes évolutionnistes, et nous expliquerons les modèles hybrides ; les algorithmes à stratégie d'évolution et la programmation logique avec contraintes, utilisés dans le cadre de ce travail.

I.6 Conclusion

Ce rapide survol de la théorie de l'ordonnancement montre bien les limites des approches formalisées dans un domaine particulièrement combinatoire.

D'un point de vue très général, on peut dire que l'amélioration d'un système industriel vise deux objectifs essentiels:

- rapprocher le mieux possible les produits des besoins de la clientèle,

- améliorer la productivité.

Le but étant toujours d'optimiser les performances des systèmes d'ordonnancement ce qui permet aux entreprises, vu le contexte économique d'être toujours compétitives.

En effet, le but du projet GRAISyHM est d'aider les entreprises de la région Nord-Pas-de Calais à trouver des solutions à leurs problèmes de gestion. La collaboration entre divers laboratoires permet de mettre au point des nouvelles techniques d'aide à l'ordonnancement, vu que les méthodes déjà existantes, qu'elles soient exactes ou approchées ont montré leurs limites dans la résolution des problèmes réels. La problématique retenue pour cette étude est de trouver une méthode basée sur les algorithmes évolutionnistes pour optimiser le temps total.

Chapitre II

Algorithmes génétiques

II.1 Introduction

Qu'essaie-t-on d'accomplir lorsqu'on optimise ?

D'après Beighter et al (1979) "Le désir humain de perfection trouve son expression dans la théorie de l'optimisation. Elle étudie comment décrire et atteindre ce qui est meilleur, une fois que l'on connaît comment mesurer et modifier ce qui est bon et mauvais..." [38].

Optimiser est-ce chercher le parfait ou plutôt améliorer l'existant ?

La vie courante est pleine de défits qui nous incitent à améliorer nos performances, il est beaucoup plus réaliste d'essayer d'être mieux que les autres que d'être parfait. Nous pouvons dire que l'objectif le plus important de l'optimisation est l'amélioration. La priorité est d'atteindre rapidement une performance de niveau satisfaisant. La recherche de solutions dans un espace complexe implique souvent un compromis entre deux objectifs apparemment contradictoires ; l'exploitation des meilleures solutions disponibles à un moment donné et une exploration robuste de l'espace de solutions possibles.

Dans la littérature on distingue principalement trois méthodes d'optimisation :

- Les méthodes fondées sur le calcul s'appliquant localement ; les extremums qu'elles atteignent sont optimaux dans un voisinage du point de départ, telles que les méth-

odes de type descente "hill-climbing". Ces derniers se basent uniquement sur l'exploitation et souffrent, par conséquent, de la non-globalité de l'optimum trouvé, Ceci est visible lors de la recherche de l'extremum dans un espace qui contient deux collines. Il est évident que débuter les procédures de recherche d'extremum dans le voisinage de la plus petite colline nous conduira à rater la colline la plus haute. De plus, ces méthodes dépendent de l'existence de dérivées alors que le monde réel à explorer est envahi de discontinuités, d'espaces de recherches bruités et multimodaux, ce qui le rend bien moins adapté au calcul.

- Les méthodes énumératives consistent à regarder les valeurs de la fonction à optimiser en chaque point de l'espace, un par un. Ceci étant possible si l'espace de recherche est petit et le nombre de possibilités est très limité. Malheureusement dans la pratique, beaucoup d'espaces de recherche sont trop grands pour que l'on puisse explorer toutes les positions une par une en ayant une chance d'obtenir une information utilisable.

- Les méthodes aléatoires explorent et mémorisent le meilleur élément mais elles sont à rejeter à cause de leur lenteur excessive, de leur manque d'efficacité et de leur ignorance aveugle.

La recherche de nouvelles méthodes d'exploration n'a cesser d'évoluer. En 1975 Johan Holland [39] a mis au point une nouvelle méthode de recherche de l'optimum appelée "les algorithmes génétiques".

II.2 Que sont les algorithmes génétiques ?

Les algorithmes génétiques (AG) constituent une classe de stratégies de recherche réalisant un compromis équilibré et raisonnable entre l'exploration et l'exploitation. Ce sont des méthodes qui utilisent un choix aléatoire comme outil pour guider une exploration hautement intelligente dans l'espace des paramètres codés.

Les algorithmes génétiques sont des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils utilisent à la fois les principes de la survie des structures les mieux adaptées, et les échanges d'information pseudo-aléatoires, pour former un algorithme d'exploration qui possède certaines des caractéristiques de l'exploration humaine.

II.2.1 Principe

Les algorithmes génétiques sont des algorithmes itératifs de recherche globale dont le but est d'optimiser une fonction prédéfinie appelée le critère ou fonction coût "fitness". Pour réaliser cet objectif, l'algorithme travaille en parallèle sur un ensemble de points représentant les solutions potentielles du problème. Cet ensemble est appelé population. Chaque point la constituant est appelé individu ou chromosome.

Un chromosome est une représentation ou un codage d'une solution d'un problème donné. Le chromosome est constitué d'un ensemble d'éléments appelés gènes, pouvant prendre plusieurs valeurs appelées allèles appartenant à un alphabet non nécessairement numérique. Dans les algorithmes de base un allèle à pour valeur "0" ou "1", un chromosome est donc une chaîne binaire de longueur finie.

Le but est donc de chercher la meilleure combinaison de ces éléments, qui donne lieu au maximum de la fonction coût. A chaque itération ou génération, est créée une nouvelle population en utilisant des parties des meilleurs éléments de la génération précédente, ainsi que des parties innovatrices à l'occasion. Bien qu'utilisant le hasard, les algorithmes génétiques ne sont pas purement aléatoires. Ils exploitent efficacement l'information obtenue précédemment pour spéculer sur la position de nouveaux points à explorer, avec l'espoir que les nouveaux individus vont tendre vers l'optimum de la fonction coût.

A partir d'une première population de chromosomes de taille fixe créée soit d'une manière aléatoire, soit par des heuristiques ou méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques, les algorithmes génétiques génèrent de nouveaux individus de telle sorte qu'ils soient plus performants que leurs

prédécesseurs. Le processus d'amélioration d'une population donnée s'effectue par l'utilisation d'opérateurs génétiques qui sont la sélection, le croisement et la mutation.

II.2.2 L'opérateur de sélection

La sélection choisit et détermine quels membres d'une population survivent et se reproduisent. Cette sélection est déterminée en fonction des valeurs de la fonction f à optimiser (dite aussi fonction coût). Intuitivement cette fonction peut être envisagée comme une mesure de profit, d'utilité ou encore de qualité, que l'on souhaite maximiser. Sélectionner des chromosomes en fonction des valeurs de leur fonction coût revient à donner aux individus dont la valeur est plus grande une probabilité plus élevée de produire un ou plusieurs descendants dans la prochaine génération et de contribuer ainsi à l'évolution de la solution.

L'opérateur de sélection peut être mis en oeuvre sous forme algorithmique de différentes façons :

- Soit d'une manière déterministe ; il s'agit de garder un certain nombre d'individus jugés bons d'après leurs fonctions coût et de rejeter le reste.

- Soit d'une manière stochastique en utilisant la technique de la roulette biaisée pour laquelle chaque individu de la population occupe une surface de la roue proportionnelle à sa fonction coût ; plus précisément, l'opérateur de sélection sélectionne chaque individu i avec une probabilité $\frac{f_i}{f_s}$ (où f_i est la valeur de la fonction coût associée au $i^{ème}$ individu, $f_s = \sum_{i=1}^{n} f_i$ et n est le nombre d'individus de la population).

Hélas, aucune sélection n'est pas parfaite [40], le risque de favoriser un individu ou un petit nombre d'individus constitue un véritable inconvénient. En effet si un chromosome est très supérieur à la moyenne, il constituera presque exclusivement la population suivante ce qui nous fait perdre l'âme des algorithmes génétiques à savoir la diversité, d'où

risque de convergence prématurée. Pour éviter ce genre d'excès, une des solutions possibles consisterait à procéder à un changement d'échelle de la fonction coût. On s'arrange pour que les fonctions coût minimum et maximum aient des valeurs données en adoptant un changement d'échelle statique ou dynamique, linéaire ou exponentiel.

La sélection ne permet pas d'explorer de nouveaux points (de nouvelles possibilités) dans l'espace des solutions admissibles. En ce sens, rien de nouveau n'est testé. L'introduction d'un nouvel espace de recherche se fait via les processus de mutation et de croisement qui, d'un point de vue algorithmique, peuvent être considérés comme des mécanismes servant à changer localement les solutions ou à les combiner.

II.2.3 L'opérateur de croisement

Le croisement est un opérateur qui assure le brassage et la recombinaison des gènes parentaux pour former des descendants aux potentialités nouvelles, il correspond à un échange des matériels génétiques entre deux reproducteurs (parents) choisis d'une manière aléatoire parmi les géniteurs sélectionnés précédemment, pour former deux nouveaux chromosomes (ou enfants) [38][39]. L'opérateur de croisement étant aléatoire il se produit selon une probabilité P_c fixée par l'utilisateur en fonction du problème à optimiser. Il existe plusieurs types d'opérateurs de croisement dont nous citons ci-dessous les plus utilisés :

- **Croisement à un point** : C'est l'opérateur de croisement le plus simple. Il consiste à choisir au hasard deux parents (individus) de la population contenant les chromosomes précédemment sélectionnés et un site de croisement (un nombre de 1 à $L-1$, avec L la longueur du chromosome et k l'indice de ce site), on obtiendra deux enfants en prenant d'une part les k premiers allèles du premier parent et les $(L-k)$ derniers allèles du second parent, et pour le deuxième enfant, les k premiers allèles du second parent et les $(L-k)$ derniers allèles du premier parent. Ceci est illustré dans l'exemple suivant en considérant que parent 1 et parent 2 ont été

sélectionnés aléatoirement pour subir un processus de croisement à un point avec une position de coupure choisie aléatoirement entre $1 \leq k \leq 9$.

Parent 1	1 0 0 0 1 1 1 1 1 0
Parent 2	1 1 1 1 0 0 0 0 0 1
Position aléatoire	$k = 4$
Enfant 1	1 0 0 0 0 0 0 0 0 1
Enfant 2	1 1 1 1 1 1 1 1 1 0

Tableau II.1 : Schéma de croisement à un poin

- **Croisement multi-points** : Il est semblable à l'opérateur de croisement à un point sauf que dans ce cas plusieurs sites de croisements sont sélectionnés. Les enfants seront construits par mélange des parties appartenant aux deux parents. Nous considérons dans l'exemple qui suit un croisement à deux points, supposons que les deux points de coupure se trouvent aux positions 3 et 5.

Parent 1	1 0 0 0 1 1 1 1 0
Parent 2	1 1 1 1 0 0 0 0 1
Positions aléatoires	*k1 = 3, k2 = 5*
Enfant 1	1 0 0 1 0 1 1 1 1 0
Enfant 2	1 1 1 0 1 0 0 0 0 1

Tableau II.2 : Schéma de croisement à deux points

- **Croisement uniforme** : ce croisement a été proposé par Syswerda [41]. Il consiste à choisir avec la même probabilité un allèle de l'un ou l'autre parent pour transmettre sa valeur à la même position des enfants. Pour bien comprendre ce type de croisement nous présentons l'exemple suivant :

Parent 1	1 0 0 0 1 1 1 1 1 0
Parent 2	1 1 1 1 0 0 0 0 0 1
Masque	*1 0 0 1 1 1 0 0 1 0*
Enfant 1	1 1 1 0 1 1 0 0 1 1
Enfant 2	1 0 0 1 0 0 1 1 0 0

Tableau II.3 : Schéma de croisement uniforme

où masque représente les tirages aléatoires pour décider de la transmission de la valeur de l'allèle à l'un ou l'autre descendant :

Si la valeur de masque à la même position que l'allèle concerné est égal à 1 alors la valeur de l'allèle de parent 1 passe à l'enfant 1 et celle de parent 2 passe à l'enfant 2, si elle est égale à 0 alors la valeur de l'allèle concerné de parent 1 passe à l'enfant 2 et celle de parent 2 passe à l'enfant 1. Des résulats théoriques et empiriques présentés par Syswerda dans [41] indiquent que l'opérateur de croisement uniforme donne généralement de meilleurs résulats que ceux produit par un croisement à un ou à deux points.

II.2.4 L'opérateur de mutation

La mutation est un changement aléatoire occasionnel (de faible probabilité notée P_m) de la valeur d'un allèle dans un chromosome .Elle ne crée généralement pas de meilleures solutions, mais elle évite une perte irréparable de la diversité. Sans elle on aurait une population uniforme, incapable d'évoluer. La mutation classique appliquée à une chaîne binaire consistera à changer un "1" en "0" ou vice versa. Pour illustrer ce processus, supposons qu'un chromosome noté A ainsi que la position $k = 5$, ont été aléatoirement choisis, il vient :

A avant mutation	1 0 0 0 1 1 1 1 1 0
Position aléatoire	*k = 5*
A après mutation	1 0 0 0 0 1 1 1 1 0

Tableau II.4 : Schéma d'une mutation classique

Comme pour le cas du croisement, il y a plusieurs types d'opérateurs de mutation [42] nous citons :

- transposition de deux allèles consécutifs (échange de deux allèles consécutifs dans un chromosome),
- transposition de deux allèles quelconques (échange de deux allèles choisis d'une manière aléatoire dans le même chromosome),
- inversion d'allèles (on tire deux positions dans un chromosome d'une manière aléatoire et on inverse l'ordre des allèles dans la zone sélectionnée).

Remarque II.1 *Pour pouvoir appliquer ces trois opérateurs, on a besoin de définir principalement trois paramètres : le nombre d'individus dans la population n, les taux de croisement P_c et de mutation P_m. Trouver de bonnes valeurs de ces paramètres est un problème parfois délicat. La valeur de n dépend fortement du problème, alors que des taux de 5% de mutation et 70% de croisement sont souvent utilisés.*

Les effets des opérateurs génétiques cités plus haut sur l'évolution d'une population donnée s'expliquent en adoptant un des concepts centraux de l'analyse des algorithmes génétiques qui est le concept de schéma.

II.2.5 Notion de schéma

Pour guider sa recherche, un algorithme génétique qui agit sur une population traite en fait des entités appelées schémas et tente de découvrir les meilleurs schémas et les combiner.

Un schéma est un motif de similarité décrivant un sous-ensemble de chromosome ayant des caractéristiques communes à certaines positions ; c'est donc une hyper-surface particulière de l'espace de recherche. Un alphabet étendu est utilisé pour représenter cette notion $(0, 1, *)$ où le symbole $(*)$ représente les positions non définies. Par exemple

le schéma 1 * 1 * 0 représente tous les chromosomes qui ont un 1 à la première et la troisième positions et un 0 à la dernière.

Tous les schémas ne sont pas créés égaux. Certains sont plus spécifiques que d'autres. Pour quantifier ceci, il est nécessaire de définir deux caractéristiques : l'ordre et la longueur d'un schéma.

- L'ordre d'un schéma H, noté $O(H)$, est le nombre de positions à valeur fixes dans ce schéma. Dans un alphabet binaire, une position à valeur fixe est égale à "1" ou à "0". pour le schéma ci-dessus l'ordre est 3.

- La longueur d'un schéma H, noté $\delta(H)$, est la distance entre la première et la dernière position occupée par l'un des caractères appartenant à l'alphabet de base (sans le symbole "*"). Pour le schéma représenté ci-dessus la longueur est $\delta(H) = 5 - 1 = 4$.

Les schémas et les propriétés que nous venons de définir sont des outils intéressants qui permettent d'étudier rigoureusement et de classer les similitudes entre chromosomes. Ils permettent aussi d'analyser les effets des différents opérateurs génétiques sur la convergence des algorithmes génétiques. Considérons d'abord les effets isolés puis combinés de chacun des ces opérateurs

Effet de la reproduction

Supposons qu'à la génération k, il y ait m exemplaires d'un schéma H contenus dans la population $A(k)$, ce que nous notons $m = m(H,k)$. Lors de la reproduction, un chromosome est sélectionné avec une probabilité $P_i = \frac{f_i}{f_s}$, après avoir choisi une population de taille n, ainsi, le nombre de schémas H espérés dans la population à la génération suivante (c'est à dire à la génération $k+1$) est :

$$m(H,k+1) = m(H,k)\, n \frac{f(H)}{f_s}$$

Où $f(H)$ est la valeur moyenne des chromosomes représentant le schéma H à la génération k. Si la valeur moyenne de la population est $f_{moy} = \sum_{i=1}^{n} \frac{f_i}{n}$, l'équation de développement par reproduction des schémas s'écrit comme suit :

$$m(H, k+1) = m(H, k) \frac{f(H)}{f_{moy}} \tag{II.1}$$

Explicitement, un schéma quelconque augmente comme le rapport entre les valeurs moyennes de la fonction coût du schéma et de la population. En d'autres termes les schémas qui ont des coûts moyens supérieurs à celui de la population seront mieux représentés dans la génération suivante, ceux qui ont un coût moyen inférieur à la moyenne le seront moins. **Mais de quelle façon ces individus vont ils se développer (ou disparaître) ?**

Supposons qu'un schéma quelconque H reste au-dessus de la moyenne d'une quantité égale à Cf_{moy} ou C est une constante, d'où $f(H) = f_{moy} + Cf_{moy}$. Il est possible de réécrire l'équation de l'évolution des schémas par la reproduction comme suit :

$$m(H, k+1) = m(H, k)(1 + C)$$

En commençant de la première génération et en supposant que C est stationnaire, on obtient l'équation :

$$m(H, k+1) = m(H, 0)(1 + C)^k \tag{II.2}$$

Ainsi la reproduction seule augmente (ou diminue) exponentiellement le nombre des schémas qui ont des performances au-dessus (au dessous) de la moyenne.

Effet du croisement

La selection en elle-même ne fait rien afin de promouvoir l'exploration de nouveaux individus de l'espace de recherche. Le croisement permet d'introduire de nouvelles configurations, en effet il consiste en un échange d'information entre les chromosomes. Le

croisement crée de nouvelles structures en évitant autant que possible de troubler la stratégie d'allocation dictée par la reproduction seule. Pour voir quels schémas sont affectés par le croisement et lesquels ne le sont pas, considérons un chromosome de longueur $L = 7$ et deux schémas représentatifs de cette dernière.

$A = 0\,1\,1\,1\,0\,0\,0$

$H1 = *\,1\,*\,*\,*\,*\,0$

$H2 = *\,*\,*\,1\,0\,*\,*$

Comme nous l'avons déjà indiqué, le croisement se réalise avec deux chromosomes et une position choisie aléatoirement (croisement classique à un point). Supposons que le chromosome A ait été retenu et que la position du croisement soit égale à 3, l'effet de croisement sur nos deux schémas est le suivant :

$H1 = *\,1\,*\,/\,*\,*\,*\,0$

$H2 = *\,*\,*\,/\,1\,0\,*\,*$

Dans ce cas, le schéma $H1$ sera détruit car ses deux positions fixes ont été séparées. Il est également clair qu'avec le même point de coupure le schéma $H2$ survivra car les deux positions fixes sont préservées dans le même descendant. Pour quantifier ces observations, nous remarquons que le schéma $H1$ a une longueur utile $\delta = 5$. Si le point de croisement est choisi uniformément au hasard parmi les $L - 1$ points possibles, alors le schéma $H1$ sera détruit avec une probabilité P_d suivante :

$$P_d = \frac{\delta(H)}{L-1}$$

Donc le schéma $H1$ sera détruit avec une probabilité $P_d = \frac{5}{6}$, par contre le schéma $H2$ qui a une longueur utile $\delta = 1$ sera détruit avec une probabilité $P_d = \frac{1}{6}$ c'est à dire lors de l'unique événement parmi les 6 possibilités pour lequel le point de coupure est choisi entre les positions 4 et 5.

Si le croisement à lieu avec une probabilité P_c, lors d'un appariement quelconque, la

probabilité de survie peut être exprimée ainsi :

$$P_{sc} = 1 - Pc\frac{\delta(H)}{L-1} \tag{II.3}$$

Effet de la mutation

La mutation est la modification aléatoire de probabilité P_m d'une position quelconque du chromosome. Pour qu'un schéma H survive, toutes ces positions instantanées doivent survivre. Comme la probabilité de survie d'un allèle est de $1 - P_m$ et comme chaque mutation est statistiquement indépendante des autres, un schéma quelconque survit quand chacune des $O(H)$ positions instantanées dans le schéma survit, par conséquent la probabilité de survie à la mutation d'un schéma H est :

$$P_{sm} = (1 - P_m)^{O(H)}$$

Etant donné que la probabilité de mutation est petite ($P_m << 1$), la probabilité de survie du schéma H peut être approximée par l'expression :

$$P_{sm} \approx 1 - O(H) P_m \tag{II.4}$$

L'effet combiné de ces trois paramètres est traduit par le théorème suivant :

Théorème II.1 *Un schéma H quelconque reçoit, du fait de la reproduction, du croisement et de la mutation, un nombre de copies donné par l'équation suivante :*

$$m(H, k+1) \geq m(H, k) \frac{f(H)}{f_{moy}} P_{sc} P_{sm} \tag{II.5}$$

En effet, les schémas courts, d'ordre faible, et de performance au-dessus de la moyenne font l'objet d'un nombre de tests exponentiellement croissants dans les générations suivantes [38][43].

L'équation II.5 est dite théorème des shémas ou théorème fondamental des AG.

II.3 Adaptabilité des algorithmes génétiques

L'un des plus importants avantages des algorithmes génétiques est leur indépendance presque totale du problème à optimiser. En effet, ils ne demandent en général qu'une mesure de la qualité de la solution. La représentation de la solution sous forme de code est une étape cruciale lors de l'application des algorithmes génétiques. Le codage du chromosome sous forme de chaîne binaire est souvent utilisé car il s'adapte aisément à un large éventail de problèmes. Toutefois pour certains types de problèmes le codage binaire devient très lourd à manipuler et se traduit souvent :

- par une chaîne très longue pour les problèmes multidimensionnels et ceux demandant une grande précision numérique,
- par un résultat erroné pour les problèmes soumis à des contraintes comme ceux rencontrés dans les problèmes d'ordonnancement (combinaison ne correspondant pas à une solution).

II.3.1 Les problèmes d'ordonnancement vus par les algorithmes génétiques

Devant le succès grandissant des algorithmes génétiques dans le domaine de la recherche opérationnelle et de l'intelligence artificielle, leur efficacité à résoudre des problèmes d'optimisation très variés allant de l'économie à la biologie, en passant par l'informatique, la reconnaissance de formes et la commande des systèmes [43][44][45]. Ainsi, leur efficacité à s'approcher assez rapidement de l'optimum global et le pouvoir de donner contrairement à la plus part des autres méthodes existantes non pas une seule solution mais tout un ensemble, leur donne un avantage considérable et constitue une raison valable à leur utilisation malgré la concurrence féroce d'autre méthodes aussi efficaces qu'eux tels que le recuit simulé [46][16] et la recherche tabou [18][21] (voir le paragraphe I.4.2). Mais si la théorie des algorithmes génétiques est bien développée pour les problèmes qui peuvent

être facilement codés par une chaîne binaire et sans ordre ou séquence de dépendance, elle ne l'est pas pour une grande partie des applications réelles et tout particulièrement pour les problèmes d'ordonnancement. Le problème du voyageur de commerce est un exemple qui illustre parfaitement les différents problèmes posés par l'utilisation des algorithmes génétiques pour résoudre des problèmes classés NP-difficiles [47][48].

Problématique du voyageur de commerce (PVC)

Un commerçant doit réaliser une tournée complète de plusieurs villes, en passant une seule fois par chaque ville tout en minimisant la distance totale parcourue. L'espace de recherche pour ce type de problème est formé par l'ensemble des permutations incluant les n villes que le commerçant doit visiter. Trouver la solution optimale consiste à trouver la permutation qui donne la distance minimale à parcourir parmi les $n!$ permutations possibles. La première tentative de résolution de ce type de problème remonte à 1759, le problème consistait à faire passer un chevalier une seule fois sur les 64 places d'un échiquier. Ce n'est qu'on 1932 que le terme PVC a été utilisé pour la première fois [43]. De nombreuses méthodes de recherche de la solution essentiellement déterministes telles que les méthodes dites par séparation et évaluation connues sous le nom de branch-and-bound et les algorithmes basés sur les heuristiques, sont souvent mises en échec à cause des contraintes de temps de calcul. Cependant des méthodes plus rapides telles que les méthodes dites de descente stochastique, les algorithmes gloutons, le recuit simulé, la recherche tabou et les algorithmes génétiques ont vu le jour. Ces méthodes ont le grand avantage d'approcher l'optimum global en un temps acceptable [48].

L'application des algorithmes génétiques à ce type de problèmes et plus généralement aux problèmes d'ordonnancement soulève certaines difficultés : la représentation de la solution (choix du codage) et l'évaluation des individus ne sont pas triviales ; souvent, des opérateurs génétiques spécifiques doivent être conçus pour aboutir à des résultats utiles [41]. La première question intéressante à se poser est : **le codage binaire est-il adéquat ?**

Supposons que notre commerçant doit passer par n villes, alors il nous faut $log_2(n)$ bits pour coder ces n villes. Si $n = 20$, il faut 5 bits pour représenter les 20 villes, mais avec 5 bits on peut avoir 32 possibilités différentes, il reste donc 12 codes qui ne correspondent à aucune ville, ce qui donne forcement lors de l'application des opérateurs génétiques des résultats irréalisables. Ainsi, afin d'aboutir à des résultats admissibles il est nécessaire de prendre en compte les diverses contraintes du problème traité. De part leur fonctionnement les opérateurs génétiques ne permettent pas la prise en compte de contraintes telles que celles rencontrées dans les problèmes d'ordonnancement. Toutefois, il existe principalement trois solutions possibles à ce genre de problème [42][43].

- Fonction objectif à pénalité : Dans le cas ou on obtient des solutions non réalisables, ces dernières sont éliminées par la fonction objectif qui leur affecte une pénalité et par conséquent diminue leur chance d'être sélectionnées pour la génération suivante.

- Algorithmes de réparation : Une solution initialement non admissible peu le devenir en lui appliquant un processus de correction.

- Opérateurs génétiques adaptés au problème : Dans ce cas de figure, plutôt que de corriger les erreurs dues au non-respect des contraintes, on préfère inclure ces contraintes dans la conception même des différents opérateurs génétiques. Le paragraphe suivant explique comment ceci peut être fait au niveau du codage.

II.3.2 Le codage dans les problèmes d'ordonnancement

Le codage est la première étape à définir lors de l'utilisation des algorithmes génétiques. En effet, les problèmes tels que celui que nous traitons dans cette thèse à savoir le problème d'ordonnancement de type job-shop flexible présentent une difficulté due principalement aux contraintes de précédence entre les opérations d'un même ordre de fabrication. La littérature est assez riche dans ce domaine où on trouve une classification détaillée des différents types de codes existants [49][50][51][2]. En considérant la classification don-

née par cette littérature abondante, on peut distinguer des représentations directes ou indirectes et qui peuvent être augmentées par des connaissances spécifiques au problème.

La représentation directe

Dans ce cas de figure, le chromosome représente directement un ordonnancement réalisable. Le plus souvent on utilise une représentation assez simple car les opérateurs génétiques s'avèrent plus compliqués à mettre en oeuvre, mais en revanche la fonction coût est directement applicable au chromosome ce qui donne une évaluation directe de la valeur de l'ordonnancement trouvé. Parmi les codages directs nous citons :

Codage du PVC : les solutions possibles peuvent être codées par des chaînes de caractères telle que ABCDEF où les lettres représentent les villes dans l'ordre dans lequel elles sont parcourues. On remarque que ce codage réduit les problèmes d'ordonnancement à des problèmes de recherche de la meilleure permutation des opérations à effectuer.

Représentation par liste d'ordre/machine/date de début : Ce codage à été principalement développé pour une catégorie de problème d'ordonnancement plus restreinte où chaque ordre est limité à une seule opération qui doit être exécutée par une seule machine. Chaque chromosome est une liste d'opérations uniques affectées chacune à une machine avec une date de début d'exécution.

Ordre A	Ordre B	. . .
M1	M1	. . .
Début A	Début B	. . .

Une variante de ce type de codage existe pour les problèmes d'ordonnancement ayant plusieurs ressources où plusieurs opérations pourront être exécutées en parallèle. Dans ce cas, le codage consiste à découper un chromosome en sous-chromosome. Chaque sous-chromosomes représente un ordre des opérations sur une ressource.

Représentation augmentée des connaissances spécifiques au problème : Ce type de codage à été proposé par Bruns [51], il consiste à inclure dans la conception même du chromosome toutes les informations particulières concernant le problème à traiter. Ce qui est très pratique pour les problèmes ayant de nombreuses contraintes, nous pouvons tous les inclure ou du moins une partie d'entre eux dans la conception du chromosome. Ce type de codage nécessite la mise au point d'opérateurs génétiques spécifiques qui sont souvent plus compliqués que les opérateurs traditionnels.

Ordre 7			Ordre 9		. . .
Opér B1	Opér B2	Opér B3	Opér A1	Opér A2	. . .
M 9	M 3	M 6	M 6	M 1	. . .
[10, 15]	[16, 17]	[18, 22]	[2, 9]	[11, 19]	. . .

Par exemple pour l'ordre 7 / opération B2 / M3 / [16, 17] : s'interprète de la manière suivante : l'opération 2 de la gamme B de l'ordre de fabrication 7 s'exécutera sur la machine M3 à partir de l'unité de temps 16 jusqu'à 17.

La représentation indirecte

Ce codage est plus souple que le précédent, il est utilisé dans la plupart des approches basées sur les algorithmes génétiques déstinées à la résolution des problèmes d'ordonnancement. Dans ce cas, les chromosmoes ne représentent plus directement un ordonnancement mais plutôt des listes de priorités sur une ressource qui permettent de déterminer un ordonnancement. La transition de la représentation du chromosome vers un ordonnancement légal doit être faite par une procédure de décodage, cette dernière doit garantir l'obtention d'un ordonnancement realisable. Deux grandes catégories de codages peuvent être dégagées : ceux indépendants du domaine et ceux spécifiques au problème.

Représentation indépendante du domaine : Le chromosome ne contient aucune information concernant le domaine du problème, il y a une totale indépendance du codage envers le problème traité. Les opérateurs génétiques traditionnels peuvent aisément être appliqués. On distingue :

Représentation binaire : Les représentations binaires ne se limitent plus aux vecteurs à une dimension. Des structures plus complexes ont été récemment utilisées dans les problèmes d'ordonnancement. L'utilisation la plus significative de cette représentation au problème du type job-shop est celle présentée dans [31]. Ce type de codage consiste à déterminer une matrice de précédence entre deux ordres de fabrication. Selon les priorités données à ces derniers s'exécutant sur la même machine, on décide entre deux ordres, lequel doit être exécuté en premier. Il est à noter que cette représentation est assez délicate et nécessite lors du processus de croisement l'utilisation d'une procédure de correction complexe. D'autres procédures de croisement sans processus de correction et une variante de ce codage ont été développés [37][52].

Représentation par liste d'ordres : Cette approche à été développée par Syswerda [41]. Un chromosome est représenté par une liste de tous les ordres à ordonnancer.

Ordre 5	Ordre 9	Ordre 3	. . .

En effet, ce type de codage se réduit à établir une liste de priorité entre les ordres. Le problème dans ce cas se réduit à une série de permutations des ordres identique à celle du PVC. De nombreux auteurs ont utilisé cette représentation plus précisément dans les problèmes flow-shop [53][54].

Représentation spécifique au problème : Ce type de représentation enrichi le chromosome, les connaissances spécifiques au problème sont explicitement représentées dans le chromosome et de nouveaux opérateurs génétiques dépendants du domaine peuvent être conçus. Généralement on les obtient en ajoutant une ou plusieurs informations aux codages indépendants du domaine. Parmi ces représentations, on distingue :

Représentation par liste d'ordres / gammes : Ce codage à été développé par Baghi [49]. Dans ce cas, la gamme de chaque ordre est indiquée dans la représentation. La responsabilité du générateur d'ordonnancement est restreinte à la sélection des machines et des intervalles de temps.

Ordre 5 / Gamme B	Ordre 9 / Gamme C	Ordre 3 / Gamme A

Représentation par liste d'ordres / gammes / ressources : On ajoute à la représentation précédente les ressources sur lesquelles doivent s'exécuter les opérations on augmente les connaissances incorporée dans le chromosome [49].

Ordre 7			Ordre 9		. . .
Opér B1	Opér B2	Opér B3	Opér A1	Opér A2	. . .
M 9	M 3	M 6	M 6	M 1	. . .

Cette représentation permet de diminuer considérablement la complexité des opérateurs génétiques à mettre en oeuvre.

Représentation par listes de préference dépendantes du temps : Dans cette représentation, une solution est codée à chaque instant (espacé par un intervalle de temps) par une liste de préférences de chaque station de travail (ST) [30]. Une liste de préférence est une liste d'ordres, à laquelle on ajoute les éléments 'wait' et 'idel'.

ST1 (0 ordre 2 wait ordre 1 idel)	ST2
(60 ordre 1 ordre 2 wait idel) (120 ...)	. . .

L'interprétation de ce codage est de montrer quel travail, chaque station préfère exécuter à un instant donné, quand elle attend ou ne rien faire. L'interprétation de l'exemple présenté plus haut est la suivante [51][2] : à l'unité de temps 60, la station de travail ST1 doit exécuter l'ordre 1 avant l'ordre 2. Si aucun des deux ordres ne demande ST1 à cet instant, alors elle doit attendre.

Nous venons de voir qu'il existe deux grandes catégories de représentations du chromosome : binaire et symbolique. C'est en fonction du problème traité que l'une ou l'autre s'avère meilleure. Pour les problèmes d'ordonnancement c'est la représentation symbolique qui est la mieux adaptée en raison de la nature contraignante du problème. Le passage du codage binaire au codage symbolique implique forcément une forte modification des autres opérateurs génétiques (croisement et mutation). Ces opérateurs vont être détaillés dans le paragraphe qui suit.

II.3.3 Le croisement pour les problèmes d'ordonnancement

En passant au codage symbolique, la nécessité de concevoir de nouveaux opérateurs de croisement s'est fait considérablement sentir. En effet, l'imagination des chercheurs été à la hauteur de leurs ambitions et de nombreux opérateurs de croisement ont vu le jour, la littérature dans ce domaine est assez considérable [53][42][37]. Dans ce paragraphe nous allons décrire successivement les principaux opérateurs de croisement utilisés en ordonnancement.

Croisement PMX

Le PMX (Partially Mapped crossover) a été conçu par Goldberg et al [55] principalement pour résoudre le problème du voyageur de commerce. Il a également été utilisé pour résoudre les problèmes d'ordonnancement de type job-shop. Ce croisement consiste à échanger une portion d'une séquence avec une portion d'une autre séquence et le reste des informations est changé après d'éventuels remplacements appropriés. Le PMX est un croisement à deux points : Après avoir choisi d'une manière aléatoire les deux chromosomes (deux parents) qui vont subir le croisement, on construit la zone de croisement par un choix aléatoire des deux points de coupure. L'exemple suivant illustre ce croisement.

Première étape : Inverser les zones de croisement des parents pour générer les enfants probablement mal formés ; cette inversion consiste à ce que la zone interne de l'enfant 2 reçoive celle de parent 1 et celle de l'enfant 1 reçoive celle de parent 2.

Deuxième étape : les zones externes de l'enfant 1 (respectivement l'enfant 2) recevrons les éléments de parent 1 (respectivement parent 2).

Pour illustrer ce processus, on considère l'exemple suivant :

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Enfant 1	I	H	D	B	C	J	A	**C**	**B**	**J**
Enfant 2	H	**G**	A	E	F	G	I	**E**	D	**F**

On remarque que les deux ordonnancements générés sont illégaux, on a dupliqué des éléments dans les deux enfants par exemples les ordres C, B, et J dans enfant 1 (en gras) et d'autres manquent comme les ordres E, F et G. Le même problème se pose pour le deuxième enfant. Pour remédier à cet handicap une seconde étape est nécessaire :

- Détecter les éléments dupliqués à l'intérieur de la zone de croisement pour l'enfant 1 (respectivement l'enfant 2).
- Chercher les positions de ces éléments (éléments dupliqués) dans la zone de croisement du parent 2 (respectivement parent 1) et prendre les éléments correspondant à ces positions du parent 1 (respectivement parent 2) et les mettre à la place des éléments dupliqués dans l'enfant 1 (respectivement l'enfant 2).

Ainsi, l'ordre C se trouve à la position 5 dans parent 2 (à l'intérieur de la zone de croisement) il lui correspond l'ordre F dans parent 1 (toujours à l'intérieur de la zone de croisement). Après avoir appliqué ce processus aux deux enfants nous obtenons :

Enfant 1	I	H	D	B	C	J	A	F	E	G
Enfant 2	H	J	A	E	F	G	I	B	D	C

Croisement MPX

Le MPX (Maximal Preservative crossover) à été proposé par Mülhenbein [42][37] pour le problème du voyageur de commerce. Il consiste à insérer une partie du parent 1 dans le parent 2 et vice versa de telle façon que les chromosomes résultants soient le plus proche possible des parents. Deux positions sont sélectionnées aléatoirement par exemple les positions 4 et 6. La première étape consiste à recopier la zone du parent 1 (respectivement du parent 2) dans l'enfant 1 (respectivement dans l'enfant 2), les éléments de l'enfant 1 (respectivement de l'enfant 2) qui ne sont pas dans la zone de croisement sont complétés de la façon suivante. Le i-ème élément du parent 2 (respectivement du parent 1) est recopié sur le i-ème élément de l'enfant 1 (respectivement de l'enfant 2) si cette recopie respecte les contraintes. Sinon, le i-ème élément du parent 1 (respectivement du parent

2) est recopié sur le i-ème élément de l'enfant 1 (respectivement de l'enfant 2) si cette recopie respecte les contraintes. Si les deux cas précédents ne peuvent pas s'appliquer, le i-ème élément de l'enfant 1 (respectivement de l'enfant 2) reçoit un élément de la zone de croisement du parent 2 (respectivement du parent 1) qui respecte les contraintes. L'exemple suivant illustre bien ce cas de croisement :

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Enfant 1	H	B	A	E	F	G	I	C	D	J
Enfant 2	I	H	D	B	C	J	A	E	F	G

Croisement d'ordre OX

Le croisement d'ordre OX (Order crossover) à été conçu par Davis [30]. Deux zones aléatoires sont choisies. Les deux enfants sont obtenus en inversant les zones de croisement des parents. Les places restantes de l'enfant 1 et de l'enfant 2 sont remplies par les éléments non contenus dans leurs zones de croisement. On commence par le premier élément de la première zone du parent 1 (respectivement parent 2), on regarde si l'élément correspondant existe dans la partie construite de l'enfant 1 (respectivement l'enfant 2), si oui, passer à la position 2, sinon mettre l'élément correspondant dans la première position en commençant par la zone 3 et on poursuit par la zone 1 de l'enfant 1 (respectivement l'enfant 2) jusqu'à ce que toutes les cases vides soient remplies. Ceci est illustré par l'exemple suivant :

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Enfant 1	F	G	A	B	C	J	I	H	D	E
Enfant 2	J	I	D	E	F	G	H	A	B	C
	Zone 1			Zone 2			Zone 3			

Croisement d'ordre LOX

Le LOX (Linear Order crossover) à été proposé par Falkenauer et al [32] pour résoudre les problèmes d'ordonnancement du type job shop. Il est similaire au croisement OX, la seule différence est que le LOX est linéaire alors que le OX est circulaire. Autrement dit, après la construction des zones de croisement des enfants 1 et 2, les places restantes sont par la suite remplies par les éléments non contenus dans la zone de croisement. On commence dans ce cas par la première zone et on poursuit par la troisième zone dans l'ordre de parent 1 pour l'enfant 1 et du parent 2 pour l'enfant 2. Ceci est illustré par l'exemple suivant :

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Enfant 1	I	H	D	B	C	J	E	F	G	A
Enfant 2	H	A	B	E	F	G	C	J	I	D
		Zone 1			Zone 2			Zone 3		

Croisement basé sur les ordres UOX

Le UOX (Uniform Order crossover) à été développé par Syswerda [41]. Il consiste à choisir un ensemble de positions d'une manière totalement aléatoire. L'ordre dans lequel les éléments des positions sélectionnées apparaissent dans un parent est imposé à l'autre parent, pour produire les deux enfants.

Dans l'exemple qui suit, on sélectionne les positions 2, 3, 5 et 8. Pour construire l'enfant 1 (respectivement l'enfant 2), l'ordre des éléments dans ces positions de parent 2 est imposé à l'enfant 1 (respectivement celui de parent 1 est imposé à l'enfant 2). Pour le parent 2, ces éléments sont G, A, C et E. Dans parent 1, ces éléments se trouvent aux positions 6, 7, 8, et 4 (donc dans l'ordre E, G, A et C qui est différent de celui du parent 2). Pour générer l'enfant 1, ces éléments doivent être réorganisés de manière à respecter

l'ordre de ces éléments dans parent 2 (c'est à dire G, A, C et D), donc leur position dans l'enfant 1 sera 4, 6, 7 et 8.

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Positions sélectionnées		*	*		*			*		
Enfant 1	I	H	D	G	F	A	C	E	B	J
Enfant 2	H	G	A	B	D	J	I	E	F	C

Autrement dit, ce croisement revient pour la génération de l'enfant 1 (respectivement l'enfant 2) à respecter l'ordre des éléments donné par le parent 2 (respectivement le parent 1) et les positions données par le parent 1 (respectivement le parent 2).

Croisement basé sur les positions UPX

Le UPX (Uniform Position crossover) Il à été également conçu par Syswreda [41]. Un ensemble de positions est sélectionné d'une manière aléatoire, dans ce cas les valeurs correspondant aux positions choisies dans parent 2 (respectivement dans parent 1) sont copiées à la même position dans l'enfant 1 (respectivement dans l'enfant 2), les positions restées vides dans l'enfant 1 (respectivement dans l'enfant 2) sont remplies avec les valeurs manquantes en les prenant dans l'ordre du parent 1 (respectivement du parent 2). Ceci est illustré par l'exemple suivant :

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	I	H	D	E	F	G	A	C	B	J
Parent 2	H	G	A	B	C	J	I	E	D	F
Positions sélectionnées		*	*		*			*		
Enfant 1	I	G	A	H	C	D	F	E	B	J
Enfant 2	G	H	D	A	F	B	J	C	I	E

Le croisement UOX nous permet de préserver un ordre pour certains éléments alors que le UPC préserve les positions de quelques éléments. Ceci permet en fonction de la nature du problème à traiter, la prise en compte dans le processus de croisement lui-même de quelques contraintes.

Croisement de cycle CX

Le CX (Cycle crossover) à été développé par Oliver [56]. Il permet de préserver les positions absolues des éléments contenus dans les parents, il a été conçu à l'origine pour la résolution du problème de voyageur de commerce. Dans une première phase, un cycle est construit à partir des deux parents et copié dans l'enfant 1 (respectivement dans l'enfant 2), dans une seconde phase, les positions vides restantes sont remplies avec les éléments de parent 2 (respectivement dans parent 1) pour l'enfant 1 (respectivement pour l'enfant 2).

La construction du cycle de la première phase s'effectue pour l'enfant 1 (respectivement pour l'enfant 2) de la manière suivante :

Etape 1 Choisir d'une manière aléatoire une position de départ appelée la position courante,

Etape 2 copier l'élément contenu dans la position courante de parent 1 (respectivement de parent 2) dans l'enfant 1 (respectivement dans l'enfant 2),

Etape 3 chercher la position de cet élément dans parent 2 (respectivement dans parent 1), cette dernière devient la nouvelle position courante,

Etape 4 si la position courante est déjà remplie dans l'enfant 1 (respectivement dans l'enfant 2), la première phase est terminée, sinon aller à l'étape 2.

Nous allons illustrer cette démarche par un exemple : la position de départ sera la position 4.

Position	1	2	3	4	5	6	7	8	9	10
Parent 1	A	B	C	D	E	F	G	H	I	J
Parent 2	C	F	A	J	H	D	I	G	B	E
Positions courante		3ème		1ère	7ème	2ème	5ème	6ème	4ème	8ème
Enfant 1	C	B	A	D	E	F	G	H	I	J
Enfant 2	A	F	C	J	H	D	I	G	B	E

Remarque II.2 *Il est à noter que ce croisement peut donner dans certain cas comme enfants exactement les mêmes parents. Dans ce cas le croisement n'apporte rien de nouveau dans la population et c'est entre autre dans des cas pareils, que l'opérateur de mutation devient utile.*

II.3.4 La mutation pour les problèmes d'ordonnancement

L'opérateur classique (modification aléatoire d'une valeur d'un gène) n'est pas applicable dans les problèmes d'ordonnancement. Les autres opérateurs de mutation (transposition ou inversion de deux gènes voisins ou aléatoire) peuvent être appliqués. Cependant, dans certains problèmes tels que le job shop, il faut tenir compte des contraintes de précédence et de ressources. Il est à noter aussi que plusieurs autres opérateurs de mutation ont été mis aux points. Parmi ceux là nous pouvons citer les opérateurs de mutation avancée, ils utilisent des méthodes de recherche locale tels que le recuit simulé, la recherche tabou pour améliorer le résultat [45][57].

II.4 Conclusion

La force des algorithmes génétiques réside essentiellement dans le fait qu'ils ne demandent pas d'hypothèses restrictives sur l'espace de recherche. A travers les différentes applications, les algorithmes génétiques sont capables de manipuler une grande variété de paramètres ainsi que des structures complexes de représentation de solutions. Les problèmes d'ordonnancement constituent un parfait exemple montrant l'adaptabilité des

algorithmes génétiques, par contre il faut préciser que cette adaptation n'est pas évidente. En effet, il faut identifier et déterminer toutes les contraintes du problème et adapter en fonction de celle-ci, les différents paramètres (codages, opérateurs de croisement, opérateurs de mutation) des algorithmes génétiques. Ces derniers offrent la possibilité d'interaction avec le concepteur, ainsi il peut inclure des solutions dans la population initiale afin d'accélérer la convergence ou tout simplement pour faire démarrer l'algorithme dans le cas des problèmes à forte contraintes. Par contre la difficulté à laquelle doit faire face le concepteur est le réglage des différents paramètres génétiques. Ces derniérs influencent fortement la convergence et l'efficacité des algorithmes. Par exemple un taux très fort de mutation peut provoquer une convergence rapide vers un optimum local. Une trop faible taille de population réduit la diversité des individus et peut aussi provoquer une convergence prématurée.

Dans le chapitre suivant nous allons montrer comment nous avons adapté, hybridé et utilisé ces algorithmes pour minimiser le temps total d'ordonnancement pour résoudre conjointement les problèmes d'ordonnancement et d'affectation dans les ateliers de type job-shop flexible.

Chapitre III

Problèmes d'ordonnancement et d'affectation : Approche par les algorithmes évolutionnistes

III.1 Introduction

Après avoir présenté les principes généraux des algorithmes génétiques, le but de ce chapitre est d'étudier comment ces techniques peuvent aider à traiter des problèmes complexes d'optimisation combinatoire tels que ceux que nous avons mis en évidence dans le chapitre 1. Notre objectif principal est d'offrir une approche et des outils qui soient à la fois rapides et capables de donner une solution pour les problèmes d'ordonnancement et de résorber le plus grand nombre d'aléas de la production. L'approche que nous proposons doit permettre non seulement d'aboutir à un ensemble d'ordonnancement admissible pour des problèmes très complexes de grande taille et mieux encore, l'ordonnancement choisi parmis cet ensemble doit être très proche de l'optimum au sens d'un critère préalablement défini.

III.2 Identification du problème considéré

Le problème traité dans le cadre de cette thèse est du type job-shop flexible c'est une extension du modèle job-shop classique. En effet, il contient du fait de sa structure, deux problèmes classiques de l'optimisation combinatoire : d'une part, le problème de l'affectation linéaire, car chaque opération peut être exécutée sur une ou plusieurs machines avec des temps d'exécution différents (dépendant de la machine sur laquelle l'opération va s'exécuter) et d'autre part, le problème de l'ordonnancement qui consiste à trouver un ordre de passage de ces différentes opérations sur l'ensemble des machines en respectant toutes les contraintes inhérentes au problème posé et en minimisant un ou plusieurs critères préalablement définis. Les données et les contraintes de ce problème sont détaillées dans les paragraphes suivants :

III.2.1 Données et hypothèses

Nous supposons que les problèmes traités possèdent les données suivantes :

- $M = \{M_1, M_2, ..., M_m\}$ est un ensemble de m machines non reliées.
- $OF = \{OF_1, OF_2, ..., OF_n\}$ est un ensemble de n ordres de fabrication (jobs). Chaque ordre de fabrication est composé d'une gamme opératoire g_{OFj} qui représente une séquence linéaire totalement ordonnée de x_{OFj} opérations. Cette séquence est indépendante des autres ordres de fabrication et par conséquent varie d'un ordre à un autre.
- une opération est notée $O_{i,OFj}$ ou OF_j est le numéro du produit auquel elle appartient, et i son numéro d'ordre dans la gamme.
- L'opération $O_{i,OFj}$ nécessite pour être réalisée, une des machines de l'ensemble existant dans le parc de l'atelier susceptible de réaliser cette opération. Son temps opératoire est noté $p_{i,OFj,Mk}$, il dépend de la machine M_k choisie (k représente le numéro de la machine affectée à cette opération ($k = 1, 2, ..., m$).

Et comme hypothèses on a :

- Les temps de préparation des machines et les durées de transport entre deux opérations sont considérés négligeables,
- les machines sont indépendantes les unes des autres.

De plus, la réalisation des opérations est soumise aux contraintes décrites dans le paragraphe suivant :

III.2.2 Les contraintes

- Les contraintes externes de disponibilité des machines : chaque machine M_k est supposée disponible en quantité égale à un,
- Les contraintes externes de dates limites : chaque produit est contraint par une date de début au plus tôt (date de disponibilité) noté $r_{i,OFj,Mk}$ et une date de fin au plus tard (date due ou deadline) $d_{i,OFj,Mk}$,
- une machine ne peut exécuter qu'une seule opération à un instant donné,
- chaque machine est disponible pendant toute la période de l'ordonnancement,
- une opération en cours d'exécution ne peut être interrompue (il n'y a pas de préemption),
- les ordres de fabrication sont indépendants les uns des autres.

Le problème ne consiste plus uniquement à trouver une séquence des opérations sur chaque machine (job-shop classique) mais auusi de déterminer une affectation judicieuse de chaque opération sur une machine. Ainsi notre objectif est de traiter conjointement les problèmes d'ordonnancement et d'affectation afin de trouver un ensemble d'ordonnancements admissibles et de choisir parmis cet ensemble la bonne solution (le bon ordonnancement) qui soit la plus proche possible de l'optimum au sens de notre critère à

savoir le temps de production minimum pour l'ensemble des ordres de fabrication (Cmax ou Makespan). Le choix de ce critère étant dicté par le fait que ce dernier permet d'optimiser le temps de fabrication et d'utiliser au mieux les ressources [1].

Les algorithmes génétiques ont vu leur utilisation s'accroître et se diversifier du fait de leur souplesse, de leur robustesse et de leur capacité à converger vers l'optimum global. Leur utilisation a connu un succès grandissant dans de nombreux domaines [43][44][45] avec plus ou moins des modifications selon le type du problème à traiter. Ces modifications peuvent être très importantes dès lors que le problème est soumis à des contraintes très fortes. Le problème d'ordonnancement étant le parfait exemple. L'application des algorithmes à stratégie d'évolution au problème d'ordonnancement nécessite essentiellement des modifications au niveau du codage. Ces modifications sont rendues nécessaires à cause des contraintes de précédence et de ressources qui doivent être prises en compte. Par conséquent, nous présenterons deux approches d'algorithmes évolutionnistes réglant conjointement les problèmes d'ordonnancement et d'affectation.

III.3 Première approche

Le choix de la représentation de la solution est une tâche importante et nécessaire pour garantir le succès de l'application des algorithmes génétiques. Traditionnellement le chromosome est une suite binaire, malheureusement ce simple codage ne permet pas ou avec beaucoup de processus de correction [31][33] de traiter des problèmes sur-contraints. Pour pallier à ce type de problème, de nombreux codages ont vu le jour (voir chapitre II). La majeure partie de ces derniers représentent le chromosome sous forme de chaîne ou une suite de lettres. De ce fait et par soucis de simplicité nous nous sommes orientés vers une représentation d'un ensemble d'affectation/ordonnancement sous formes des chromosomes conviviaux et exploitables directement par le chef d'atelier.

Le premier codage que nous avons mis au point est nommé codage parallèle des machines (CPM) [58] décrit dans les paragraphes suivantes :

III.3.1 Codage parallèle des machines (CPM)

un individu (ou chromosome) est représenté par un ensemble de machines placées en parallèle, chaque machine est un vecteur contenant les opérations à exécuter sur cette machine ; une opération est représentée par trois champs, le premier indique l'ordre de l'opération dans la gamme, le second représente le numéro de l'ordre de fabrication et le troisième champ contient la date de début d'exécution de l'opération si son affectation sur cette machine est définitive dans le chromosome. La configuration du codage parallèle des machines est la suivants :

M_1	$(i,OF_j, t_{i,OFj,M1})$	...
M_2	$(i',OF_{j1}, t_{i',OFj1,M2})$	
...	...	...
M_m	...	

Figure III-1 : Codage parallèle des machines

d'une façon plus générale une ligne d'un chromosome se présente comme suit :

M_k : (i, OF_j, $t_{i,OFj,Mk}$), (i', OF_{j1}, $t_{i',OFj1,Mk}$),...

Où i est l'opération à exécuter, OF_j est l'ordre de fabrication auquel appartient l'opération i et $t_{i,OFj,Mk}$ est la date de début d'exécution de l'opération i de l'OF_j sur la machine M_k. Cette date est calculée en tenant compte des contraintes de ressources et de précédence.

Exemple III.1 *Nous considérons les trois ordres de fabrication suivants :*

Les données de cet exemple seront utilisées dans tous les exemples qui vont suivre

OF 1 : O1,1 , O2,1 , O3,1

OF 2 : O1,2 , O2,2 , O3,2

OF 3 : O1,3 , O2,3

Le temps d'exécution de chaque opération est indiqué dans le tableau suivant :

	M_1	M_2	M_3	M_4	M_5
O1, 1	1	9	3	7	5
O2, 1	3	5	2	6	4
O3, 1	6	7	1	4	3
O1, 2	1	4	5	3	8
O2, 2	2	8	4	9	3
O3, 2	9	5	1	2	4
O1, 3	1	8	9	3	2
O2, 3	5	9	2	4	3

Tableau III-1 : Temps d'exécution des opérations sur différentes machines

Une représentation possible d'un chromosome se présente comme suit :

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	(2, 1, 3)	(3, 2, 5)
M_4	(3, 1, 5)		
M_5	(1, 3, 0)		

Figure III-2 : Exemple de la représentation du CPM

L'avantage de ce codage est qu'il permet de voir directement l'ordonnancement obtenu, il contient toutes les informations dont le chef d'atelier à besoin c'est à dire quelle opération et sur quelle machine elle va être exécutée et à quelle date, quelles sont les machines chargées et celles qui sont sous-utilisées, ceci permet au chef d'atelier de gérer au mieux son parc de machines ainsi que sa production.

III.3.2 Création de la population initiale

Le choix de la population initiale joue un rôle primordial dans l'évolution de la solution tout le long des générations. En effet, il faut obtenir une population 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. Mais il faut rappeler aussi que la création de la première population n'est guère une tâche facile. Il existe plusieurs façon de mettre au point une première population ; soit d'une manière totalement aléatoire, ce qui n'est pas très recommandé quand il faut respecter un certain nombre de contraintes telles que celles présentes dans notre problème (contraintes de précédences et de ressources) ou en utilisant des heuristiques. Dans le cadre de nos travaux, nous avons utilisé une combinaison de plusieurs méthodes que nous présentons ci-dessous :

1^{er} cas : Le problème est de petite taille et une solution peut être trouvée en un temps assez raisonnable par les méthodes exactes telle que branch & bound, ou par la programmation dynamique. On procède alors à la duplication, la modification et à l'évolution de cette solution en appliquant les différents processus de mutation et de croisement jusqu'à obtention des n chromosomes initiaux requis [42][59]

$2^{ème}$ cas : Créer la population initiale en résolvant le problème par la méthode de la programmation avec contraintes (PC), l'ensemble donné par cette dernière sera utilisé comme ensemble de départ pour les algorithmes génétiques. Nous avons procédé de la manière suivante [60] :

1. choisir un ensemble d'affections aux différentes opérations sur l'ensemble des machines existantes en respectant toutes les contraintes du problème,

2. trouver à l'aide de la programmation logique avec contraintes un ensemble d'ordonnancements en fonction de l'ensemble des affectations proposées avec toujours le respect de toutes les contraintes du problème traité,

3. après une sélection adéquate de la représentation ainsi que des différents opérateurs génétiques les algorithmes génétiques sont utilisés pour minimiser le Cmax de

notre problème d'ordonnancement en utilisant comme population initiale l'ensemble de solutions données par la programmation logique avec contraintes. Ces étapes peuvent être représentées par l'organigramme suivant :

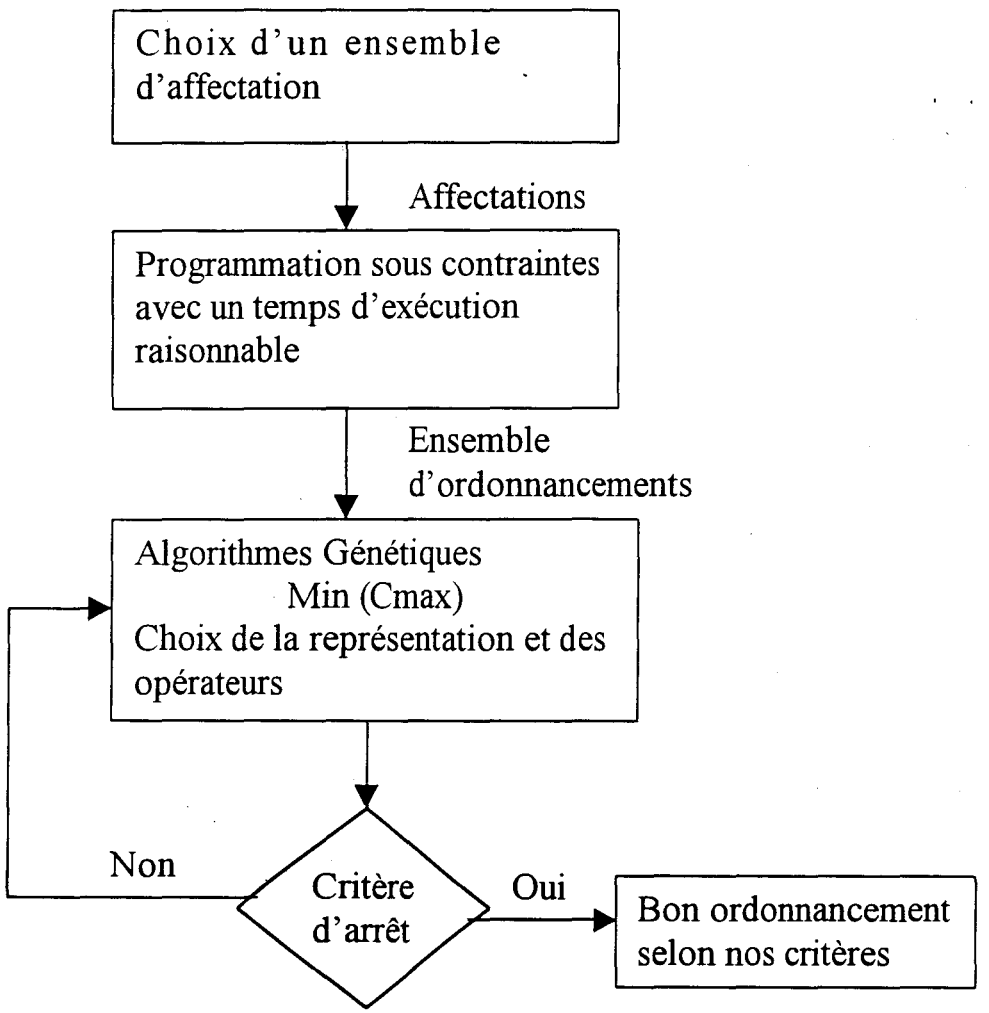

Figure III-3 : Organigramme d'hybridation de PLC - AG

$3^{ème} cas$: Créer la population initiale en utilisant une combinaison des règles de priorité. En effet, un ensemble de solutions sera construit en appliquant des règles de priorité qui arbitrent les conflits entre plusieurs opérations disponibles au pied de la machine (ordonnancement sans délai) ou sur le point d'arriver (ordonnancement actif). Chaque règle peut donner naissance à une ou plusieurs heuristiques. Nous avons utilisé dans notre approche un ensemble de règles classiques [59]. Nous citons ci-dessous les deux règles les plus utilisées [61] :

- SPT : (Shortest Processing Time) priorité à l'opération ayant la plus petite durée d'exécution,
- LPT : (Latest Processing Time) priorité à l'opération ayant la plus grande durée d'exécution,

Ainsi, l'ensemble d'ordonnancement admissible obtenu par ces règles sera utilisé comme point de départ par les algorithmes génétiques.

III.3.3 L'opérateur de sélection

La sélection consiste à choisir les individus à partir desquels on va créer la génération suivante. Nous utilisons dans notre approche la procédure classique basée sur le principe de la roulette, qui permet de retenir les individus les plus forts et d'exclure les plus faibles (voir le paragraphe II.2.2).

III.3.4 L'opérateur de croisement pour le CPM

Le croisement est un opérateur qui assure le brassage et la recombinaison des gènes parentaux pour former des descendants aux potentialités nouvelles, il correspond à un échange des matériels génétiques entre deux reproducteurs (parents) choisis d'une manière aléatoire parmi les meilleurs géniteurs, pour former deux nouveaux chromosomes (ou enfants). Nous présentons ici l'algorithme de notre opérateur de croisement [62] afin de construire d'abord l'enfant 1. Puis, pour obtenir l'enfant 2, il suffit d'inverser le rôle de parent 1 et de parent 2 à partir de 2.

Etape 1 : *Choisir d'une manière aléatoire deux individus (deux parents) et une machine*

Nous supposons que parent 1, parent 2 et la machine M_k aient déjà été choisis

Etape 2 : *Copier les opérations de la $k^{ième}$ ligne de parent 1 à la $k^{ième}$ ligne de l'enfant 1 (la $k^{ième}$ ligne représente la machine M_k)*

Etape 3 : *Parcourir les machines (les lignes) => $I = 1$*

Tantque ($I < m$) ***Faire*** *(m étant le nombre total des machines dans l'atelier)*

Si ($I \neq k$) **alors**

Copier les opérations non existantes de M_I *de parent 2 dans enfant 1*

Fin Si

$I = I + 1$

Fin Tant que

Etape 4: $I = K$

Si *(au moins une opération* $O_{i,OFj}$ *de l'enfant 1 est manquante de la ligne k)* **alors**

Si ($i = 1$) **alors**

Mettre $O_{i,OFj}$ *à la première position de la séquence d'opérations affectée à la machine* M_k

Fin si

Si ($i = x_{OFj}$) **alors**

Mettre $O_{i,OFj}$ *à la dernière position de la séquence courante d'opérations déjà affectée à la machine* M_k

Fin si

Si$(i \in]1, x_{OFj}[)$ **alors**

Trouver les rangs R1 de $O_{i-1,OFj}$ *et R2 de* $O_{i+1,OFj}$ *dans l'enfant 1*

Mettre $O_{i,OFj}$ *entre R1 et R2 (s'ils existent) dans la séquence courante de la machine* M_k *de l'enfant 1*

Fin si

Fin si

Fin

L'étape 4 de l'algorithme nous protège contre des éventualités d'avoir un ordonnancement irréalisable, principalement quand la taille du problème est importante. L'inadmissibilité de cet ordonnancement est due à la création d'un cycle dans le graphe des contraintes de précédence explicité par cet exemple :

Supposons que nous ayons à ordonnancer ces deux ordres de fabrication

OF 1 : O1,1 , O2,1

OF 2 : O1,2 , O2,2.

Le problème se pose quand nous obtenons la solution suivante :

M_1	(2, 2, ?)	(1, 1, ?)
M_2	(2, 1, ?)	(1, 2, ?)

Figure III-4 : Ordonnancement inadmissible

La machine M_1 doit en premier lieu exécuter l'opération 2 de l'ordre 2, malheureusement ceci ne peut être fait avant l'exécution de l'opération 1 de l'ordre 2. De la même façon la machine M_2 ne peut exécuter l'opération 2 de l'ordre 1 avant l'exécution de l'opération 1 de l'ordre 1. Cette situation de blocage conduit à un ordonnancement irréalisable. Cette situation est solutionnée :

1. soit par la prise en compte des contraintes du problème dans l'opérateur de croisement comme ce qui à été fait dans l'étape 4 de l'algorithme de croisement,

2. soit par la prise en compte des ces contraintes lors de la conception du codage ce que nous avons fait et que nous présenterons par la suite.

Nous allons illustrer ce croisement par cet exemple :

Exemple III.2 *Etape 1 : Nous supposons que parent* 1, *parent* 2 *et la machine* M_4 *ont déjà été choisis*

Parent 1

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	(2, 1, 3)	(3, 2, 5)
M_4	(3, 1, 5)		
M_5	(1, 3, 0)		

Parent 2

M_1	(1, 1, 0)		
M_2	(3, 2, 6)		
M_3	(2, 3, 2)		
M_4	(1, 2, 0)	(2, 1, 3)	(3, 1, 9)
M_5	(1, 3, 0)	(2, 2, 3)	

Etape 2 : Copier les opérations affectées à la machine M_4 de parent 1 (respectivement de parent 2) dans l'enfant 1 (respectivement dans l'enfant 2), à la même machine (M_4).

Enfant 1 en construction

M_1			
M_2			
M_3			
M_4	(3, 1, ?)		
M_5			

Enfant 2 en construction

M_1			
M_2			
M_3			
M_4	(1, 2, ?)	(2, 1, ?)	(3, 1, ?)
M_5			

Etape 3 : copier les opérations non existantes affectées aux autres machines (dans notre exemple il s'agit des machines M_1, M_2, M_3 et M_5) de parent 2 (respectivement de parent 1) dans l'enfant 1 (respectivement dans l'enfant 2).

Enfant 1 en construction

M_1	(1, 1, ?)		
M_2	(3, 2, ?)		
M_3	(2, 3, ?)		
M_4	(3, 1, ?)		
M_5	(1, 3, ?)	(2, 2, ?)	

Enfant 2 en construction

M_1	(2, 2, ?)		
M_2	(2, 3, ?)		
M_3	(1, 1, ?)	(3, 2, ?)	
M_4	(1, 2, ?)	(2, 1, ?)	(3, 1, ?)
M_5	(1, 3, ?)		

Etape 4 : Dans cette étape on se pose la question : **est-ce qu'il y a des opérations manquantes dans les enfants qu'on vient de construire ?**

La réponse est non pour l'enfant 2, les huit opérations à effectuer sont affectées et ordonnancées. Par contre, il manque deux opérations à l'enfant 1 qui n'ont pas été affectées. Il s'agit de l'opération 1 de l'ordre 2 et de l'opération 2 de l'ordre 1. Pour commencer, on place les premières opérations des gammes. Dans notre exemple il n'en existe qu'une seule, c'est l'opération 1 de l'ordre 2, elle sera placée à la première position sur la machine M_4 conformément à l'étape 4 de notre algorithme de croisement. On

déplace alors les opérations se trouvant sur cette machine d'un cran vers la droite et on libère ainsi la première position pour placer la première opération manquante on obtient alors le chromosme suivant :

Enfant 1 en construction

M_1	(1, 1, ?)		
M_2	(3, 2, ?)		
M_3	(2, 3, ?)		
M_4	(1, 2, ?)	(3, 1, ?)	
M_5	(1, 3, ?)	(2, 2, ?)	

On passe maintenant à la deuxième opération manquante, c'est l'opération 2 de l'ordre de fabrication 1. Cette dernière n'est ni la première ni la dernière dans la gamme opératoire de l'ordre 1. On cherche alors le rang de l'opération qui la précède et celle qui la suit. Celle qui la précède étant la première dans la gamme de l'ordre 1, elle se trouve à la position 1 sur la machine M_1 alors $R1 = 1$. L'opération qui la suit étant la troisième dans la gamme, elle se trouve après insertion de la première opération manquante à la deuxième position sur la machine M_4 ce qui donne $R2 = 2$. Vu que les opérations qui précèdent et celle qui suit l'opération à placer ne sont pas affectées à la même machine, nous prenons en compte lors du calcul de l'emplacement de l'opération manquante uniquement celles qui se trouvent affectées à la même machine. Dans notre cas, en prend en compte uniquement l'opération 3, son rang est égal à 2, l'opération manquante précède cette dernière dans sa gamme et doit s'exécuter avant elle, on place alors notre opération à la deuxième position sur la machine M_4 vu que la première est occupée par la machine précédemment placée. On déplace encore les autres opérations affectées à cette machine à partir de la position 2 d'un cran vers la droite.

Après avoir construit correctement les deux enfants (toutes les opérations de l'atelier sont affectées et les contraintes du problème sont respectées), nous passons au calcul des dates de début d'exécution de chaque opération en respectant les contraintes de ressources

et de précédence selon la méthode indiquée plus loin (voir le paragraphe III.5.1). Après ce processus de croisement, nous obtenons les deux enfants suivants.

Enfant 1

M_1	(1, 1, 0)		
M_2	(3, 2, 6)		
M_3	(2, 3, 2)		
M_4	(1, 2, 0)	(2, 1, 3)	(3, 1, 9)
M_5	(1, 3, 0)	(2, 2, 3)	

Enfant 2

M_1	(2, 2, 3)		
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	(3, 2, 5)	
M_4	(1, 2, 0)	(2, 1, 3)	(3, 1, 9)
M_5	(1, 3, 0)		

III.3.5 Les opérateurs de mutation pour le CPM

L'opérateur de mutation change quelques caractéristiques des individus résultant du croisement et qui ont hérité des caractéristiques de leurs parents. La mutation permet de générer de nouvelles configurations qui ne seraient pas atteignables en se basant simplement sur le croisement. Cet opérateur permet d'éviter une perte irréparable de la diversité. Sans elle, on aurait une population uniforme, incapable d'évoluer. Nous avons utilisé dans notre approche deux opérateurs de mutation [62] :

Mutation par affectation

Dans ce cas de mutation nous utilisons la flexibilité de notre problème, les opérations peuvent être exécutées par une ou plusieurs machines. La mutation par affectation consiste à réaffecter une opération prise d'une manière aléatoire à une autre machine si ceci est possible . L'algorithme de cette procédure est le suivant :

1. *Un chromosome et une opération sont choisis aléatoirement*
2. *Réaffecter l'opération précédemment sélectionnée sur une autre machine à la même position si ceci est possible (si cette nouvelle machine peut exécuter cette opération) tout en respectant les contraintes de précédence et de ressources.*

Illustrons cette mutation sur un exemple.

Exemple III.3 *Supposons qu'à l'issu de l'étape* 1 *de l'algorithme, le chromosome ci-dessous ainsi que l'opération* 2 *de l'ordre de fabrication* 1 *ont été aléatoirement sélectionnés. Cette opération est affectée sur la machine* M_3 *et elle est ordonnancée à la deuxième position.*

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	**(2, 1, 3)**	(3, 2, 5)
M_4	(3, 1, 5)		
M_5	(1, 3, 0)		

Figure III-5 : Chromosome avant mutation par affectation

Nous allons maintenant réaffecter cette opération sur une machine qui sera aussi choisie d'une façon aléatoire. Nous supposons que ce choix nous donne la machine M_5. La solution obtenue après l'application de cet opérateur de mutation est la suivante :

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	(3, 2, 3)	
M_4	(3, 1, 7)		
M_5	(1, 3, 0)	**(2, 1, 3)**	

Figure III-6 : Chromosome après mutation par affectation

Mutation par permutation

La mutation par permutation consiste à permuter une ou plusieurs opérations (à choix multiple sur les machines) sélectionnées d'une manière aléatoire. Contrairement à la mutation par affectation qui fait déplacer une seule opération, la mutation par permutation opère sur au moins deux opérations. Cette mutation suit l'algorithme suivant :

1. *Un chromosome, deux machines, une position et une direction sont aléatoirement choisis.*

2. **Si** *(direction = gauche)* **alors**

   Faire une permutation à gauche de la position sélectionnée.

   Si *(direction = droite)* **alors**

   Faire une permutation à droite de la position sélectionnée.

   Fin

Nous allons illustrer cette mutation sur l'exemple suivant :

Exemple III.4 *Supposons que dans le chromosome suivant, les machines M_1 et M_3 ainsi que la position 2 ont été sélectionnées d'une manière aléatoire.*

Position	1	2	3
M_1	**(1, 2, 0)**	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	**(1, 1, 0)**	(2, 1, 3)	(3, 2, 5)
M_4	(3, 1, 5)		
M_5	(1, 3, 0)		

Figure III-7 : Chromosome avant mutation par permutation

1^{er} cas : Supposons que le choix aléatoire nous a donné direction = gauche, dans ce cas on fait des permutations à gauche de la position 2 entre les machines M_1 et M_3, ce qui donne le chromosome suivant :

M_1	**(1, 1, 0)**	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	**(1, 2, 0)**	(2, 1, 5)	(3, 2, 7)
M_4	(3, 1, 7)		
M_5	(1, 3, 0)		

Figure III-8 : Chromosome après une mutation à gauche

$2^{ème}$ cas : Supposons maintenant que le choix aléatoire donne direction = droite, la permutation ce fait donc à droite de la position 2, ce qui donne la solution suivante :

M_1	(1, 2, 0)	(2, 2, 1)	**(3, 2, 3)**
M_2	(2, 3, 2)		
M_3	(1, 1, 0)	(2, 1, 3)	
M_4	(3, 1, 5)		
M_5	(1, 3, 0)		

Figure III-9 : Chromosome après une mutation à droite

Remarque III.1 *la permutation doit prendre en compte le respect des contraintes de précédence et de ressources de notre problème. Il se pourrait lors d'une permutation qu'il y'ait violation des contraintes de précédence, il faut avant de placer chaque opération s'assurer que les contraintes du problème sont respectées, par conséquent déplacer les opérations permutées afin de respecter ces contraintes.*

Nous illustrons ce cas sur l'exemple suivant :

Exemple III.5 *A l'issu de l'étape* 1, *les machines* M_3 *et* M_4 *ainsi que la position* 2 *ont été sélectionnées aléatoirement. Nous gardons toujour le même chromosome*

Nous choisissons direction = gauche (permutation à gauche de la position 2) ce qui donne

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	**(3, 1, ?)**	(2, 1, ?)	(3, 2, ?)
M_4	**(1, 1, 0)**		
M_5	(1, 3, 0)		

Figure III-10 : Chromosome non valide après mutation par permutation

On obtient un ordonnancement inadmissible, la machine M_3 est totalement bloquée car il y'a violation de la contrainte de précédence de l'ordre 1. La dernière opération est placer au pied de la machine avant celle qui la précède dans la gamme. Dans ce cas il faut procéder au déplacement de cette opération. Ceci se fera en appliquant l'étape 4 de l'algorithme de croisement précédemment présenté. Nous obtenons alors la solution suivante :

M_1	(1, 2, 0)	(2, 2, 1)	
M_2	(2, 3, 2)		
M_3	(2, 1, 7)	**(3, 1, 9)**	(3, 2, 10)
M_4	**(1, 1, 0)**		
M_5	(1, 3, 0)		

Figure III-11 : Chromosome valide après mutation par permutation

Ces deux opérateurs de mutation peuvent être utilisées de façon séparée. Un des opérateurs peut être tiré au hasard avec une probabilité P_{mut1} de sélectionner l'opérateur de mutation par affectation ou la probabilité $1 - P_{mut1}$ de sélectionner l'opérateur de mutation par permutation.

III.4 Deuxième approche

La résolution des problèmes d'ordonnancement est fortement conditionnée par les contraintes très fortes qu'il faut respecter. Toute méthode de résolution doit prendre en compte cette spécificité pour aboutir à un résultat acceptable. Lors de la mise au point de notre méthode de résolution basée sur les algorithmes à stratégie d'évolution, nous avons en premier lieu intégré ces contraintes dans la conception des opérateurs génétiques à savoir le croisement et la mutation. La deuxième possibilité consiste à les intégrer dans la conception même du codage et alléger ainsi les opérateurs de croisement et de mutation.

Notre préoccupation majeure lors de la conception de notre deuxième codage nommé codage parallèle des ordres de fabrication (CPOF), était de rester fidèle aux objectifs

que nous nous sommes fixés : concevoir un codage robuste qui représente directement un ordonnancement réalisable, respectant toutes les contraintes et donnant toutes les informations nécessaires au chef d'atelier. La souplesse de ce codage permettra une gestion plus satisfaisante de la production et ce quels que soient les opérateurs génétiques appliqués. Ainsi le CPOF réalisé dans cet esprit se présente comme suit [63] :

III.4.1 Codage parallèle des ordres de fabrication (CPOF)

Le chromosome est représenté par une matrice, chaque ligne représente la gamme de chaque ordre de fabrication. Chaque colonne de cette ligne (représentant une opération) contient deux termes. Le premier indique le numéro de la machine qui est affectée à l'exécution de cette opération, le second représente la date de début d'exécution de l'opération concernée sur la machine qui lui a été affectée. La configuration du codage parallèle des ordres de fabrication est la suivante :

OF 1	(M_1, t_{M1})	(M_2, t_{M2})	. . .
OF 2	(M_3, t_{M3})	(M_1, t_{M1})	
. . .		. . .	
OF n	(M_2, t_{M2})	(M_5, t_{M5})	

Figure III-12 : Codage parallèle des ordres de fabrication

Plus généralement, une ligne d'un individu se présente comme suit :

OF_j : (M_k, t_{Mk}), (M_l, t_{Ml}), . . .

Où chaque colonne (opération) de cet OF contient la machine sur laquelle doit être exécutée cette opération et sa date de début d'exécution. Pour l'OF_j l'opération 1 est exécutée sur la machine M_k à la date t_{Mk}, l'opération 2 est exécutée sur la machine M_l à la date t_{Ml}.

Exemple III.6 *L'application de ce codage à l'exemple représenté par la figure III-2 donne le chromosome suivant :*

OF 1	(M_3, 0)	(M_3, 3)	(M_4, 5)
OF 2	(M_1, 0)	(M_1, 1)	(M_3, 5)
OF 3	(M_5, 0)	(M_2, 2)	

Figure III-13 : Exemple de représentation du CPOF

Cette représentation permet de prendre en compte les contraintes de précédence dans la construction même du chromosome, puisque les ordres de fabrication sont placés dans l'ordre de leurs gammes de production. Les contraintes de ressources seront respectées lors du calcul des dates de début d'exécution de chaque opération (voir le paragraphe III.5.2).

Pour les problèmes qui n'ont pas de contraintes à respecter, le choix aléatoire de la population de départ est souvent utilisé. Malheureusement pour la majeure partie des problèmes d'ordonnancement le respect d'un certain nombre de contraintes est une partie intégrante de leur résolution. On doit faire appel à une série de méthodes pour aider à mettre au point cette population de départ (voir le paragraphe III.3.2), par contre si les contraintes du problème sont prises en compte lors de la conception du codage ce que nous avons fait dans le codage parallèle des ordres de fabrication, il est alors possible de créer cette population initiale d'une façon totalement aléatoire. Toutefois, il serait profitable d'utiliser aussi les solutions trouvées par les autres méthodes ce qui permettrait d'accélérer la convergence et de réduire considérablement le temps d'exécution de l'algorithme utilisé.

III.4.2 Opérateurs de croisement pour le CPOF

Le but du croisement reste toujours le même à savoir obtenir par mélange de solutions d'autres chromosomes susceptibles d'améliorer le résultat. Nous présenterons dans ce paragraphe deux opérateurs de croisement ; croisement ligne qui manipule les ordres de fabrication et le croisement colonne manipulant plutôt un ensemble d'opérations appartenant à des ordres différents.

Croisement ligne

c'est le premier croisement que nous avons mis au point, l'algorithme de cet opérateur est le suivant :

Etape 1 : *Choisir aléatoirement deux parents et un ordre de fabrication*

Nous supposons que parent 1, parent 2 et l'ordre F ont déjà été choisis

Etape : *Les opérations de l'OF F recevront les mêmes machines affectées à l'exécution des opérations du même ordre (l'OF F) de parent 1*

Etape 3 : *Parcourir les ordres de fabrication (les lignes)* $L = 1$

Tant que $((L < n)$ *et* $(L \neq F))$ ***faire*** *(n est le nombre total des OF)*

Copier les machines affectées aux opérations des ordres de fabrication de parent 2 aux même ordres de l'enfant 1.

$L = L + 1$

Fin Tant que

Fin.

Pour obtenir l'enfant 2, aller à l'étape 2 et inverser le rôle de parent 1 et parent 2. Illustrons cet opérateur par l'exemple qui suit :

Exemple III.7 *Etape 1 : Nous supposons que parent* 1, *parent* 2 *et l'ordre* 2 *(la ligne* 2 *de la matrice qui représente le chromosome) ont déjà été choisis.*

Parent 1

OF 1	(M_3, 0)	(M_3, 3)	(M_4, 5)
OF 2	(M_1, 0)	(M_1, 1)	(M_3, 5)
OF 3	(M_5, 0)	(M_2, 2)	

Parent 2

OF 1	(M_1, 0)	(M_4, 3)	(M_4, 9)
OF 2	(M_4, 0)	(M_5, 3)	(M_2, 6)
OF 3	(M_5, 0)	(M_3, 2)	

Etape 2 : Les opérations de l'ordre 2 de l'enfant 1 (respectivement de l'enfant 2) recevrons les même machines affectées à l'exécution des opérations du même ordre (l'OF 2) de parent 1 (respectivement de parent 2).

Enfant 1 en construction

OF 1			
OF 2	(M_1, ?)	(M_1, ?)	(M_3, ?)
OF 3			

Enfant 2 en construction

OF 1			
OF 2	(M_4, ?)	(M_5, ?)	(M_2, ?)
OF 3			

Etape 3 : Copier les machines affectées aux opérations des autres ordres de fabrication (dans notre exemple il s'agit des odres 1 et 3) de parent 2 (respectivement de parent 1) aux même ordres de l'enfant 1 (respectivement de l'enfant 2).

Enfant 1 en construction

OF 1	(M_1, ?)	(M_4, ?)	(M_4, ?)
OF 2	(M_1, ?)	(M_1, ?)	(M_3, ?)
OF 3	(M_5, ?)	(M_3, ?)	

Enfant 2 en construction

OF 1	(M_3, ?)	(M_3, ?)	(M_4, ?)
OF 2	(M_4, ?)	(M_5, ?)	(M_2, ?)
OF 3	(M_5, ?)	(M_2, ?)	

On remarque ici que les deux enfants sont parfaitement valides, toutes les contraintes de précédence ont été respectées. On peut calculer maintenant les dates de début d'exécution de chaque opération en tenant compte des contraintes de ressources. Après ce processus de croisement nous obtenons les deux ordonnancements suivants :

Enfant 1

OF 1	(M_1, 0)	(M_4, 1)	(M_4, 7)
OF 2	(M_1, 1)	(M_1, 2)	(M_3, 4)
OF 3	(M_5, 0)	(M_3, 2)	

Enfant 2

OF 1	(M_3, 0)	(M_3, 3)	(M_4, 5)
OF 2	(M_4, 0)	(M_5, 3)	(M_2, 11)
OF 3	(M_5, 0)	(M_2, 2)	

Pour la construction du croisement ligne, nous avons résonné sur les ordres de fabrication, alors que dans la deuxième variante nous allons nous concentrer sur les opérations prises individuellement dans chaque ordre de fabrication.

Croisement colonne

C'est notre deuxième croisement, son algorithme se présente comme suit :

Etape 1 : *Choisir aléatoirement deux parents et une opération*

Nous supposons que parent 1, parent 2 et l'opération i ont été choisis

Etape 2 : *L'opération i de tous les OF de l'enfant 1 recevra les mêmes machines affectées à l'opération i de tous les OF de parent 1.*

Etape 3 : *Parcourir toutes les autres opérations (les colonnes)* $C = 1$

Tant que $((C < I)$ *et* $(C \neq i))$ ***faire*** *(I est le nombre total d'opérations)*

Copier le reste des machines affectées aux autres opérations de tous les OF de parent 2 dans les mêmes opérations de l'enfant 1.

$C = C + 1$

Fin Tant que

Fin.

Pour obtenir l'enfant 2 aller à l'étape 2 et inverser le rôle de parent 1 et parent 2. Illustrons cet opérateur par l'exemple suivant :

Exemple III.8 *Etape 1 : L'opération 2 a été choisie aléatoirement en utilisant les parents de l'exemple 3.7.*

Etape 2 : L'opération 2 de tous les ordres de fabrication de l'enfant 1 (respectivement de l'enfant 2) recevrons les mêmes machines affectées à l'opération 2 de tous les ordres fabrication de parent 1 (respectivement de parent 2).

Enfant 1 en construction

OF 1		(M_3, ?)	
OF 2		(M_1, ?)	
OF 3		(M_2, ?)	

Enfant 2 en construction

OF 1		(M_4, ?)	
OF 2		(M_5, ?)	
OF 3		(M_3, ?)	

Etape 3 : Copier le reste des machines affectées aux autres opérations de tous les ordres de fabrication de parent 2 (respectivement de parent 1) dans les mêmes opérations de l'enfant 1 (respectivement de l'enfant 2).

Enfant 1 en construction

OF 1	(M_1, ?)	(M_3, ?)	(M_4, ?)
OF 2	(M_4, ?)	(M_1, ?)	(M_2, ?)
OF 3	(M_5, ?)	(M_2, ?)	

Enfant 2 en construction

OF 1	(M_3, ?)	(M_4, ?)	(M_4, ?)
OF 2	(M_1, ?)	(M_5, ?)	(M_3, ?)
OF 3	(M_5, ?)	(M_3, ?)	

Les deux enfants découlant de cet opérateur de croisement sont parfaitement valides, on peut passer maintenant au calcul des dates de début d'exécution de chaque opération sur la machine qui lui a été affectée. La procédure étant la même que pour les autres opérateurs. Finalement, nous obtenons les deux ordonnancements suivants :

Enfant 1

OF 1	(M_1, 0)	(M_3, 1)	(M_4, 3)
OF 2	(M_4, 0)	(M_1, 3)	(M_2, 11)
OF 3	(M_5, 0)	(M_2, 2)	

Enfant 2

OF 1	(M_1, 0)	(M_4, 1)	(M_4, 7)
OF 2	(M_1, 1)	(M_5, 2)	(M_3, 5)
OF 3	(M_5, 0)	(M_3, 2)	

Des variantes de ces deux opérateurs de croisement peuvent être conçues. Nous pouvons par exemple pour accélérer la convergence, choisir un des deux parents en fonction de sa fonction coût, on prendra alors le meilleur chromosome comme un des deux reproducteurs.

III.4.3 Opérateur de mutation pour le CPOF

Le rôle essentiel de la mutation est d'introduire une certaine diversification dans la population que l'opérateur de croisement ne peut apporter. Nous voulions profiter de ce phénomène pour améliorer le résultat obtenu. Nous présenterons dans ce paragraphe un opérateur de mutation qui nous permettra d'équilibrer les charges des machines. L'algorithme est le suivant :

Etape 1 : *Choisir un chromosome et une opération de l'ensemble des opérations affectées à la machine la plus chargée*

Etape 2 : *Affecter à cette opération la machine la moins chargée si ceci est possible.*

Nous allons illustrer cette mutation sur l'exemple suivant :

Exemple III.9 *Etape 1 : Supposons que le chromosome suivant ait été choisi :*

Parent

OF 1	(M_1, 0)	(M_4, 3)	(M_4, 9)
OF 2	(M_4, 0)	(M_5, 3)	(M_2, 6)
OF 3	(M_5, 0)	(M_3, 2)	

Nous allons calculer maintenant le temps d'occupation de ces machines il représente leur charge horaire exprimée en unités de temps.

Machines	Charge
M_1	1
M_2	5
M_3	2
M_4	13
M_5	5

Figure III-14 : Charge des machines du chromosome choisi avant mutation

Nous remarquons que la machine M_4 est la plus chargée par contre la machine M_1 est la moins chargée. Supposons que l'opération 2 de l'OF 1 ait été choisie aléatoirement parmi les opérations à exécuter sur la machine la plus chargée à savoir la machine M_4.

OF 1	(M_1, 0)	**(M_4, 3)**	(M_4, 9)
OF 2	(M_4, 0)	(M_5, 3)	(M_2, 6)
OF 3	(M_5, 0)	(M_3, 2)	

Figure III-15 : Chromosome avant mutation

Etape 2 : Réaffecter à cette opération la machine la moins chargée, dans notre exemple il s'agit de la machine M_1.

Après le calcul des dates de début d'exécution de chaque opération, le nouveau chromosome se présente comme suit :

OF 1	(M_1, 0)	**(M_1, 1)**	(M_4, 4)
OF 2	(M_4, 0)	(M_5, 3)	(M_2, 6)
OF 3	(M_5, 0)	(M_3, 2)	

Figure III-16 : Chromosome après mutation

Recalculons maintenant les nouvelles charges des machines.

Machines	Charge
M_1	4
M_2	5
M_3	2
M_4	7
M_5	5

Figure III-17 : Charge des machines du chromosome après mutation

Nous pouvons dire que cet opérateur de mutation contribue à équilibrer la charge temporelle des machines.

III.5 Calcul de la date de début d'exécution

Une bonne utilisation des algorithmes génétiques dans les problèmes combinatoire passe essentiellement par une mise au point d'un codage efficace, ce dernier doit être à la fois robuste, simple et lisible. Lors de la construction de nos codages, nous avons pris en compte ces trois paramètres, en intégrant en plus des informations utiles à savoir la date de début d'exécution de chaque opération sur chaque machine dans chaque chromosome. En effet, le chef d'atelier peut directement exploiter l'ensemble d'ordonnancement proposé sans aucune autre mise au point. Nous présenterons dans ce qui suit les algorithmes de calcul des dates de début d'exécution de chaque opération sur les machines concernées après la fin de la construction de la population courante.

III.5.1 Algorithme de calcul des dates pour le CPM

T_F : vecteur contenant la date de fin d'exécution de la dernière opération ordonnancée de l'ordre F (Vecteur de taille n).

D_{Mk} : vecteur contenant la date de fin d'exécution de la dernière opération ordonnancée sur la machine M_k (vecteur de taille m).

Début

$i = 1$

Tant que $(i < I)$ ***faire***

$j = 1$

Tant que $(j < n)$ **faire**

Si $(O_{i,OFj}$ *est la* $1^{ère}$ *opération de l'ordre* $OF_j)$ **alors**

Calculer $(t_{i,OFj,Mk})$

Mise à jour $(T_F[j], D_{Mk}[i])$

Fin Si

Sinon

Si $(O_{i-1,OFj}$ *est calculé*) **alors**

Calculer ($t_{i,OFj,Mk}$)

Mise à jour ($T_F[j]$, $D_{Mk}[i]$)

Fin Si

Fin Sinon

Fin Tant que

Fin Tant que

Fin.

Procédure *(calcule* ($t_{iOF,j,Mk}$))

Début

Si ($T_F[j] < D_{Mk}[i]$) **alors**

$t_{i,OFj,Mk} = D_{Mk}[i]$

Sinon

$t_{i,OFj,Mk} = T_F[j]$

Fin

Procédure *(Mise à jour* ($T_F[j]$, $D_{Mk}[i]$))

Début

$T_F[j] = t_{i,OFj,Mk}+p_{i,OFj,Mk}$ *(Mise à jour de* T_F*)*

$D_{Mk}[i] = t_{i,OFj,Mk}+p_{i,OFj,Mk}$ *(Mise à jour de* D_{Mk}*)*

Fin

III.5.2 Algorithme de calcul des dates pour le CPOF

Vu que les opérations sont ordonnées d'après leurs gammes respectives. Le calcul se fait en commençant toujours par la première opération de chaque OF, il se fera donc ligne par ligne, selon la priorité donnée par le chef d'atelier aux différents ordres de fabrication. Toutes les combinaisons sont possibles ; nous pouvons commencer par le 1^{er} ordre ($1^{ère}$ ligne) ou le $2^{ème}$ ordre, etc. L'algorithme se présente comme suit :

Nous gardons les mêmes définitions pour les vecteurs T_F et D_{Mk}.

Début

$i = 1$

Tant que $(i < I)$ ***faire***

$j = 1$

Tant que $(OF_j < n)$ **faire**

Calculer $(t_{i,OFj,Mk})$

Mise à jour $(T_F[j], D_{Mk}[i])$

Fin Tant que

Fin Tant que

Fin.

III.6 Calcul de la fonction d'évaluation

Le succès grandissant de l'utilisation des algorithmes génétiques dans des différents domaines est dû principalement à leur capacité à se reconfigurer et de s'auto-organiser face à un environnement inconnu. Le faible taux d'informations demandé sur le problème traité leur donne un atout considérable par rapport aux autres méthodes classiques d'optimisation telles que la méthode de Newton ou du simplex [40]. En effet, les algorithmes génétiques ne demandent qu'une mesure de la valeur de la fonction étudiée nommée fonction coût ou fonction d'adéquation (traduction du terme anglais fitness).

La fonction coût est le lien entre les algorithmes génétiques et le problème qu'on tente de résoudre. Elle reçoit le chromosome comme entrée et retourne en sortie, souvent sous forme de nombre, une mesure de la performance de ce dernier.

L'évaluation des chromosomes est une partie très importante dans l'application des algorithmes génétiques à l'optimisation de tout problème, elle permet d'indiquer quelles sont les solutions à retenir ou à rejeter selon un critère préalablement défini par l'utilisateur. Dans le cadre de notre thèse, nous avons choisi de minimiser le temps total d'ordonnancement des ordres de fabrication noté généralement Cmax ou Makespan. Ce

choix est motivé par le fait que ce dernier permet en plus de l'optimisation du temps de fabrication, une utilisation efficace des ressources [1]. Le temps total d'ordonnancement s'exprime de deux manières :

$1^{ère}$) il correspond à la date à la quelle la machine la plus tardive fini d'exécuter sa dernière opération.

$2^{ème}$) il correspond à la date à la quelle l'ordre de fabrication le plus tardif prend fin.

Le calcul du Cmax ce fait selon l'algorithme suivant :

Etape 1 : **Début** $M_c = M1$

Etape 2 : **Tant que** $(M_c < m)$ ***faire*** (M_c : la machine courante)

Calculer $C\max M_c = c_{i,OFj,Mc}$ (date de fin d'exécution de la dernière opération ordonnancer sur la machine M_c)

$M_c = M_{c+1}$

Fin Tant que

Etape 3 : $Cmax = Max(C\max 1, C\max 2, \ldots, C\max m)$

Cet algorithme correspond à la première définition du Cmax donnée précédemment, elle intervient au niveau des machines. Cet algorithme est plutôt applicable au codage parallèle des machines vu sa configuration. Par contre, malgré que le raisonnement reste le même pour le codage parallèle des ordres de fabrication, il faut dans ce cas faire quelques petites modifications sur cet algorithme, il devient alors :

Etape 1 : **Début** $F = 1$

Etape 2 : **Tant que** $(F < n)$ **faire**

Calculer $C\max F = c_{i,OFF,Mk}$ (date de fin d'exécution de la dernière opération ordonnancer de l'OF F)

$F = F + 1$

Fin Tant que

Etape 3 : $Cmax = Max(C\max 1, C\max 2, \ldots, C\max n)$

Ainsi, nous obtenons dans les deux cas de figures un $Cmax$ qui représente le temps total d'ordonnancement.

L'organigramme mettant en oeuvre nos approches basées sur les algorithmes évolutionniste est présenté dans la figure suivante :

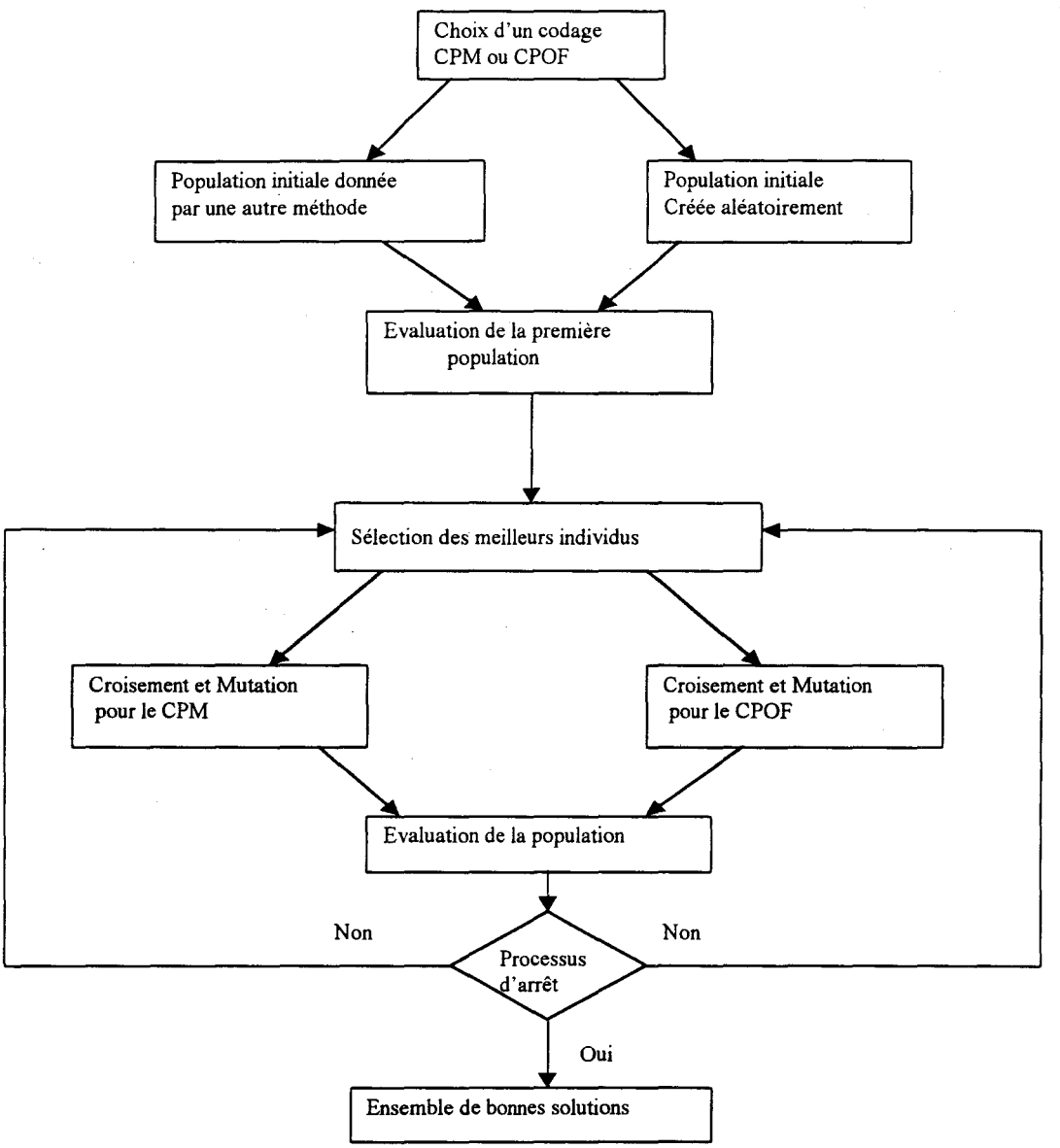

Figure III-18 : Organigramme de nos approches évolutionnistes

III.7 Conclusion

Deux approches d'algorithmes évolutionnistes ont été présentées dans ce chapitre, pour la résolution conjointe du problème d'ordonnancement et d'affectation de type job shop flexible de façon à minimiser le temps total d'ordonnancement. Ces deux approches se différencient essentiellement par leurs codages.

Dans le premier codage, le chromosome donne une information visible de la charge des machines et sur la répartition des opérations sur ces dernières, il permet au chef d'atelier de mieux utiliser son parc de machines. Pour démarrer la recherche, la création de la population initiale est faite de trois manières différentes, en utilisant des processus d'hybridations soit avec les méthodes classiques, la programmation sous contraintes soit encore en utilisant un ensemble d'heuristiques issues des règles de priorités classique. Nous avons montrer aussi comment nous avons pris en compte dune part, toutes les contraintes de précédence lors de la conception des opérateurs de croisement et de mutation, et d'autre part les contraintes de ressources lors du calcul des différentes dates de début d'exécution de chaque opération sur la machine qui lui a été préalablement attribuée.

Dans la deuxième approche, le chromosome donne plutôt une information lisible sur la répartition des machines sur les différentes opérations de la construction. La prise en compte de toutes les contraintes étant remarquablement intégrée dans le chromosome, les opérateurs génétiques sont ainsi allégé permettant par conséquent leur amélioration en vue d'une accélération de la convergence et/ou une utilisation efficace des ressources.

Le chapitre suivant va illustrer ces deux approches sur des exemples de tailles et de complexités différentes.

Chapitre IV

Application et résultats

Dans ce chapitre, nous allons appliquer notre approche basée sur les algorithmes à stratégie d'évolution pour résoudre les problèmes d'ordonnancement de type job shop flexible présentés dans le chapitre 1. Les résultats expérimentaux concernent des problèmes de tailles et de complexités différentes.

Dans la première partie, nous appliquons le premier codage à des problèmes de différentes tailles en détaillant les différentes méthodes permettant l'obtention de la population initiale.

La deuxième partie sera consacrée à l'application de la deuxième approche, nous venons aussi comment il est possible grâce aux opérateurs génétiques d'accélérer la convergence de l'algorithme et de répartir au mieux la charge de travail sur l'ensemble du parc de machines.

IV.1 Première approche

Nous allons en premier lieu appliquer le premier codage à plusieurs exemples de tailles et de flexibilités différentes.

IV.1.1 Exemple de petite taille

Nous proposons en premier lieu de traiter un exemple de petite taille, 3 ordres de fabrication et 5 machines. Ce dernier présente une flexibilité totale, chaque opération peut être exécutée par n'importe quelle machine. Les données de ce problème sont présentées dans le chapitre précédent (voir exemple III.1). Nous appliquons deux approches différentes pour créer la population initiale

1^{er} cas : Une première solution est trouvée en utilisant la décomposition temporelle. Cette solution est représentée par le digramme de Gantt suivant :

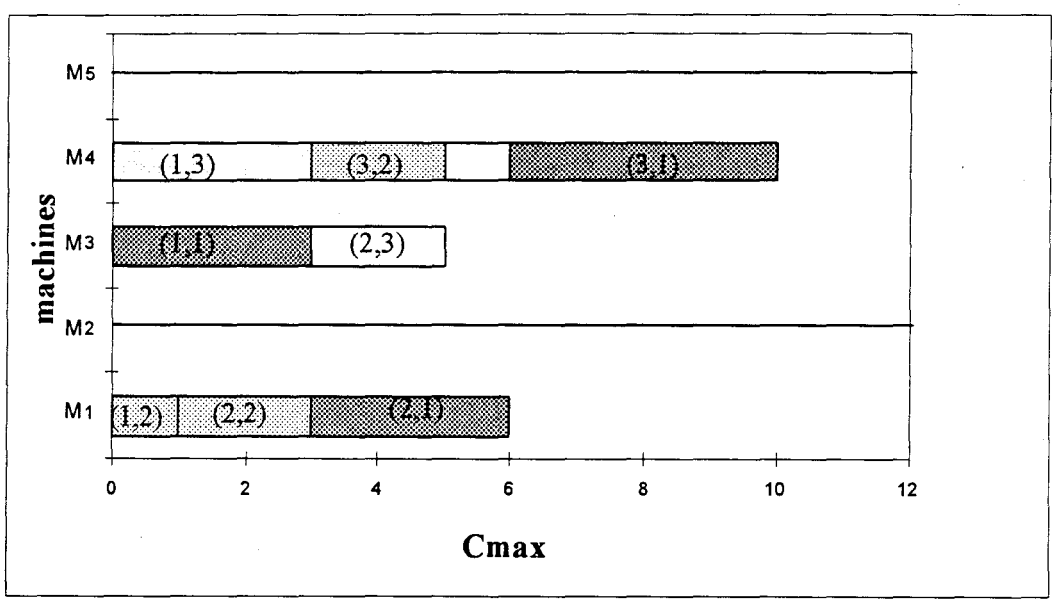

Figure IV-1 : Diagramme de Gantt du bon ordonnancement obtenu

Cette solution est dupliquée et modifiée en utilisant l'opérateur de mutation en procédant comme suit :

- appliquer l'opérateur de mutation par affectation avec un taux de 75 %,
- appliquer l'opérateur de mutation par permutation avec un taux de 25 %,
- arrêt du processus après l'obtention de 50 individus valides.

Le Cmax de cette première population est représenté par la figure suivante :

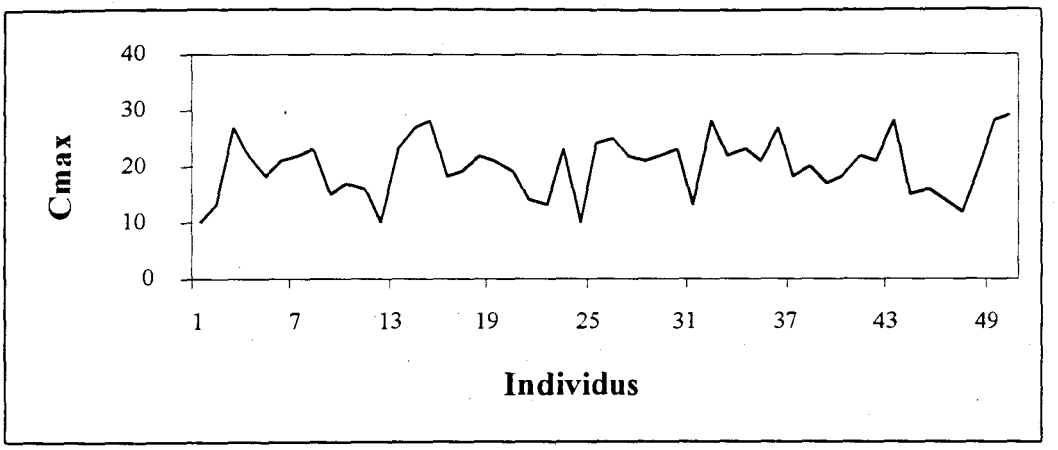

Figure IV-2 : Cmax de la première population obtenue par duplication

$2^{ème}$ cas : La population initiale est obtenue par la programmation avec contraintes.

- choisir un ensemble d'affectations des différentes opérations sur l'ensemble des machines susceptibles de les exécuter,
- trouver un ordonnancement admissible en appliquant la programmation avec contraintes.

On a obtenu 20 solutions.

Le Cmax de l'ensemble des solutions trouvées est représenté par la figure suivante :

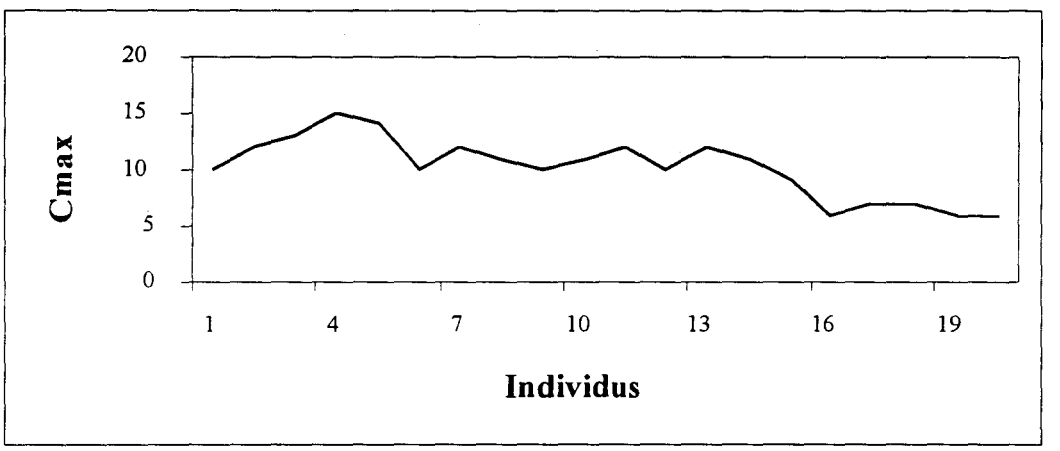

Figure IV-3 : Cmax de la première population obtenue par la PC

Les paramètres génétiques de nos algorithmes sont indiqués ci dessous :

pour le premier cas

- Taille de la population = 50
- Probabilité de croisement = 0,65
- Probabilité de mutation par affectation = 0,015
- Probabilité de mutation par permutation = 0,015
- Nombre de générations = 60.

Pour le deuxième cas

- Taille de la population = 20
- Probabilité de croisement = 0,65
- Probabilité de mutation par affectation = 0,025
- Probabilité de mutation par permutation = 0,025
- Nombre de générations = 60.

Nous remarquons d'après les courbes représentant les deux populations initiales, que la qualité des individus dans le deuxième cas (population trouvée par la programmation logique avec contrainte) est meilleure. On trouve déjà des solutions plus proches de l'optimum que celles trouvées par la décomposition temporelle. Nous nous trouvons très probablement dans une zone contenant la meilleure valeur que nous pouvons espérer obtenir. Par conséquent une taille de population moins importante que dans le premier cas suffit à améliorer la solution, elle permet en plus de réduire considérablement le temps de calcul. Voici la courbe d'évolution du Cmax en fonction des générations dans les deux cas de la création de la population initiale.

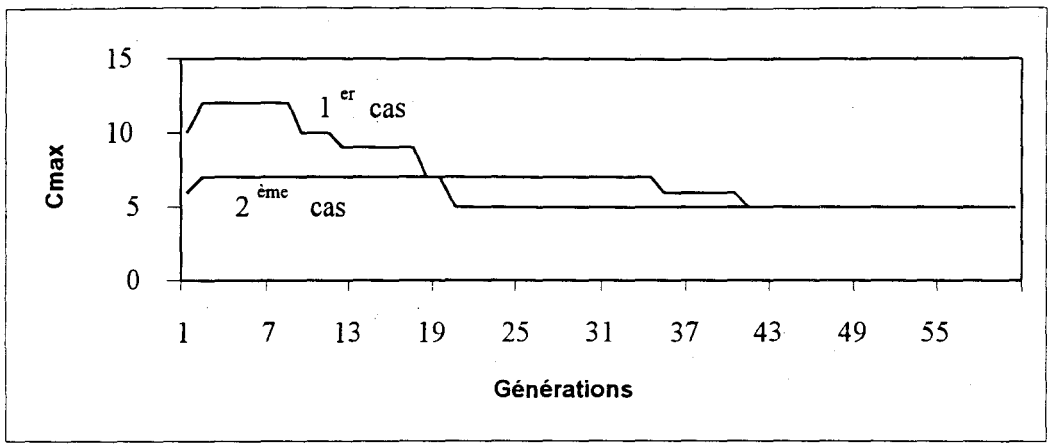

Figure IV-4 : Evolution du Cmax pour le problème 3 x 5.

Il apparaît très clairement que la qualité de la population initiale influence la convergence de l'algorithme. Dans le deuxième cas, on s'approche plus vite de l'optimum, la valeur de 5 unités de temps est obtenue après une vingtaine de générations, alors que dans le premier cas elle n'est atteinte qu'après une quarantaine de générations. L'ordonnancement obtenu est présenté ci dessous :

M_1	(1, 1, 0)	(1, 2, 1)	(2, 2, 2)
M_2			
M_3	(2, 1, 1)	(3, 1, 3)	(3, 2, 4)
M_4			
M_5	(1, 3, 0)	(2, 3, 2)	

Figure IV-5 : Meilleur individu trouvé

Ainsi, la solution donnée sous cette forme est parfaitement lisible pour le chef d'atelier. Elle représente une reproduction fidèle du diagramme de Gantt présenté ci dessous :

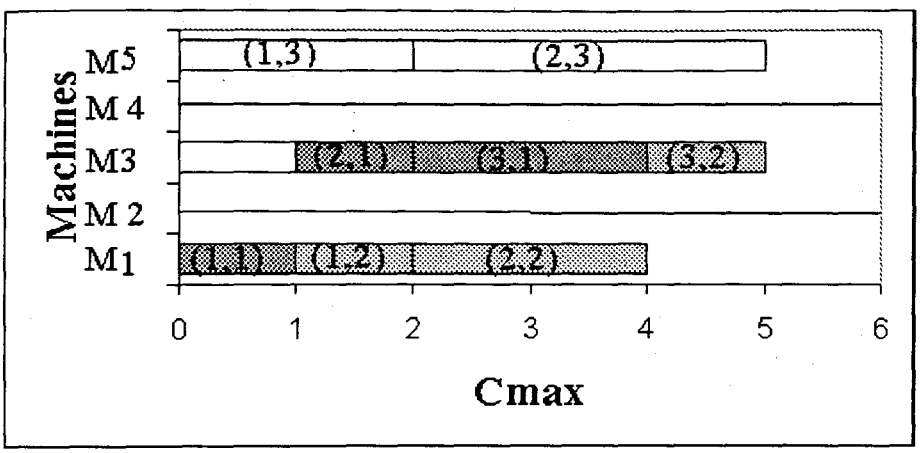

Figure IV-6 : Diagramme de Gantt du meilleur ordonnancement obtenu pour le CPM

Vu cet exemple, nous pouvons dire que notre approche fonctionne correctement pour les problèmes de petite taille. **Qu'en est-il des problèmes de taille plus grande ?**

IV.1.2 Exemple de taille importante

On se propose maintenant de résoudre un problème de taille importante. L'exemple comporte 10 ordres de fabrication, chaque OF comporte 3 opérations, toutes les opérations peuvent être exécutées par l'ensemble des 10 machines existant dans le parc avec des temps d'exécution différents (flexibilité totale). Le problème se résume à ordonnancer les 30 opérations sur les 10 machines disponibles en respectant les différentes gammes de fabrication de chaque OF. L'ordre de précédence est indiqué dans la gamme, ainsi par exemple pour l'OF 1, les opérations seront exécutées dans l'ordre suivant : opération 1, puis l'opération 3 et après, l'opération 2. Le tableau suivant représente les différentes gammes des ordres de fabrication ainsi que les différents temps d'exécution de chaque opération sur chaque machine :

	Opérations	Gammes	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
OF 1	O1,1	1,3,2	1	4	6	9	3	5	2	8	9	5
	O2,1		3	2	5	1	5	6	9	5	10	3
	O3,1		4	1	1	3	4	8	10	4	11	4
OF 2	O1,2	2,1,3	4	8	7	1	9	6	1	10	7	1
	O2,2		2	10	4	5	9	8	4	15	8	4
	O3,2		6	11	2	7	5	3	5	14	9	2
OF 3	O1,3	1,2,3	8	5	8	9	4	3	5	3	8	1
	O2,3		9	3	6	1	2	6	4	1	7	2
	O3,3		7	1	8	5	4	9	1	2	3	4
OF 4	O1,4	1,2,3	5	10	6	4	9	5	1	7	1	6
	O2,4		4	2	3	8	7	4	6	9	8	4
	O3,4		7	3	12	1	6	5	8	3	5	2
OF 5	O1,5	2,3,1	6	1	4	1	10	4	3	11	13	9
	O2,5		7	10	4	5	6	3	5	15	2	6
	O3,5		5	6	3	9	8	2	8	6	1	7
OF 6	O1,6	1,2,3	8	9	10	8	4	2	7	8	3	10
	O2,6		7	3	12	5	4	3	6	9	2	15
	O3,6		4	7	3	6	3	4	1	5	1	11
OF 7	O1,7	3,2,1	5	4	2	1	2	1	8	14	5	7
	O2,7		3	8	1	2	3	6	11	2	13	3
	O3,7		1	7	8	3	4	9	4	13	10	7
OF 8	O1,8	3,1,2	8	3	10	7	5	13	4	6	8	4
	O2,8		6	2	13	5	4	3	5	7	9	5
	O3,8		5	7	11	3	2	9	8	5	12	8
OF 9	O1,9	1,2,3	3	9	1	3	8	1	6	7	5	4
	O2,9		4	6	2	5	7	3	1	9	6	7
	O3,9		8	5	4	8	6	1	2	3	10	12
OF 10	O1,10	3,2,1	9	2	4	2	3	5	2	4	10	23
	O2,10		3	1	8	1	9	4	1	4	17	15
	O3,10		4	3	1	6	7	1	2	6	20	6

Tableau IV.1 : Temps d'exécution des opérations sur différentes machines exemple 10x10

La résolution de ce problème passe en premier lieu par la création d'une première population, cette dernière doit être suffisamment grande et diversifiée pour garantir une exploration assez large de l'espace de recherche. Nous appliquons deux méthodes différentes pour l'obtention de la population initiale.

$1^{ère}$ méthode : Un premier ordonnancement est obtenu par la méthode de décomposition temporelle, son Cmax est égal à 16 unités de temps (voir la figure A.1 de l'annexe A). On duplique cette solution en appliquant une succession de mutation comme suit :

- appliquer l'opérateur de mutation par affectation avec un taux de 75%,
- applique l'opérateur de mutation par permutation avec un taux de 25%,

- le processus est arrêté après l'obtention de 50 individus.

Nous avons appliquer notre algorithme à stratégie d'évolution avec les paramètres suivants :

- Taille de la population = 50
- Probabilité de croisement = 0,75
- Probabilité de mutation par affectation = 0,025
- Probabilité de mutation par permutation = 0,025 (soit au total une probabilité de mutation égale à 5%)
- Nombre de générations = 5000.

La première remarque que nous pouvons faire est le nombre très grand de générations par rapport au problème de petite taille, ceci est dû à un espace de recherche considérable. La figure suivante montre l'évolution du Cmax le long des générations.

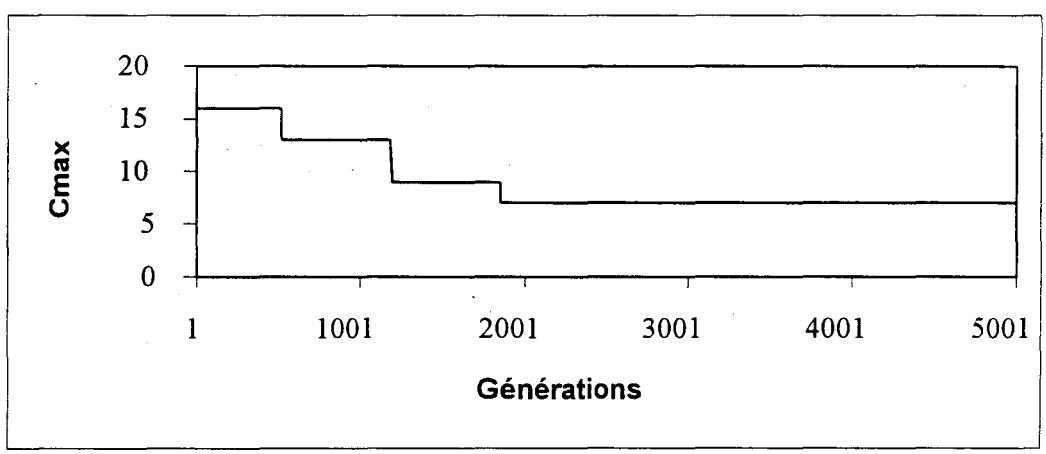

Figure IV-7 : Evolution du Cmax pour le problème 10 x 10 à faible taux de mutation

Le choix des paramètres est fait en fonction de la nature et de la taille du problème traité. Nous avons étudié l'influence des taux de croisement et de mutation sur la convergence de notre algorithme, pour notre premier cas nous avons choisi les paramètres suivants :

- Probabilité de croisement = 0,75
- Probabilité de mutation par affectation = 0,25
- Probabilité de mutation par permutation = 0,25 (soit au total un taux de mutation égal à 50%)

La courbe obtenue est représentée par la figure suivante :

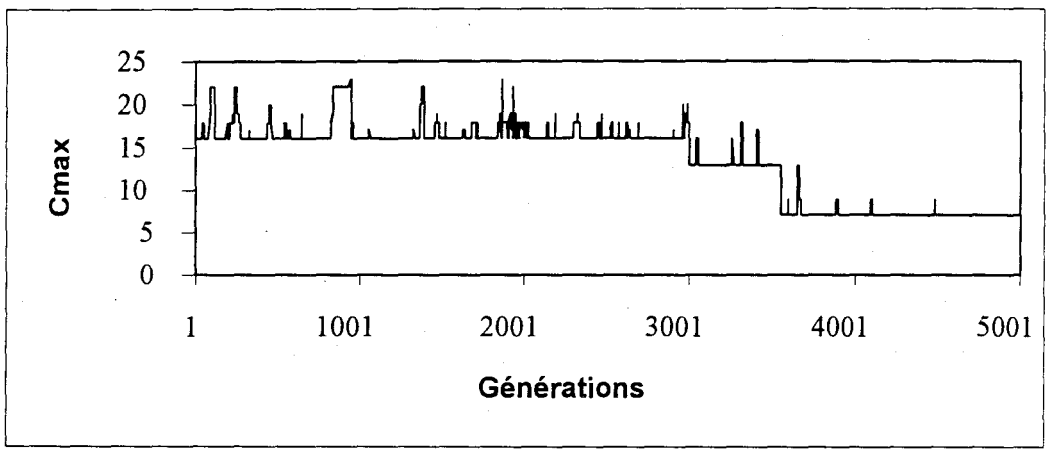

Figure IV-8 : Evolution du Cmax pour le problème 10 x 10 à grand taux de mutation

Nous remarquons une perte de temps à autre de la bonne solution, ceci est dû au taux de mutation qui s'avère trop élevé. Cet opérateur devient un perturbateur et fait perde très souvent la bonne valeur et ralenti par conséquent la convergence. Le meilleur Cmax n'est obtenu qu'après plus de 3500 générations alors que dans le cas précédent il l'est après 2000 générations.

Que ce passe t'il lorsque le taux de mutation est plus grand que celui du croisement ?

- Probabilité de croisement = 0,4
- Probabilité de mutation par affectation = 0,35
- Probabilité de mutation par permutation = 0,35

Nous représentant le Cmax obtenu par la figure suivante :

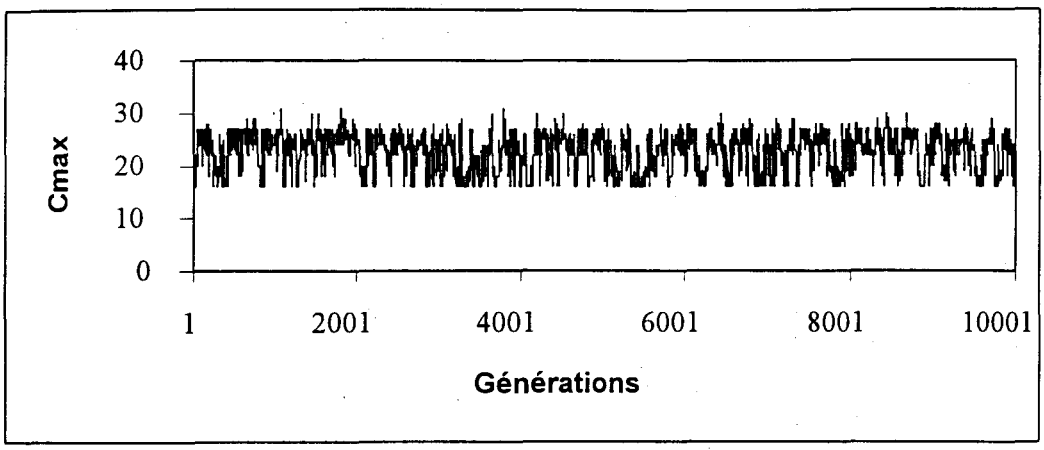

Figure IV-9 : Evolution du Cmax pour le problème 10 x 10 avec Pm > Pc

Dans ce cas, le taux de mutation est plus important que celui du croisement, le rôle d'élément perturbateur pour l'opérateur de mutation se confirme. On oscille autour de la valeur trouvée par la décomposition temporelle sans jamais être au-dessous. On ne converge pas même après avoir doublé le nombre de générations (10 000 générations).

Il apparaît très clairement que le taux de mutation influence grandement la convergence de l'algorithme vers l'optimum. **Qu'en est-il de la qualité de la population initiale ?**

Nous allons dans ce qui suit voir l'influence de la qualité de la population initiale sur la convergence de notre algorithme en obtenant la première population par une autre méthode.

$2^{ème}$ méthode : Nous avons utilisé la programmation avec contrainte selon l'organigramme de la figure III-3 présenté dans le chapitre précédent, pour obtenir un premier ensemble de solutions utilisé comme une population initiale. La population a une taille de 20 individus (voir la figure A.2 de l'annexe A).

Notre algorithme est appliqué avec les paramètres génétiques suivants :

- Taille de la population = 20
- Probabilité de croisement = 0,75
- Probabilité de mutation par affectation = 0,025

- Probabilité de mutation par permutation = 0,025
- Nombre de générations = 5000.

On otient le résultat présenté sur la figure suivante :

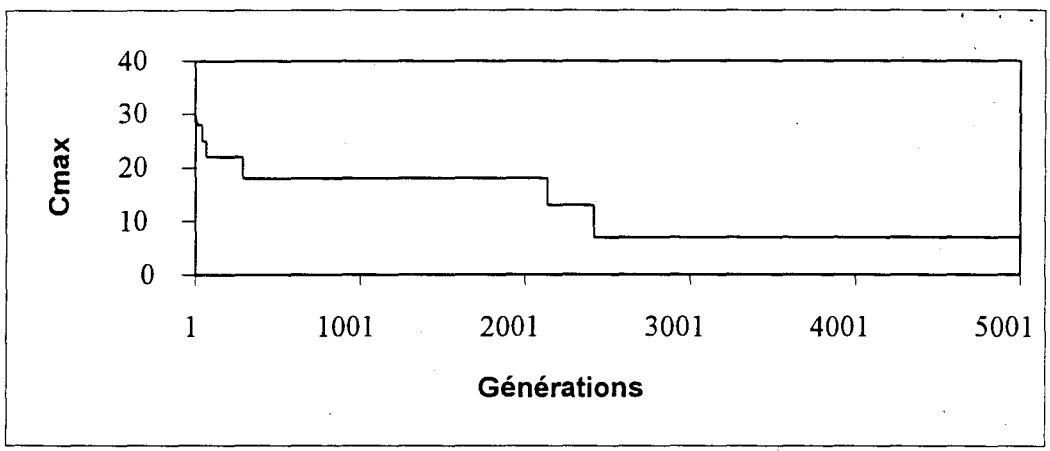

Figure IV-10 : Evolution du Cmax pour le problème 10 x 10 à faible taux de mutation

Nous remarquons que la qualité de la solution est très satisfaisante, les paramètres génétiques choisis pour les deux méthodes permettant d'avoir la population initiale s'avèrent très efficaces. Malgré une population moins grande que dans le premier cas la convergence est assurée dans de très bonnes conditions.

Changeant maintenant les taux de mutation tout en gardant la même population initiale.

- Probabilité de mutation par affectation = 0.1
- Probabilité de mutation par permutation = 0.1 (soit au total un taux de mutation égal à 20%)

Nous obtenons le résultat suivant :

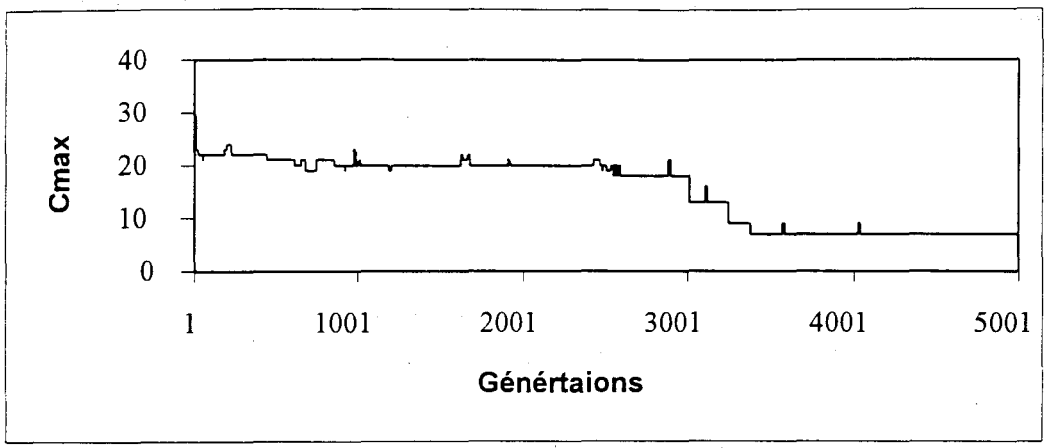

Figure IV-11 : Evolution du Cmax pour le problème 10 x 10 à grand taux de mutation

Le choix du taux de mutation est une opération délicate et très importante dans l'évolution de notre algorithme, elle conditionne très fortement la convergence vers l'optimum voir figure IV-11. Un taux de mutation important, dépassant dans certains cas les 30% bloque totalement la convergence et détruit systématiquement la majeure partie des bonnes solutions que nous espérons trouver. Ceci est illustré par la figure IV-12, on a choisi un taux de mutation égal à 35%. On n'arrive pas à converger malgré un nombre de génération très important (10 000 générations), on oscille autour d'une valeur moyenne déjà existante dans la population initiale. Elle est égale à 22 unités de temps. Cette dernière est très loin de la valeur déjà trouvée avec des opérateurs génétiques raisonnables.

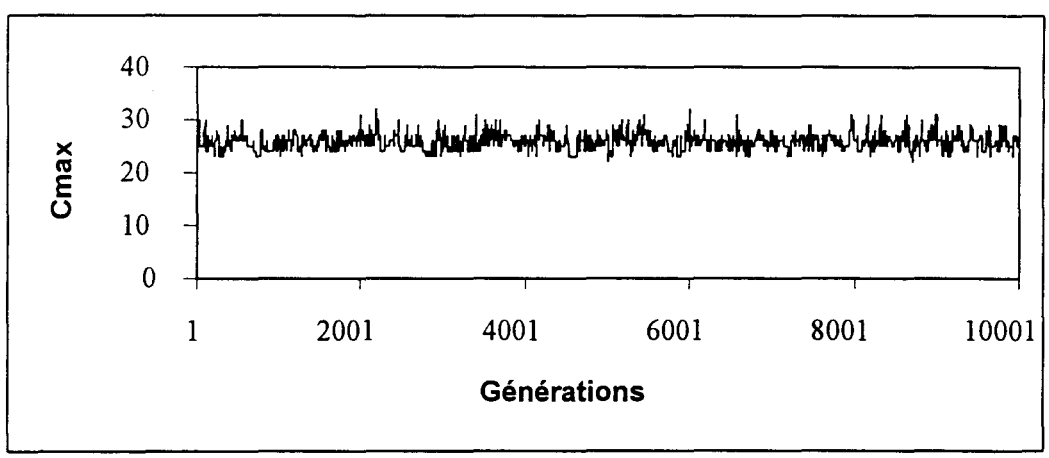

Figure IV-12 : Evolution du Cmax pour le problème 10 x 10 avec Pm > 30%

Nous allons dans ce qui suit, appliquer la deuxième approche basée sur notre deuxième codage.

IV.2 Deuxième approche

On se propose en premier lieu de résoudre le même problème de grande taille que précédemment (10 ordres de fabrication avec 10 machines). Vu que les contraintes de précédence sont prises en compte lors de la construction du codage et que le problème possède une flexibilité totale, il est possible de générer aléatoirement notre population initiale. Chaque opération est affectée aléatoirement à une machine, les contraintes de ressources sont prises en compte lors du calcul des dates de début d'exécution de chaque opération par la machine qui lui a été affectée. Le Cmax de la première population est représenté par la figure suivante :

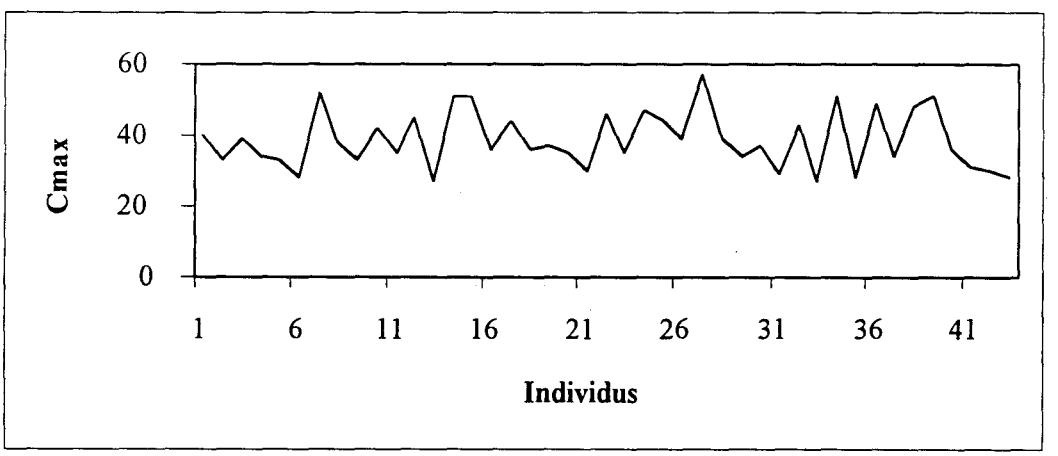

Figure IV-13 : Cmax de la première population générée aléatoirement

Nous appliquons par la suite à cette première population notre algorithme avec les paramètres suivants :

- Taille de la population = 42
- Taux de croisement = 75 %
- Taux de mutation = 10 %.

Au bout de 5000 générations, le Cmax a évolué de la façon suivante :

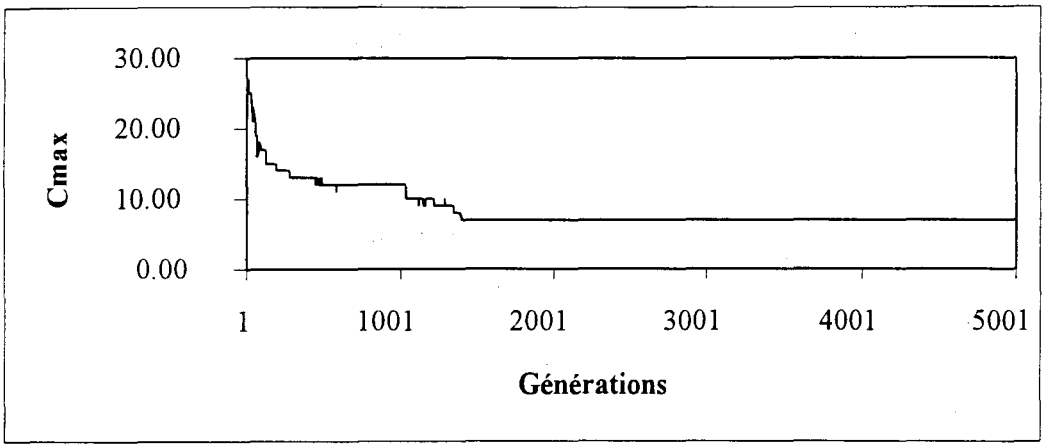

Figure IV-14 : Evolution du Cmax pour le problème 10 x 10 à faible taux de mutation

Il apparaît très clairement que l'algorithme converge plus rapidement que dans le premier cas (utilisation de la première approche). La valeur de 7 unités de temps est atteinte après moins de 1500 générations. Le chromosome représentant le meilleur ordonnancement obtenu est indiqué par la figure suivante :

	Ordre d'exécution des opérations			
OF 1	Op 1, Op 2, Op 3	(M1, 0)	(M2, 1)	(M4, 4)
OF 2	Op 2, Op 1, Op 3	(M1, 1)	(M4, 3)	(M3, 5)
OF 3	Op 1, Op 2, Op 3	(M10, 0)	(M8, 1)	(M8, 2)
OF 4	Op 1, Op 2, Op 3	(M7, 0)	(M3, 1)	(M10, 4)
OF 5	Op 2, Op 3, Op 1	(M9, 0)	(M9, 2)	(M6, 3)
OF 6	Op 1, Op 2, Op 3	(M6, 0)	(M9, 3)	(M9, 5)
OF 7	Op 3, Op 2, Op 1	(M4, 0)	(M3, 4)	(M5, 5)
OF 8	Op 3, Op 1, Op 2	(M5, 0)	(M2, 2)	(M2, 5)
OF 9	Op 1, Op 2, Op 3	(M3, 0)	(M7, 3)	(M8, 4)
OF 10	Op 3, Op 2, Op 1	(M7, 1)	(M7, 4)	(M7, 5)

Figure IV-15 : Meilleur individu trouvé

Les résultats sont très satisfaisants avec des taux de croisement et de mutation raisonnables, **qu'en est-il si le taux de mutation est nettement supérieur à celui du croisement ?**

Avec des taux de mutation élevés, le comportement de notre algorithme est sensiblement le même pour les deux types de codage. L'opérateur de mutation devient un élément perturbateur et n'arrive pas à atteindre la bonne valeur précédemment obtenue de la fonction à optimiser. Ceci est illustré par la figure suivante :

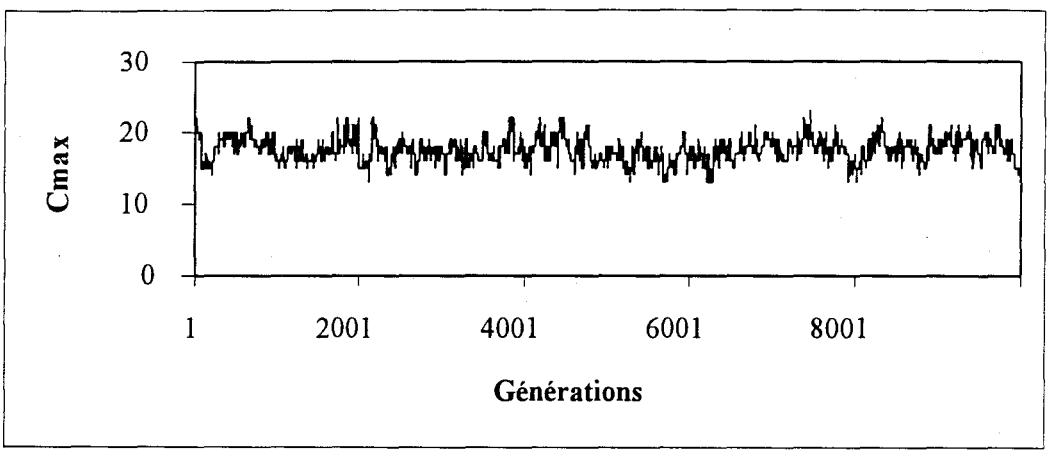

Figure IV-16 : Evolution de Cmax pour le problème 10 x 10 avec Pm > Pc

Cette courbe a été obtenue avec les paramètres génétiques suivants :

- Probabilité de croisement = 0.4
- Probabilité de mutation = 0.7

Malgré un doublement du nombre de générations par rapport au cas précédent (10 000 générations), la meilleure valeur du Cmax que nous avons pu obtenir est égale à 13 unités de temps. Cette dernière est meilleure que celle trouvée par la première approche dans les mêmes conditions (taux de mutation plus grand de celui du croisement), mais elle reste loin de la valeur de 7 unités de temps précédemment trouvée avec des taux de croisement et de mutation raisonnablement choisis. Le taux de mutation est trop élevé pour permettre l'amélioration ou même le maintien de la valeur du Cmax obtenu.

Il apparaît très clairement dans cet exemple (10 ordres de fabrication et 10 machines avec une flexibilité totale) que nos deux approches permettent d'avoir une très bonne valeur du critère à optimiser avec des taux de croisement et de mutation parfaitement choisis. Il serait très intéressant de voir comme notre algorithme se comporte pour des problèmes moins flexibles.

L'exemple que nous présentons dans ce qui suit est formé de 8 ordres de fabrications. Chaque OF est composé de 3 ou 4 opérations pouvant s'effectuer sur une ou plusieurs machines parmi les 8 existant dans le parc. Les données de cet exemple sont présentées dans le tableau suivant :

	Opérations	Gammes	M1	M2	M3	M4	M5	M6	M7	M8
OF 1	O1,1	1, 2, 3	5	3	5	3	3	x	10	9
	O2,1		10	x	5	8	3	9	9	6
	O3,1		x	10	x	5	6	2	4	5
OF 2	O1,2	1, 2, 4, 3	5	7	3	9	8	x	9	x
	O2,2		x	8	5	2	6	7	10	9
	O3,2		10	8	9	6	4	7	x	x
	O4,2		x	10	x	5	6	4	1	7
OF 3	O1,3	3, 2, 1	1	4	5	6	x	10	x	7
	O2,3		x	10	6	4	8	9	10	x
	O3,3		10	x	x	7	6	5	2	4
OF 4	O1,4	1, 3, 2	3	1	6	5	9	7	8	4
	O2,4		4	6	2	10	3	9	5	7
	O3,4		12	11	7	8	10	5	6	9
OF 5	O1,5	1, 2, 3 4	3	6	7	8	9	x	10	x
	O2,5		10	x	7	4	9	8	6	x
	O3,5		x	9	8	7	4	2	7	x
	O4,5		11	9	x	6	7	5	3	6
OF 6	O1,6	2, 3, 1	10	5	9	10	11	x	10	x
	O2,6		6	7	1	4	6	9	x	10
	O3,6		11	x	9	9	9	7	6	4
OF 7	O1,7	1, 3, 2	5	4	2	6	7	x	10	x
	O2,7		x	8	9	3	8	6	x	10
	O3,7		x	9	x	9	11	9	10	5
OF 8	O1,8	1, 2, 4, 3	2	8	5	9	x	4	x	10
	O2,8		7	4	7	8	9	x	10	x
	O3,8		9	x	3	7	1	5	8	x
	O4,8		9	9	x	8	5	6	7	1

Tableau IV.2 : Temps d'exécution des opérations sur différentes machines exemple 8x8

Cet exemple ne présente pas de flexibilité totale, certaines opérations ne sont pas exécutée par certaines machines.

La résolution par la méthode de décomposition temporelle a donné un Cmax égal à 19 unités de temps [64].

Notre algorithme avec le deuxième codage (CPOF) et les paramètres suivants :

- Population initiale créée aléatoirement
- Taille de la population = 42
- Probabilité de croisement = 0,85
- Probabilité de mutation = 0,05
- Nombre de générations = 12 000.

donne le résultat suivant :

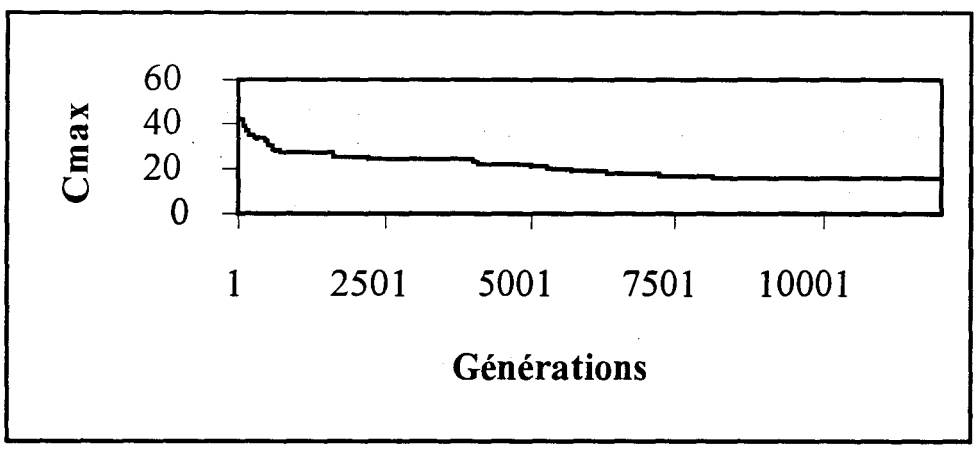

Figure IV-17 : Evolution du Cmax pour le problème 8 x 8 à flexibilité partielle.

Nous remarquons que moins le problème est flexible, plus il est long à converger, ceci est dû principalement à un espace de recherche plus compliqué que dans le précédent cas (flexibilité totale). Le domaine contient des points de singularité représenté par des ordonnancements inadmissibles du fait qu'ils possèdent des combinaisons interdites à cause de la flexibilité limitée de l'atelier de fabrication.

	Ordre d'exécution des opérations				
OF 1	Op 1, Op 2, Op 3	(M5, 0)	(M5, 3)	(M6, 6)	
OF 2	Op 1, Op 2, Op 4, Op 3	(M3, 0)	(M4, 3)	(M7, 9)	(M5, 10)
OF 3	Op 3, Op 2, Op 1	(M7, 0)	(M4, 5)	(M1, 9)	
OF 4	Op 1, Op 3, Op 2	(M2, 0)	(M6, 1)	(M1, 10)	. . .
OF 5	Op 1, Op 2, Op 3, Op 4	(M1, 0)	(M7, 3)	(M6, 9)	(M7, 11)
OF 6	Op 2, Op 3, Op 1	(M3, 3)	(M8, 4)	(M2, 8)	
OF 7	Op 1, Op 2, Op 3	(M2, 1)	(M8, 8)	(M4, 13)	
OF 8	Op 1, Op 2, Op 4, Op 3	(M1, 3)	(M3, 5)	(M8, 13)	(M5, 14)

Figure IV-18 : Meilleur individu trouvé

La figure ci-dessus donne le meilleur ordonnancement obtenu, ce dernier n'est obtenu qu'après plus de 7500 générations, il présente un Cmax égal à 16 unités de temps, il est 15% plus petit que celui trouvé par la décomposition temporelle (19 unités de temps).

IV.3 Durée d'exécution

Le temps d'exécution dépend principalement de la taille de la population et du nombre de générations. Tous nos exemples ont été exécutés sur un PC équipé d'un microprocesseur Intel Pentium à 166 MHZ.

Les moyennes des durées d'exécution des différentes séries d'expérience considérée sont présentées ci-dessous

Durée d'exécution pour la première approche, exemple 10 OF et 10 machines. Ce problème possède une flexibilité totale. Il faut ordonnancer 30 opérations sur 10 machines. Le temps CPU pour 5000 générations et 20 individus est égal à 55 secondes et il est de 33 secondes pour 12 individus. Après plusieurs expériences pour des valeurs différentes des nombres de générations et de populations, il apparaît que le coût moyen en temps CPU moyen pour un individu et par génération est de $5,5\ 10^{-4}$s.

Durée d'exécution pour la deuxième approche, exemple 10 OF et 10 machines. Dans ce cas, Le temps CPU pour 5000 générations et 20 individus est de 10 secondes et il est de 21 secondes pour 5000 générations et 42 individus. Après plusieurs expériences et avec des jeux de données différents nous avons établi le temps CPU moyen par génération et par individu pour ce cas de figure, il est égal à 10^{-4}s. Nous remarquons que pour le même exemple, la deuxième approche relative au deuxième codage (CPOF) est 5 fois plus rapide que la première approche basée sur le premier codage (CPM) ; ceci est dû principalement au processus de correction (voir paragraphe III.3.4) que nous avons mis au point pour aboutir à des solutions admissibles. Ce processus s'avère très coûteux en temps de calcul, il est multiplié par 5.

Durée d'exécution pour la deuxième approche, exemple 8 OF et 8 machines. Pour avoir une idée plus claire de l'influence de la complexité du problème sur le temps d'exécution, nous allons calculer ce dernier pour l'exemple 8 OF et 8 machines avec une flexibilité réduite. Il s'agit dans ce cas d'ordonnancer au total 27 opérations sur 8 machines. Le temps CPU pour 12 000 générations et 42 individus est de 59 secondes et il est de 28 secondes pour 20 individus. Après plusieurs expériences et avec des jeux de données différents nous avons établi le temps CPU moyen par génération et par individu pour ce cas de figure, il est égal à $1,167\ 10^{-4}$s. Ce qui représente une augmentation de 16% par rapport au cas précédent.

Remarque IV.1 *Les taux de croisement et de mutation conditionnent aussi le temps CPU, faire plus ou moins de croisement et de mutation augmente ou diminue la durée d'exécution. Dans nos calculs du CPU nous avons pris soin de prendre les mêmes taux de croisement et de mutation pour les différents exemples. Ces taux sont indiques ci-dessous.*

- *Taux de croisement = 75%*
- *Taux de mutation = 5%.*

IV.4 Conclusion

Dans ce chapitre, Nous avons mis en évidence de façon concrète, l'apport des algorithmes à stratégie d'évolution et en particulier des méthodes que nous avons développées pour aider les responsables d'atelier en leur donnant non seulement un ensemble d'ordonnancement admissible très proche de l'optimum selon le ou les critères choisis par le chef d'atelier. Nous avons montré suite aux résultats que nous avons exposé dans ce chapitre, que nos approches étaient capables d'améliorer les résultats trouvés par d'autres méthodes et de s'approcher très efficacement de l'optimum, ceci avec des temps de calcul très courts. Dans tous les cas présentés une très bonne solution à été trouvée au bout de quelques secondes et même dans le cas le plus délicat un bon résultat est trouvé en moins d'une minute. Le deuxième codage est plus souple, plus rapide car il permet d'intégrer les contraintes de précédence dans la configuration même du chromosome, il permet aussi de se détacher des autres méthodes car nous pouvons créer notre première population d'une façon totalement aléatoire ce qui permet de faire démarrer l'algorithme. Grâce aux différentes simulations présentées dans ce chapitre, nous pouvons dire que les meilleurs taux de croisement et de mutation sont respectivement plus de 65 % et moins de 10 %.

Conclusion et perspectives

Nous avons étudié dans ce mémoire la contribution des algorithmes à stratégie d'évolution pour résoudre conjointement les problèmes d'affectation et d'ordonnancement des ateliers flexibles. Notre objectif dans ce travail est la résolution approchée du problème d'ordonnancement d'atelier de type job-shop flexible à machines non reliées. Le critère considéré est la minimisation de la durée totale de l'ordonnancement (le Cmax ou Makespan).

Les problèmes d'ordonnancement sont extrêmement difficiles à résoudre parce qu'ils appartiennent à la classe dite NP-difficile, ils demandent un espace de recherche hautement combinatoire, de traitement particulièrement complexe. C'est pourquoi les méthodes exactes telles que la branch and bound et la programmation dynamique demandent un temps d'exécution considérable et/ou des formulations mathématiques complexes, particulièrement quand la taille du problème est importante. Pour palier à ce type de problème, nous avons proposé deux approches basées sur le principe des algorithmes génétiques.

La revue de la littérature que nous avons présentée au chapitre 2, met en évidence les difficultés liées à leur utilisation pour résoudre les problèmes que nous nous sommes posés. Il apparaît clairement que leur utilisation est conditionnée par certaines modifications à apporter notamment dans la représentation de la solution (le codage), ainsi que dans les différents opérateurs génétiques (le croisement et la mutation), d'où le nom des algorithmes à stratégie d'évolution.

Nous avons développé deux approches pour résoudre les problèmes d'ordonnancement et en particulier ceux du type job-shop flexible.

La première approche est relative au premier codage (le CPM : Codage Parallèle des Machines). Il nous permet de traiter conjointement les deux problèmes présents

dans un job-shop flexible, à savoir l'affectation et l'ordonnancement. Après avoir généré la première population nécessaire pour le démarrage de l'algorithme en utilisant une seule méthode ou une forme combinée des méthodes classiques (programmation avec contraintes ou la décomposition temporelle), le résultat obtenu est non seulement un ordonnancement réalisable mais en plus il est optimal ou du moins proche de l'optimum. Le choix de la représentation parallèle permet une compréhension directe de la solution. Les opérateurs génétiques relatifs à ce codage permettent de prendre en charge toutes les corrections nécessaires pour garantir une solution respectant les contraintes de précédence et de ressources du problème considéré.

Le processus de correction étant très coûteux en temps, nous avons donc procédé à une modification du codage en intégrant les contraintes de précédence dans la conception même du chromosome, ce qui a abouti tout naturellement à la deuxième approche basée sur le Codage Parallèle des Ordres de Fabrication (CPOF). Ce dernier permet de respecter l'ordre de précédence des OF. Il permet aussi de distinguer notre algorithme des autres méthodes. En effet il n'est plus nécessaire d'avoir une population initiale pour démarrer l'algorithme, il est capable de générer aléatoirement sa propre première population et de l'améliorer par la suite en appliquant les différents opérateurs génétiques conçus spécialement pour le CPOF.

Nous avons appliqué ces deux approches à des exemples de tailles et de flexibilités différentes, les résultats obtenus montrent bien que nous aboutissons à des solutions admissibles et nettement meilleures que celles obtenues par les autres méthodes. La deuxième approche s'avère nettement plus rapide et plus souple que la première. Néanmoins ces méthodes sont capables de traiter un grand nombre de machines et de gammes. En effet, seule la capacité de mémoire de l'ordinateur sur lequel l'algorithme est lancé peut limiter le nombre des OF et ou des machines selon le codage utilisé (la taille de l'individu dépend du nombre des opérations et des machines).

Les résultats que nous avons obtenus sont très encourageants. Il serait intéressant de pouvoir développer certains aspects ce que nous envisageons dans nos prochains travaux

de recherche.

1) L'optimisation multi-critère : l'optimisation d'un seul critère peut dans plusieurs cas être insuffisante pour une utilisation efficace de toutes les possibilités de l'entreprise. En effet, il est plus intéressant d'optimiser plusieurs critères. Ceci peut être fait soit par l'agrégation de plusieurs critères en une fonction traduisant un compromis entre les différents buts à atteindre soit encore, par une division de la population en plusieurs sous-ensemble dont chacun optimise un critère donné.

2) Développer des moyens théoriques à partir de la notion de schéma pour évaluer les performances des codages, ainsi que des opérateurs génétiques destinés à explorer les espaces de recherche.

3) Nous avons restreint notre étude au cas des gammes déterministes, vu le contexte de fabrication actuel, il serait très intéressant de prendre en compte d'autres types de gammes telles que les gammes multiples.

Bibliographie

[1] Carlier. J, Chrétienne. P, Problèmes d'ordonnancement modélisation/complexité/algorithmes, Edition Masson, Paris, France, 1988.

[2] Ghedjati-Guessoum. F, Résolution par des heuristiques dynamiques et des algorithmes génétiques du problème d'ordonnancement de type job-shop généralisé, Thèse de doctorat de l'université Paris 6, Décembre 1994.

[3] Blazewicz. J, Cellary. W, Slowinski. R, Weglarz. J, Scheduling under resource constraints : deterministic models. Annals of oper. Res., J.C. Baltze, A.G. Basel. 1986.

[4] Molet. H, La nouvelle gestion de production, Edition Hermès, Paris, 1989.

[5] Farreny. H, Ghallab. M, Eléments de l'intelligence artificielle, Edition Hermès, Paris, 1987.

[6] Tournassoud. P, Géométrie et intelligence artificielle pour les robots, Edition Hermès, Paris, 1988.

[7] Garey. M.R, Jhonson. D.S, Computers and intractability: A guide to theroy of NP-completencess, Freeman and Company, New York, 1979.

[8] Baker. K.K, Introduction to sequencing and scheduling, Wiley, New-York, 1974.

[9] French. S, Sequencing and scheduling: An introduction to the mathematics of the job-shop, Wiley, 1982.

[10] Bellman. R, Esogbue. A.O, Nabeshima. I, Mathematical aspects of scheduling and applications, Pergamon press, 1982

[11] Carlier. J, Pinson. E, An algorithm for solving the job-shop problem. Management Science, No 35, p 164-176, février 1989.

[12] Borne. P, Dauphin-Tanguy. G, Richard. J.P, Rotella. F, Zambettakis. I, Commande et optimisation des processus, Edition Technip, Paris, 1990.

[13] Portmann. M.C, Méthodes de décomposition spatiales et temporelles en ordonnancement de la production, Thèse d'état, Université de Nancy 1, 1987.

[14] Kirkpatrick. S, Gelatt. C.D, Vecchi. M.P, Optimization by simulated annealing. Sience, Volume 220, No 4598, p 671-680, 1983.

[15] Aarts. E et Korst. J, Simulated annealing and boltzmann machines, John Willy and Sons Ltd. 1989.

[16] Siarry. P, La méthode du recuit simulé : théorie et application, Actes des journées d'études : Ordonnancement et entreprise, application concrètes et outils pour le futur, 16 et 17 juin, p 47-67, Toulouse, France.

[17] Chengbin. C, Proth. J.M, L'ordonnancement et ses applications, Edition Masson. Paris, 1996.

[18] Hertz. A, Widmer, M, La méthode tabou appliquée aux problèmes d'ordonnancement, Actes des journées d'études : Ordonnancement et entreprise, application concrètes et outils pour le futur, 16 et 17 juin, p 97-125, Toulouse, France.

[19] Golver. F, Tabou search: part I, ORSA journal on computing, Volume 1, No 3, p 190-206, Summer 1989.

[20] Golver. F, Tabou search: part II, ORSA journal on computing, Volume 2, No 1, p 4-32, Winter 1990.

[21] Golver. F, Taillard. E, De werra. D, A user's guide to tabou search, Annals of operations research, volume 41, p 3-28, 1993.

[22] Farremy. H, Actes de la journée nationale, Intelligence Artificielle, PRC-GRECO, 14-15 Mars, Toulouse, France, 1988.

[23] Kerr. R.M, Ebsary. R.V, Implementation of an expert system for production systems, European journal of Operations research, No 3, p 17-29, 1988.

[24] Le pape. C, Sauve. B, SOJA : un Système d'Ordonnancement Journalier d'Atelier, 5ème journée internationales les systèmes experts et leur applications, p 849-867, Avignon, France, 1985.

[25] Bell. G, Bensana. E, Dubois. D, Construction d'ordonnancement prévisionnels : un compromis entre approches classiques et systèmes experts, APII, Volume 22, p 509-534, 1988.

[26] Bensana. E, Bel. G, Dubois. D, OPAL: A multi-knowledge-based system for industrial job-shop scheduling. Int. journal of production resarch, Volume 26, No 5, p 795-819, 1988.

[27] Codognet. P, Programmation logique avec contraintes : une introduction, RAIRO, TSI, Edition Hermès, Volume 14, No 6, p 665 -692, 1995.

[28] Esquirol. P, Lopez. P, Programmation logique avec contraintes et ordonnancement : Automatique-Productique-Informatique Industrielle, dition Hermès, Volume 29, N4-5, p 379-407, 1995.

[29] Varnier. C, Baptiste. P, Constraint logic programming and scheduling problems, Proceedings de IEEE/SMC Conference, Beijing, China, volume 4, Octobre 14-17, p 2942-2946, 1996.

[30] Davis. L, Job shop scheduling with genetic algorithms, Proceedings of the first international conference on genetic algorithms, 1985, p 136-140.

[31] Nakano. R, Yamada. T, Conventional genetic algorithm for job shop problems, Proceedings of the fourth international conference on genetic algorithms, University of california, san-Diego, July 13-16, 1991, p 474-479.

[32] Falkenauer. E, Bouffoux. S, A genetic algorithm for job shop, Proceedings of the IEEE International conference on Robotics and Automation, Sacramento, California, April, 1991, p 824-829.

[33] Yamada. T, Nakano. R, A genetic algorithm applicable to large-scale job-shop problems, Parallel problem solving from nature, Volume 2, Editeur R. Männer and B. Manderick, North Holland, Amsterdam, p 281-290, 1992.

[34] Bruns. R, Direct chromosome representation and advanced genetic operators for production scheduling, Proceedings of the fifth international conference on genetic algorithms, University of Illinois at Urbana-Champaign, July 17-21, p 352-359, 1993.

[35] Della Croce. F, Tadei. R, Volta. G, A genetic algorithm for the job shop problem, Computer and operations research, Volume 22, No 1, p 15-24, 1995.

[36] Shi. G, Iima. H, Sannomiya. N, A new encoding scheme for solving job shop problem by genetic algorithm, Proceedings of the 35th CDC, Kobe, Japan, December, p 4395-4400, 1996.

[37] Portmann. M.C, Genetic algorithms and scheduling: a state of the art and some propositions, Proceedings of the workshop on production planning and control, Mons, Belgium, p i-xxiv, September 9-11, 1996.

[38] Goldberg D.E., Genetic algorithms in search,optimization, and machine learning, Addison-wesley, 1989.

[39] Holland John.H., Adaptation in natural and artificial systems, MIT press Bradford Books edition 1992.

[40] Rendres J.-M., Algorithmes génétiques et réseaux de neurones, édition Hermes, Paris, 1995.

[41] Syswerda. G., Schedule optimization using genetic algorithm, In handbook of genetic algorithm, p 332-348 Van Nostrand Reinhold, New york, 1991.

[42] Caux. C, Pierreval. H et Portmann. M.C., Les algorithmes génétiques et leur applications aux problèmes d'ordonnancement, Actes des journées d'études : Ordonnancement et entreprise, application concrètes et outils pour le futur, 16 et 17 juin, p 5-45, Toulouse, France.

[43] Michalewicz. Z, Genetic Algorithms + Data Structures = Evolution programs, Springer Verlag,1992.

[44] Man. K.F, Tang. K.S, Kwong. S et Halang. W.A, Genetic algorithms for control and signal processing, Springer 1997.

[45] Davics. L, handbook of genetic algorithm, Van Nostrand Reinhold, New-York, 1991.

[46] Aarts. E et Korst. J, simulated annealing and boltzmann machines, John Willy and Sons Ltd. 1989.

[47] Garey. M, Johnson. D, Computer and interactbility : A Guide to the theory of NP-Completeness, Freeman. W.H, San Francisco, 1979.

[48] Charon. I, Germa. A, Hudry. O, Méthodes d'optimisation combinatoire, éditions Masson, 1996.

[49] Baghi. S, Uckun. S, Miyab. Y, Kawamura. K, Exploring problem-specific recombination operators for job shop scheduling, Proceedings of the fourth international conference on genetic algorithms, University of california, san-Diego, July 13-16, 1991, p 10-17.

[50] Uckun. S, Baghi. S, Kawamura. K, Managing genetic search in job-shop scheduling, IEEE Expert, Volume 8, Numéro 5, 1993, p 15-24.

[51] Bruns. R, Direct chromosome representation and advanced genetic operators for production scheduling, Proceedings of the fifth international conference on genetic algorithms, University of Illinois at Urbana-Champaign, July 17-21, 1993, p 352-359.

[52] Djerid. L, Portmann. M.C, Genetic algorithm operators restricted to precedent constraint sets: genetic algorithm designs with or without branch and bound approach for solving scheduling problems with disjunctive constraints, Proceedings of IEEE /SMC, Beijing, China, volume 4, October 14-17, 1996, p.2922-2927.

[53] Starkweather. T, McDaniel. S, A comparison of genetic sequencing operators, Proceedings of the fourth international conference on genetic algorithms, University of california, san-Diego, July 13-16, 1991, p 69-76.

[54] Whitely. D, Starkweather. T, Shaner. D, The traveling selesman and sequence schedulling: quality solution using genetic edge recombination, In handbook of genetic algorithm, p 350-372 Van Nostrand Reinhold, New york, 1991.

[55] Goldberg. D. E, Lingel. R, Alleles, loci, and the traveling salesman problem, Proceedings of the first international conference on genetic algorithms, 1985, p 154-159.

[56] Oliver. I.M, Smith. D.J, Holland. J.R.C, A study of permutation crossover operators on the traveling salesman problem, Proceedings of the second international conference on genetic algorithms, 1991, p 224-230.

[57] Fleurent. C, Ferland. J.A, Algorithmes génétiques hybrides pour l'optimisation combinatoire, Recherche opérationnelle, Volume 30, No 4, 1996, p 373-398.

[58] Mesghouni. K, Hammadi. S, Borne. P., Production job-shop scheduling using genetic algorithms, proceedings of IEEE /SMC, Beijing, China, vol 2, October 14-17, 1996, p. 1519-1524.

[59] Mesghouni. K, Hammadi. S, Borne. P., On Modeling genetic algorithm for flexible job-shop scheduling problem, Studies in Informatics and Control Journal, Volume. 7, No. 1, March 1998, p. 37-47.

[60] Mesghouni. K, Pesin. P, Hammadi. S, Tahon. C, Borne. P., Genetic algorithms - constraint logic programming. Hybrid method for job shop scheduling, Proceedings of the OE/IFIP/IEEE International Conference on Integrated And Sustainable Industrial Production, Lisbon, Portugal, May 14-16 1997 , p.151-160.

[61] Chu. C, Portmann. M.C, Application of artificial memory in job-shop scheduling, Proceedings of the second International workshop on Project Management and Scheduling, Compiegne, France, June, 20-22, 1990, p. 154-159.

[62] Mesghouni. K, Hammadi. S, Borne. P, Parallel genetic operators for flexible job-shop scheduling à paraître dans journal Engineering Design & Automation, Communication présentée au 1st International Conference on Engineering Design And Automation, Bangkok, Thaïlande, Mars 18-21, 1997 et repris pour publication.

[63] Mesghouni. K, Hammadi. S, Borne. P, Evolution programs for job-shop scheduling, Proceedings of IEEE /SMC, Orlando, U.S.A, volume 1, October 12-15, 1997, p. 720-725.

[64] Chetouane. F, Ordonnancement d'atelier à tâches généralisées, perturbations, réactivité, rapport de DEA de l'institut national polytechnique de Grenoble 1995

Annexe A

Nous présentons dans cette annexe, le diagramme de Gantt de l'exemple IV.1.2 représentant un problème de taille importante. Cette solution à été trouvée par la méthode de décomposition temporelle (DT). Le Cmax de cet ordonnancement est égal à 16 unités de temps.

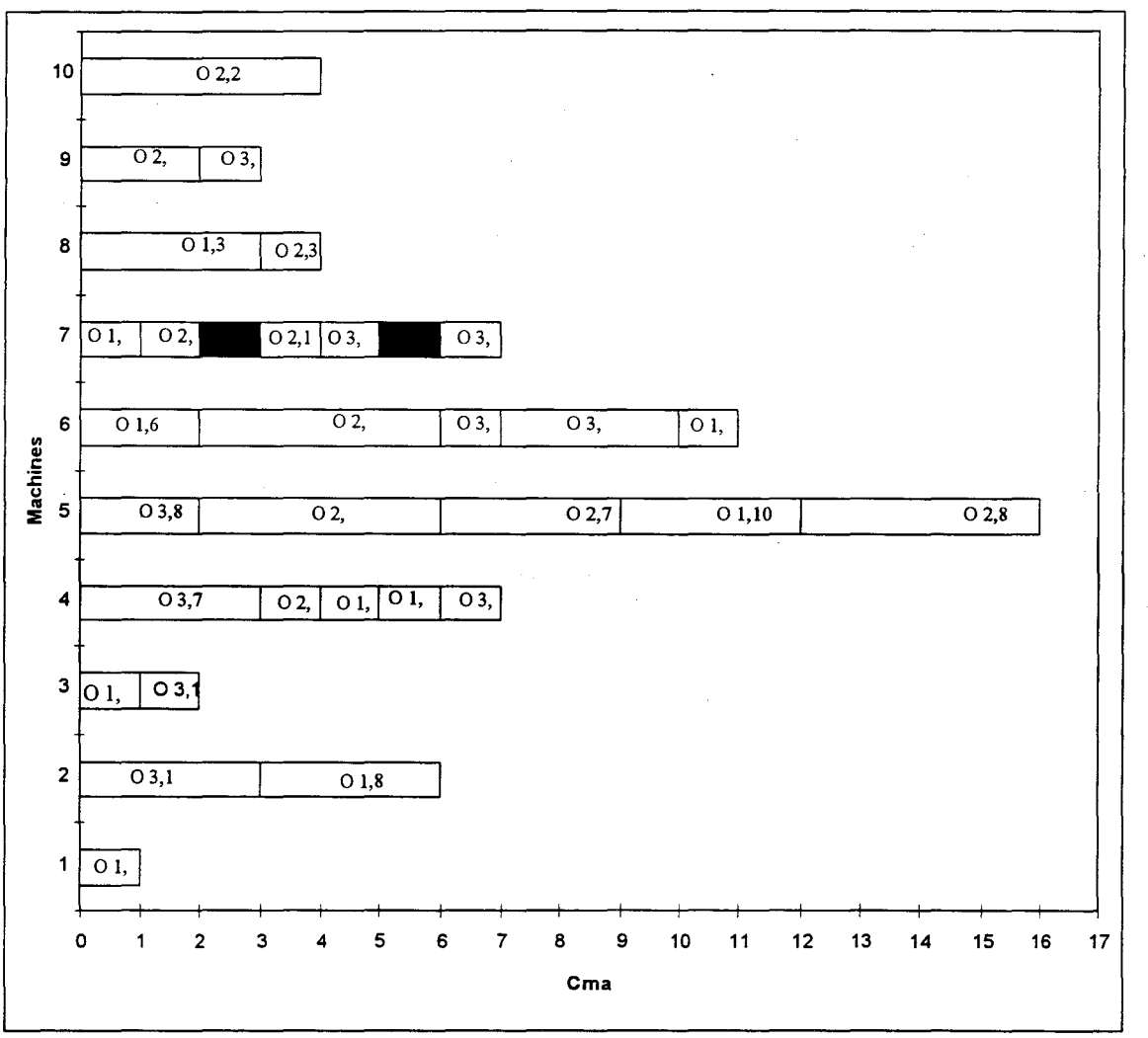

Figure A.1 : Diagramme de Gantt de la meilleure solution obtenue par DT

La figure suivante représente la première population obtenue grâce à la méthode de

programmation avec contrainte (PC). Elle à une taille de 20 individus. Le meilleur ordonnancement obtenu dans ce cas est l'individu 8, il présente un Cmax égal à 30 unités de temps.

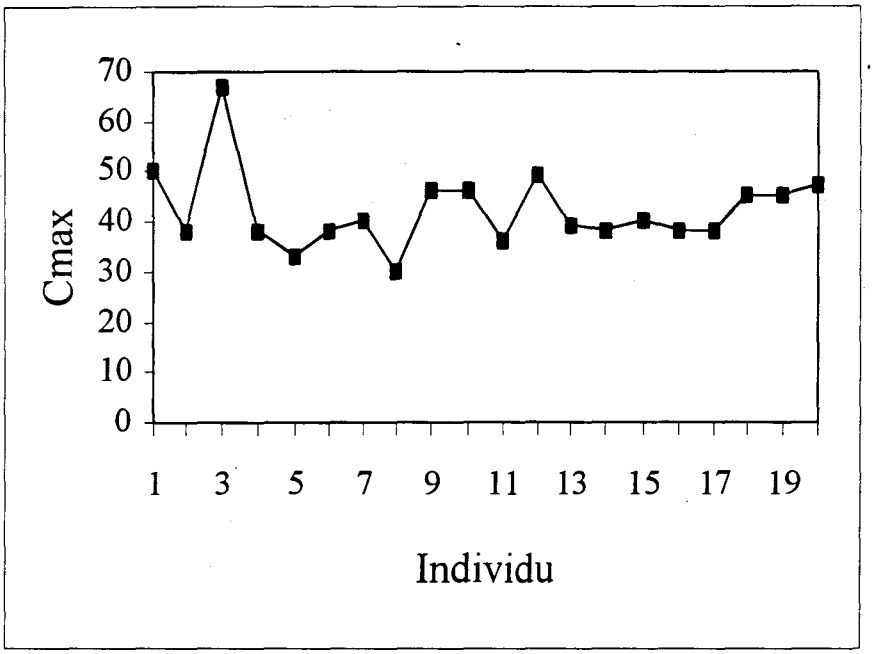

Figure A.2 : Cmax de la première population obtenue par PC

La figure suivante représente le diagramme de Gantt du meilleur ordonnancement obtenu en appliquant notre approche évolutionniste. En plus de l'optimisation du Cmax (7 unités de temps au lieu de 16), la charge des machines est mieux répartie dans ce cas que dans celui représenter par la figure A1. En effet, la machine la moins chargée dans la solution obtenue en utilisant l'approche évolutionniste travail pour une durée de 3 unités de temps, alors que la plus chargée fonctionne durant 7 unités de temps. Par contre pour le cas de la solution obtenue par la décomposition temporelle, la machine la moins chargée travail pendant une durée de 1 unité de temps et la plus chargé fonctionne pendant 16 unités de temps. Il y'a donc une mauvaise utilisation du parc des machines.

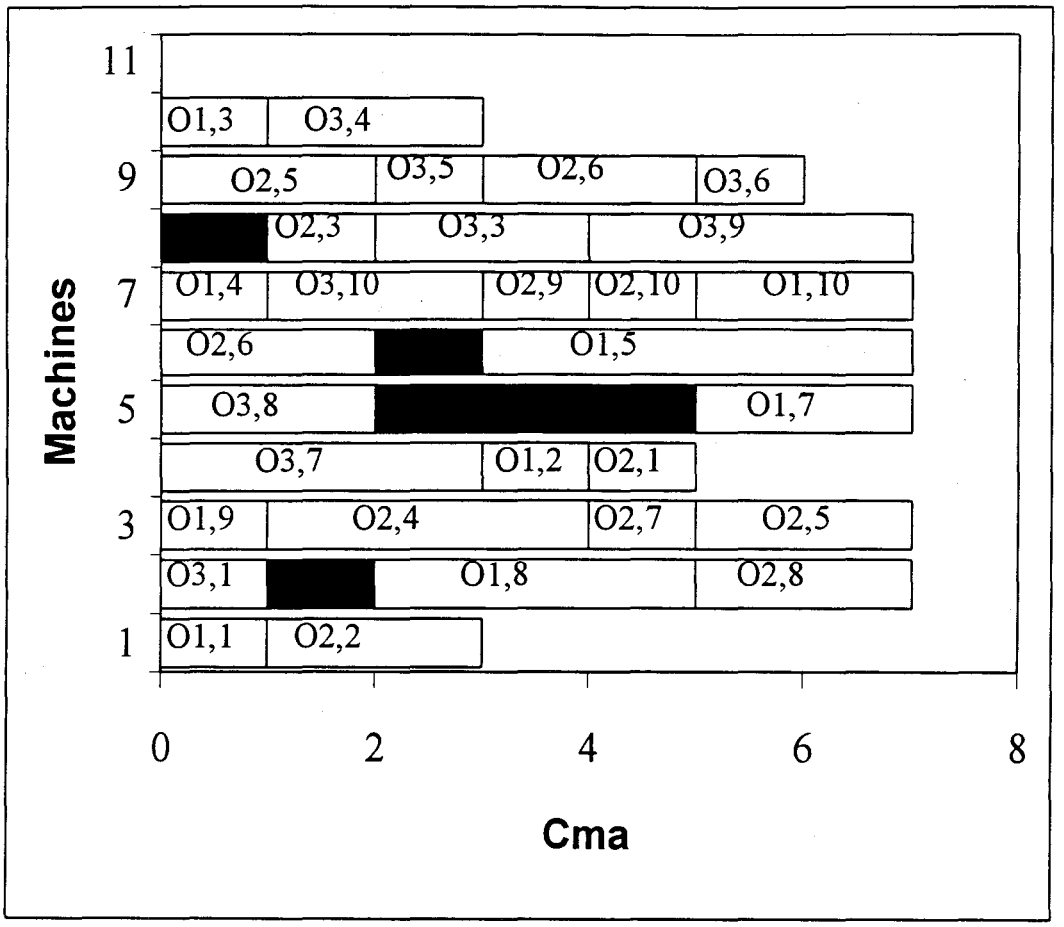

Figure A.3 : Diagramme de Gantt de la meilleure solution obtenue par Notre approche.

Annexe B

Nous présentons dans cette annexe, le diagramme de Gantt du meilleur ordonnancement obtenue pour l'exemple ayant 8 ordres de fabrication et 8 machines en utilisant la méthode de décompostion temporelle (DT). Le cmax de cette solution est égale à 19 unités de temps.

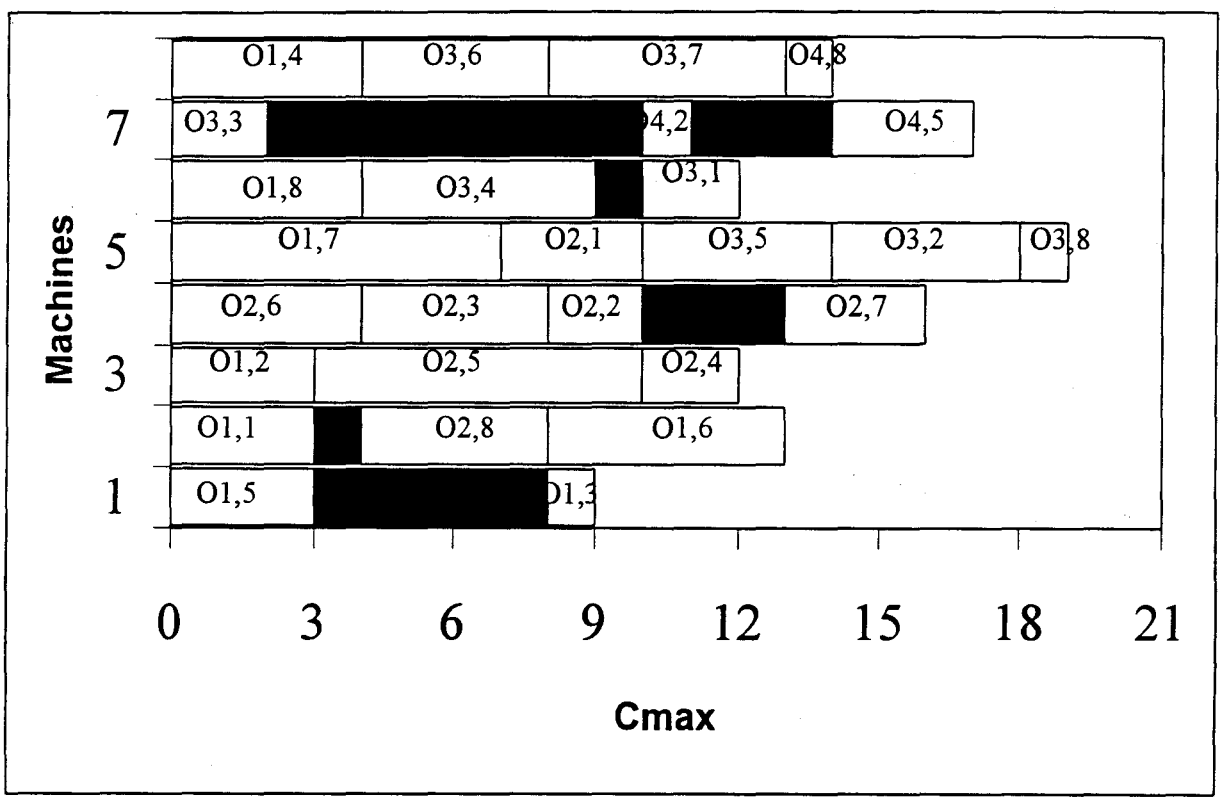

Figure B.1 : Diagramme de Gantt de la meilleure solution obtenue par DT

La figure suivante représente le diagramme de Gantt du meilleur ordonnancement obtenu en appliquant notre approche évolutionniste (AE). Elle permet d'obtenir une solution meilleure que celle donnée par la décomposition temporelle, 16 unités de temps au lieu de 19 unités de temps, soit un gain de plus de 15% d'unités de temps.

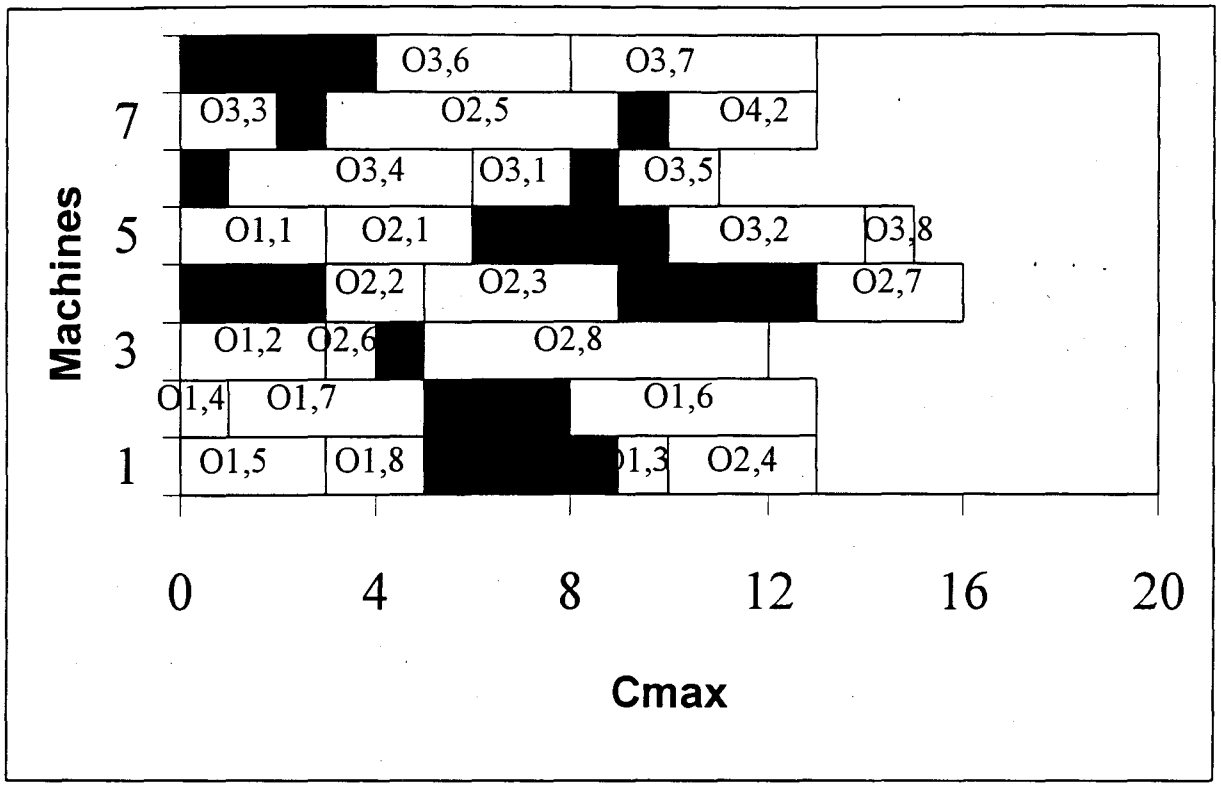

Figure B.2 : Diagramme de Gantt de la meilleure solution obtenue par AE.