# Contribution à l'ordonnancement conjoint de la production et de la maintenance : Application au cas d'un Job Shop.

Youssef Harrath

N° 997

Année : **2003**

# THESE

présentée à
**L'UFR des Sciences et Techniques**
**de l'Université de Franche-Comté**

pour obtenir le

## GRADE DE DOCTEUR DE L'UNIVERSITE DE FRANCHE-COMTE

**en Automatique et Informatique**
*(Ecole Doctorale Sciences Physiques pour l'Ingénieur et Microtechniques)*

## CONTRIBUTION A L'ORDONNANCEMENT CONJOINT DE LA PRODUCTION ET DE LA MAINTENANCE : APPLICATION AU CAS D'UN JOB SHOP

par

**Youssef HARRATH**

Soutenue le 16 décembre devant la Commission d'examen :

| | |
|---|---|
| Rapporteurs | ARTIBA A. Professeur, FUCAM, Belgique |
| | HENNET J-C. Directeur de recherche, LAAS-CNRS |
| Examinateurs | DI MASCOLO M. Chargée de recherche, LAG-CNRS |
| (Président de jury) | POMORSKI D. Professeur, Université Lille I |
| Directeur de thèse | ZERHOUNI N. Professeur, ENSMM |
| Co-encadrant | CHEBEL-MORELLO B. Maître de conférences, UFR de Franche Comté |

# Sommaire

# Table des figures

# Liste des tableaux

# Table des algorithmes

*Avec mes profonds sentiments d'amour et de reconnaissance, je dédis ma thèse à l'hommage de l'esprit saint et chaleureux, du coeur irremplaçable et de l'effort acharné de mon défunt père, source divine et éternelle de force ...*

*A ma mère, à qui j'exprime un profond amour et respect*

*à ma chère épouse, la rose de ma vie*

*à mon cher frère mohamed*

*à ma famille*

# Remerciements

Ce travail a été réalisé au Laboratoire d'Automatique de Besançon (LAB, UMR CNRS 6596) au sein de l'équipe Maintenance et sûreté de fonctionnement. Il n'aurait pu voir le jour sans le soutien de nombreuses personnes que je tiens à remercier.

La première personne, mon directeur de thèse, Monsieur Noureddine Zerhouni, Professeur à l'École Nationale Supérieure de Mécanique et de Microtechniques de Besançon, qui m'enseigna la rigueur d'un travail de longue haleine. Ses conseils tout au long de la thèse m'ont permis d'acquérir une maturité suffisante pour continuer dans le chemin de la recherche et de l'enseignement.

Je suis particulièrement reconnaissant à Madame Brigitte Chebel-Morello mon co-encadreur, Maître de conférences à l'UFR de Franche Comté, qui m'a soutenu durant les trois années de la thèse, m'a poussé à un travail intensif et a suivi mes travaux dans les moindres détails.

Je tiens à remercier sincèrement Monsieur Abdelhakim Artiba, Professeur aux Facultés Universitaires Catholiques de Mons en Belgique et Monsieur Jean-Claude Hennet, directeur de recherche au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS) d'avoir accepté de rapporter ce mémoire, pour leur gentillesse, leur disponibilité, leurs conseils et leur soutien. Je tiens également à exprimer toute ma gratitude à monsieur Denis Pomorski, Professeur à l'Université Lille I et à Madame Maria Di Mascolo, Chargée de recherche au Laboratoire d'Automatique de Grenoble (LAG-CNRS), qui ont bien voulu examiner ce travail.

Je remercie sincèrement toutes les personnes du Laboratoire qui contribuent à ce que le travail se fasse dans une ambiance chaleureuse.

# Introduction générale

Nos travaux présentés dans cette thèse concernent la résolution du problème d'ordonnancement dans un atelier de production de type job shop. Une première étude, nous a conduit à traiter le problème en supposant les machines toujours disponibles et en s'intéressant à l'optimisation du critère $C_{max}$. Tandis que dans un deuxième temps, nous avons tenu compte de l'état de disponibilité des machines, dépendant de l'activité de maintenance pouvant se planifier, à savoir des tâches de maintenance préventives périodiques. De par leur nature, ces tâches ne peuvent pas être considérées comme des opérations classiques dans un système de production, et nécessitent l'élaboration d'un critère approprié. Nous chercherons donc à déterminer un ordonnancement conjoint de la production et de la maintenance en optimisant deux critères, l'un lié à la production et l'autre à la maintenance.

Dans le premier chapitre, nous situons notre travail dans le cadre de l'ordonnancement des systèmes de production. Pour cela, nous présentons dans le premier chapitre la problématique de l'ordonnancement. Nous rappelons en premier lieu les différents éléments qui composent un problème d'ordonnancement, ainsi que les notations utilisées permettant de le caractériser. Nous présentons en second lieu une typologie des problèmes d'ordonnancement qui permet de distinguer les différents types d'ateliers. Cette classification nous amène à étudier la complexité des problèmes pour spécifier le degré de difficulté de notre problème d'ordonnancement. Ensuite, nous décrivons une méthode de modélisation des problèmes basée sur les graphes de précédence et une méthode de visualisation des solutions par le diagramme de Gantt. L'espace des ordonnancements peut être reparti en plusieurs zones selon l'affectation des opérations dans le temps. Nous identifions ainsi les quatre types d'ordonnancements : admissibles, semi actifs, actifs et sans retard. Notre intérêt s'est ensuite focalisé sur les méthodes de résolution développées dans la littérature. Ces méthodes sont classées en deux catégories : les méthodes exactes et les méthodes approchées, méthodes que nous exploitons dans la suite de notre travail. La deuxième partie du chapitre décrit le type d'atelier étudié dans cette thèse.

Nous présentons ainsi sa formalisation, ses contraintes et ses objectifs, sa complexité et un état de l'art des principales méthodes développées pour sa résolution. Enfin, nous parlons de la nécessité de l'ordonnancement conjoint au sein des ateliers.

Notre deuxième chapitre présente les algorithmes génétiques comme méthode approchée de résolution des problèmes d'ordonnancement en général et du problème de job shop en particulier. Pour cela, une description complète du fonctionnement des algorithmes génétiques est faite. Nous présentons en détail les opérateurs génétiques participant à l'exploration de l'espace de recherche et les paramètres nécessaires pour la convergence vers des bonnes solutions. Les algorithmes génétiques sont très utilisés pour la résolution des problèmes d'optimisation multiobjectif. Nous montrerons les différences entre un algorithme génétique monobjectif et celui optimisant plusieurs objectifs. Un état de l'art des méthodes génétiques multiobjectif sera classé selon la notion de l'optimisation Pareto optimale. La dernière partie du deuxième chapitre applique les algorithmes génétiques aux problèmes d'ordonnancement en général et au problème du job shop en particulier. Un état de l'art sur les différents types de codage et sur l'adaptation des opérateurs génétiques au traitement du problème de job shop sont présentés.

La mise au point des algorithmes génétiques est une tâche délicate. En effet, le choix des paramètres génétiques qui dépendent de la taille du problème traité, n'est pas toujours évident. Cette difficulté apparaît essentiellement lorsque les algorithmes génétiques sont utilisés pour la résolution des problèmes de job shop de grande taille. Pour cela, nous proposons dans ce troisième chapitre une démarche d'élaboration d'une heuristique facile à implanter pour la résolution du job shop. Dans un premier temps, nous travaillons dans l'espace de solutions d'un ensemble de problèmes de job shop, pour générer un ensemble de bonnes solutions à l'aide d'un algorithme génétique. Nous cherchons ensuite à expliquer cet espace de bonnes solutions, par les caractéristiques du problème se trouvant dans l'espace de définition, à l'aide du processus d'Extraction des Connaissances à partir des Données. Les données étant les ordonnancements solutionnant le problème de job shop. La connaissance, étant les règles d'ordonnancement obtenues à partir de ces solutions. Dans un second temps, nous étudions des solutions déterminées par un algorithme génétique, afin de substituer à cette métaheuristique, dont les paramètres ne sont pas aisés à trouver, une heuristique performante facile à mettre en oeuvre et résolvant la même catégorie de problèmes.

L'Extraction de Connaissance à partir de Données (ECD) ou Knowledge Discovery in Databases (KDD) est un processus non trivial d'identification de structures inconnues, valides et potentiellement exploitables. Son but est l'extraction d'information utile conte-

nue dans les bases de données, à travers la mise en exergue des relations dominantes entre les exemples qui les composent. Elle se réfère à une démarche complète d'exploitation des données que l'on peut résumer en quatre phases distinctes : L'acquisition des données, le pré-traitement, la fouille de données et le post-traitement. Nous avons adapté ce processus étape par étape, à la résolution de problème d'ordonnancement dans un atelier de type job shop.

La première étape consiste à générer un ensemble de bonnes solutions de plusieurs problèmes de job shop en utilisant un algorithme génétique. Ses solutions seront caractérisées par des attributs numériques liés au problème, tels que les temps opératoires, les charges des machines, la durée totale d'un job etc. L'étape de pré-traitement consiste à transformer les solutions sous la forme de liste d'opérations devant chaque machine. Dans un souci de généralisation, nous avons utilisé après un état de l'art dans le domaine, un algorithme de discrétisation des attributs numériques. Cet algorithme dit supervisé, tient compte à la fois des caractéristiques des individus ainsi que les classes auxquelles ils appartiennent. Puis, l'étape importante de fouille de solutions générées par l'algorithme génétique, permet la découverte d'un ensemble de règles de priorité (ou règles d'affectation des opérations) en établissant les liens entre la position de chaque opération sur sa machine et ses caractéristiques. La dernière étape, celle du post-traitement, consiste à transformer les règles trouvées en une heuristique permettant de résoudre plusieurs problèmes de job shop de différentes tailles.

Il s'avère que la dernière étape de cette démarche d'ECD, généralise en effet la résolution du problème à des tailles supérieures, mais ne garantit pas une solution optimale. Pour obtenir des résultats précis par cette démarche, on n'applique pas la dernière étape de l'ECD (c'est un compromis entre précision et généralisation). Nous avons appliqué la même démarche de l'ECD à l'exception de la dernière étape de post traitement à un problème de taille 15x5 (15 jobs 5 machines). Les règles d'ordonnancement obtenues sont appliquées sur des problèmes de même taille. Les résultats expérimentaux ont montré l'efficacité de cette deuxième méthode de résolution.

Le quatrième chapitre présente une adaptation des algorithmes génétiques pour la résolution d'un problème multiobjectif prenant en compte l'aspect de la maintenance. En effet, une grande partie de la littérature dédiée aux problèmes d'ordonnancement se place dans le contexte de disponibilité totale des ressources. Cette hypothèse n'est pourtant pas fidèle à la réalité des ateliers de production. En effet, les différentes ressources qu'elles soient humaines ou matérielles peuvent, pour diverses raisons, être indisponibles. Pour ces raisons, nous étudierons dans le quatrième chapitre l'ordonnancement conjoint de la

production et de la maintenance au sein du même type d'atelier qu'au chapitre précédent.

Nous présentons en premier lieu un état de l'art sur la maintenance et ses différents types. Nous optons pour la maintenance préventive systématique pour l'appliquer dans l'atelier de job shop. L'une des difficultés majeures de ce type de maintenance est le choix des périodes d'interventions. Nous proposons dans ce cadre deux méthodes de choix de périodes systématiques. La première consiste à fixer des périodes de maintenance préventive indépendamment des charges de travail des machines. La deuxième prend en compte ce dernier paramètre. Nous développons ensuite un algorithme génétique multiobjectif pour la résolution du problème d'ordonnancement conjoint de la production et de la maintenance. Cet algorithme génétique génère des solutions pareto optimales ; solutions que nous validerons par des bornes inférieures.

# Chapitre 1

# Introduction à l'ordonnancement des systèmes de production

***Résumé :*** *Les problèmes d'ordonnancement constituent un domaine très vaste de la recherche opérationnelle. Une des problématiques difficiles à gérer dans un système de production est celle de l'ordonnancement. Les ouvrages et les articles traitant cette problématique sont nombreux. Ce premier chapitre n'est autre qu'une introduction à l'ordonnancement en général et à l'atelier de production de type job shop en particulier. Nous rappelons en premier lieu les définitions et les différentes typologies et notations existantes qui seront utiles dans la suite de ce mémoire. Une classification des problèmes d'ordonnancement ainsi qu'un rappel sur les notions de complexité sont par la suite abordés.*

*La deuxième partie de ce chapitre est consacrée à la description du problème d'ordonnancement dans un atelier de type job shop. Ce problème, constitue en effet le centre de nos intérêts dans cette thèse. Les études de complexité ont montré que la majorité des instances du job shop sont $\mathcal{NP}$-difficiles. Néanmoins, quelques algorithmes polynômiaux sont développés pour résoudre des instances particulières de taille limitée. Un état de l'art détaillé des méthodes de résolution exactes et approchées est présenté à la fin de ce chapitre. La plupart de ces approches traitent le cas d'optimisation mono objectif. Les rares travaux s'intéressant à étudier plusieurs objectifs ne considèrent que des objectifs liés à la production (le makespan, le retard max, la somme des retards, etc). Nous allons par conséquent, montrer l'intérêt de tenir compte de l'aspect de la maintenance au sein du job shop. Ceci nous mènera à proposer dans le quatrième chapitre une approche génétique pareto optimale pour l'ordonnancement conjoint de la production et de la maintenance dans un job shop.*

## 1.1 Introduction

L'ordonnancement occupe une place particulière dans la gestion informatisée des flux de production au sein de l'entreprise. C'est généralement le point de rencontre entre un système hiérarchisé et informatisé de production et le système de production lui-même. C'est le lieu de l'interface entre l'élaboration globale de la commande et la partie opérationnelle. La gestion de production crée les ordres de fabrication qui déterminent globalement ce qui doit être fait dans une période donnée (généralement la semaine). L'ordonnancement consiste à prévoir l'enchaînement de toutes les opérations élémentaires nécessaires à la réalisation de ces ordres de fabrication sur les ressources de production, tout en tenant compte des ressources secondaires (telles que les opérateurs, les outillages, etc.), des contraintes extérieures (maintenance préventive, calendrier de travail, etc.) et de l'existant (reste de la semaine précédente, tâches en cours de réalisation, etc.).

L'ordonnancement est un processus de décision qui apparaît dans la plupart des systèmes de production et de transport ainsi que dans la gestion de projet [Pin95]. L'ordonnancement trouve sa place essentiellement dans les ateliers ayant une production diversifiée en flux poussé (tels que le flow shop et le job shop), gérée par des ordres de fabrication. En effet, les systèmes de production mono produit ou aux ateliers gérés en Kanban, les flux de produit s'écoulent naturellement et doivent en principe s'autoréguler. Concrètement, l'ordonnancement d'atelier tel que le job shop, consiste à régler le passage de chaque ordre de fabrication (ou chaque produit) sur l'ensemble des machines, en respectant les contraintes d'ordre (gamme), de date (respect des calendriers et des opérations de maintenance), et de ressources secondaires (outillages, opérateurs) dans le contexte. L'ordonnancement peut être vu comme un problème d'optimisation combinatoire. Il faut trouver une bonne solution, voire optimale au regard d'un ou de plusieurs critères d'évaluation.

Nous pouvons distinguer deux catégories de problèmes d'ordonnancement. En fonction du degré de connaissance que l'on a du problème à résoudre, on parlera de :

- **problème statique** : lorsque les tâches à ordonner sur une période ainsi que l'état initial de l'atelier sont connus au début de la période ;
- **problème dynamique** : lorsque les décisions sont à prendre sur la période mais toutes les tâches à réaliser sur cette période ne sont pas connues au début de la période.

Nous traiterons dans cette thèse un problème statique.

Le résultat de l'ordonnancement est un calendrier précis des tâches à réaliser. Il se décompose en trois grandeurs fondamentales :

- **l'affectation** : consistant à choisir, s'il y a lieu, les ressources nécessaires à une tâche ;
- **le séquencement** : qui donne l'ordre de passage des tâches sur chaque ressource ;
- **le datage** : qui donne pour chaque tâche une date de début et une date de fin.

De plus, un ordonnancement se décompose en deux parties toujours présentes dans l'atelier, mais pouvant revêtir une importance variable :

- **l'ordonnancement prédictif** : consiste à prévoir "à priori" un certain nombre de décisions en fonction de données prévisionnelles et d'un modèle de l'atelier ;
- **l'ordonnancement réactif** : consiste à adapter les décisions prévues en fonction de l'état courant du système et des déviations entre la réalité et le modèle (ce qui est prévu en théorie).

L'ordonnancement est un élément crucial dans l'ensemble des tâches liées au pilotage d'atelier, que l'on peut décrire par un cycle d'ordonnancement, de contrôle, de suivi et de réaction. Dans notre travail, nous nous limiterons à l'aspect ordonnancement qui joue un rôle central, surtout lorsque les ateliers ne sont pas entièrement automatisés. En effet, dans ce cas, le contrôle et la supervision revêtent un caractère moins important. Notre étude se concentrera sur l'ordonnancement prédictif. Les notions générales de l'ordonnancement seront définies dans ce premier chapitre, et seront décrites dans le cadre plus particulier d'un atelier de type job shop. Nous rappelons dans ce chapitre les éléments d'un problème d'ordonnancement ainsi que les notations pouvant être utilisées dans le reste du mémoire. Ensuite, une typologie des problèmes en fonction du type d'ateliers et quelques notions de complexité seront présentées. Nous aborderons aussi les méthodes exactes et approchées les plus connues pour résoudre un problème d'ordonnancement.

## 1.2 Notions générales d'ordonnancement

**Définition 1.1.** *[Pin95] L'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, etc.) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les tâches. Un ordonnancement décrit l'ordre d'exécution des tâches et l'allocation des ressources au cours du temps, afin de satisfaire un ou plusieurs critères d'optimisation.*

### 1.2.1 Les éléments d'un problème d'ordonnancement

D'après la définition de l'ordonnancement cités ci-dessus, nous en déduisons qu'un ordonnancement est constitué principalement de quatre éléments suivants : les *tâches*, les *ressources*, les *contraintes* et les *objectifs* ou les *critères* d'optimisation. Notions que nous définissons dans ce paragraphe.

#### 1.2.1.1 Les tâches

Ces sont toutes les opérations élémentaires de travail à effectuer pour la fabrication d'un produit (tel que le perçage d'un trou). Une tâche $i$ est généralement caractérisée par une *date de début* $s_i$ et/ou par une date de *fin* $c_i$ (completion time) et une durée d'exécution $p_i$. Une tâche peut également être définie par une date limite $\bar{d}$ appelée aussi date absolue au delà de laquelle il n'est plus possible de l'exécuter. Certaines contraintes techniques ou économiques peuvent associer aux tâches des dates de début au plus tôt $r_i$ (ready time) ou des dates de fin au plus tard $d_i$ (due date). Une tâche peut se réaliser par morceaux, il s'agit alors de tâche préemptive, ou sans interruption, on parle alors de tâche non-préemptive. Certaines tâches présentent une *priorité* d'exécution pour les clients, avant de commencer à chercher un ordonnancement, on leur attribue des poids $w_i$ proportionnels à leurs degrés de priorité.

#### 1.2.1.2 Les ressources

L'exécution des différentes tâches nécessite la mise en oeuvre d'un ensemble de moyens techniques et d'opérateurs humains. Cet ensemble représente donc les ressources indispensables à la réalisation des tâches durant des intervalles de disponibilité. La capacité d'une ressource est en réalité limitée. Nous ne tenons pas compte de cette limitation dans la première partie de notre étude. Par contre, cette hypothèse est considérée dans le quatrième chapitre où l'on suppose que les ressources sont sujettes à des pannes et nécessitent des interventions de maintenance.

Il existe plusieurs types de ressources. Si après son utilisation la ressource est à nouveau disponible avec la même capacité, on parle de ressource *renouvelable* (tels que les hommes et les machines). Par contre, si après son utilisation la ressource est disponible avec une capacité inférieure ou nulle, on parle de ressource *consommable* (l'argent est un bon exemple de ressource consommable). Certaines ressources sont capables de réaliser plusieurs tâches en parallèle, elles sont dites *cumulatives* (une station de travail composée de plusieurs machines est une ressource cumulative). Cette dernière caractéristique est considérée généralement comme étant une hypothèse forte. Les ressources sont souvent

supposées *disjonctives* et ne permettent de réaliser qu'une seule opération à la fois (un ouvrier s'occupant de plusieurs machines est une ressource disjonctive).

#### 1.2.1.3 Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent avoir les variables de décision. Leur prise en compte permet d'avoir un ordonnancement réalisable. Il existe deux classifications possibles des contraintes. La première est classique, elle comporte les contraintes temporelles et celles liées aux ressources. La deuxième présentée par Kacem [Kac03] distingue les contraintes endogènes qui sont liées directement au système de production et à ses performances, des contraintes exogènes indépendantes du système. Nous adoptons la première classification pour décrire les différents types de contraintes qu'un problème d'ordonnancement peut avoir.

- **les contraintes temporelles** : souvent, certaines opérations ne peuvent s'exécuter qu'après une date de début au plus tôt, c'est-à-dire en satisfaisant l'expression suivante : $c_i - p_i \geq r_i$ et ne finissent qu'avant une date de fin au plus tard ce qui revient à vérifier la formule suivante : $r_i + p_i \leq r_i$. Ceci reflète la non disponibilité continue des ressources (pour les dates au plus tôt), et la nécessité de délivrer les produits en respectant les délais (pour les dates au plus tard). Ce type de contraintes peut aussi définir des relations de précédence entre les opérations. Ce sont surtout des contraintes technologiques qui imposent un ordre d'exécution aux opérations (ex : perçage d'un trou, puis insertion d'une vis). La relation de précédence entre deux tâches $i$ et $j$ (c.à.d $i$ précède $j$), peut s'écrire sous la forme suivante : $s_i + p_i \leq s_j$ ;
- **les contraintes liées aux ressources** : les caractéristiques des ressources peuvent induire des contraintes indiquant les conditions de passage des tâches. Dans le cas d'une ressource consommable, seules les tâches ayant une consommation inférieure au égale à la capacité de la ressource à l'instant $t$ peuvent être exécutées. Par contre, pour une ressource renouvelable, ce type de contraintes permet de limiter sa capacité. Cette limitation apparaît surtout dans le cas des ressources cumulatives où l'on ne peut exécuter qu'un certain nombre de tâches en parallèle (exécuter simultanément plusieurs opérations sur la même ressource rend l'ordonnancement inadmissible. Il est par exemple impossible d'effectuer simultanément trois opérations de perçage si l'on ne dispose que de deux foreuses). Certaines ressources permettent d'interrompre l'exécution d'une tâche pour s'occuper d'une autre, on parle alors de contraintes de préemption (les machines multi-processeurs réalisent ce type de contraints). Les contraintes d'indisponibilité pendant des périodes bien définies

sont rarement prises en compte. En réalité, ce type de contraintes est fréquemment rencontré surtout pour planifier des périodes de maintenance systématiques.

#### 1.2.1.4 Les objectifs

Lorsque l'on aborde un problème d'ordonnancement, il est crucial de définir un certain nombre d'objectifs à atteindre. Il s'agit ici d'optimiser (maximiser ou minimiser) une fonction d'évaluation en respectant un certain nombre de contraintes. Le credo des ateliers reste le triptyque coût/qualité/délais. Toute évaluation d'un ordonnancement n'aura donc de sens aux yeux de la production que dans la mesure où elle porte sur ces trois facettes.

Le délai est l'élément du triptyque qui se décline le mieux sur l'axe des critères classiques de l'ordonnancement. Il évalue en termes de temps les performances d'un système, par exemple : le temps de cycle $F_i$, le retard algébrique d'une opération $L_i$ qui seront définis ultérieurement. L'objectif du respect des délais, consiste souvent à définir un ordre de fabrication durant une période donnée. Notons que l'objectif "délais" se traduit plus volontiers par des contraintes que par un critère à minimiser. En effet, le but d'un responsable d'atelier est de respecter la période de temps qui lui est affectée pour effectuer un ensemble de produits. Dans la majeure partie des cas, l'ajustement de la capacité à la charge se fait par les opérateurs (ressource humaine plus flexible) alors que les moyens de production sont eux-mêmes sur-capacitifs.

Il est rare d'associer la qualité des produits à l'évaluation d'un ordonnancement. Notons toutefois que les responsables de production associent souvent la qualité au volume d'encours[1] : la fluidité du flux est souvent assimilée à un facteur de qualité. Dans cette hypothèse, il est clair que l'axe qualité peut se traduire par la minimisation de l'encours. Malheureusement, l'encours n'est pas un critère utilisé en ordonnancement théorique. Or ce type de critère est régulier. Cela signifie qu'il peut s'améliorer si on accepte de retarder le début d'une opération.

Le coût est aussi assez rarement pris en compte. Si l'on excepte le coût de réglage, le coût "machine" est souvent invariant. La différence de coût entre deux ordonnancements relève plus souvent de l'utilisation de la main d'oeuvre. En effet, de multiples petits arrêts des machines immobilisent du personnel improductif.

1. Les encours représentent les produits inachevés pendant une période de production. Ils sont notés $F_i$ et peuvent être déterminés par le temps de présence des travaux dans l'atelier soit $F_i = C_i - r_i$, où $C_i$ est la date de fin du job $i$ et $r_i$ est sa date d'arrivée. La minimisation de ce critère permet de réduire le coût du stockage.

Minimiser le makespan[2] peut concourir à minimiser le coût de la production en ce sens que le besoin en opérateurs est borné par $M * makespan$ si $M$ est le nombre de machines. Malheureusement cette borne est rarement atteinte et il est alors plus efficace de réfléchir à d'autres solutions. La contrainte d'ordonnancement la plus proche est la contrainte de "no-idle". Cette contrainte interdit l'arrêt machine et impose donc une activité continue. Elle n'a été mise au point (et encore rapidement) qu'en flow shop. En effet, dans le cas du job shop, elle n'est pas toujours satisfaisante. La perte financière est associée à la longueur de l'arrêt, il faudrait donc des modèles plus fouillés pour exprimer réellement le coût d'un ordonnancement.

Pour conclure, il est clair que les critères utilisés en ordonnancement classique ne reflètent pas exactement les préoccupations réelles des opérateurs en production. Ce manque de convergence explique sans doute en partie le nombre limité d'applications concrètes des méthodes classiques.

Le tableau 1.1 résume les objectifs (appelés aussi critères) les plus utilisés pour mesurer la qualité d'un ordonnancement en terme de coût/qualité/délais.

### 1.2.2 Notation des problèmes

Plusieurs notations sont apparues dans la littérature pour simplifier la description des problèmes d'ordonnancement. Nous allons adopter la notation proposée par Graham et al. [GLLK79] et par Blazewicz et al. [BLK83]. Cette notation est constitué de trois champs $\alpha/\beta/\gamma$.

#### 1.2.2.1 Le champ $\alpha$ : les machines (ou processeurs)

Le champ $\alpha$ permet d'identifier le nombre et le type de machines disponibles, il est généralement constitué de trois éléments : $\alpha_1$ , $\alpha_2$ et $\alpha_3$.

- $\alpha_1 \in \{o, P, Q, R, O, F, J\}$ ;
  - $\alpha_1 = o$ : l'atelier dispose d'une seule machine pour exécuter les tâches. Par conséquent, les jobs sont constitués d'une seule tâche chacun. Le temps d'exécution d'une tâche $j$ est noté $p_j$ ;
  - $\alpha_1 = P$ : un ensemble de machines parallèles identiques est susceptible de réaliser les tâches. La durée d'exécution d'un job donné est la même sur toute les machines ;

2. Le makespan étant la date de fin de la dernière opération exécutée. Ce paramètre est très utilisé pour l'évaluation de la qualité d'un ordonnancement.

- $\alpha_1 = Q$ : les machines disponibles pour la réalisation des jobs sont dites parallèles uniformes. Chaque machine possède une vitesse appropriée notée $b_i$. La durée d'exécution d'un job $j$ sur la machine $M_i$ notée $P_{ij}$ est égale à $p_j/b_i$ ;
- $\alpha_1 = R$ : les machines disponibles sont parallèles quelconques. Le temps de réalisation d'une activité dépend de la machine sur laquelle elle sera affectée. On parlera de : *Unrelated Parallel Machine* ;
- $\alpha_1 \in \{O,\ F,\ J\}$ : l'atelier est composé d'un ensemble de machines pour la réalisation des jobs. Le type de l'atelier dépend de la nature des gammes de production. L'atelier est dit open shop si les jobs ne possèdent pas de gamme. Dans le cas où les jobs se réalisent suivant des gammes linéaires il s'agit d'un atelier de type flow shop. Si les jobs suivent des gammes différentes les unes des autres, l'atelier est de type job shop.

- $\alpha_2 \in \mathbb{Z}+$ : le nombre de machines $m$ est constant ;
- $\alpha_3 = o$ : le nombre de machines $m$ est variable.

#### 1.2.2.2 Le champ $\beta$ : les contraintes

Le champ $\beta$ caractérise les conditions d'exécution des jobs ainsi que les états des ressources présentes dans l'atelier. Il indique l'existence ou non des contraintes de précédence entre les tâches, la possibilité de tolérer la préemption etc. Ce champ se décompose généralement en cinq sous champs $\beta_1$, $\beta_2$, $\beta_3$, $\beta_4$ et $\beta_5$.

- $\beta_1 \in \{pmtn,\ o,\ split\}$ ;
  - $\beta_1 = pmtn$ : dans ce cas, la préemption entre les tâches est permise. On peut alors interrompre l'exécution d'une tâche et la reprendre plus tard ;
  - $\beta_1 = o$ : pas de préemption entre les tâches ;
  - $\beta_1 = split$ : si une tâche est décomposable en plusieurs sous tâches, alors il est autorisé de l'exécuter simultanément sur plusieurs machines.

- $\beta_2 \in \{Prec,\ Tree,\ o\}$ ;
  - $\beta_2 = Prec$ : les tâches des jobs sont reliées par des relations de précédence ;
  - $\beta_2 = Tree$ : les relations de précédence entre les tâches sont sous la forme d'un arbre ;
  - $\beta_2 = o$ : pas de relations de précédence entre les tâches.

- $\beta_3 \in \{r_j, o\}$ ;
  - $\beta_3 = r_j$ : chaque tâche $j$ possède une date début au plus tôt "ready time" à partir de laquelle elle peut s'exécuter ;
  - $\beta_3 = o$ : toutes les tâches sont exécutables à partir de la date $t = 0$.
- $\beta_4 \in \{p_j = 1, p_{ij} = 1, o\}$ ;
  - $\beta_4 = p_j = 1$ : si $\alpha_1 \in \{o, P, Q\}$ alors toutes les tâches possèdent une durée d'exécution égale à une unité de temps ;
  - $\beta_4 = p_{ij} = 1$ : si $\alpha_1 \in \{O, F, J\}$ alors toutes les tâches possèdent une durée d'exécution égale à une unité de temps ;
  - $\beta_4 = o$ : les durées d'exécution des tâches sont toutes positives.
- $\beta_5 \in \{S_{nsd}, R_{nsd}, R_{sd}, b_{j,j+1}\}$.
  - $\beta_5 = S_{nsd}$ : les machines nécessitent un temps de montage qui est indépendant des séquences ;
  - $\beta_5 = R_{nsd}$ : les machines nécessitent un temps de démontage qui est indépendant des séquences ;
  - $\beta_5 = R_{sd}$ : les machines nécessitent un temps de démontage qui dépend des séquences ;
  - $\beta_5 = b_{j,j+1}$ : possibilité de stockage limité entre les machines $M_j$ et $M_{j+1}$.

#### 1.2.2.3 Le champ $\gamma$ : les critères

Le champ $\gamma$ représente le critère d'optimisation étudié. Afin de décrire les critères les plus utilisés pour caractériser un ordonnancement donné, nous introduisons les notations suivantes :

- $r_j$ : date de disponibilité de la tâche $j$ ou encore date au plus tôt (ready time) ;
- $d_j$ : date échue ou date au plus tard (due date)de la tâche $j$;
- $w_j$ : poids (définit la priorité de l'exécution de la tâche $j$ pour les clients) ;
- $c_j$ : date d'achèvement de la tâche $j$ (completion time) ;
- $f_j = c_j - r_j$ : durée de flot indiquant la durée d'attente et d'exécution de la tâche $j$ dans le système (flow time) ;

- $l_j = c_j - d_j$ : avance ou retard algébrique (lateness) de la tâche $j$ ;
- $t_j = max\{c_j - d_j, 0\}$ : retard absolu de la tâche $j$;
- $e_j = max\{d_j - c_j, 0\}$ : avance absolue de la tâche $j$;
- $u_j$ : indiquant si la tâche $j$ est en retard par rapport à sa date de début d'exécution, elle est donnée par l'expression suivante :

$$u_j = \begin{cases} 1 & si \ \ c_j > d_j \\ 0 & si \ \ non \end{cases} \tag{1.1}$$

Le tableau 1.1 résume les critères les plus utilisés pour caractériser un ordonnancement donné.

| Critères | Définitions |
|---|---|
| $C_{max} = \max c_j$ | date d'achèvement de l'ordonnancement (makespan) |
| $F = \sum f_j$ | flot total |
| $\overline{F} = \frac{1}{n} \sum f_j$ | flot moyen |
| $\overline{F}_w = \sum w_j f_j / \sum j w_j$ | flot moyen pondéré |
| $L = \sum l_j$ | somme des retards algébriques |
| $L_{max} = \max l_j$ | retard algébrique maximum |
| $T = \sum t_j$ | somme des retards |
| $T_{max} = \max\{t_j\}$ | retard maximum par rapport aux dates de début au plus tard |
| $\overline{T} = \frac{1}{n} \sum t_j$ | retard moyen |
| $\overline{T}_w = \sum w_j t_j / \sum w_j$ | retard moyen pondéré |
| $E = \sum e_j$ | somme des avances |
| $u = \sum u_j$ | nombre de tâches en retard |

TAB. 1.1 – *Critères d'optimisation*

### 1.2.3 Typologie des problèmes d'ordonnancement

Les problèmes d'ordonnancement peuvent être classés en deux catégories et ceci en fonction du nombre de machines nécessaires pour réaliser chaque tâche. La première catégorie regroupe les problèmes pour lesquels chaque tâche nécessite une seule machine, la deuxième, ceux pour lesquels chaque tâche demande plusieurs machines pour son exécution. La première catégorie de problèmes concerne les ateliers à machines parallèles ou non dédiées. Ce type d'atelier se caractérise par le fait que plusieurs machines sont disponibles pour l'exécution d'un travail qui n'en nécessite qu'une seule.

Il s'agit ici d'une généralisation du problème à une seule machine. Il est possible de distinguer trois types d'ateliers selon la vitesse d'exécution des machines :

- les ateliers à machines identiques : toute tâche peut s'exécuter sur n'importe quelle machine avec une même durée opératoire (à condition que la machine soit libre) ;
- les ateliers à machines uniformes : chaque machine possède sa propre vitesse et ceci indépendamment de la tâche à exécuter ;
- les ateliers à machines indépendantes (ou non reliées) : la vitesse des machines dépend de la tâche à effectuer.

La deuxième classe de problèmes, quant à elle, englobe les ateliers composés de $m$ stations différentes. Une station comporte une ou plusieurs machines en parallèle. Les travaux à réaliser sont constitués d'un ensemble de $n$ opérations élémentaires. chacune d'elles doit s'exécuter sur une machine différente de celles des autres opérations. En fonction du mode de passage des opérations sur les machines (appelé aussi gamme opératoire), trois classes d'atelier sont distinguées, à savoir le flow shop, le job shop et le open shop.

#### 1.2.3.1 Flow shop

Le *flow shop* se rencontre dans les ateliers disposant de lignes de production dédiées à la production de masse de peu de variété de produit. Un tel atelier est aussi appelé «atelier en ligne» ou à cheminement unique où toutes les gammes sont identiques. Chaque job est constitué de $m$ opérations (où $m$ est le nombre de machines). Chacune de ces opérations doit être exécutée sur une machine différente. L'ordre de passage des opérations sur les machines est le même pour tous les jobs. Il existe plusieurs versions de flow shop qui diffèrent par le type des gammes opératoires. On parle de *flow shop hybride* s'il est possible de trouver plusieurs machines pour réaliser une opération donnée. Une autre version de flow shop appelée *flow shop de permutation* consiste à affecter les jobs dans un ordre identique pour toutes les machines.

#### 1.2.3.2 Job shop

Dans cette classe d'atelier, chaque tâche possède son propre mode de passage sur les machines. A titre d'exemple, considérons le problème de teinture de tissus. Suivant le type de tissus et la couleur désirée, l'ordre de passage dans les différents bains ne sera pas le même. Le *job shop* est notre cas d'étude, nous allons détailler dans la deuxième partie

de ce chapitre, son formalisme mathématique ainsi qu'un état de l'art des méthodes développées pour le résoudre.

#### 1.2.3.3 Open shop

Dans un atelier *open shop*, le cheminement des jobs est multiple, mais à la différence du *job shop*, les jobs ne possèdent pas de gammes. L'ordre de passage des opérations est ainsi quelconque (atelier à cheminement libre). Ce paramètre est déterminé lors de l'ordonnancement. Nous citons l'exemple de l'organisation des tests médicaux dans un hôpital [Esp98] où l'ordre de réalisation des examens n'a pas d'importance.

### 1.2.4 Notions générales sur la complexité

Nous nous intéressons à des problèmes d'ordonnancement de différents niveaux de difficulté. La difficulté de ces problèmes peut être spécifiée en fonction des données telles que le nombre de machines, le nombre de tâches etc. La résolution de ces problèmes s'est faite par différents types d'algorithmes dont la complexité varie. Avant d'aborder les méthodes de résolution, nous donnons des notions générales sur la complexité. En effet, la théorie de la complexité offre un cadre d'étude mathématique dans lequel les problèmes peuvent être classés en problèmes faciles ou difficiles.

#### 1.2.4.1 Complexité des algorithmes

Le temps et l'espace mémoire sont les paramètres les plus utilisés pour déterminer la performance des algorithmes résolvant les problèmes d'optimisation. C'est pourquoi il faut choisir parmi les algorithmes pouvant résoudre un problème donné, celui qui nécessite le minimum d'espace mémoire et qui converge le plus vite. Nous nous intéressons ici à mesurer la performance d'un algorithme par rapport au temps de calcul nécessaire.

**Définition 1.2.** *On appelle complexité en temps d'un algorithme, la fonction $f(n)$ qui représente le nombre maximum d'opérations élémentaires effectuées pour résoudre un problème de taille $n$ ($n$ étant le nombre de variables décrivant le problème). On associe ainsi une unité de temps à chaque opération élémentaire.*

**Définition 1.3.** *On dit que $f(n) \in \mathcal{O}(g(n))$, s'il existe une constante $c > 0$ et un entier $n_0$ tels que $\forall\ n \geq n_0$, $|f(n)| \leq c|g(n)|$.*

**Définition 1.4.** *Un algorithme est dit polynômial si sa fonction de complexité $f(n) \in \mathcal{O}(p(n))$, où $p$ est un polynôme en $n$, c'est- à-dire, s'il existe une constante $k > 0$ telle que $f(n) \in \mathcal{O}(n^k)$.*

Tout algorithme dont la fonction de complexité ne peut pas être majorée par un polynôme est dit *exponentiel*. Pour montrer qu'un algorithme est exponentiel, il suffit de minorer sa fonction de complexité $f(n)$ par une fonction exponentielle de type $ck^n$ où $c > 0$ et $k > 1$. En pratique, les fonctions de complexité des algorithmes polynômiaux en ordonnancement sont des polynômes en nombre de tâches $n$, de machines $m$, en $\log(maxp_j)$, etc.

#### 1.2.4.2 Complexité des problèmes

Nous faisons ici la distinction entre un problème d'*optimisation* et un problème de *décision*. Un problème d'optimisation est un problème pour lequel on doit chercher une solution admissible optimisant au mieux une fonction objectif. Un problème de décision est un énoncé auquel la réponse peut être uniquement oui ou non [Fin99]. Chaque problème d'optimisation possède un problème de décision correspondant. A titre d'exemple, considérons le problème d'optimisation $P//Cmax$ avec $n$ tâches et soit $y$ un entier positif. Un problème de décision associé à ce problème d'optimisation peut être le suivant : existe-t-il un ordonnancement avec $Cmax \leq y$ ?

Les problèmes de décision sont divisés selon leur degré de complexité en plusieurs classes. Les problèmes de décision qui peuvent être résolus par un algorithme polynômial appartiennent à la classe $\mathcal{P}$. La classe $\mathcal{NP}$ regroupe quand à elle les problèmes de décision pour lesquels on possède pour chaque instance $I$ de taille $L(I)$ et ayant une réponse "oui" un schéma de vérification polynômial en $L(I)$, vérifiant en un temps polynômial la validité de la réponse "oui". Il est clair que $\mathcal{P} \subseteq \mathcal{NP}$. Au jour actuel $\mathcal{P} \neq \mathcal{NP}$ car la classe $\mathcal{NP}$ contient des problèmes difficiles pour lesquels aucun chercheur n'est arrivé, jusqu'à maintenant, à trouver d'algorithmes polynômiaux pour les résoudre. Ces problèmes sont appelés *$\mathcal{NP}$-complets*.

La notion principale, pour définir la *$\mathcal{NP}$-complétude*, est celle de la réduction polynomiale. Un problème de décision $\pi_1$ est dit réductible polynomialement à un autre problème de décision $\pi_2$, et on note $\pi_1 \ \alpha \ \pi_2$, s'il existe une fonction polynomiale $f$ qui transforme toute instance de $\pi_1$ en une instance de $\pi_2$ de telle manière que la réponse pour $\pi_1$ est "oui" si, et seulement si, la réponse pour $\pi_2$ est "oui".

**Définition 1.5.** *Un problème de décision $\pi_1$ est $\mathcal{NP}$-complet si $\pi_1 \in \mathcal{NP}$ et $\forall\ \pi_2 \in \mathcal{NP}$, on a $\pi_2 \ \alpha \ \pi_1$.*

A partir de la dernière définition, il en découle que si $\pi_1$ et $\pi_2$ sont deux problèmes de décision tels que $\pi_1$ $\alpha$ $\pi_2$ alors :

- si $\pi_2 \in \mathcal{P}$ alors $\pi_1 \in \mathcal{P}$ ;
- si $\pi_1$ est $\mathcal{NP}$-*complet* alors $\pi_2$ l'est aussi.

Dans la classe de problèmes $\mathcal{NP}$-*complets*, on distingue deux types de problèmes : les problèmes $\mathcal{NP}$-*complets au sens faible* et les problèmes $\mathcal{NP}$-*complets au sens fort*. Un problème est dit $\mathcal{NP}$-*complet* au sens faible, s'il est $\mathcal{NP}$-*complet* et qu'il existe un algorithme *pseudo-polynômial* pour le résoudre. Sachant, qu'un algorithme est pseudo-polynômial si sa fonction de complexité est de la forme $\mathcal{O}(p(L(I),|I|_{max})$ où $|I|_{max}$ est la longueur maximale d'une instance I du problème à résoudre. Un problème $\Pi$ est $\mathcal{NP}$-*complet* au sens fort, s'il existe un polynôme $p$ fonction de $|I|_{bin}$ (selon un codage binaire), tel que, pour toute instance $I$ de $\Pi$, $|I|_{max} \leq p(|I|_{bin})$.

Un problème d'optimisation est dit $\mathcal{NP}$-*difficile* si le problème de décision qui lui correspond est $\mathcal{NP}$-*complet*.

### 1.2.5 Modélisation et représentation des ordonnancements

Il existe deux types de modélisation d'un problème ordonnancement. La modélisation graphique sous forme de graphe de précédence et la représentation analytique sous forme de programme mathématique. Nous avons opté pour le premier type de modélisation qui est très utilisé dans la littérature et qui présente un caractère visuel facilitant l'interprétation des solutions. Nous détaillons par la suite les graphes de précédence et nous présentons une méthode de visualisation d'un ordonnancement.

#### 1.2.5.1 Modélisation

Un graphe de précédence est constitué d'un ensemble de noeuds et de deux types d'arcs. Les noeuds représentent les tâches à réaliser. Les arcs conjonctifs valués de la durée de réalisation de la tâche d'où l'arc sort, représentent les contraintes de précédence. Les arcs disjonctifs à deux sens représentent les contraintes de ressources. Pour représenter le début et la fin de l'ordonnancement, deux noeuds fictifs sont rajoutés au graphe.

La figure 1.1 représente un problème d'ordonnancement composé de trois ressources notées A, B et C et 8 tâches numérotées de 1 à 8. Les tâches 1, 5 et 6 doivent se réaliser sur la même ressource, pareil pour les tâches 2, 4 et 7 et 3 et 8. Le début et la fin de l'ordonnancement sont représentés respectivement par les noeuds fictifs "s" et "p". Le fait de fixer un sens à chaque arc disjonctif réalise un ordonnancement [RS64].

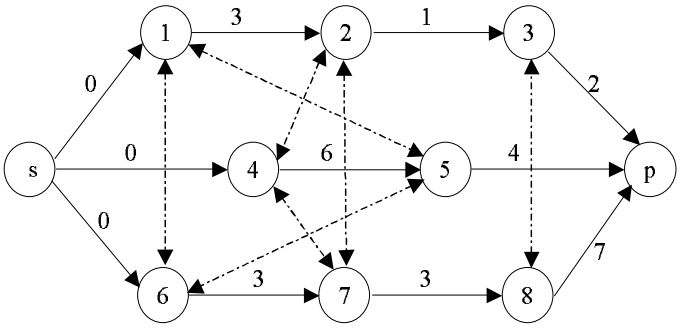



FIG. 1.1 – *Exemple de graphe de précédence*

#### 1.2.5.2 Représentation des ordonnancements

Le diagramme de Gantt est certainement le type de représentation le plus ancien, le plus répandu et le plus simple pour visualiser graphiquement l'exécution des tâches et/ou l'occupation des ressources au cours du temps. Le diagramme de Gantt consiste à placer les tâches en ordonnées sur un tableau où le temps est en abscisse et les ressources en ordonnées. A chaque tâche est associé un segment, ou barre, horizontal de longueur proportionnelle à la durée de traitement.

La figure 1.2 visualise le diagramme de Gantt d'un ordonnancement réalisable du problème représenté par le graphe de précédence de la figure 1.1.

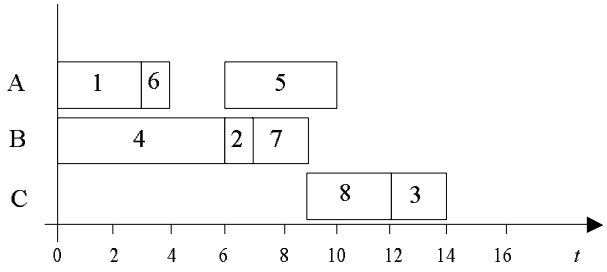



FIG. 1.2 – *Diagramme de Gantt*

### 1.2.6 Classes d'ordonnancement

Avant de développer les méthodes de résolution, nous définissons les différentes classes de l'ordonnancement. L'espace des ordonnancements peut être partagé en sous classes en fonction de l'affectation des tâches dans le temps. Ces classes d'ordonnancements présentent des propriétés de dominance vis à vis de tout critère régulier.

**Définition 1.6.** *Un critère est dit* ***régulier*** *si sa performance ne se dégrade pas en avançant l'exécution d'une tâche.*

Afin de mieux comprendre les différentes caractéristiques d'un ordonnancement, nous allons utiliser l'exemple d'un atelier de production composé d'une machine cumulative renouvelable à capacité finie $\Omega_0 = 4$. Le problème consiste à trouver un ordonnancement admissible qui permet l'affectation de six tâches sur la machine et qui optimise le $C_{max}$. Les tâches sont liées par des contraintes de précédence représentées par la figure 1.3. Chaque tâche $j$ est caractérisée par sa durée d'exécution $p_j$ et par son volume d'utilisation de la machine notée $\Omega_j$. Le tableau 1.2 résume les données des tâches.

| Tâches $j$ | $\Omega_j$ | $p_j$ |
|---|---|---|
| $a$ | 4 | 1 |
| $b$ | 3 | 1 |
| $c$ | 2 | 2 |
| $d$ | 4 | 1 |
| $e$ | 3 | 2 |
| $f$ | 5 | 2 |

TAB. 1.2 – *Données des tâches*

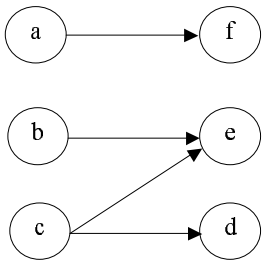



FIG. 1.3 – *Graphe de précédence*

#### 1.2.6.1 Ordonnancement admissible

Un ordonnancement est *admissible* si l'ordre d'affectation de ces tâches respecte les contraintes du problème. Un exemple d'un ordonnancement admissible est donné par la figure 1.4. Cet ordonnancement respecte les contraintes de précédence entre les différentes tâches ainsi que la capacité maximale de la machine à tout instant $t$. La valeur du critère d'optimisation de cet ordonnancement est égale à 11.

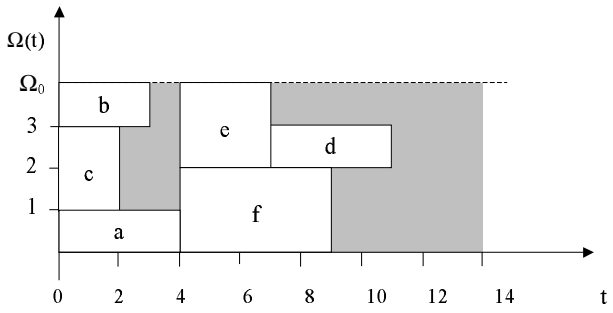



FIG. 1.4 – *Ordonnancement admissible*

#### 1.2.6.2 Ordonnancement semi actif

Il est possible d'améliorer les performances d'un ordonnancement admissible pour un critère régulier tel que le makespan. Il s'agit de réaliser des décalages à gauche de certaines tâches pour combler des vides à condition de ne pas violer les contraintes imposées par le problème (telles que les dates de début au plus tôt). Si aucun décalage à gauche n'est possible, c'est-à-dire qu'on ne peut plus avancer une tâche sans modifier la séquence initiale, alors l'ordonnancement obtenu est *semi actif*. Pour l'ordonnancement de la figure 1.4, un décalage à gauche d'une unité des tâches $e$ et $d$ permet d'avoir un ordonnancement semi actif avec un $C_{max}$ amélioré d'une unité (Voir figure 1.5).

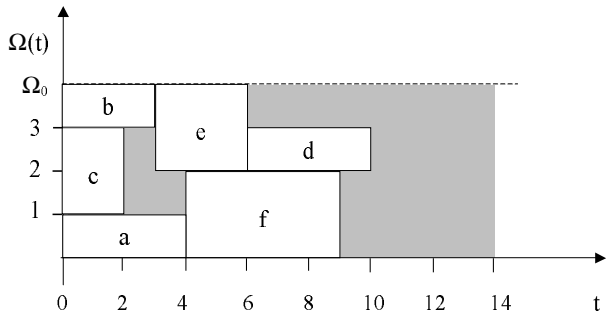



FIG. 1.5 – *Ordonnancement semi actif*

#### 1.2.6.3 Ordonnancement actif

Un ordonnancement est *actif* si aucun décalage ou permutation de ces tâches n'est possible [Pin95]. Aucune tâche ne peut commencer plus tôt sans reporter le début d'une autre. Par exemple, Une permutation des tâches $e$ et $d$ dans l'ordonnancement semi actif de la figure 1.5 permet d'avoir un ordonnancement actif (Voir figure 1.6) optimisant le critère étudié ($C^*_{max} = 9$).

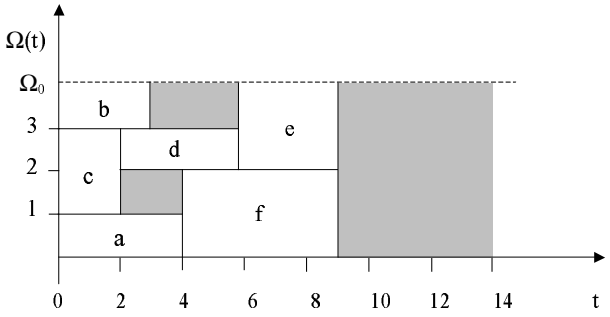



FIG. 1.6 – *Ordonnancement actif*

**Propriété 1.1.** *Un ordonnancement actif est semi actif. Cependant l'inverse n'est pas toujours vrai.*

#### 1.2.6.4 Ordonnancement sans retard

Pour obtenir un ordonnancement sans retard, il suffit d'affecter les tâches vérifiant les contraintes du problème sur les machines disponibles sans retard. Ceci veut dire que dès qu'une ressource est libre et qu'il existe une tâche en attente, alors il faut l'affecter sans aucun retard. Un ordonnancement sans retard n'est pas forcément optimal. La figure 1.7 montre un ordonnancement sans retard d'une valeur de $C_{max} = 11$.

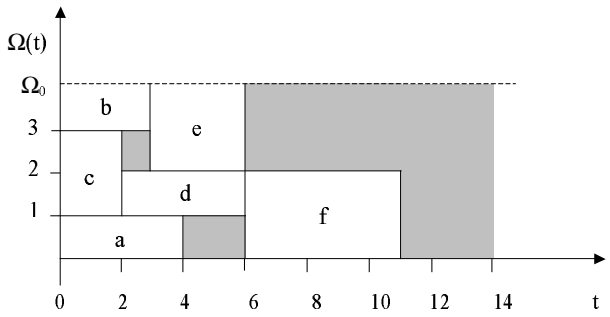



FIG. 1.7 – *Ordonnancement sans retard*

#### 1.2.6.5 Caractérisation des solutions

Souvent la résolution des problèmes d'ordonnancement est contrainte à l'explosion combinatoire due au nombre grandissant de variables et de contraintes. Pour limiter l'espace de recherche, il est important de caractériser des ensembles de solutions qui le constituent. On appelle ainsi un ensemble de solutions par *dominant* pour l'optimisation

d'un critère régulier, s'il contient au moins un optimum pour ce critère. La recherche d'une solution optimale ou proche de l'optimale peut ainsi se limiter au plus petit ensemble dominant. Il est important de noter que l'ensemble des ordonnancements semi actifs est dominant pour tout critère régulier de même pour l'ensemble des ordonnancements actifs [Pin95]. Puisque tout ordonnancement actif est semi actif, donc l'ensemble des ordonnancements actifs s'avère être le plus petit ensemble dominant. L'ensemble des ordonnancements sans retard est un sous ensemble des ordonnancements actifs, mais ne constituent pas un ensemble dominant vis à vis d'un critère régulier [EL01]. Néanmoins, il est pratique de générer des ordonnancements sans retard qui constituent généralement des bonnes solutions. La figure 1.8 présente une classification des différents types d'ordonnancements pour un critère régulier.

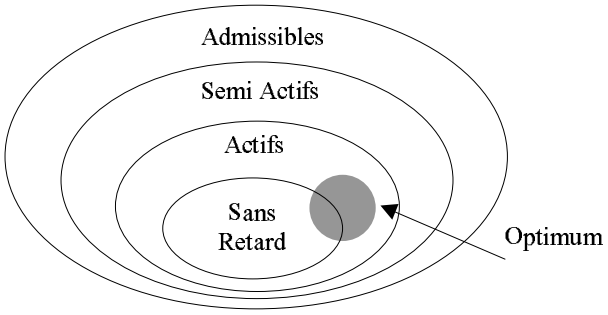



FIG. 1.8 – *Classification des ordonnancements pour un critère régulier*

## 1.3 Les méthodes de résolution

Les méthodes de résolution sont aussi variées que les problèmes d'ordonnancement. Dans la littérature ([LR01], [Kac03], [Cav00], ...), on distingue essentiellement deux classes de méthodes : les méthodes exactes assurant la résolution des problèmes en un temps polynômial (si la taille des problèmes en question ne dépasse pas une certaine limite), et les méthodes approchées, appelées aussi heuristiques permettant de trouver une solution proche de l'optimal en un temps tolérable.

### 1.3.1 Les méthodes exactes

Elles sont généralement utilisées pour résoudre des problèmes de petite taille. Dans ce cas, le nombre de combinaisons possibles est suffisamment faible pour pouvoir explorer l'espace de solutions dans un temps raisonnable. Kacem [Kac03] a distingué trois sous

classes de méthodes exactes : la procédure de séparation et d'évaluation (Connue dans la littérature anglophone sous le nom branch and bound), la programmation dynamique et la programmation linéaire.

#### 1.3.1.1 Procédure de séparation et d'évaluation

La procédure de séparation et d'évaluation notée PSE, est une méthode d'exploration par énumération implicite de l'espace de recherche. Elle permet la construction d'une arborescence dont les sommets représentent chacun un sous-problème et les arcs issus d'un même sommet représentent chacun une décomposition possible du problème situé au sommet de l'arbre en sous-problèmes de taille réduite. Cette arborescence est explorée de façon à éviter les branches ne contenant pas des solutions réalisables et les branches n'amenant pas à des solutions meilleures que la solution courante. D'après [LR01], les PSE reposent sur quatre composantes essentielles :

- la technique de *séparation*, qui permet de décomposer un problème en le partitionnant en sous-problèmes de taille réduite ;
- la méthode d'évaluation qui associe une borne au critère d'optimisation sur l'ensemble des solutions d'un sous-problème (borne supérieure dans le cas de minimisation. Cette borne permet d'éviter l'exploration d'un sommet dont la solution partielle correspondante, à une valeur du critère dépassant la borne) ;
- la méthode de sondage, qui permet de déterminer si un sommet est terminal (il ne contient pas de solution admissible, ou c'est une solution optimale, ou l'on peut obtenir polynomialement la solution optimale de ce sous-problème), ou s'il mérite d'être séparé ;
- la méthode de sélection, ou stratégie d'exploration, qui décrit comment choisir le sous-problème à séparer, lorsque plusieurs sont candidats. On peut distinguer deux stratégies d'exploration, celle qui favorise les meilleurs d'abord appelée aussi procédure par séparation et évaluation progressive, et celle de type profondeur d'abord et retour arrière (procédure par séparation et évaluation séquentielle).

#### 1.3.1.2 Programmation dynamique

La programmation dynamique est une méthode d'optimisation opérant par phases (ou encore par séquences). Son efficacité repose sur le principe d'optimalité de Bellman à savoir «toute politique optimale est composée de sous-politiques optimales» [BD74]. Ce principe permet une résolution ascendante, qui détermine une solution optimale d'un

problème à partir des solutions de tous les sous-problèmes. Chaque étape correspond à un sous-problème à résoudre optimalement en tenant compte des informations obtenues des étapes précédentes. Ceci nécessite une formulation du critère sous forme d'une relation de récurrence liant deux niveaux successifs.

Cette méthode est destinée à résoudre des problèmes d'optimisation à vocation plus générale que la méthode de séparation et évaluation. Par contre, la taille des problèmes qu'elle permet d'aborder est plus limitée.

#### 1.3.1.3 Programmation linéaire

La programmation linéaire est l'une des techniques classiques de la recherche opérationnelle. Elle permet la modélisation d'un problème d'optimisation d'une fonction objectif $Z$ de plusieurs variables en présence de contraintes sous la forme d'un programme mathématique [GPS00]. Le programme est dit *linéaire* si la fonction et les contraintes sont toutes des combinaisons linéaires de variables. Il comporte $n$ variables non négatives (1.4), $m$ contraintes d'égalité ou d'inégalité (1.3) et la fonction objectif à optimiser (1.2). Le coefficient de coût ou de profit de la variable $x_j$ est noté $c_j$, celui de la variable $x_j$ dans la contrainte $i$ est noté $a_{ij}$. La contrainte $i$ a un second membre constant $b_i$. Les contraintes simples de positivité ne sont pas incluses dans les $m$ contraintes, car elles sont gérées à part par les algorithmes.

$$\max \ ou \ \min Z = \sum_{j=1}^{n} c_j x_j \tag{1.2}$$

$$\forall \ i = 1...m : \sum_{j=1}^{n} a_{ij} x_j \leq \ ou \ \geq b_i \tag{1.3}$$

$$\forall \ j = 1...n : x_j \geq 0 \tag{1.4}$$

Si les variables sont astreintes à être entières, on a un *programme linéaire en nombres entiers* (PLNE). Un programme linéaire en 0-1 est un cas particulier de PLNE dont les variables ne peuvent prendre que deux valeurs 0 ou 1 ; ces variables sont dites *booléennes*, *binaires* ou de *décision*. Enfin, à partir du moment où au moins une contrainte ou la fonction objectif n'est pas une combinaison linéaire de variables, on parle alors d'un programme non linéaire PNL. Les PLNE et PL en 0-1 sont plus difficiles à résoudre que les PL classiques. Les PNL sont encore plus difficiles.

### 1.3.2 Les méthodes de résolution approchée

Malgré l'évolution permanente des calculateurs et les progrès fulgurants de l'informatique, il existe pour plusieurs problèmes d'optimisation combinatoire une taille critique de l'espace de solutions admissibles. La méthode permettant d'obtenir une solution optimale est bien évidement celle de l'énumération complète de l'espace de recherche. Cette dernière est dans la plupart des cas prohibitive. Compte tenu de ces difficultés, la plupart des spécialistes de l'optimisation combinatoire ont orienté leur recherche vers le développement des méthodes heuristiques. Une méthode heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible [LR01]. Bien que l'obtention d'une solution optimale ne soit pas garantie, l'utilisation d'une méthode heuristique offre de multiples avantages par rapport à une méthode exacte :

- la recherche d'une solution optimale peut être totalement impossible dans certaines applications pratiques en raison de la dynamique caractérisant l'environnement de travail, du nombre de variables et de contraintes étudiées, de l'antagonisme entre les objectifs à atteindre et parfois même de l'imprécision des données récoltées ;
- l'exécution des méthodes heuristiques est souvent rapide, en effet, ces dernières fournissent en un temps polynômial une ou plusieurs solutions admissibles de bonne qualité ;
- l'application des méthodes heuristiques est à la portée des utilisateurs non expérimentés. En effet, une méthode heuristique se base sur des règles et des principes simples et "intelligents" ce qui leurs permet d'être compréhensibles ;
- l'adaptation ou la combinaison d'une méthode heuristique avec d'autres méthodes est souvent facile. Cette flexibilité permet d'augmenter la performance de la méthode hybride résultante et de garantir parfois l'obtention des bonnes solutions.

Avant de présenter une typologie des métaheuristiques développées pour résoudre un problème d'ordonnancement, nous définissons les termes heuristique et métaheuristique. Une heuristique est une méthode de calcul pour un problème générique d'optimisation produisant une solution non nécessairement optimale. Par contre, une métaheuristique est un ensemble de concepts applicables à un large ensemble de problèmes d'optimisation combinatoire pour créer de nouvelles heuristiques.

Une classification des différentes types de métaheuristiques est présentée par [WHC01]. Les auteurs considèrent qu'il existe trois types de métaheuristiques : les métaheuristiques constructives basées sur l'idée de la diminution progressive de la taille du problème, les

métaheuristiques basées sur la recherche locale et les métaheuristiques évolutives inspirées des phénomènes réels, telles que les algorithmes génétiques et les algorithmes de fourmi etc. Afin de pouvoir expliquer en détail le principe de chaque type d'approche, nous allons utiliser les notations suivantes: Soit $X$ l'ensemble des solutions admissibles du problème traité (nous supposons que cet ensemble est fini). Soit $f$ une fonction d'évaluation des solutions admissibles. Résoudre un problème d'optimisation combinatoire d'une manière optimale en respectant les contraintes qui lui sont associées (dans le cas de minimisation), consiste à déterminer une solution $s^* \in X$ vérifiant :

$$f(s^*) = \min_{s \in X} f(s) \tag{1.5}$$

#### 1.3.2.1 L'approche constructive

Les méthodes constructives fournissent des solutions réalisables de la forme $s = (x\{1\},x\{2\},...,x\{n\})$ en partant d'une solution initiale vide $s[0]$ et en insérant, à chaque étape $k(k = 1,...,n)$, une composante $x_{o(k)}$ $(o(k) \in \{1,2,...,n\} \setminus \{o(1),o(2),...,o(k-1)\})$ dans la solution partielle $s[k-1]$. Notons que la construction d'une solution réalisable est statique, par conséquent, les prises des décisions intermédiaires ne sont jamais remises en cause. Ce type de représentation vectorielle convient très bien pour les problèmes d'affectation, en effet, les positions du vecteur $s$ correspondent aux objets alors que les composantes $x_i$, $1 \leq i \leq n$ définissent les ressources affectées aux objets $i$.

L'idée principale derrière une approche constructive est de diminuer progressivement la taille de l'espace de recherche en fixant à chaque étape la valeur d'une variable du problème (Figure 1.9). Mathématiquement parlant, ceci revient à trouver un sous-ensemble $X^k \subseteq X$ toujours de taille plus petite. Une approche constructive fournit une ou plusieurs solutions optimales lorsque chaque sous ensemble $X^k$ contient au moins une solution optimale $s^* \in X$. Ceci est généralement difficile à réaliser pourvu que la construction d'une solution est statique et ne permet pas de mettre en question les choix fixés antérieurement. La majorité des méthodes constructives sont de type glouton et sont souvent considérées comme myopes. A chaque étape, la solution courante est complétée de la meilleure façon sans tenir compte des conséquences que cela entraîne au niveau de la qualité de la solution finale.

Les méthodes constructives se distinguent par leur rapidité et leur grande simplicité. On obtient en effet très rapidement une solution admissible sans avoir recours à des connaissances approfondies dans le domaine de l'optimisation combinatoire. Le principal défaut de ces méthodes réside dans la qualité des solutions fournies. En effet, le fait de

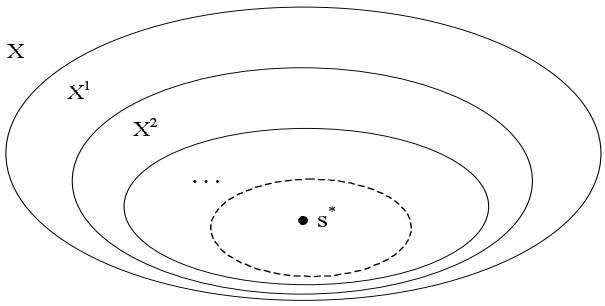



FIG. 1.9 – *Exploration de l'espace d'états par une approche constructive*

vouloir opérer rapidement et sans tenir compte du contexte global du problème a souvent des conséquences négatives sur le coût de la solution obtenue. Il est donc préférable de mettre au point des procédures anticipant les effets secondaires occasionnés par les décisions prises lors de la construction d'une solution admissible.

#### 1.3.2.2 L'approche de recherche locale

Les méthodes de recherche locale sont des algorithmes itératifs qui explorent l'espace d'états $X$ en partant d'une solution admissible $s_0$ choisie arbitrairement ou à l'aide d'une heuristique. Le principe de la recherche locale consiste à modifier légèrement à chaque itération $k$, et à l'aide d'un opérateur la solution $s_k$. Ce procédé prend fin lorsqu'une condition d'arrêt est satisfaite. Les conditions d'arrêt peuvent être un seuil d'itérations ou l'évaluation de la qualité de la solution courante. Dans ce dernier cas, si l'application de l'opérateur à la solution courante ne permet pas de l'améliorer, l'algorithme s'arrête. Nous allons citer par la suite trois approches de recherche locale.

- **la méthode de descente** : la méthode de descente décrite par l'algorithme 1.1, est l'une des approches de recherche locale. Cette méthode explore l'espace $X$ en choisissant à chaque fois la meilleure solution voisine de la solution courante. Ce procédé continue aussi longtemps que la valeur de la fonction objectif diminue. La recherche s'interrompt dès lors qu'un minimum local de $f$ est atteint. Historiquement, les méthodes de descente ont connu un grand succès pour le traitement des problèmes d'optimisation combinatoire. Toutefois, elles comportent deux obstacles majeurs qui limitent considérablement leur efficacité :
  - suivant la taille et la structure du voisinage d'une solution $s \in X$, la recherche de la meilleure solution voisine est un problème qui peut être aussi difficile que le problème initial ;

– une méthode de descente peut fournir un minimum local. En effet, le processus de recherche s'arrête dès qu'un minimum local est atteint. Or les problèmes d'optimisation combinatoire comportent typiquement de nombreux *optima* locaux pour lesquels la fonction objectif peut être fort éloignée de la valeur optimale. La figure 1.10 montre l'inconvénient de la méthode de descente se basant sur un choix aléatoire d'un voisin meilleur et s'arrête s'il n'existe plus.

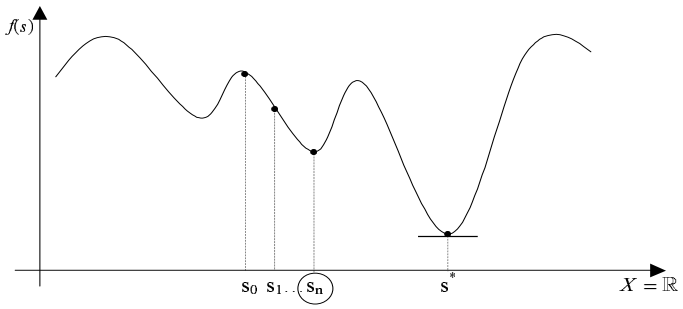



FIG. 1.10 – *Blocage dans un optimum local de f*

**Algorithme 1.1** La méthode de descente

**Entrée :** $s \in X$ *{une solution initiale admissible choisie aléatoirement ou à l'aide d'une heuristique}*

**Début**

$s^* \leftarrow s$

**Tant que** le critère d'arrêt n'est pas satisfait **faire**

générer un voisinage de $s : N(s)$

déterminer la meilleure solution $s' \in N(s)$

$s \leftarrow s'$

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

$s^* \leftarrow s$

**Fin si**

**Fin tant que**

**Fin**

Pour faire face à ces carences, d'autres méthodes de recherche locale plus sophistiquées ont été développées au cours de la dernière décennie. Ces méthodes analysent des solutions voisines moins bonnes afin d'éviter les optimums locaux.

Les méthodes les plus connues seront introduites aux paragraphes suivants. Il y a deux différences majeures entre ces méthodes.

- la manière d'explorer les solutions voisines d'une solution courante ;
- le critère d'arrêt de la recherche (qui est souvent difficile à choisir puisque la solution optimale est rarement connue.)

– **le recuit simulé** : le recuit simulé est une méthode de recherche locale dont les origines remontent aux expériences de Metropolis et al. en 1953 [MRR$^+$53]. Leurs travaux consistaient à étudier la stabilité thermique d'un système physique. Kirkplatrick [KJV83] et al. ont été les premiers à s'inspirer d'une telle technique pour l'appliquer à des problèmes d'optimisation combinatoire. Le recuit simulé démarre par une solution initiale admissible et continue l'exploration de l'espace d'états en effectuant des perturbations mineures à la solution courante. Si la solution voisine obtenue améliore le critère cherché alors elle est retenue. Si, au contraire, elle provoque une détérioration $\Delta E$ du critère, elle est retenue avec une probabilité $P = \exp(-\Delta E/T)$, où $T$ est un paramètre inversement proportionnel au nombre d'itérations.

  Les avantages du recuit simulé sont nombreux. Tout d'abord, cette approche est en mesure de fournir des résultats satisfaisants dès que l'on sait trouver une solution initiale admissible et une manière d'explorer des solutions voisines. En plus, cette méthode permet de traiter les problèmes d'optimisation combinatoire pour lesquels il n'existe aucune méthode de résolution. Le fait de garder des solutions moins bonnes permet de couvrir un espace de recherche plus grand et d'éviter une convergence prématurée vers un optimum local. Le recuit simulé a été abondamment utilisé pour résoudre des problèmes pratiques d'optimisation tel que le job shop.

– **la méthode Tabou** : cette méthode a été créée dans les années 1970 et a été présentée pour la première fois par Glover en 1986 [Glo86]. La formulation finale de cette technique est apparue en deux parties [Glo89], et [Glo90]. Cette méthode est basée sur deux principes. Le premier consiste à améliorer à chaque itération la valeur de la fonction objectif en choisissant à chaque fois la meilleure solution voisine si elle existe. Si aucune amélioration n'existe dans le voisinage, le choix se fait sur le moins mauvais des voisins. Ce principe utilisé seul, présente un inconvénient majeur. Si un optimum local se trouve dans un pic très grand (éventuellement ou au fond d'une vallée profonde s'il s'agit d'un problème de minimisation), il sera impossible de l'atteindre (ou d'en ressortir). C'est pour cette raison que la méthode

Tabou s'appuie sur un deuxième principe qui consiste à garder en mémoire les dernières solutions visitées et à interdire le retour vers celles-ci pendant un nombre d'itérations fixé.

La méthode Tabou est une métaheuristique souvent utilisée pour résoudre des problèmes industriels car elle présente une amélioration importante des algorithmes de plus forte pente. On parle également de recherche guidée. Plusieurs problèmes de type flow shop ou job shop sont traités par cette méthode (voir section état de l'art à la fin de ce chapitre).

#### 1.3.2.3 L'approche évolutive

De tout temps, les sciences de la vie et les processus naturels ont constitué des bases d'inspiration et d'imitation pour les chercheurs et les ingénieurs. Il y a bien longtemps, pour citer quelques exemples, les boutons à pressions issus des sèches, les fermetures éclair etc. Des disciplines entières se sont développées telles que la bio-informatique, l'éthologie. Plus proche de l'ordonnancement, des techniques liées aux fourmis ont été mises au point tel que les ants systems. Les mécanismes du monde vivant sont à l'origine des systèmes artificiels utilisables dans des contextes variés. Les méthodes évolutives qui sont présentées dans cette section constituent la base d'un nouveau champ de la programmation informatique en pleine effervescence.

Contrairement aux méthodes constructives et de recherche locale qui font intervenir une solution unique, les méthodes évolutives traitent un ensemble de solutions admissibles (appelé aussi population d'individus). L'idée principale consiste à appliquer des opérateurs spécifiques à certaines solutions distinguables pour faire évoluer la population vers un niveau acceptable de performance moyenne. En général, la taille $n$ de la population reste constante tout au long du processus cyclique d'exploration de l'espace de solutions $X$. Après avoir généré une population initiale, généralement d'une manière aléatoire, une méthode évolutive tente d'améliorer la qualité moyenne de la population courante en ayant recours à des principes d'évolution naturelle. Le processus cyclique qui est à la base d'une méthode évolutive est composé d'une phase de coopération et d'une phase d'adaptation individuelle qui se succèdent à tour de rôle.

Lors de la phase de coopération, les solutions de la population courante sont comparées puis combinées entre elles dans le but de produire des solutions inédites et de bonne qualité. L'échange d'information qui en résulte, se traduit par l'apparition de nouvelles solutions admissibles qui héritent des caractéristiques prédominantes contenues dans les solutions de la population courante. Dans la phase d'adaptation individuelle, les solu-

tions reçoivent sans aucune interaction les unes des autres des modifications mineures en restant évidemment admissibles. Une nouvelle génération de solutions est créée au terme de chaque phase d'adaptation individuelle.

Les algorithmes génétiques constituent au jour actuel, l'approche la plus utilisée parmi les méthodes évolutives. A l'inverse des méthodes exactes, telles que la programmation dynamique et la procédure de séparation et d'évaluation, et des méthodes approchées telle que la recherche locale, les algorithmes génétiques permettent de trouver une bonne solution, voire la meilleure solution en un temps de calcul très réduit. Leur aspect se basant sur l'aléatoire et l'évolution à l'aide des opérateurs spécifiques, leur permet de visiter toutes les zones de l'espace de recherche. Les algorithmes génétiques fournissent en général toute une population de bonnes solutions. Cette caractéristique est très avantageuse dans plusieurs situations où l'on aura besoin de choisir entre plusieurs alternatives, ou si on veut changer de solution. De plus, les algorithmes génétiques sont très bien adaptés à l'optimisation multiobjectif. Le choix d'une telle technique, nous aidera alors à considérer plusieurs objectifs et par conséquent se rapprocher de la réalité des problèmes industriels ou de la vie pratique, où l'on cherche à optimiser plusieurs critères simultanément.

## 1.4 Ordonnancement dans un atelier job shop

Le *job shop* est associé à des lignes de production dédiées à la production de moyenne et de petite série où les changements de produit sont fréquents (généralement, les produits à fabriquer présentent des caractéristiques différentes). Contrairement au flow shop, le job shop est considéré comme étant un atelier à cheminement multiple. Chaque travail passe donc sur les machines selon une gamme fixée. En cas de job shop, cette gamme peut être différente pour chaque travail. On parle ainsi d'un atelier à flot multidirectionnel. D'autre part, si une opération a la possibilité d'être exécutée sur plus qu'une machine, il s'agit alors d'un job shop à «machines dupliquées».

### 1.4.1 Formalisation du job shop

Nous présentons dans ce paragraphe un formalisme mathématique du job shop définit par Vaessens en 1994 [VL94]. Un problème $P$ de job shop de taille $n \times m$ est constitué d'un ensemble $O$ de $l$ opérations, d'un ensemble $M$ de $m$ machines et d'un ensemble $J$ de $n$ jobs. La $j^{me}$ opération du $i^{me}$ job notée $o_{ij}$ lui correspond un temps opératoire $p_{o_{ij}} \in \mathbf{N}$ et une unique machine $M_{o_{ij}} \in M$ sur laquelle elle s'effectue. Chaque job $J_i$ est constitué d'une séquence $O_i = (O_{i1},O_{i2},...,O_{ik_i})$ de $k_i$ opérations. L'ordre de séquencement exprime

en réalité les contraintes technologiques. Les jobs peuvent être associés à des dates de début au plus tôt $r_i$ avant lesquelles il n'est possible d'effectuer aucune exécution.

Les machines demandent généralement une certaine durée de préparation et ne peuvent être disponibles qu'à partir d'une date $u_i$. Les différents paramètres de temps: $p$, $r$ et $u$ sont généralement discrets (des entiers).

Une relation binaire $A$ est définie dans l'ensemble $O$ représentant les relations de précédence entre les opérations. Si $(v,w) \in A$ alors $v$ doit s'exécuter avant $w$. $A$ induit un ordre total entre les opérations d'un même job. Aucune contrainte de précédence n'existe entre les opérations des différents jobs. De plus, si $(v,w) \in A$, et il n'existe aucune opération $u \in O$ telle que $(v,u) \in A$ et $(u,w) \in A$ alors $M(v) \neq M(w)$. Un ordonnancement $S$ est défini comme suit :

$$\begin{aligned} S : O &\longrightarrow \mathbf{N} \\ v &\longmapsto S(v) \end{aligned}$$

L'application $S$ détermine les dates de début d'exécution des opérations de l'ensemble $O$. Un ordonnancement $S$ est admissible si les relations suivantes sont vérifiées :

$$\forall\ v \in O : S(v) \geq 0 \tag{1.6}$$

$$\forall\ v,\ w \in O,\ (v,\ w) \in A\ :\ S(w) \geq S(v) + p(v) \tag{1.7}$$

$$\forall\ v,\ w \in O,\ v \neq w,\ M(v) = M(w) : S(w) \geq S(v) + p(v)\ ou\ S(v) \geq S(w) + p(w) \tag{1.8}$$

Le job shop est fortement lié aux deux autres types d'atelier de production, à savoir le flow shop et le open shop. Le flow shop est un cas particulier du job shop, dans lequel les ordres de passage des séquences $O_i$ sur les machines sont les mêmes pour tous les jobs. Cette condition n'implique pas forcément que les jobs soient identiques puisque les temps opératoires des jobs peuvent être différents. Le problème d'ordonnancement de type open shop est équivalent au job shop avec l'exception de l'absence des contraintes technologiques. Par conséquent, les opérations d'un même job $j_i$ sont à exécuter dans n'importe quel ordre.

### 1.4.2 Contraintes et objectifs

Le problème du job shop présente des hypothèses à respecter et des conditions particulières sur l'environnement d'exécution concernant à la fois les ressources et les contraintes de temps. Nous allons par la suite citer les hypothèses liées au job shop.

- les machines ne sont pas identiques : chaque machine est unique. Lors de l'affectation, aucune question de type "sur quelle machine une telle opération doit s'effectuer" ne peut se poser ;
- les opérations d'un même job s'exécutent séparément sur les machines en respectant l'ordre des séquences $O_i$ ;
- la préemption des opérations n'est pas permise : une fois commencée, l'exécution d'une opération ne peut pas être interrompue ;
- toutes les données du job shop sont connues d'avance : il s'agit alors du job shop statique (en particulier, il ne doit pas y avoir des nouveaux jobs lors de l'exécution des anciens) ;
- les machines ont une disponibilité infinie. Elles sont supposées fiables et ne subissent aucune panne pouvant perturber ou arrêter l'exécution des opérations. Contrainte que nous relaxons dans la deuxième partie de notre travail, en tenant compte des opérations de maintenance préventives systématiques ;
- les problèmes de job shop classiques, appelés aussi carrés, traitent généralement des jobs ayant un nombre d'opérations égal au nombre de machines. Ainsi, toutes les opérations d'un même job s'exécutent sur des machines différentes.

Les critères d'optimisation cités dans le tableau 1.1 à l'exception du nombre de tâches en retard sont utilisés pour mesurer la performance d'un ordonnancement du job shop. Tous ces critères sont à minimiser. Le critère $E = \sum e_j$ exprimant la somme des avances des tâches exécutées avant les dates au plus tôt $r_i$, est souvent utilisé conjointement avec d'autres critères d'optimisation (tel que la somme des retards $T$) pour optimiser une fonction multicritère de la forme $\alpha T + \beta E$. Cette dernière fonction est utile pour minimiser les coûts des inventaires de certains problèmes de job shop.

Nous étudions un problème de job shop carré qui satisfait toutes les contraintes citées auparavant à l'exception de la prise en compte de la maintenance qui sera traité conjointement avec la production dans le quatrième chapitre. Le critère d'optimisation étudié est la longueur de l'ordonnancement appelé aussi date d'achèvement.

Ce critère noté $C_{max}$, est donné par l'equation 1.9. Selon les notations définies dans la section 1.1.2, le problème de job shop étudié peut s'écrire sous la forme suivante : $J//C_{max}$.

$$C_{max} = \max_{v \in O} (S(v) + p(v)) \tag{1.9}$$

Même si le critère de $C_{max}$ n'est pas considéré comme étant une bonne fonction objectif théorique, il est largement utilisé en industrie comme dans les études académiques. Ce critère est le premier à être utilisé par les chercheurs pour résoudre des problèmes ayant une explosion combinatoire de l'espace de recherche. D'un point de vue mathématique, le traitement et la formulation de ce critère sont faciles. Par conséquent, il a été le critère le plus utilisé par les académiciens, d'autant qu'il permet de détecter la difficulté combinatoire sous-jacente dans la détermination d'un ordonnancement optimal. La nature générique et la flexibilité de ce critère est démontrée par Demirkol et al [DMU97]. Ainsi, une solution obtenue en considérant le critère $C_{max}$, engendre en moyenne des bonnes performances pour les critères $\sum C_i$, $\sum T_i$ et $max$. Un problème d'ordonnancement ayant pour objectif la minimisation du makespan, est analogue au problème type du voyageur de commerce. Cependant, résoudre le job shop classique ou le voyageur de commerce facilite la résolution de plusieurs autres problèmes n'ayant pas de critère de performance régulier.

### 1.4.3 Complexité du job shop statique

La plupart des variantes du job shop à l'exception de quelques formulations limitées à une ou deux machines sont connues pour être *$\mathcal{NP}$-difficiles*. Un algorithme polynomial se basant sur l'algorithme de Johnson[3] a été publié par Jackson [Jac56] pour résoudre le problème $J_2//C_{max}$. Le principe d'affectation des opérations de ce type particulier de problème est détaillé dans l'algorithme 1.2.

Les problèmes de job shop ayant un nombre $m > 2$ de machines, et optimisant les critères $C_{max}$ et $F$ sont *$\mathcal{NP}$-difficiles* au sens fort [Jen01]. Théoriquement, le problème du job shop $J//C_{max}$ peut se réduire polynomialement aux problèmes $J//T_{max}$ et $J//L_{max}$. De même, le problème $J//F$ peut se réduire aux problèmes $J//T$ et $J//L$. Par

3. L'algorithme de Johnson résoutdsa d'une manière optimale le problème $F_2//C_{max}$ par un ordonnancement de permutation (même ordre sur $M_1$ et $M_2$). Il s'agit de déterminer en premier lieu l'ensemble $S = J_j \ / \ p_{1j} \leq p_{2j}$ où $p_{ij}$ représente le temps d'exécution de l'opération du $j^{me}$ job sur la machine $M_i$. Ensuite, ordonnancer les opérations de $S$ en ordre croissant des $p_{1j}$ et les opérations de $\bar{S}$ en ordre décroissant des $p_{2j}$.

conséquent, ces quatre dernières instances du job shop sont $\mathcal{NP}$-*difficiles* au sens fort [4]. L'expérience a montré que ces problèmes de job shop avec un nombre $m \geq 2$ de machines sont difficiles à résoudre même en utilisant des heuristiques. En effet, une instance de job shop de taille 10x10 proposée par Muth et Thomson en 1963 n'est optimalement résolue qu'en 1989 par Carlier et Pinson [CP89].

**Algorithme 1.2** L'algorithme de Jackson : job shop à deux machines

**Entrée :** $J_2//C_{max}$

**Début**

classer les jobs en 4 catégories :

$J_1$ : les jobs ayant une seule opération sur $M_1$ ;

$J_2$ : les jobs ayant une seule opération sur $M_2$ ;

$J_{12}$ : les jobs ayant une opération sur $M_1$ puis une opération sur $M_2$ ;

$J_{21}$ : les jobs ayant une opération sur $M_2$ puis une opération sur $M_1$.

***Étape 1 :*** appliquer séparément l'algorithme de Johnson aux opérations de $J_{12}$ et de $J_{21}$. L'ordre d'affectation des opérations de $J_1$ et de $J_2$ est arbitraire ;

***Étape 2 :*** affecter les opérations sur $M_1$ en ordre $J_{12}$,$J_1$,$J_{21}$ et sur $M_2$ en ordre $J_{21}$,$J_2$,$J_{12}$.

**Fin**

### 1.4.4 Représentation graphique du job shop

Le job shop peut être décrit par un graphe de précédence $G = (V, C, D)$ où $V$ représente l'ensemble des opérations des jobs (les noeuds sur le graphe), avec deux opérations spéciales (une source $s$ et un puits $p$ représentant respectivement le début et la fin de l'ordonnancement). $C$ est l'ensemble des arcs conjonctifs représentant les séquences technologiques entre les opérations. $D$ est l'ensemble des arcs disjonctifs regroupant les paires d'opérations devant s'exécuter sur la même machine. Chaque noeud du graphe $G$ est pondéré par le temps opératoire de l'opération considérée. Pour illustrer la représentation du job shop par les graphes de précédence, nous allons utiliser un exemple d'un atelier composé de trois machines pour fabriquer trois types différents de produits notés $J_1$, $J_2$, $J_3$. Le tableau 1.3 résume les données de cet exemple. Une représentation sous forme de graphe de précédence est donnée par la figure 1.11. Le problème de job shop peut ainsi être résolu en déterminant l'ordre d'affectation des opérations devant

4. Voir la section 1.1.5 sur la complexité.

s'exécuter sur la même machine. Ceci revient à fixer le sens d'orientation de chaque arc disjonctif sans créer des cycles.

| Les jobs | $M(v)$,$p(v)$ | | |
|---|---|---|---|
| $J_1$ | 1,3 | 2,3 | 3,3 |
| $J_2$ | 1,2 | 3,3 | 2,4 |
| $J_3$ | 2,3 | 1,2 | 3,1 |

TAB. 1.3 – *Job shop de taille 3x3*

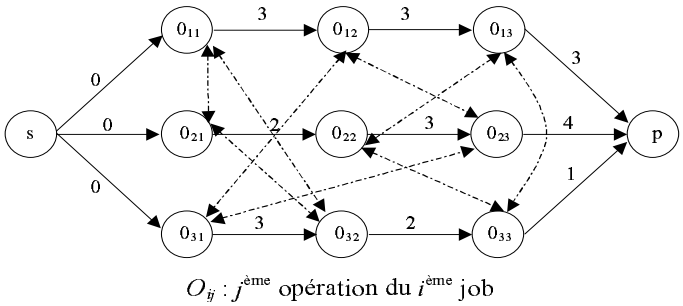



FIG. 1.11 – *Graphe de précédence du job shop 3x3*

Une fois généré, l'ordonnancement est généralement représenté par un diagramme de *Gantt*. Ce dernier est actuellement le type de représentation le plus répandu et le plus simple pour visualiser graphiquement l'exécution des tâches au cours du temps. Pour des problèmes d'atelier tel que le job shop, le diagramme de Gantt est composé de deux axes : l'axe vertical contenant les ressources et l'axe horizontal représentant l'écoulement du temps. Ce diagramme permet ainsi de visualiser les périodes d'inactivité des ressources ainsi que l'ordre de passage des opérations et la durée totale de l'ordonnancement. A titre d'exemple, nous représentons par la figure 1.12, une solution du problème de job shop de taille $3 \times 3$ cité auparavant.

### 1.4.5 État de l'art

Un état de l'art exhaustif sur les méthodes de résolution du job shop a été publié par Jain et Meeran en 1998 [JM98b], [JM98c]. Ces auteurs ont étudié l'origine, le présent et le futur d'une telle problématique ainsi que les différents algorithmes exacts et méthodes approchées de résolution publiés depuis les années 50. Nous allons classer cet état de l'art en deux catégories : les méthodes exactes et les méthodes approchées.

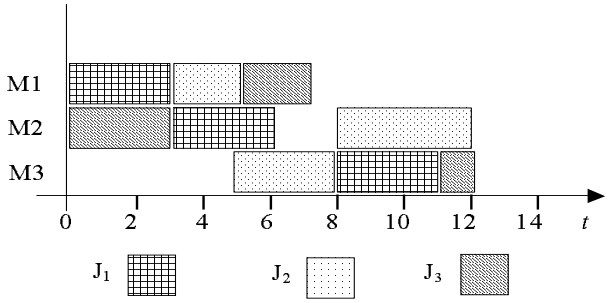



FIG. 1.12 – *Diagramme de Gantt représentant l'ordonnancement du job shop* $3 \times 3$

#### 1.4.5.1 Les méthodes exactes

Le job shop shop tel qu'on le connaît sous sa formulation classique est considéré pour la première fois pendant les années 50. Akers et Freidman [AF55] sont probablement les premiers à avoir étudié ce problème en 1955. Ces auteurs ont proposé une approche de résolution basée sur l'algèbre de Boole pour déterminer les séquences d'affectation des opérations. Ensuite, Jackson [Jac56] a adapté en 1956 au job shop, l'algorithme de Johnson [Jho54] proposé pour résoudre le flow shop. Roy et Sussmann [RS64] sont les premiers à utiliser le graphe de précédence disjonctif en 1964 qui a été ensuite utilisé en 1969 par Balas [Bal69] pour développer une approche énumérative.

Plusieurs méthodes efficaces permettant de résoudre le job shop en un temps polynomial ont été proposées. Ces méthodes traitent des instances particulières de taille réduite : $2 \times m$ [Ake56], $n \times 2$ (chaque job contient au maximum deux opérations) [Jac56] et $n \times 2$ (où tous les temps opératoires des opérations sont unitaires) [HA82].

Durant les années 60, l'intérêt des chercheurs s'est orienté vers la résolution exacte des problèmes de job shop en utilisant des algorithmes énumératifs. Ces algorithmes se basent sur des modèles mathématiques sophistiqués. La majorité de ces algorithmes est incapable de résoudre plusieurs problèmes de moyenne et de grande taille et par conséquent leur utilisation en pratique est restée limitée à la recherche des bornes inférieures. La stratégie énumérative la plus connue est celle de séparation et évaluation (Branch and Bound). Cette méthode construit un arbre dynamique représentant l'espace de tous les ordonnancements réalisables. Une grande partie de cet arbre est élaguée en utilisant des bornes supérieures. Plusieurs algorithmes de type Branch and Bound sont développés pour résoudre le job shop. A titre d'exemple, Carlier et Pinson [CP89] ont réussi grâce à cette technique exacte à résoudre le problème FT10[5] 25 ans après sa publication.

5. FT10 est un benchmark de taille $10 \times 10$ publié par Fisher et Thomson en 1963 [FT63].

D'autres instances de job shop de taille plus grande sont aussi résolues en utilisant une méthode de Branch and Bound [CP90].

Pratiquement, les problèmes résolus sont de taille limitée (le nombre d'opérations est inférieur à 250). En plus, la performance de telles techniques est très sensible au choix de la borne supérieure initiale et au type de la complexité de l'instance étudiée [LLKS93].

#### 1.4.5.2 Les méthodes approchées

La complexité des problèmes d'optimisation auxquels font face chercheurs et industriels les a amenés à développer des méthodes heuristiques. En effet, en raison de la limitation des techniques exactes d'énumération, les méthodes approchées (heuristiques) sont devenues une alternative fiable. Ces approches sont largement utilisées pour résoudre des problèmes de grande taille bien qu'elles ne permettent pas de garantir l'obtention de la solution optimale. Les premiers algorithmes approximatifs sont les règles d'affectation. Ces règles utilisent des caractéristiques du problème traité (telles que les durées opératoires des tâches, les charges des machines, etc) pour affecter une priorité d'exécution à chaque tâche. Ses priorités permettent de sélectionner à tout instant $t$ la tâche à exécuter. En général la mise en oeuvre et l'application de ce genre de règles sont faciles à réaliser. En plus, elles ne demandent pas un temps opératoire excessif. Par contre, le champs de leur application est limité à certains objectifs. Panwalker et Iskander [PI77] ont publié un état de l'art complet sur les règles d'affectation. Ces mêmes auteurs ont proposé une classification en deux catégories des règles : les règles de priorité et les règles heuristiques d'ordonnancement. Les règles de priorité sont basées sur des informations liées aux opérations, aux jobs ou aux machines pour déterminer la priorité d'allocation des ressources. Les règles heuristiques nécessitent des considérations beaucoup plus compliquées et utilisent d'autres paramètres tel que la charge des machines. Blackstone et al. [BPH82] ont conclu que la règle SPT[6] est la plus adaptée au job shop sans considérer les retards.

Adams et al ont publié en 1988 l'heuristique "Shifting Bottleneck Procedure (SBP)" [AED88]. Les auteurs traitent les machines consécutivement dans un ordre de priorité décroissant. Ainsi, la machine ayant l'ordre le plus important est sélectionnée. A chaque fois une seule machine est considérée en affectant d'une manière optimale ses opérations sans tenir compte des autres machines. Une fois une machine traitée, les séquences des machines déjà ordonnancées sont révisées. L'heuristique SBP est considérée parmi les

6. La règle SPT : Shortest processing Time, ordonnance les tâches en attente devant une ressource selon un ordre croissant des temps opératoires.

premiers algorithmes d'approximation efficaces dédiés au problème de job shop depuis que la recherche s'est focalisée sur les règles d'affectation et les méthodes d'énumération complète.

Les débuts des années 90 ont connu la formulation de certains nouveaux algorithmes innovants incluant : la recherche tabou [Tai94] et la simulation [YN95]. Lors de cette même époque, les chercheurs commencent à s'intéresser aux techniques de l'intelligence artificielle pour résoudre le job shop. Nous citerons les algorithmes génétiques [CTV95], [YN92], [YN97], les réseaux de neurone [JM98a], [Jon97] et les algorithmes de fourmi (ants systems) [ZM99].

Malgré leur capacité d'apprentissage et de généralisation, les réseaux de neurones sont peu utilisés pour l'ordonnancement des systèmes de production en général, et pour l'ordonnancement de job shop en particulier. En effet, des réseaux tels que celui développé par Jonsson [Jon97] n'ont pas fourni des résultats performants. Ceci est dû d'une part à la complexité grandissante du job shop, et d'autre part à la difficulté de trouver une bonne fonction d'activation des neurones. Un état de l'art sur l'application des réseaux de neurones à la résolution du job shop, donné par Jain [JM98a], montre que les réseaux de neurones ne sont pas utilisés dans le cadre de l'apprentissage. Ils sont plutôt utilisés pour résoudre des problèmes particuliers sans pouvoir généraliser le réseau à des variétés d'instances (ils sont parfois non applicables à des tailles supérieures à celui pour lequel le réseau est construit). La plupart des réseaux de neurones sont basés sur les modèles de «Hopfield» ou «back error propagation». Ils cherchent à optimiser une fonction d'énergie $E$ (par exemple, la somme des dates de débuts des opérations).

L'algorithme de la fourmi (ants systems) est une méthode évolutive dont les mécanismes de recherche s'inspirent fortement du comportement collectif d'une colonie de fourmis. Les rares travaux utilisant ces algorithmes pour résoudre le job shop ont rencontré d'énormes difficultés d'adaptation. A titre d'exemple, Zwaan et Marques [ZM99] ont proposé un algorithme de fourmi pour résoudre le job shop. Les résultats obtenus sur des benchmarks de taille allant de $10 \times 10$ à $20 \times 10$ présentent des écarts par rapport à l'optimum variant de 8% à 31.9%.

Les algorithmes génétiques sont largement appliqués à la résolution du problème de job shop. A l'inverse des autres approches, ces algorithmes manipulent plusieurs solutions en parallèle. La phase la plus importante dans cette approche et qui affecte la qualité des solutions finales est la représentation génétique des solutions (c'est à dire le codage). Davis [Dav85] est le premier à avoir proposé et démontré en 1985 la faisabilité des algorithmes génétiques pour la résolution du job shop sous sa forme la plus simple.

Ponnambalam et al. ont évalué dans [PRJ01a] les opérateurs de codage les plus utilisés pour étudier le job shop. L'adaptation des algorithmes génétiques aux job shop ainsi que les différents opérateurs seront détaillés dans le chapitre suivant.

## 1.5 Nécessité de l'ordonnancement conjoint

Dans le domaine de la production industrielle, les tendances actuelles indiquent que les systèmes manufacturiers performants doivent s'adapter rapidement aux fluctuations du marché où les demandes de produits deviennent aléatoires, et aux perturbations internes à l'atelier dues aux pannes des machines. Les machines doivent pouvoir fabriquer plusieurs types de produits simultanément et de manière efficace. Dans un tel contexte, la planification optimale de la production et le contrôle en temps réel de ces machines deviennent de plus en plus préoccupants tant pour les investisseurs et producteurs que pour les consommateurs. Dans ces conditions, la détermination d'un rythme de production, d'une politique de maintenance des équipements, et des règles d'ordonnancement et d'affectation des produits aux machines est un problème préoccupant dans le domaine de l'optimisation des systèmes industriels.

Une des préoccupations majeures du monde industriel est d'avoir un système de production performant permettant de garantir au mieux la qualité des produits fabriqués, le maintien des délais demandés et la minimisation des coûts de production. Toutefois, aujourd'hui dans bon nombre de PME/PMI, la production est ordonnancé en considérant que les machines sont disponibles à tout moment pour exécuter des tâches de production sur un horizon planifié. Cependant, elles peuvent tomber en panne, être en révision, en entretien ou bien en réparation. La maintenance sur un équipement n'est pas exceptionnelle mais suit un certain planning fourni souvent par le constructeur. Malgré cela, bien souvent le calendrier prévisionnel de fabrication est élaboré indépendamment du planning de maintenance préventive. Il arrive qu'il y ait antagonisme entre les deux plannings, ce qui ne contribue pas à une optimisation du fonctionnement de l'atelier. L'expérience montre que souvent les machines restent inoccupées longtemps en attendant l'intervention des équipes de maintenance [LC00]. Également, des tâches de production peuvent être reportées à cause des interventions prématurées de maintenance (pourtant les machines fonctionnent bien). Ceci revient au manque de coordination entre les équipes de production et de maintenance. Ce manque de coordination entre les équipes de maintenance et de production, nuit au bon fonctionnement de l'atelier. Il serait judicieux, d'exploiter les périodes d'arrêt de production pour faire la maintenance systématique, ne serait ce qu'en jouant sur la plage de période, requise pour la maintenance des machines.

L'équipement est donc une ressource partagée entre les équipes de production et de maintenance et la fonction maintenance est directement impliquée dans le rendement de l'atelier au même titre que la production. Une politique de maintenance est donc nécessaire pour planifier les interventions régulières sur les machines de l'atelier. Pour améliorer les performances de l'atelier, et influer au mieux sur le triptyque coût, qualité et délais, il est indispensable d'ordonnancer conjointement la production et la maintenance.

Rares sont les travaux qui traitent de l'ordonnancement conjoint de la production et de la maintenance au sein du job shop. Les cas étudiés sont limités à des ateliers composés d'une ou de deux machines. A notre connaissance, cet aspect d'ordonnancement conjoint n'est pas considéré dans les ateliers de type job shop à cause de leur complexité. C'est la raison la plus motivante qui nous a poussé à nous intéresser à ce type de problème.

## 1.6 Conclusion

A travers ce chapitre, nous avons présenté les notions de base de l'ordonnancement, et en particulier celles concernant les ateliers de type job shop. Nous avons montré que les problèmes d'ordonnancement dans les ateliers de type job shop sont largement étudiés dans la littérature et sont des problèmes $\mathcal{NP}$-*difficiles* à l'exception de quelques instances particulières. Pour faciliter la recherche d'une solution optimale, il est préférable de se limiter à l'ensemble des ordonnancements actifs. Dans ce cas, il faut que le critère d'optimisation considéré soit régulier.

La complexité du problème de job shop limite l'efficacité des approches existantes malgré l'évolution spectaculaire de la puissance des calculateurs. On constate que la tendance actuelle est à l'utilisation de métaheuristiques, de méthodes issues de l'ethologie, d'algorithmes évolutifs parmi lesquels les algorithmes génétiques. Ces derniers semblent être une piste prometteuse et laissent espérer une résolution de pas mal de problèmes d'ordonnancement. En effet, depuis leur adaptation au début des années 80 au job shop, ces techniques connaissent un succès considérable. Le deuxième chapitre sera consacré à la présentation des algorithmes génétiques ainsi qu'à leurs applications aux problèmes d'ordonnancement d'une façon générale, et plus particulièrement au job shop.

# Chapitre 2

# Les algorithmes génétiques et l'ordonnancement

***Résumé :*** *Dans ce chapitre, nous introduisons les algorithmes génétiques : leur principe de base, les opérateurs participant à l'exploration de l'espace de recherche, les paramètres nécessaires pour la convergence vers des bonnes solutions, etc. Les algorithmes génétiques sont très bien adaptés au traitement des problèmes d'optimisation multiobjectif. Nous montrerons les différences entre un algorithme génétique mono objectif et celui optimisant plusieurs objectifs. Plusieurs classifications des méthodes génétiques multiobjectif sont apparues dans la littérature. Ces classifications se basent sur l'utilisation ou non du concept de Pareto ou sur l'agrégation ou non des différents objectifs. Nous allons détailler le principe de ces différentes méthodes en utilisant une classification selon l'utilisation ou non de l'optimisation au sens Pareto. La dernière partie de ce chapitre est consacrée à l'étude de l'application des algorithmes génétiques aux problèmes d'ordonnancement en général et au problème du job shop en particulier. Les différents types de codage utilisés pour traiter le problème de job shop seront étudiés ainsi que l'adaptation des opérateurs génétiques.*

## 2.1 Introduction

Parmi les méthodes de l'intelligence computationnelle[1] telles que les réseaux de neurones et les systèmes flous (figure 2.1), les méthodes basées sur les algorithmes évolutifs constituent une approche originale. Ces algorithmes stochastiques sont inspirés des mécanismes de l'évolution naturelle élaborée par Charles Darwin (Sélection, adaptation, reproduction, recombinaison et mutation). Avec ce type de méthodes, il ne s'agit pas de trouver une solution analytique exacte ou une bonne approximation numérique, mais de trouver des solutions satisfaisantes. Ces méthodes ne permettent pas de trouver à coup sûr la solution optimale de l'espace de recherche, du moins peut-on constater que les solutions fournies sont généralement meilleures que celles obtenues par des méthodes plus classiques, pour un même temps de calcul.

Suite à leur apparition, plusieurs variantes de ces algorithmes évolutifs ont été développées isolément et à peu près simultanément par différents chercheurs [Koz] :

- les programmes évolutifs se basent sur la création aléatoire d'une population d'individus. Chaque individu subit une mutation. Ensuite, les nouveaux individus sont évalués pour sélectionner ceux qui vont former la population suivante ;
- les stratégies d'évolution dont la plus simple est la stratégie d'évolution (1+1) ((1+1)-evolution strategy), font intervenir un seul individu pour la reproduction. L'évolution est réalisée uniquement par un opérateur de mutation ;
- les programmes génétiques appliquent les principes de l'évolution génétique à la recherche de programmes (au sens général du terme) avec des opérateurs adaptés aux structures manipulées (arbre, pile, graphe). Le principe de la programmation génétique s'inspire du principe général des algorithmes génétiques. De petits programmes constituant un chromosome, évoluent et s'assemblent afin de fournir un programme permettant de résoudre le problème donné ;
- les algorithmes génétiques sont les plus utilisés parmi les méthodes évolutives pour résoudre des problèmes d'optimisation. Ces algorithmes se distinguent par l'opérateur de croisement qui permet de combiner deux individus pour générer deux autres, généralement plus performants.

Le choix des algorithmes génétiques s'est imposé tout naturellement à nous pour la recherche des bonnes solutions du problème du job shop que nous étudions au troisième

1. L'intelligence computationnelle est un terme populaire utilisé pour référencer les techniques basées sur la puissance de calcul des ordinateurs et sur l'intelligence humaine pour résoudre des problèmes difficiles.

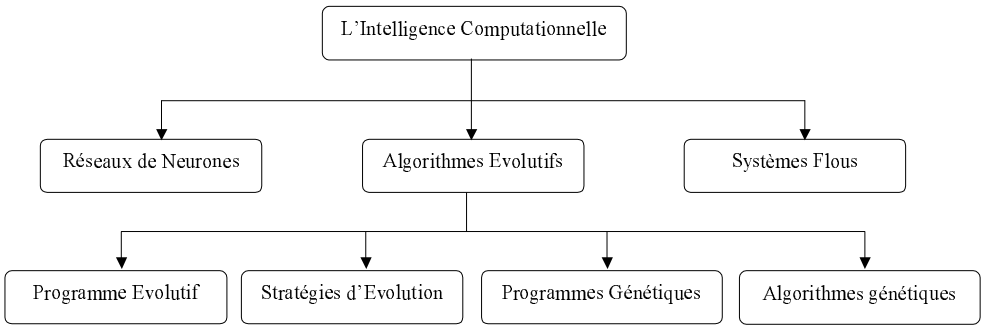



FIG. 2.1 – *Différentes méthodes de l'intelligence computationnelle*

chapitre. Les éléments constituant ces algorithmes ainsi que le principe de leur fonctionnement seront abordés dans la première partie de ce chapitre. Les algorithmes génétiques permettent de travailler sur des problèmes d'optimisation multiobjectif que nous développons dans la deuxième partie de ce chapitre. Finalement, nous étudions l'application des algorithmes génétiques à la résolution des problèmes d'ordonnancement en général et du job shop en particulier.

## 2.2 Les algorithmes génétiques

Les Algorithmes Génétiques sont des méthodes d'exploration fondées sur des techniques dérivées de la génétique et l'évolution naturelle. Dans ce cas, il s'agit de simuler l'évolution naturelle d'organismes (individus), génération après génération, en respectant des phénomènes d'hérédité et une loi de survie. Ils n'appliquent pas une méthode de résolution à un espace de définition pour obtenir une solution, mais, ils s'appuient sur un autre concept. Ils balaient l'espace de solutions, et ils valident cet espace par vérification de contraintes, et évolution d'une population de solutions.

Les algorithmes génétiques ont déjà une histoire relativement ancienne. De tels algorithmes furent développés dès 1950 par des biologistes qui utilisaient des ordinateurs pour simuler des organismes. C'est Holland [Hol75], ses collègues et ses étudiants qui ont développé les algorithmes génétiques en 1975 à l'université de Michigan. Goldberg [Gol89] apportera également beaucoup à la création de ces algorithmes et à leur adaptation pour la recherche de solutions à des problèmes d'optimisation. Cette adaptation est réalisée en développant une analogie entre un individu dans une population et une solution d'un problème dans un ensemble de solutions.

Les algorithmes génétiques sont largement utilisés pour résoudre des problématiques

appartenant à divers domaines. A titre d'exemple, nous pouvons citer le traitement d'image, l'optimisation des fonctions analytiques, l'ordonnancement des ateliers (et plus particulièrement le problème du job shop), etc. Les algorithmes génétiques constituent également un outil très efficace pour la résolution des problèmes d'optimisation multiobjectif contrairement aux autres méthodes. Plusieurs versions sont apparues dans la littérature permettant de traiter sous différentes formes l'optimisation multiobjectif. Ces formes seront développées dans la deuxième partie de ce chapitre.

## 2.3 Terminologie et éléments de base

Un algorithme génétique recherche les extrêmes d'une fonction définie sur un espace de données appelé population. Par analogie avec la génétique, chaque individu de cette population est un chromosome et chaque caractéristique de l'individu est un gène. Dans un cas simple, un gène sera représenté par un bit (0 ou 1), un chromosome par une chaîne de bits et un individu par un ensemble de chaîne de bits. Chaque gène représente une partie élémentaire du problème, il peut être assimilé à une variable. Il peut prendre des valeurs différentes appelées *alleles*. La position du gène dans le chromosome se nomme *locus*.

On parle également de génotype et de phénotype. Le génotype représente l'ensemble des valeurs des gènes du chromosome alors que le phénotype représente la solution réelle après transformation du chromosome. Lors de la génération d'une nouvelle population, des opérateurs génétiques tels que la sélection, le croisement et la mutation sont nécessaires pour la manipulation des chromosomes.

Le tableau 2.1 présente une récapitulation de la terminologie naturelle et celle utilisée par les algorithmes génétiques.

| Nature | Algorithme génétique |
|---|---|
| Chromosome | Chaîne |
| Gène | Trait, Caractéristique |
| Allèle | Valeur de la caractéristique |
| Locus | Position dans la chaîne |
| Génotype | Structure |
| Phénotype | Ensemble de paramètres, structure décodée |

TAB. 2.1 – *Comparaison de la terminologie naturelle et celle des algorithmes génétiques*

Les algorithmes génétiques s'inspirent principalement du fonctionnement de l'évolution de la nature. Ils sont basés sur le principe d'*évolution* d'une *population d'individus*. Dans une population d'individus, ce sont en général les plus forts, c'est-à-dire les mieux adaptés au milieu, qui survivent et engendrent des progénitures. À partir des données du problème, on crée (généralement aléatoirement) une "population" de solutions admissibles. Puis, on évalue chacune des solutions. On élimine une partie infime de celles qui se sont montrées inutiles ou désastreuses, et on recombine les gènes des autres afin d'obtenir de nouveaux individus-solutions. Ainsi, à chaque génération un nouvel ensemble de créatures artificielles (des chaînes de caractères) est crée en utilisant des parties des meilleurs éléments de la génération précédente ainsi que des parties innovatrices. Selon la théorie évolutionniste, cette nouvelle génération sera globalement plus adaptée au problème que la précédente. Ce procédé est alors répété jusqu'à la naissance d'une solution que l'on jugera satisfaisante.

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

- d'une *représentation génétique* du problème, c'est-à-dire un codage approprié des solutions sous la forme de chromosomes. Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne le succès des algorithmes génétiques ;
- d'un mécanisme de génération de la population initiale. Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connaît rien sur le problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche ;
- d'une fonction d'évaluation pour mesurer la force de chaque chromosome ;
- d'un mode de sélection des chromosomes à reproduire ;
- des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'états ;
- des valeurs pour les paramètres qu'utilise l'algorithme : taille de la population, nombre total de générations ou critère d'arrêt, probabilités de croisement et de mutation.

Un algorithme génétique fonctionne typiquement à travers un cycle simple de quatre étapes :

1. création d'une population de chromosomes ;
2. évaluation de chaque chromosome ;
3. sélection des meilleurs chromosomes ;
4. manipulation génétique, pour créer une nouvelle population de chromosomes.

Ce cycle décrit par le schéma 2.2, est inspiré de la terminologie génétique. Lors de chaque cycle, une nouvelle génération de solutions du problème est produite. Avant d'exécuter ce cycle, une population initiale de solutions admissible doit être fournie. Chaque individu-solution de la population est codé sous forme d'une chaîne de caractères (chromosome) afin d'être manipulé par les opérateurs génétiques. L'étape suivante consiste à évaluer la qualité de chaque chromosome à l'aide de la fonction d'évaluation : *fitness*. En se basant sur les fitness des chromosomes, un mécanisme de sélection permet de garder les individus les plus adaptés pour être manipulés par les opérateurs génétiques (croisement et mutation) et ensuite créer une nouvelle génération. Les étapes d'évaluation et de sélection constituent le processus de reproduction.

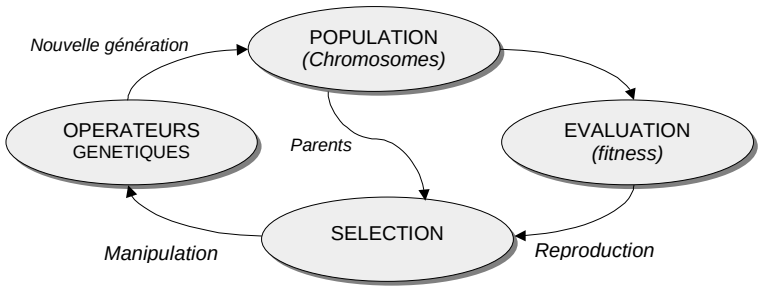



FIG. 2.2 – *Cycle génétique*

### 2.3.1 Codage

Le codage est une modélisation d'une solution d'un problème donné sous forme d'une séquence de caractères appelée chromosome où chaque caractère, dit aussi gène, représente une variable ou une partie du problème. La tâche principale consiste à choisir le contenu des gènes qui facilite la description du problème et respecte ses contraintes. Il existe dans la littérature deux types de codage : le codage direct et le codage indirect [GRG01]. Si le passage du génotype au phénotype est immédiat, le codage est dit direct.

Par contre, si le génotype nécessite une transformation pour obtenir le phénotype, alors il s'agit du codage indirect. A titre d'exemple, pour résoudre le problème du voyageur du commerce avec un algorithme génétique, il est possible d'utiliser un codage constitué des lettres représentants les noms des villes à visiter. Dans ce cas, les chromosomes obtenus à la fin de la dernière exécution du cycle génétique ne nécessitent aucune transformation pour extraire les solutions effectives : c'est l'ordre des villes dans chaque chromosome qui est prise en compte.

Le codage classique utilise l'alphabet binaire : 0, 1. Dans ce cas, le chromosome représente simplement une suite de 0 et de 1. Goldberg [Gol89] a démontré dans sa théorie de schème [2] que le codage binaire permet d'avoir le nombre maximal de schemata possible, ce qui permet une exploration plus exhaustive de l'espace de recherche. Le codage binaire est également indépendant des opérateurs génétiques (croisement et mutation). Ces derniers ne nécessitent aucune spécification ou adaptation. En effet, toute manipulation d'un chromosome donne naissance à un nouveau chromosome valide. Dans la pratique, le codage binaire peut présenter des difficultés [Woo97], [ClPP95]. En effet, il est parfois très difficile ou très lourd de coder des solutions de cette manière. D'autre part, dans certains cas la place mémoire requise peut devenir prohibitive. En plus, si le nombre de valeurs discrètes qu'un gène peut prendre n'est pas une puissance de deux, alors l'algorithme génétique peut générer des chromosomes non valides. Prenons par exemple un problème à dix variables devant se coder en 0 et 1. La taille des chromosomes nécessaire pour coder toutes les variables est de quatre. Ceci implique qu'il est possible d'avoir $2^4 - 10$ soit 6 chromosomes non admissibles.

Le codage binaire a permis certes de résoudre beaucoup de problèmes, mais il s'est avéré que pour des problèmes d'optimisation numérique ou des problèmes d'ordonnancement il est plus pratique d'utiliser un codage réel des chromosomes. Un gène est ainsi représenté par un nombre réel au lieu d'avoir à coder les réels en binaire puis de les décoder pour les transformer en solutions effectives. Le codage réel permet d'augmenter l'efficacité de l'algorithme génétique et d'éviter des opérations de décodage supplémentaires. Un tel type de codage nécessite l'utilisation des opérateurs génétiques appropriés pour générer des chromosomes valides. Cette dernière propriété permet d'incorporer des informations spécifiques au problème lors de la construction des opérateurs ce qui leurs rend plus efficaces et plus adaptés.

2. Un schème est un motif de similarité décrivant un sous-ensemble de chaînes avec des similarités à des positions définies. Concernant le codage binaire, la mise en oeuvre de schémas nécessite d'ajouter à l'alphabet des gènes un symbole supplémentaire * qui représente indifféremment 0 ou 1. Ainsi, un schéma 011* représente les deux chromosomes 0110 et 0111.

Un chromosome codé en réels est plus court que celui codé en binaire. Un nombre réduit de gènes signifie une convergence plus rapide de l'algorithme génétique et une perte de la diversité de la population. Ceci influence éventuellement la qualité de la solution finale.

### 2.3.2 Population initiale

Une fois le codage choisi, une population initiale formée de solutions admissibles du problème doit être déterminée. Plusieurs mécanismes de génération de la population initiale sont utilisés dans la littérature [ClPP95]. La méthode la plus classique consiste à générer aléatoirement les chromosomes constituant la population initiale. Cette méthode répond à la nécessité d'avoir une population variée permettant d'explorer des zones diverses de l'espace de recherche. Des heuristiques peuvent être utilisées pour générer des solutions admissibles, dans l'espoir de les voir s'améliorer dans les générations futures. Cependant, une telle pratique peut amener l'algorithme génétique à converger vers des optimums locaux. Dans le but d'accélérer la convergence de l'algorithme génétique, et en même temps de garder une diversité nécessaire de la population initiale, les deux méthodes précédentes peuvent être utilisées simultanément. Lorsque le problème est fortement contraint et qu'il est difficile de générer plusieurs solutions, on peut être amené à trouver par une heuristique ou demander à un expert une seule solution que l'on dupliquera par croisement ou par mutation jusqu'à l'obtention de plusieurs solutions.

Le problème principal dans cette étape est le choix de la taille de la population. Les biologistes ont introduit le concept de *diversité requise* de la population. Ainsi, pour survivre, une espèce doit être suffisamment hétérogène. Par ailleurs, une population trop grande augmente le temps de calcul et demande un espace mémoire considérable. Il faut donc trouver le bon compromis. Nous allons discuter dans la section des paramètres de l'algorithme génétique, la taille possible qu'une population doit avoir.

### 2.3.3 Evaluation : fitness

Afin de mesurer les performances de chaque individu qui correspond à une solution donnée du problème à résoudre, on introduit une fonction d'évaluation. Cette fonction correspond au profit ou à l'utilité de la solution par rapport au problème. Elle permet de quantifier la capacité d'un individu à survivre en lui affectant un poids couramment appelé *fitness*. La force de chaque chromosome de la population est calculée afin que les plus forts soient retenus (étape de sélection) puis modifiés (croisement et mutation). L'algorithme génétique tend alors à maximiser la force des individus de la population.

La complexité de la fonction d'évaluation dépend essentiellement du problème et de ses contraintes. Dans le cas des problèmes d'ordonnancement, le choix de cette fonction est presque évident (la date de fin de l'ordonnancement, le retard, le nombre de tâches en retard, etc.).

Si le problème étudié consiste à maximiser un critère donné, ce dernier peut servir directement pour l'évaluation des individus. Prenons l'exemple d'un problème d'affectation d'un ensemble de $n$ ouvriers à un ensemble de $m$ machines. Supposons que l'ouvrier $i$ apporte le gain $g_{ij}$ en travaillant sur la machine $j$. L'objectif est de trouver la combinaison (ouvrier, machine) optimale qui maximise le gain total. Dans ce cas, la fonction d'évaluation peut s'écrire sous la forme suivante :

$$F = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} g_{ij} \tag{2.1}$$

Où $a_{ij}$ est une variable booléenne indiquant si l'ouvrier $i$ est affecté sur la machine $j$ ou pas.

Dans le cas où le critère d'optimisation est à minimiser, il est nécessaire de chercher son complémentaire pour se ramener au cas du maximisation. Si $C(x)$ représente la valeur du critère à optimiser pour l'individu $x$, alors l'expression de la fonction d'évaluation peut être la suivante :

$$F(x) = \begin{cases} C_{max} - C_x & si \;\; C(x) > 0 \\ 0, & sinon \end{cases} \tag{2.2}$$

Où $C_{max}$ peut être un coefficient fixé ou la plus grande valeur observée de $C(x)$ dans la population courante, soit depuis le début de l'algorithme.

Certains problèmes abordés ne peuvent pas être modélisés par un outil mathématique, dans ce cas, les individus sont évalués par des simulateurs [Cav00]. C'est généralement le cas des fonctions non linéaires, non continues et/ou non dérivables.

### 2.3.4 Sélection

La sélection est un procédé dans lequel chaque individu est choisi en fonction de sa valeur d'évaluation. C'est l'étape qui sélectionne les individus à partir desquels la population suivante sera créée. Cet opérateur génétique inspiré de la sélection naturelle (appelé aussi opérateur de reproduction), est un processus qui permet de choisir parmi la population courante d'individus, ceux les plus adaptés pour se présenter au croisement et à la mutation. Ce choix est crucial pour l'évolution de la performance globale de la

population. Vladimir a démontré dans [Vla96] que lorsqu'un algorithme génétique est utilisé pour maximiser une fonction objectif, alors c'est le processus de sélection qui assure la convergence vers un optimum global.

Il existe de nombreuses techniques de sélection. Nous présentons ici les trois les plus utilisées parmi elles :

- **la sélection par classement** : elle consiste à ranger les individus de la population dans un ordre croissant (ou décroissant selon l'objectif) et à retenir un nombre fixé de génotypes. Ainsi, seuls les individus les plus forts sont conservés. L'inconvénient majeur de cette méthode est la convergence prématurée de l'algorithme génétique. Il est parfois nécessaire de garder quelques individus jugés faibles pour créer la diversité au niveau de la population. En plus, les individus faibles contiennent parfois des gènes intéressants et pourront contribuer à l'apparition de bonnes solutions. Une autre difficulté consiste à fixer une limite à la sélection ce qui empêche parfois de garder des bons candidats pour les futures générations ;

- **la sélection par la roulette** : elle consiste à créer une roue de loterie biaisée pour laquelle chaque individu de la population occupe une section de la roue proportionnelle à sa valeur d'évaluation. Ainsi, même les individus les plus faibles ont une chance de survivre. Si la population d'individus est de taille égale à $N$, alors la probabilité de sélection d'un individu $x_i$ notée $p(x_i)$ est égale à :

$$p(x_i) = \frac{F(x_i)}{\sum_{k=1}^{N} F(x_k)} \tag{2.3}$$

  En pratique, on calcule pour chaque individu $x_i$ sa probabilité cumulée $q_i = \sum_{j=i}^{i} p(x_i)$ et on choisi aléatoirement un nombre $r$ compris entre 0 et 1. L'individu retenu est : $x_1$ si $q_1 \geq r$ ou $x_i (2 \leq i \leq N)$ si $q_{i-1} < r \leq q_i$. Ce processus est répété $N$ fois. Avec une telle sélection, un individu fort peut être choisi plusieurs fois. Par contre, un individu faible a moins de chance d'être sélectionné.

  Le tableau 2.2 présente un exemple d'une population de quatre individus $I_1$ jusqu'à $I_4$ avec leurs valeurs d'évaluation respectives et leurs probabilités de sélection. La roue associée à cet exemple est représentée par la figure 2.3 ;

- **la sélection par tournoi** : elle consiste à choisir aléatoirement deux ou plusieurs individus et à sélectionner le plus fort. Ce processus est répété plusieurs fois jusqu'à l'obtention de $N$ individus. L'avantage d'une telle sélection est d'éviter qu'un individu très fort soit sélectionné plusieurs fois.

| Individu | Fitness | $P_{selection}$ |
|---|---|---|
| $I_1$ | 135 | 0,5625 |
| $I_2$ | 60 | 0,2500 |
| $I_3$ | 30 | 0,1250 |
| $I_4$ | 15 | 0,0625 |

TAB. 2.2 – *Exemple de sélection à la roulette*

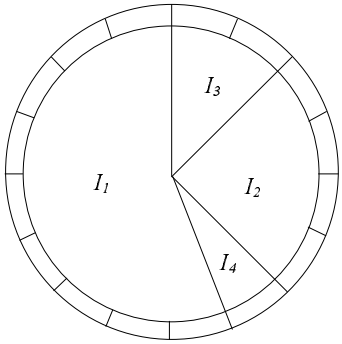



FIG. 2.3 – *Sélection à la roulette de Goldberg*

On pourra toutefois introduire la notion d'élitisme dans cette méthode. Si l'individu le plus fort n'a pas été sélectionné, il est copié dans la génération suivante à la place d'un autre choisi aléatoirement.

### 2.3.5 Croisement

Afin de donner naissance à un nouvel individu, il suffit de prendre aléatoirement une partie des gènes de chacun des deux parents. Ce phénomène, issu de la nature est appelé *croisement* (crossover). Il s'agit d'un processus essentiel pour explorer l'espace des solutions possibles. Une fois la sélection terminée, les individus sont aléatoirement répartis en couples. Les chromosomes parents sont alors copiés et recombinés à fin de produire chacun deux descendants ayant des caractéristiques issues des deux parents. Dans le but de garder quelques individus parents dans la prochaine population, on associe à l'algorithme génétique une probabilité de croisement notée $P_{cross}$ qui permet de décider si les parents seront croisés entre eux ou s'ils seront tout simplement recopiés dans la population suivante.

Il existe dans la littérature plusieurs opérateurs de croisement. Ils différent selon le type de codage adapté et la nature du problème traité.

Nous allons citer par la suite les opérateurs de croisement les plus utilisés en les classant en deux catégories : le croisement binaire et le croisement réel.

#### 2.3.5.1 Croisement binaire

- **croisement en 1-point** : c'est le croisement le plus simple et le plus connu dans la littérature. Il consiste à choisir au hasard un point de croisement pour chaque couple de chromosomes. Les sous-chaînes situées après ce point sont par la suite inter-changées pour former les deux fils (voir la figure 2.4) ;

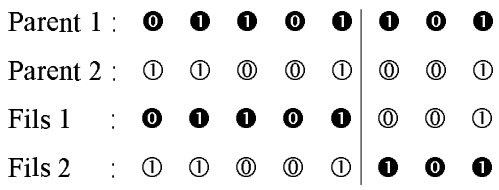



FIG. 2.4 – *Croisement en un point de deux chromosomes*

- **croisement en n-points** : ce type de croisement est utilisé en choisissant aléatoirement $n$ points de coupure pour dissocier chaque parent en $n+1$ fragments. Pour former un fils, il suffit de concaténer alternativement $n+1$ sous-chaînes à partir des deux parents. Ce croisement cherche à explorer tout l'espace de solutions possibles en créant des descendants ayant des caractéristiques très loin des parents ;

- **croisement en 2-points** : c'est un cas particulier du croisement en n-points. On choisit aléatoirement deux points de coupure pour créer les descendants ;

- **croisement uniforme** : cette technique génère des progénitures gène par gène à partir des deux parents. Il existe deux versions de ce croisement. La première, consiste à recopier les gènes du premier parent, à la même position, sur le premier fils ou sur le deuxième, avec une probabilité égale à 0.5. Les gènes manquants de chaque fils sont complétés à partir du père opposé sans changement de positions. La deuxième utilise un masque pour donner naissance à des progénitures. Si la valeur du masque est égale à 1, l'enfant 1 reçoit l'allèle correspondant du parent 1 et l'enfant 2 reçoit celui du parent 2. Sinon, l'échange se fait dans l'autre sens (Figure 2.5).

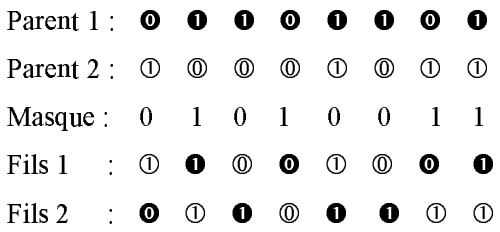



FIG. 2.5 – *Croisement uniforme de deux chromosomes*

#### 2.3.5.2 Croisement réel

Nous avons signalé auparavant que le codage réel nécessite des opérateurs génétiques spécifiques pour la manipulation des chromosomes. En effet, les opérateurs développés pour le codage binaire, peuvent s'ils sont utilisés dans le cadre d'un codage réel, générer des solutions non admissibles qui ne respectent pas les contraintes du problème étudié. Nous allons par la suite décrire à titre d'exemple trois opérateurs de croisement : le croisement d'ordre de base cyclique, le croisement uniformément continu et le croisement d'ordre maximal. Pour plus de détails sur d'autres opérateurs de croisement, le lecteur peut se référer à [Woo97].

- **ordre de base cyclique** : pour créer un fils, il suffit de copier une sous-chaîne d'un parent et de compléter les gènes manquants à partir de l'autre parent en maintenant l'ordre des gènes. Généralement, une fois deux chromosomes parents sélectionnés pour le croisement, deux points de coupures sont choisis aléatoirement sur chaque parent. Ensuite on place les sous-chaînes entre les points de coupures sur les deux fils dans la même position que les parents. Pour compléter les gènes manquants du fils 1, on commence par insérer les gènes situés à droite du deuxième point de coupure du parent 2 tout en gardant l'ordre des gènes et en ignorant les gènes déjà pris. Le deuxième fils est complété à partir du parent 1 de la même manière que le fils 1. La figure 2.6 montre sur un exemple les étapes de ce type de croisement ;

- **croisement uniformément continu** : ce croisement est proposé par [Alt95]. Les auteurs ont suggéré cet opérateur pour produire des chromosomes valides. Un chromosome $X = (x_1, x_2, ..., x_n)$ est valide lorsque : $\sum_{i=1}^{n} x_i = 1$. Étant donnés deux chromosomes valides $X = (x_1, x_2, ..., x_n)$ et $Y = (y_1, y_2, ..., y_n)$, les descendants $X' = (x'_1, x'_2, ..., x'_n)$ et $Y' = (y'_1, y'_2, ..., y'_n)$ sont définis de la façon suivante : $x'_i = sx_i + (1-s)y_i$ et $y'_i = (1-s)x_i + sy_i$. Où $s$ est une constante choisi à chaque itération aléatoirement dans l'intervalle $[-0.5\ ,\ 0.5]$ ;

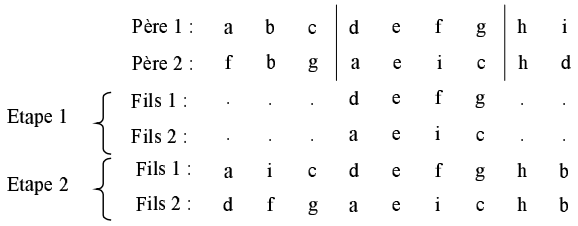



FIG. 2.6 – *Croisement d'ordre de base cyclique*

– **croisement d'ordre maximal** : ce type de croisement a pour objectif de garder le maximum possible les positions et l'ordre des gènes. On commence par choisir aléatoirement deux points de coupure. Les sous-chaînes situées au milieu sont interchangées. Les gènes manquants sont par la suite complétés à partir de chaque père en allant de gauche à droite et en choisissant le premier caractère disponible. A la différence du croisement de base cyclique, le fils 1 est complété à partir du parent 1 et le fils 2 à partir du parent 2. La figure 2.7 illustre un exemple de croisement d'ordre maximal.

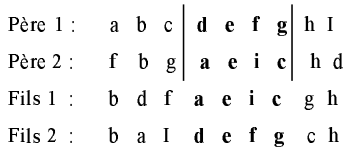



FIG. 2.7 – *Croisement d'ordre maximal*

### 2.3.6 Mutation

La mutation est définie comme étant la modification aléatoire d'une partie d'un chromosome. C'est un phénomène qui joue le rôle de bruit et empêche l'évolution de se figer. Elle permet d'assurer une recherche aussi bien globale que locale, selon le poids et le nombre de bits mutés.

Il existe de nombreuses manières de mutation d'un chromosome. Pour un problème utilisant le codage binaire, la mutation la plus connue consiste à inverser la valeur d'un bit choisi aléatoirement. Pour le codage réel, les opérateurs de mutation les plus connus sont les suivants :

– **l'opérateur d'inversion simple** : il consiste à choisir aléatoirement deux points de coupure et inverser les positions des bits situés au milieu ;

- **l'opérateur d'insertion** : cet opérateur consiste à sélectionner au hasard un bit et une position dans le chromosome à muter, puis à insérer le bit en question dans la position choisie ;
- **l'opérateur d'échange réciproque** : c'est un opérateur de mutation qui permet de sélectionner deux bits et les inter changer.

La mutation joue un rôle secondaire par rapport au croisement. Elle est utilisée pour introduire de légères modifications à certains individus de la population. C'est pour ça qu'on lui attribue généralement une faible probabilité (choisie entre 0.001 et 0.01 ou fixée à la valeur inverse de la longueur du chromosome).

### 2.3.7 Valeurs des paramètres

Les paramètres qui conditionnent la convergence d'un algorithme génétique sont la taille de la population d'individus, le nombre maximal de générations, la probabilité de croisement et la probabilité de mutation. Les valeurs de tels paramètres dépendent fortement de la problématique étudiée : de sa taille, du nombre de variables, etc. Des recherches ont été menées dans ce domaine par [EHM99], [Jon75] et [LT94] ont montré la difficulté de fixer de tels paramètres. Il existe néanmoins des valeurs, proches de celles de la nature qui permettent d'obtenir de bons résultats. Par exemple, la probabilité de croisement appartient à l'intervalle [0.7 , 0.99]. De même, la probabilité de mutation est souvent choisie dans l'intervalle [0.0001 , 0.01]. En effet, une mutation avec une grande probabilité perturbe la convergence en induisant une oscillation de la valeur moyenne du critère à optimiser. En revanche, un faible taux de mutation permet d'assurer une bonne exploration de l'espace de recherche. Afin d'accélérer la convergence de l'algorithme génétique, la taille de la population n'excède pas en général la valeur de 1000. Le choix d'une population de faible effectif conduira à l'obtention d'un optimum local. Par contre, une grande population engendrera un temps de calcul excessif.

Bien évidement, ces valeurs couramment utilisées ne sont là qu'à titre d'exemple. Des algorithmes génétiques différents auront certainement des valeurs différentes des paramètres. Pour trouver les bonnes valeurs des paramètres, [Woo97] a suggéré l'exécution de l'algorithme génétique plusieurs fois avec des valeurs différentes des paramètres génétiques. Parfois, il n'est pas évident de fixer expérimentalement ces valeurs, il est nécessaire donc d'avoir recours à l'expérience humaine et à l'intuition.

## 2.4 Les algorithmes génétiques et l'optimisation multiobjectif

Nous vous avons décrit, la résolution par les algorithmes génétiques, d'une fonction mono objectif. Par contre, dans le problème que l'on rencontre, la modélisation suivant un seul critère est trop restrictive et donne des résultats tronqués. C'est pour cela que l'on s'intéresse aux problèmes d'optimisation multiobjectif.

Dans un problème multiobjectif on ne parle plus de notion d'optimalité. On aborde une nouvelle notion, à savoir la Pareto optimalité. Mathématiquement, le concept de Pareto optimalité peut être défini de la manière suivante : considérons (sans perdre de généralité), le problème de maximisation défini par l'équation 2.4 où $m$ est le nombre de paramètres (ou variables de décision), $n$ est le nombre d'objectifs, $x = (x_1, x_2, \ldots, x_n) \in X$ ($X$ est l'espace de solutions réalisables) et $y = (f_1, f_2, \ldots, f_n) \in Y$.

$$y = f(x) = (f_1(x), f_2(x), ..., f_n(x)) \tag{2.4}$$

Un vecteur $x \in X$ *domine* un vecteur $x' \in X$, on note aussi $x \succ x'$ ssi :

$$\forall \;\; i \in \{1, 2, ..., n\} \quad f_i(x) \geq f_i(x') \tag{2.5}$$

$$\exists \;\; j \in \{1, 2, ..., n\} \;\; / \;\; f_j(x) > f_j(x') \tag{2.6}$$

Un vecteur $x \in X$ est dit Pareto optimal s'il n'est dominé par aucun autre vecteur de $X$.

Contrairement aux problèmes mono objectif, le but d'une optimisation multiobjectif consiste à trouver plusieurs solutions Pareto optimales. Or les méthodes classiques de résolution ne génèrent qu'une seule solution Pareto optimale à la fois ou proche de Pareto optimale. Pour obtenir plusieurs solutions Pareto optimales, ces méthodes sont appliquées plusieurs fois. Cependant, les algorithmes génétiques sont très utilisés dans le traitement d'un problème d'optimisation multiobjectif. En témoigne le nombre important de travaux qui ont été publiés dans ce domaine [Alt95], [FP95], [MRTD01] etc. Le fait que les algorithmes génétiques manipulent une population de plusieurs individus, ceci permet de visiter plusieurs parties de l'espace de recherche, et par conséquent d'augmenter la probabilité de trouver plusieurs solutions Pareto optimales en une seule itération.

L'adaptation des algorithmes génétiques aux problèmes d'optimisation se fait par le bias de l'opérateur de sélection. Nous allons par la suite passer en revue les différents

types de sélection des algorithmes génétiques traitant de l'optimisation multiobjectif. Plusieurs classifications de ces algorithmes ont été proposées par Fonseca et al. [FP95], Zitzeler et al. [ZT98b], [Col02] et Talbi [Tal99]. Ces classifications se basent sur plusieurs notions telles que l'utilisation ou non de la notion de Pareto et l'agrégation des objectifs pour se ramener à un problème mono objectif.

Nous allons par la suite expliquer le principe des méthodes d'optimisation multiobjectif les plus utilisées dans la littérature. Les méthodes citées seront classées selon l'utilisation ou non du principe d'optimalité au sens Pareto.

### 2.4.1 Les méthodes non Pareto

Ces méthodes privilégient, ou bien la recherche dans une seule direction en agrégeant les fonctions objectif dans une seule fonction scalaire, ou bien la recherche dans des directions multiples mais en ignorant les relations de dependence entre les objectifs.

#### 2.4.1.1 La méthode de pondération d'objectifs

Cette méthode basée sur l'agrégation des objectifs a été proposée par Hajela et Lin [HL92]. C'est probablement parmi les méthodes classiques les plus simples. Le principe d'une telle méthode consiste à combiner les $n$ fonctions objectifs $f_i$ en une seule fonction objectif notée $Z$:

$$Z = \sum_{i=1}^{n} w_i f_i(x), \tag{2.7}$$

Les poids $w_i$ sont des réels appartenant à l'intervalle ]0 , 1[ et vérifient la condition suivante: $\sum_{i=1}^{n} = 1$. Pour faciliter le traitement, les poids $w_i$ sont généralement codés à l'intérieur du chromosome. Avec cette méthode, la solution optimale est contrôlée par le vecteur poids $w$. Il est clair d'après l'équation 2.7 que la préférence d'un objectif par rapport aux autres objectifs peut être changée en modifiant la valeur du poids correspondant. Mathématiquement, une solution obtenue par des poids équivalents des objectifs engendre moins de conflits.

En général, chaque objectif est optimisé à part et ensuite les valeurs des autres fonctions objectifs sont calculées. Par conséquent, un vecteur poids est choisi selon le degré d'importance de chaque objectif et ensuite, le problème mono objectif défini avec l'équation 2.7 est utilisé pour trouver la solution désirée. L'avantage de cette méthode est la possibilité d'optimiser chaque objectif à part et donc l'obtention d'une solution Pareto optimale.

#### 2.4.1.2 La méthode basée sur une métrique

Cette méthode a été présentée par Srinivas et Deb dans [SD94]. Avec une telle technique, l'agrégation des différentes fonctions objectif est effectuée en utilisant un vecteur seuil $\bar{y}$ donné en général par un expert. Ce vecteur reflète le niveau demandé à atteindre par chaque fonction objectif (il peut être une borne supérieure ou inférieure dans le cas d'un problème d'ordonnancement). La fonction mono objectif résultant de l'agrégation de toutes les fonctions objectif est la suivante :

$$Z = \left[\sum_{i=1}^{n} |f_i(x) - \bar{y}_i|^r\right]^{1/r} \quad , \quad 1 \leq r \leq \infty \tag{2.8}$$

Généralement, une distance métrique $r = 2$ est utilisée, en fixant $\bar{y}$ aux valeurs optimales des fonctions objectif (en considérant chaque objectif indépendamment des autres). Il est clair que la solution obtenue en résolvant l'équation 2.8 dépend largement du vecteur seuil $\bar{y}$. Un choix arbitraire de ce vecteur peut inhiber la génération de solutions Pareto optimales.

Cette méthode est semblable à celle de pondération des objectifs. La seule différence est que dans cette méthode, la valeur optimale (à la limite une bonne valeur) de chaque fonction objectif est exigée. Par contre, dans la méthode précédente, l'importance relative de chaque objectif est demandée.

#### 2.4.1.3 La méthode Vector Evaluated Genetic Algorithm : VEGA

Schaffer [Sch85] est probablement le premier à avoir appliqué les algorithmes génétiques à l'optimisation multiobjectif. Il a proposé une méthode génétique pour traiter un problème d'optimisation multiobjectif sans avoir à agréger les fonctions objectif en une seule fonction.

Cette méthode, appelée *Vector Evaluated Genetic Algorithm* (VEGA), permet la sélection des individus en considérant chaque fois un seul objectif. Le principe du VEGA consiste à répartir la population initiale d'individus en $n$ groupes ($n$ étant le nombre de fonctions objectif). A chaque groupe est associée une fonction objectif. Cette fonction objectif permet de déterminer l'efficacité d'un individu au sein du groupe. Ensuite, les individus sont mélangés et l'algorithme génétique classique (sélection, croisement et mutation) est appliqué.

Collette [Col02] considère que l'inconvénient de cette méthode est l'obtention, en fin d'optimisation, d'une population constituée d'individus moyens dans tous les objectifs.

Une telle population ne permet pas d'obtenir une surface de compromis bien dessinée. De plus, il a été montré que cette méthode est équivalente à la méthode de pondération des fonctions objectif.

### 2.4.2 Les méthodes dites Pareto

Ces méthodes sont basées sur le calcul des rangs des individus de la population en s'appuyant sur la notion de dominance au sens Pareto. La sélection des individus est effectuée selon les valeurs d'évaluation des individus (qui dépendent directement des rangs).

#### 2.4.2.1 La méthode Multiple Objective Genetic Algorithm : MOGA

Cette méthode est présentée dans plusieurs articles et ouvrages dont [FP95]. Elle se base sur le principe de dominance au sens de Pareto pour déterminer l'efficacité (*fitness*) de chaque individu. Cette efficacité est représentée en fonction d'un rang indiquant le nombre d'individus qui dominent l'individu considéré. A tous les individus non dominés on affecte le rang 1. Les individus dominés auront alors un rang important et par conséquent seront pénalisés et auront moins de chance d'être sélectionnés pour la phase de croisement.

Pour le calcul de l'efficacité, il suffit de classer les individus en fonction de leur rang, ensuite, utiliser une fonction $f(rang)$ (souvent linéaire : voir figure 2.8) pour l'interpolation.

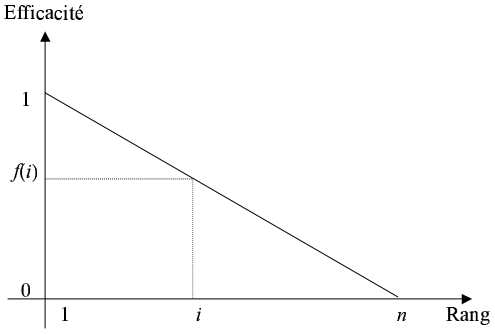



FIG. 2.8 – *Exemple de fonction d'efficacité*

L'avantage de cette méthode est la recherche multidirectionnelle dans l'espace de solutions admissibles. Néanmoins, cette recherche favorise les individus non dominés ou peu dominés en leur affectant une valeur d'efficacité importante.

#### 2.4.2.2 La méthode Non dominated Sorting Genetic Algorithm : NSGA

Cette méthode proposée par Srinivas et Deb [SD94] se base sur le principe de calcul des rangs utilisé par la méthode MOGA. En plus, une méthode de niche est utilisée pour maintenir stables certaines sous-populations de bonnes solutions. NSGA diffère de l'algorithme génétique classique par l'opérateur de sélection. Les opérateurs de croisement et de mutation restent identiques. Avant d'effectuer la sélection, la population d'individus est hiérarchisée en couches selon le principe de dominance au sens de Pareto. Ensuite, les individus non dominés de la population sont identifiés pour constituer la première couche. A cette couche, on affecte une efficacité $F$ très importante. Afin de maintenir la diversité dans la population, les individus non dominés sont par la suite répartis uniformément au sein de cette couche. La répartition est effectuée en affectant à chaque individu une nouvelle valeur d'efficacité en divisant la valeur d'efficacité de la couche considérée par le nombre d'individus autour de l'individu en question. Le nombre d'individus, noté $m_i$ autour d'un individu $i$ est calculé de la manière suivante :

$$m_i = \sum_{j=1}^{k} Sh(d_{ij}) \tag{2.9}$$

$$Sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\sigma_{share}})^2 & si \;\; d_{ij} < \sigma_{share} \\ 0 & si \;\; non \end{cases} \tag{2.10}$$

Ici, $k$ désigne le nombre d'individus dans la couche considérée et $d_{ij}$ est la distance[3] entre l'individu $i$ et l'individu $j$. $\sigma_{share}$ est la distance phénotypique maximale permise entre deux individus pour faire partie de la même niche. Elle permet de définir une zone d'influence pour le calcul de l'efficacité d'un individu.

Tous les individus de la première couche sont par la suite éliminés de la population courante. Le processus précédent de construction de couches et de répartition continue avec le reste de la population. A chaque nouvelle couche on affecte une valeur d'efficacité $F$ décroissante. Ceci privilégie lors de la sélection les individus des premières couches. Par contre, les dernières couches contenant les solutions les moins adaptées au problème, seront moins considérées. Cette propriété permet d'avoir une convergence plus rapide vers la surface de compromis. Le partage permet de maintenir une répartition uniforme sur cette surface. Collette [Col02] considère que la réduction des objectifs en une valeur d'efficacité obtenue en utilisant le classement en fonction du rang, augmente l'efficacité de cette méthode. Néanmoins, cette méthode présente l'inconvénient d'être sensible au choix

3. La plupart des algorithmes utilisent la distance phénotypique appelée aussi distance de Hamming. Cette distance se base sur le nombre de phénotype différents entre deux individus.

de la valeur de $\sigma_{share}$. Pour cette méthode, le nombre de comparaisons effectuées sur une population pour une génération est le même que pour la méthode MOGA. Cependant, un surplus de calculs dû à la repartition apparaît. Ce surplus est proportionnel à $N.(N-1)$ où $N$ est la taille de la population initiale.

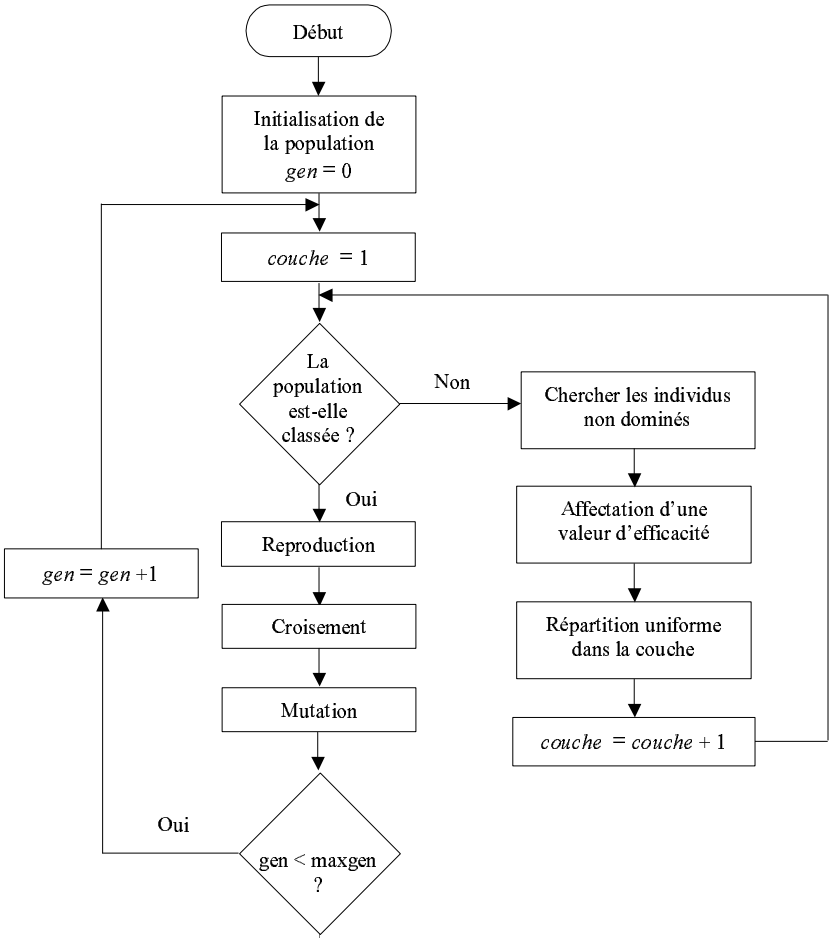



FIG. 2.9 – *Diagramme de NSGA*

#### 2.4.2.3 La méthode Weighted Average Ranking: WAR

Cette méthode s'inspire de la méthode MOGA. la différence réside dans la manière dont la relation de dominance est établie entre deux solutions. En effet, la méthode WAR consiste à calculer le rang de chaque individu de la population par rapport aux différents objectifs séparément. Le rang d'un individu $x$ est calculé selon l'équation 2.11. Ensuite, l'efficacité des individus sera calculée de la même manière que pour la méthode MOGA.

$$rang(x) = \sum_{i=1}^{n} rang_{f_i}(x) \tag{2.11}$$

#### 2.4.2.4 La méthode élitiste

La méthode élitiste consiste à maintenir une sorte de population archive qui contiendra les meilleurs solutions non dominées rencontrées au long de la recherche. Cette population participera aux étapes de sélection et de reproduction. La population élitiste peut être initialisée comme elle peut être construite lors de la recherche. L'initialisation est effectuée en optimisant un seul objectif tout en gardant les autres fixes. De cette façon, les solutions générées par l'algorithme génétique seront reparties entre ces solutions extrêmes. En effet, ces solutions non dominées de génération en génération, transmettront leurs caractéristiques aux autres individus. Parmi les méthodes d'optimisation multiobjectif élitiste les plus récentes, nous pouvons citer celle de Zitzler et Thiele [ZT98a].

## 2.5 Les algorithmes génétiques et l'ordonnancement

Les algorithmes génétiques sont de plus en plus utilisés pour résoudre des problèmes d'optimisation combinatoire et en particulier pour résoudre des problèmes d'ordonnancement [HM94]. Les problèmes d'ordonnancement traités par les algorithmes génétiques sont divers. Ils concernent à la fois l'ordonnancement de la production et l'ordonnancement de la maintenance [BS98]. La plupart des problèmes d'ordonnancement traités s'intéressent à l'optimisation d'un seul objectif (le makespan, le retard max, la somme des retards, le nombre de tâches en retard, etc). Avec l'apparition des algorithmes génétiques multiobjectif, le cas d'optimisation de plusieurs critères est considéré [MRTD01], [PRJ01b].

Une classification de ces problèmes selon une complexité croissante est introduite par Portmann et al. dans [PV01]. Les premiers problèmes considérés sont des problèmes à une machine pour lesquels le codage est un codage de permutation. Le cas de l'ordonnancement d'atelier de type job shop est ensuite considéré afin de mettre en évidence le fait que les codages de permutation ne sont pas satisfaisants pour ce type de problème et qu'il faut les adapter ou changer de type de codage. Le cas des machines parallèles est introduit en considérant l'ordonnancement des ateliers de type flow shop hybride. L'ordonnancement des ateliers de type open shop est également étudié en utilisant les algorithmes génétiques [FRC93]. Nos travaux abordent l'ordonnancement dans un atelier de type job shop en utilisant les algorithmes génétiques.

### 2.5.1 Les algorithmes génétiques appliqués à l'ordonnancement du job shop

Davis [Dav85] est le premier à avoir démontré la possibilité d'utiliser un algorithme génétique pour résoudre le job shop, et depuis, un nombre important d'articles et d'ouvrages utilisant cette technique artificielle a vu le jour. La majeure différence entre ces algorithmes réside dans la représentation génétique des chromosomes. Deux types de codages peuvent être distingués. Le premier utilise un codage direct dans lequel toute l'information est présente dans le chromosome. Le deuxième déporte la difficulté du problème d'ordonnancement à l'extérieur du codage, en utilisant un codage indirect.

#### 2.5.1.1 Codage direct

Dans un codage direct, le chromosome doit représenter une solution complète du problème traité. Ce dernier doit contenir toutes les informations utiles à la création de l'ordonnancement. Toutes les dates de début des tâches ainsi que leur affectation doivent être représentées dans le gène pour chaque occurrence. De plus, les informations dans le gène doivent tenir compte des contraintes de précédence. Dans ce cas, les opérateurs de croisement et de mutation sont adaptés avec ce type de codage pour ne pas générer des solutions inadmissibles (ne satisfaisant pas au moins une contrainte). Pour éviter un tel problème, on peut ajouter au sein de l'algorithme génétique un mécanisme de détection de solutions inadmissibles pour effectuer un traitement de reparation. La difficulté d'un tel codage réside dans la représentation lourde des données au sein des chromosomes ainsi que dans la difficulté d'adapter les opérateurs génétiques.

Le codage le plus simple pour résoudre le problème de job shop est celui présenté par Yamada et Nakano [YN92]. Il consiste à insérer dans le chromosome les dates de fin des tâches ainsi qu'un algorithme pour réaliser des ordonnancements actifs. Le tableau 2.3 montre un exemple de chromosome utilisant ce type de codage réalisé sur le problème du job shop de taille $3 \times 3$ cité dans le tableau 1.3 du chapitre précédent. Le symbol $o_{ij}(p_{ij})$ représente le fait que la $j^{me}$ opération du $i^{me}$ job se termine à la date $p_{ij}$.

| Les jobs | $o_{ij}(p_{ij})$ | | |
|---|---|---|---|
| $J_1$ | $o_{11}(3)$ | $o_{12}(6)$ | $o_{13}(11)$ |
| $J_2$ | $o_{21}(5)$ | $o_{22}(8)$ | $o_{23}(12)$ |
| $J_3$ | $o_{31}(3)$ | $o_{32}(7)$ | $o_{33}(12)$ |

TAB. 2.3 – *Codage direct de Yamada et Nakano*

#### 2.5.1.2 Codage indirect

Ce codage est le plus utilisé pour décrire les chromosomes d'un problème de job shop. En effet, il n'y a pas besoin d'introduire toutes les informations décrivant le problème au sein du chromosome. Au contraire, il suffit de générer des chromosomes dont les gènes contiennent des listes de priorité ou des heuristiques. Il est nécessaire dans ce cas d'utiliser un ordonnanceur pour transformer tout chromosome en une solution effective. Ce dernier s'appuie sur les contenus du chromosome pour déterminer les dates de début des opérations et par la suite, de déduire la valeur du critère étudié. Il existe deux types d'ordonnanceur :

- le premier,*First In First Out* (appelé aussi FIFO) affecte les opérations dans leurs ordres d'apparition dans le chromosome. Aucun retard n'est permis quand à l'affectation des opérations : dès que la ressource est disponible, la première opération sur la liste d'attente est prise en compte. Ce type d'ordonnanceur génère des ordonnancements semi actifs ;
- le deuxième, *left-shift* permet d'affecter les opérations sur les ressources le plus tôt possible. Comme son nom l'indique, cet ordonnanceur décale chaque opération à gauche sans faire tarder aucune autre opération. Cette manière d'ajustement permet parfois de changer le séquencement des opérations donné par l'algorithme génétique. Les ordonnancements obtenus après ajustement sont actifs. L'utilisation de cet ordonnanceur perfectionne les résultats de l'algorithme génétique.

Les codages indirects les plus utilisés dans la littérature sont cités dans le tableau 2.4. Une étude présentée par Ponnambalam et al. [PRJ01a] dans laquelle les auteurs ont évalué et comparé les quatres premiers types de codage cités dans le même tableau. Nous allons par la suite développer ces quatre opérateurs à travers le même exemple utilisé auparavant.

- **codage basé sur les opérations** : cette représentation génétique utilise des entiers pour coder les ordonnancements au sein des chromosomes. Il existe deux versions différentes de ce codage. La première, consiste à désigner les opérations différemment les unes des autres par des entiers (comme le codage du problème du voyageur du commerce où chaque ville reçoit un symbol différent). Malheureusement, ce codage peut générer des chromosomes amenant à des solutions inadmissibles dont les contraintes de précédence ne sont pas respectées. Pour éviter cet inconvénient, Gen et al [GTK94] ont proposé un nouveau codage des opérations. Les auteurs ont désigné les opérations d'un même job par le même symbole et

| Type de codage | Référence |
|---|---|
| Codage basé sur des listes de préférence | Croce et al. [?] |
| Codage basé sur des règles de priorité | Dorndorf et Pesch [DP95] |
| Codage basé sur les opérations | Gen et al. [GTK94] |
| Codage basé sur les opérations | Fang et al. [FRC93] |
| Codage basé sur les jobs | Holsapple [HVPZ93] |
| Codage basé sur le graphe disjonctif | Tamaki et Nishikawa [TN92] |
| Codage basé sur les relations entre les paires des jobs | Nakano et Yamada[NY91] |
| Codage basé sur des listes de préférence | Davis [Dav85] |

TAB. 2.4 – *État de l'art des codages génétiques du job shop*

ensuite, ces mêmes opérations sont interprétées selon leur occurrence d'apparition dans le chromosome. Chaque job apparaît exactement $m$ fois dans la séquence ($m$ étant le nombre de machines). Il est évident que tout chromosome généré utilisant ce type de codage est valide. Pour transformer le chromosome en ordonnancement, il suffit d'extraire à partir du chromosome, l'ordre de passage des opérations sur les machines.

A titre d'exemple, considérons le même exemple présenté au tableau 1.3 du chapitre 1. Supposons qu'un chromosome généré par l'algorithme génétique est comme suit : (2 3 3 1 2 1 3 2 1). L'entier 1 désigne les opérations du job $J_1$, l'entier 2 désigne les opérations du job $J_2$ et l'entier 3 désigne les opérations du job $J_3$. Chaque job apparaît exactement trois fois parce qu'il contient trois opérations. Par exemple, le job $J_1$ possède trois 1 dans la séquence. Le premier 1 désigne la première opération du $J_1$, le second 1 désigne la deuxième opération du $J_1$ et le troisième 1 désigne la troisième opération du même job. L'ordre de passage des opérations sur les machines est donné par la figure 2.10 ;

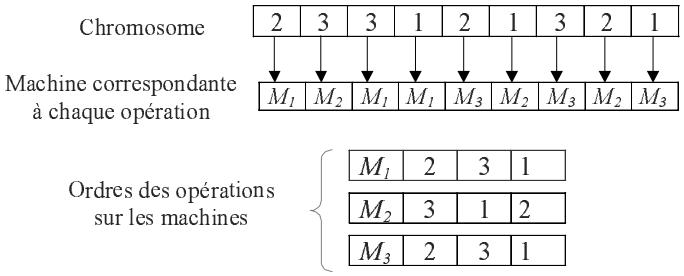



FIG. 2.10 – *Codage basé sur les opérations : [GTK94]*

– **codage basé sur les jobs** : un chromosome est formé par une liste de $n$ entiers représentant les numéros des différents jobs. Pour obtenir un ordonnancement à partir du chromosome, il suffit d'affecter aux machines les opérations de chaque job séparément et selon l'ordre d'apparition du job dans le chromosome.

  Cette représentation génétique permet de n'obtenir que des chromosomes valides, ce qui évite un traitement supplémentaire pour supprimer les solutions non admissibles (ou parfois d'effectuer des opérations de réparation). Comparé au codage précédent, ce codage fournit des chromosomes de taille très réduite. Ceci ne permet pas une exploration exhaustive de l'espace des ordonnancements admissibles.

  Pour illustrer le fonctionnement de ce codage, prenons toujours le même exemple que précédemment. Un chromosome peut être sous la forme suivante : (3 2 1). Dans ce cas on exécute toutes les opérations du job $J_3$ (le plutôt possible), ensuite les opérations du job $J_2$ et enfin les opérations du job $J_1$. La figure 2.11 montre avec un diagramme de Gantt les différentes étapes de transformation de ce chromosome en ordonnancement ;

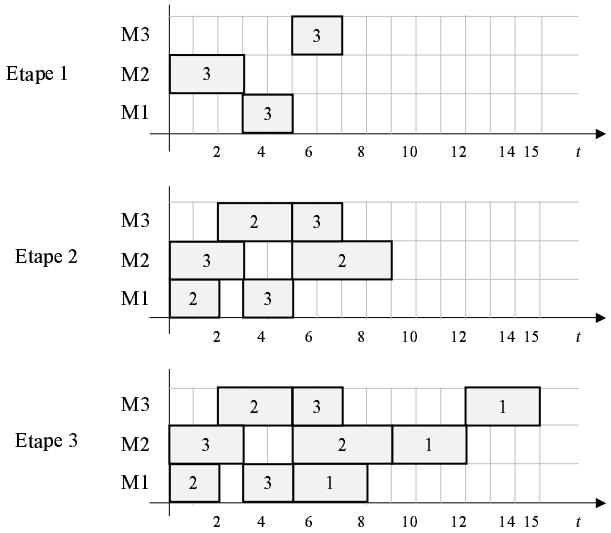



FIG. 2.11 – *Codage basé sur les jobs : [HVPZ93]*

– **codage basé sur des listes de préférence** : ce codage a été utilisé pour la première fois par Davis en 1985 [Dav85] pour l'étude d'un problème d'ordonnancement. Ensuite, Falkenauer et Bouffouix [FB91] l'ont adapté en 1991 au problème du job shop avec des dates au plus tôt et des dates au plus tard. La première utilisation de ce codage pour le job shop revient en 1995 par Croce et al. [CTV95].

Pour un job shop constitué de $n$ jobs et $m$ machines, le codage basé sur des listes de préférence est comme suit : un chromosome est subdivisé en $m$ sous-chaînes de taille identique égale à $n$. Chaque sous-chaîne représente une machine particulière avec les opérations devant s'exécuter sur elle. L'ordre d'apparition des opérations dans chaque sous-chaîne ne décrit pas forcément le séquencement des opérations sur la machine, mais plutôt un ordre de préférence. Nous obtenons ainsi à partir d'un chromosome, une liste de préférence de passage des opérations pour chaque machine. Pour obtenir un ordonnancement effectif, on peut utiliser une simulation qui analysera l'état de la file d'attente devant chaque machine et affectera les opérations en utilisant (si nécessaire) les listes de préférence (i.e l'opération qui figure en tête de la liste de préférence est sélectionnée). Ce type de codage ne génère que des chromosomes valides.

Nous allons maintenant illustrer le fonctionnement de ce codage à travers le même exemple étudié précédemment. Considérons le chromosome : [(3 1 2) (1 2 3) (2 3 1)]. Le premier gène (3 1 2) est la liste de préférence de la machine $M_1$, le deuxième gène (1 2 3) est la liste de préférence de la machine $M_2$ et le troisième gène (2 3 1) est la liste de préférence de la machine $M_3$. L'affectation des opérations sur les machines se fait à chaque fois en déterminant la liste des opérations candidates (celles qui figurent en tête de liste et respectant les contraintes de précédence). Si à un moment donnée, aucune opération tête de liste n'est ordonnançable, alors on passe aux opérations suivantes de chaque liste. Les différentes étapes d'affectation des opérations selon le chromosome ci-dessus sont montrées dans le tableau 2.5.

| Étapes | Opérations ordonnançables : $O_{or}$ |
|---|---|
| Étape 1 | $O_{or} = \{\emptyset\}$ |
| Étape 2 | $O_{or} = \{J_1 \text{ sur } M_1\}$ |
| Étape 3 | $O_{or} = \{J_1 \text{ sur } M_2\}$ |
| Étape 4 | $O_{or} = \{\emptyset\}$ |
| Étape 5 | $O_{or} = \{J_2 \text{ sur } M_1, J_3 \text{ sur } M_2, J_1 \text{ sur } M_3\}$ |
| Étape 6 | $O_{or} = \{J_3 \text{ sur } M_1, J_2 \text{ sur } M_3\}$ |
| Étape 7 | $O_{or} = \{J_2 \text{ sur } M_2, J_3 \text{ sur } M_3\}$ |

TAB. 2.5 – *Codage basé sur des listes de préférence : affectation des opérations*

– **codage basé sur des règles de priorité** : comme c'est indiqué dans le premier chapitre, les règles de priorité sont largement utilisées pour résoudre le problème du job shop. Cependant ces règles sont utilisées indépendamment les unes des autres.

Il revient à Dorndorf et Pesh en 1995 [DP95] de proposer une méthode combinant plusieurs règles d'affectation pour l'exécution des opérations. En effet, ces auteurs ont développé un nouveau codage génétique basé sur une liste de règles d'ordonnancement (voir tableau2.6). Un chromosome est ainsi formé par une séquence de codes représentant ces règles. Le rôle de l'algorithme génétique est de chercher la bonne séquence de règles ramenant à un bon ordonnancement.

Pour un problème de job shop constitué de $n$ jobs et $m$ machines, un chromosome codé selon les règles de priorité est une séquence de $n \times m$ éléments $(p_1, p_2, ..., p_{nm})$. Chaque élément $p_i$ représente une règle d'affectation (une règle peut apparaître plusieurs fois) et permet de résoudre le conflit pouvant exister à la $i^{me}$ itération de l'affectation des opérations. Si plusieurs opérations ont la même valeur du critère envers la règle, alors on choisie aléatoirement l'opération à exécuter. Le principe d'affectation des opérations se basant sur ce type de codage est détaillé par l'algorithme 2.1.

Pour illustrer le fonctionnement du codage basé sur les règles de priorité, considérons toujours le même exemple de job shop de taille $3 \times 3$. Supposons qu'un chromosome généré par l'algorithme génétique soit le suivant : (1, 0, 3, 7, 5, 2, 4, 6, 3). Chaque élément du chromosome représente le code d'une règle d'affectation permettant de régler un éventuel conflit (voir tableau 2.6).

- **itération 1** :
  - $S_1 = \{J_1, J_2, J_3\}$ ;
  - $d_1^* = min\{3, 2, 3\} = 2$ ;
  - $M^* = M_1$ ;
  - $C_1 = \{J_1, J_2\}$ ;
  - le premier élément du chromosome étant le chiffre 1 représentant la règle LOT. Utilisant cette règle pour choisir parmi les opérations en conflit celle devant s'exécuter en premier (soit la première opération du job $J_1$).
- **itération 2** :
  - $S_2 = \{J_1, J_2, J_3\}$ ;
  - $d_2^* = min\{3, 2, 3\} = 2$ ;
  - $M^* = M_1$ ;
  - $C_2 = \{J_2\}$ ;

**Algorithme 2.1** Ordonnancement basé sur les règles de priorité

**Entrée :** $OP_t$ : *ordonnancement partiel contenant t opérations* ;

$S_t$ : *ensemble d'opérations ordonnançables à la* $t^{me}$ *itération* ;

$r_i$ : *date de début au plus tôt de l'opération* $i \in S_t$ ;

$d_i$ : *date de fin au plus tôt de l'opération* $i \in S_t$ ;

$C_i$ : *ensemble d'opérations en conflit à l'itération* $i$ ;

$(p_1, p_2, ..., p_{nm})$ : *un chromosome généré par l'algorithme génétique.*

**Début**

$t \leftarrow 1$

$OP_1$ : ordonnancement partiel vide

$S_1 \leftarrow$ {opérations sans prédécesseurs}

**Tant que** il existe des opérations non affectées **faire**

- déterminer $d_t^* = \min_{i \in S_t}\{d_i\}$ ;
- déterminer la machine $M^*$ sur laquelle l'opération devant commencer à $d_t^*$. S'il existe plusieurs machines, choisir une aléatoirement ;
- déterminer l'ensemble $C_t = \{i \in S_t \ / \ r_i < d_t^* \ et \ M(i) = M^*\}$ ;
- soit $i^* \in C_t$ l'opération sélectionnée en utilisant la règle $p_t$. Si plusieurs opérations sont sélectionnées, choisir une aléatoirement ;
- $OP_{t+1} \leftarrow OP_t \bigcup \{i^*\}$ ;
- $t \leftarrow t + 1$.

**Fin tant que**

**Fin**

**Sortie :** l'ordonnancement $OP_{n \times m}$

– on affecte la première opération du job $J_2$ sans avoir recours à aucune règle (pas de conflit).

– **itération 3** :

– $S_3 = \{J_1, J_2, J_3\}$ ;

– $d_3^* = min\{3, 3, 3\} = 3$ ;

– $M^* \in \{M_2, M_3\}$ ;

– choisir aléatoirement une machine (soit $M^* = M_2$) ;

– $C_3 = \{J_1, J_3\}$ ;

– le troisième élément du chromosome étant le chiffre 3 représentant la règle LPT. L'opération sélectionnée étant la deuxième opération du job $J_1$.

– continuer ces itérations jusqu'à l'obtention d'un ordonnancement complet à partir du chromosome donné. L'ordonnancement correspondant est donné par la figure 2.12.

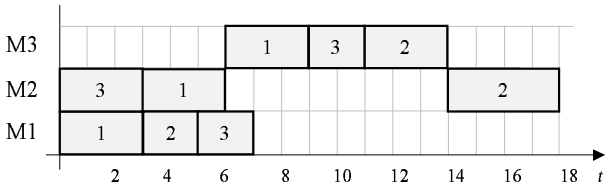



FIG. 2.12 – *Ordonnancement basé sur les règles de priorité*

| Code | Règle de priorité | Description |
|---|---|---|
| 0 | SOT (Shortest Operation Time) | L'opération ayant le temps opératoire le plus court sur la machine considérée |
| 1 | LOT (Longest Operation Time) | L'opération ayant le temps opératoire le plus long sur la machine considérée |
| 2 | SPT (Shortest Processing Time) | Le job ayant la somme des temps opératoires la plus courte |
| 3 | LPT (Longest Processing Time) | Le job ayant la somme des temps opératoires la plus longue |
| 4 | SNRO (Smallest Number of Remaining Operations) | L'opération ayant le petit nombre de successeurs |
| 5 | LNRO (Largest Number of Remaining Operations) | L'opération ayant le grand nombre de successeurs |
| 6 | LRPT (Longest Remaining Processing Time) | L'opération ayant le temps restant du job le plus long |
| 7 | SRPT (Shortest Remaining Processing Time) | L'opération ayant le temps restant du job le plus court |

TAB. 2.6 – *Exemple de règles de priorité*

Une évaluation de ces quatre types de codage a travers un ensemble de benchmarks est réalisée par Ponnambalam et al. [PRJ01a]. Selon le critère d'optimisation étudié ($C_{max}$), le codage basé sur des listes de préférence permet d'avoir les meilleurs résultats. Par contre, il est le moins rapide au niveau du temps CPU. Le codage le moins performant au niveau du temps CPU et du critère d'optimisation est le codage basé sur les règles de priorité. En effet, ce type de codage ne permet pas de régler les conflits entre les opérations ayant le même classement par l'une des règles d'affectation. Il est probablement plus pratique de constituer les gènes du chromosome par plusieurs règles. Si plusieurs opérations en conflit ont le même classement par la première règle, on utilise la deuxième règle et ainsi de suite.

#### 2.5.1.3 Les opérateurs génétiques

La limitation du codage binaire a fait que les algorithmes génétiques utilisés pour résoudre le problème du job shop se basent tous sur des codages réels. Ceci nécessite des opérateurs génétiques bien spécifiques pour la production de chromosomes valides. Ainsi, selon le type de codage, on peut adapter l'un des opérateurs de croisement et de mutation définis dans la première partie de ce chapitre.

Nous allons décrire par la suite deux autres opérateurs de croisement créés spécialement pour le job shop et basés sur le codage des opérations. Le premier, appelé GOX (Generalized Order Crossover), a été proposé par Bierwirth en 1995 [Bie95]. Il permet de respecter l'ordre relatif des opérations. Le principe de ce croisement est le suivant : choisir aléatoirement une sous-chaîne du premier parent, ensuite, supprimer tous ces éléments du deuxième parent en respectant leurs indices relatifs. Pour obtenir un fils, on complète les éléments du deuxième parent en insérant la sous chaîne dans la position de son premier élément avant la procédure de suppression. A titre d'exemple, considérons les deux chromosomes parents $p_1$ et $p_2$ suivants :

$$p_1 = 3\ 2\ \underline{2\ 2\ 3\ 1}\ 1\ 1\ 3$$
$$p_2 = \mathbf{1}\ 1\ 3\ 2\ \underline{\mathbf{2}}\ 1\ \mathbf{2}\ \mathbf{3}\ 3$$

La sous-chaîne constituée des éléments soulignés est sélectionnée du chromosome $p_1$. Elle est composée de deux opérations du job $J_2$ (la deuxième et la troisième), d'une opération du job $J_3$ (la deuxième opération) et d'une opération du job $J_1$ (la première). Ces opérations sont supprimées du parent $p_2$ (les éléments en gras). La sous-chaîne est ensuite insérée au parent $p_2$ dans la position de la deuxième opération du deuxième job (la position de l'opération en gras et soulignée). Le chromosome fils résultant de ce croisement est le suivant :

$$f = 1\ 3\ 2\ \underline{2\ 2\ 3\ 1}\ 1\ 3$$

On peut appliquer ce type de croisement sur les deux parents en même temps afin de produire deux chromosomes fils.

Mattfeld [Mat96] propose de modifier le croisement précédent pour tenir compte des positions absolues des opérations plutôt que de leurs positions relatives. La différence entre ce nouveau opérateur de croisement appelé GPX (Generalized Position Crossover) et le croisement GOX réside dans la dernière étape d'insertion de la sous-chaîne sélectionnée du premier parent. En effet la sous-chaîne d'opérations sélectionnée est insérée dans le deuxième parent dans la même position où elle était dans le premier parent.

Si on considère les mêmes chromosomes précédents, le croisement GPX produit le fils suivant : $f = 1\ 3\ \underline{2\ 2\ 3\ 1}\ 2\ 1\ 3$.

#### 2.5.1.4 Les valeurs des paramètres génétiques

dans le cadre de l'ordonnancement du job shop avec les algorithmes génétiques, Mattfeld [Mat96] a suggéré un taux de croisement $p_c = 0.6$ et un taux de mutation $p_m = 0.03$. Le taux de mutation doit tenir compte des mutations implicites produites par le croisement. En effet, les opérateurs de croisement tels que GOX et GPX introduisent une quantité considérable de mutations implicites. Par conséquent, un taux de mutation relativement petit est suffisant pour maintenir la diversité de population.

La taille de la population est un paramètre crucial pour une bonne performance de l'algorithme génétique. Si cette taille est trop petite, alors la convergence de l'algorithme est souvent prématurée. Au contraire, si la taille de la population est trop grande, on obtient en général une saturation de la mémoire utilisée. Il y a deux tendances pour fixer la taille de la population. La première consiste à travailler avec une population de petite taille et exécuter l'algorithme plusieurs fois. La deuxième consiste à utiliser une population de grande taille et de se limiter au niveau de nombre d'exécutions.

Concernant le critère d'arrêt, la littérature propose trois types différents. Le premier consiste à fixer un nombre d'itérations de l'algorithme (le nombre de cycles génétiques). Le deuxième permet d'arrêter la procédure de recherche si au bout de quelques itérations aucune amélioration n'est produite. Le troisième critère d'arrêt est plus flexible, il permet d'effectuer divers tests sur la population (par exemple, calculer l'entropie[4] de la population).

## 2.6 Conclusion

Ce chapitre a permis d'introduire les principes de base des algorithmes génétiques. Ces techniques stochastiques s'inspirent des phénomènes réels tels que l'évolution et la sélection pour explorer et exploiter des espaces de recherche de grande taille.

Les algorithmes génétiques résolvent des problèmes $\mathcal{NP}$-*difficiles*. En particulier, les problèmes d'ordonnancement tel que le job shop. En effet, l'application des algorithmes génétiques pour résoudre le job shop a connu et connaît encore un grand essor.

4. L'entropie est une mesure numérique utilisée pour calculer la diversité génotypique de la population [Mat96]. Pour le job shop, Mattfeld a proposé de calculer l'entropie en fonction de la fréquence des arcs dans le chemin Hamiltonien des machines.

De nombreux travaux sur le codage et sur les opérateurs génétiques ont déjà permis d'obtenir des résultats encourageants.

Le choix des paramètres génétiques tels que la taille de la population d'individus, les probabilités de croisement et de mutation et le critère d'arrêt, demeure le souci que l'on rencontre lorsqu'on veut utiliser les algorithmes génétiques. Souvent il faut faire appel à une expérimentation importante pour trouver les valeurs des paramètres les plus performants. De plus, le codage des solutions est parfois difficile à réaliser.

Nous avons montré dans ce chapitre l'adaptation des algorithmes génétiques au traitement d'un problème d'optimisation multiobjectif. La manipulation et la génération d'une population de plusieurs individus, permettent de visiter plusieurs parties de l'espace de recherche en fonction d'objectifs différents, ce qui fait augmenter la probabilité de trouver plusieurs solutions Pareto optimales en une seule itération.

Les résultats obtenus par les algorithmes génétiques sont probants, et nous prenons appui sur les solutions obtenues, pour proposer une méthode de résolution du job shop au chapitre suivant. L'étude de la population de bonnes solutions générées par les algorithmes génétiques, par des méthodes analysant les données comme le processus de l'ECD (Extraction de Connaissance à partir des Données), nous permettra d'expliquer les solutions obtenues, par des caractéristiques du problème de job shop. Cette étude permettra d'extraire des règles d'ordonnancement ou de connaissance dans un atelier de job shop. Ces règles d'ordonnancement constitueront une heuristique ayant les mêmes performances que l'algorithme génétique dans la résolution du même type de problèmes, tout en évitant la phase délicate de réglage des paramètres génétiques.

# Chapitre 3

# L'apprentissage des solutions génétiques pour la résolution du job shop

---

*Nous étudions dans ce chapitre l'ordonnancement dans un atelier de type job shop. Cette étude prend en compte uniquement les contraintes de production en supposant que les machines sont disponibles et ne sont pas sujettes à d'éventuelles pannes. Nous proposons une démarche particulière qui consiste à étudier un ensemble de bonnes solutions du problème, et à faire le lien entre cet espace solution et l'espace de définition du problème. Pour ce faire, nous extrayons des règles d'ordonnancement, en adaptant un processus d'Extraction de Connaissance à partir des Données. Ce processus composé de quatre étapes, a permis d'élaborer une heuristique en utilisant des règles d'ordonnancement générées à partir d'un ensemble de bonnes solutions. Ces solutions fournies par un algorithme génétique, concernent des problèmes ayant six jobs et six machines. Nous validerons cette heuristique en comparant ses résultats obtenus sur des benchmarks de différentes tailles, à d'autres méthodes approchées.*

*La même démarche d'ECD à l'exception de l'étape de post-traitement sera de nouveau appliquée à un problème de job shop de taille plus grande (quinze jobs et cinq machines). Les règles trouvées seront directement utilisées pour l'affectation des opérations sur les machines. Pour valider ces règles d'ordonnancement, nous sélectionnons un ensemble de benchmarks tous de même taille* ($15 \times 5$).

---

## 3.1 Introduction

Dans ce chapitre, nous nous intéressons au problème du job shop sans tenir compte des contraintes de disponibilité des machines. Ces contraintes, seront prises en compte dans le chapitre suivant. Le job shop fréquemment rencontré en milieu industriel dit de production de petites et moyennes séries de plusieurs types de produits. On s'intéresse au problème d'ordonnancement noté $J//C_{max}$ pour minimiser le makespan ($C_{max}$).

Nous proposons dans ce chapitre une approche de résolution du job shop basée sur l'exploitation des solutions issues d'un algorithme génétique. L'étude de ces solutions, permettra d'extraire des règles d'ordonnancement du problème de job shop, qui constitueront la connaissance obtenue à l'issue de notre travail. Nous resituons notre démarche dans le processus d'Extraction de Connaissance à partir de Données, que l'on décrira en premier lieu au premier paragraphe de ce chapitre. Puis nous adapterons cette démarche au problème de résolution du job shop, qui sera présentée phase par phase. Ce chapitre se termine par la validation de l'approche de résolution en utilisant des problèmes de job shop types.

## 3.2 Extraction de connaissance à partir des données

L'Extraction de Connaissance à partir de Données (ECD) ou Knowledge Discovery in Databases (KDD) est un processus non trivial d'identification de structures inconnues, valides et potentiellement exploitables dans les bases de données [FPSS96]. Son but est donc l'extraction d'information utile contenue dans les bases de données, à travers la mise en exergue des relations dominantes entre les exemples qui les composent. Elle se réfère à une démarche complète d'exploitation des données que l'on peut résumer en quatre phases distinctes [LM98] et [FPSS96] dont l'enchaînement est présenté dans la figure 3.1. Le déroulement de ces phases n'est pas nécessairement séquentiel, des effets de retour peuvent être introduits lors de l'enchaînement du processus.

### 3.2.1 Les étapes de l'ECD

On s'intéresse à la découverte de connaissances, on a donc besoin de techniques d'exploration de données pour trouver des formes intéressantes qui aident à expliciter une information auparavant cachée dans les données. Kodratoff [Kod99] considère que les problèmes fondamentaux de la découverte de connaissances sont la représentation des connaissances, la sélection des attributs, la prise en compte des données manquantes, bruitées et rares et la découverte de formes "intéressantes", "utiles".

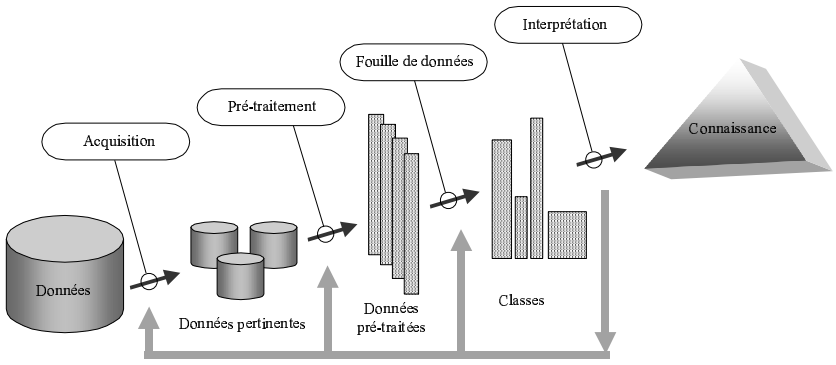



FIG. 3.1 – *Extraction de Connaissance à partir des Données*

Le processus de l'ECD n'est pas une exploitation pure et simple, mais un processus compliqué. On peut dire que l'extraction de connaissances se fait en quatre étapes principales. Nous décrivons par la suite l'objectif de chacune d'elles.

- **l'acquisition des données** (Data Warehousing) : réalise la collecte des données à partir des multiples sources d'information et sous différents formats ;
- **le pré-traitement des données** (Preprocessing) : consiste à nettoyer les données brutes afin de supprimer les bruits et de garder que les données pertinentes. Ensuite, des transformations des données sous des formes telles qu'on puisse les analyser par des algorithmes statistiques sont réalisées. En ECD, cette étape, quoique peu spectaculaire, est cruciale car elle souligne les traits importants et élimine les traits secondaires ;
- **la fouille de données** (Data Mining) : est l'étape d'extraction des structures sous-jacentes des données [FPSS96]. Il s'agit d'un processus d'exploration dans de grandes bases de données qui intègre à la fois le choix de la modélisation adéquate et de la méthode à utiliser ainsi que son application à la découverte des relations entre les données jusqu'alors inconnues. La fouille de données correspond donc à l'ensemble des techniques et des méthodes qui à partir de données permettent d'obtenir de la *connaissance* exploitable. Cette phase permet la création de modèles explicatifs et/ou prédictifs, en général sous forme de règles de décision en format analytique "Si alors" ou sous forme d'arbre ;
- **le post-traitement** (postprocessing) : consiste d'une part à évaluer et à interpréter les conclusions émises afin de s'assurer qu'elles correspondent à des mécanismes réels, et d'autre part à mettre ces conclusions sous forme intelligible et réutilisable.

### 3.2.2 Adaptation de l'ECD à l'ordonnancement du job shop

Dans notre démarche décrite par la figure 3.2, nous avons choisi les méthodes d'apprentissage pour leur intelligibilité, dont un état de l'art sera présenté au paragraphe suivant. La méthode d'acquisition des données n'est pas conventionnelle, et sera remplacée par un algorithme génétique résolvant le problème de job shop considéré.

Pour préparer les données à l'étape de fouille de données, le pré-traitement a consisté dans notre cas, aux choix des attributs (variables) caractérisant le problème de job shop, indépendamment de sa solution. L'espace de définition de ces caractéristiques étant en général numérique, nous les avons discrétisées dans un souci de généralisation. Un état de l'art des différents algorithmes de discrétisation est présenté en section 3.2.5.

Après ces deux premières étapes, la première préparant l'espace solution du problème, en proposant une variable dite endogène et la deuxième s'intéressant à l'espace caractérisant le problème, que l'on appellera variables exogènes, l'étape de fouille de données, permettra d'expliquer la variable endogène par les variables exogènes. Plusieurs techniques existantes dans la littérature permettent la fouille de données en apprentissage automatique. Un état de l'art de ces techniques sera présenté au paragraphe 3.2.4. Lors de cette phase, nous avons réalisé un apprentissage à partir des solutions fournies par l'algorithme génétique.

Finalement, l'étape de post-traitement est utilisée pour interpréter les résultats fournis par l'apprentissage et élaborer une heuristique pour la résolution du job shop.

### 3.2.3 Contexte de travail : l'apprentissage automatique

L'apprentissage automatique (machine learning) a suscité un engouement particulièrement développé au cours des dernières décennies. Ce domaine particulier de l'I.A. consiste à reproduire la capacité de l'homme à apprendre, c'est-à-dire à se servir de l'expérience passée, et à adapter son comportement afin d'accomplir au mieux une tâche similaire dans l'avenir.

#### 3.2.3.1 L'apprentissage inductif

Par opposition à la déduction, les données initiales de l'induction sont des faits spécifiques plutôt que des axiomes généraux. Le but de l'inférence est de formuler des énoncés généraux plausibles qui expliquent les faits donnés et sont capables de prédire de nouveaux faits.

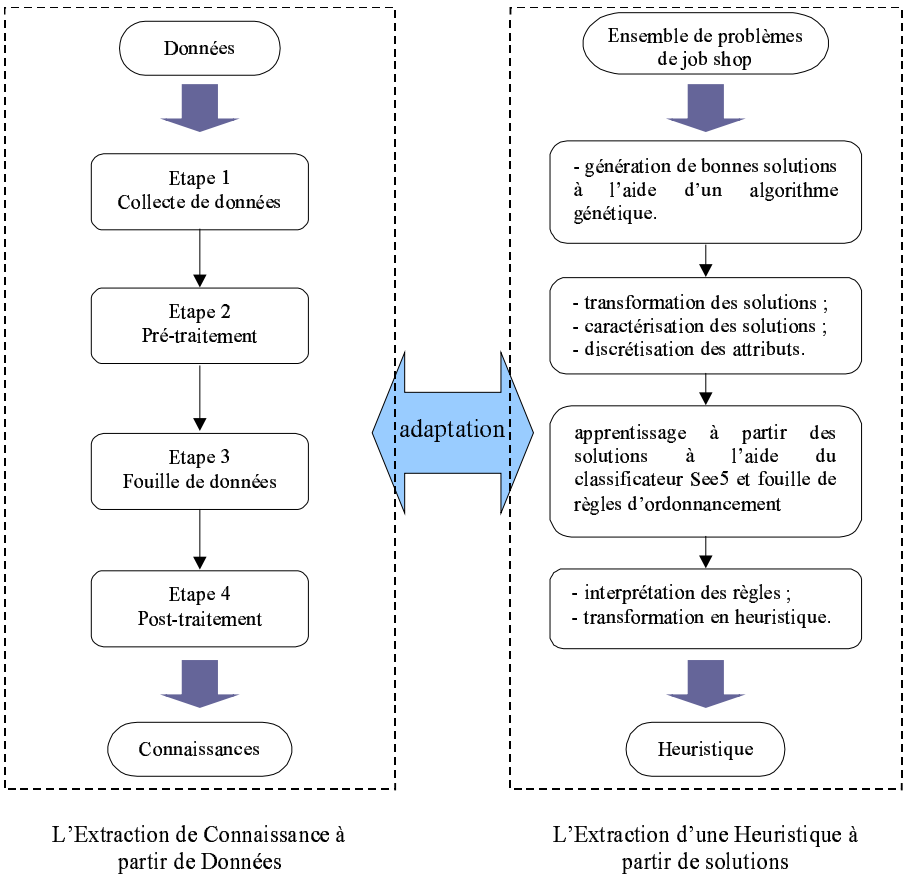



FIG. 3.2 – *Adaptation du processus de l'ECD à la résolution du job shop*

En d'autres termes, l'inférence inductive essaie de dériver une description complète et correcte d'un phénomène donné à partir d'observations spécifiques. L'apprentissage inductif est un processus d'acquisition de connaissances par application d'inférences inductives sur des faits donnés par un professeur ou fournis par l'environnement. Son intérêt réside dans sa capacité à réaliser des généralisations à partir de quelques faits ou à découvrir des schémas dans des ensembles d'observations. Il cherche donc à extraire, à partir de données brutes, l'information utile dans un but décisionnel et/ou prévisionnel. L'apprentissage se déroule dans un contexte soit supervisé soit non supervisé.

#### 3.2.3.2 L'apprentissage inductif supervisé/non supervisé

Il existe principalement deux méthodes permettant à une machine d'acquérir des connaissances. La première, connue sous le nom *d'apprentissage de concepts à partir d'observations* ou "apprentissage sans maître" [Mic99], utilise des données dont on ne

connaît pas les classes. Dans ce cas, on travaille avec des exemples non étiquetés, on parle aussi de partitionnement (clustering). La deuxième méthode est l'*apprentissage de concepts à partir d'exemples*. Cette méthode, plus utilisée en fouille de donnée que la première, est identifiée comme procédé de classification. Elle se base sur des données dont les classes sont connues. Ces deux méthodes sont considérées comme les formes principales d'apprentissage inductif et sont également connues sous les noms respectifs d'apprentissage supervisé et apprentissage non supervisé [Mic93]. L'auteur qualifie ces deux formes particulières d'apprentissage inductif conceptuel.

- **apprentissage de concepts à partir d'observations** (non supervisé) : appelé également classification en analyse des données (*clustering* dans la littérature anglaise), commence avec un ensemble non structuré d'exemples et fournit des classes. Ainsi, la classe est inconnue et le but d'apprentissage est de déterminer des groupes significatifs (les classes) d'exemples appartenant à cette même classe. En généralisation descriptive, ou en apprentissage non supervisé, le but est de déterminer une description générale, c-à-d d'élaborer de nouveaux concepts ou de nouvelles théories, caractérisant une collection d'observations. Pour résumer, l'apprentissage non supervisé fournit des descriptions en extension à partir d'exemples non structurés.

  Un algorithme d'apprentissage utilise l'ensemble des variables explicatives (variables exogènes) $\{y_1, ..., y_k, ..., y_r\}$ et constitue des groupes dans lesquels les observations diffèrent très peu. On parle alors de minimisation de l'inertie intra-classe. L'algorithme cherche également à créer ces groupes de manière à ce que les observations diffèrent le plus possible d'un groupe à l'autre. Il s'agit ici de la maximisation de la variance inter-classes ;

- **apprentissage de concepts à partir d'exemples** : ce type d'apprentissage connu aussi sous le nom d'apprentissage supervisé, revient à l'invention en intention de description de classes fournies (partiellement) en extension. En d'autres termes, la classe est connue et elle est décrite par des exemples de ses membres. Le but est de construire un système capable de classer correctement de nouveaux exemples de classe inconnue. Par exemple, supposons que les données soient relatives à des patients souffrant de cancer. Une liste des traitements appliqués constitue les classes. Le but est de trouver des façons de décrire comment appliquer le meilleur traitement possible à un nouveau patient et, pour ceci, de décrire en intention les traitements appliqués dans le passé. Cet apprentissage produit des réseaux neuronaux, des arbres de décision, ou des règles de la forme : **Si** (ensemble de caractéristiques du patient) **Alors** (traitement approprié).

Dans l'apprentissage de concepts à partir d'exemples, les observations caractérisent des objets pré-classés par un expert en une ou plusieurs classes (concepts) représentées par les valeurs (modalités) $m_1^{but}$, ..., $m_u^{but}$, ..., $m_p^{but}$ de la variable à expliquer (variable endogène) $y_{but}$. L'hypothèse induite peut être vue comme une règle de reconnaissance de concept $m_u^{but} = f(y_1, ..., y_r)$ telle que si un objet vérifie cette règle, alors il représente le concept donné. Mickalski [MK90] résume ainsi la situation : si les données fournies à une méthode d'apprentissage consistent en exemples classés par une source de connaissance indépendante, par exemple un professeur, un expert ou un modèle de simulation, alors il s'agit d'apprentissage à partir d'exemples.

L'apprentissage de concepts à partir d'exemples se résume en trois étapes. la première consiste à créer un classificateur à partir de l'ensemble des données. La deuxième étape classe des nouvelles instances via le classificateur. Enfin, la dernière étape consiste à mesurer la performance du classificateur. Pour cette tâche, la méthode la plus répandue consiste à diviser l'ensemble d'apprentissage en deux sous-ensembles : un ensemble d'entraînement (training set) et un ensemble de test. La performance d'un classificateur peut être mesurée par sa vitesse d'apprentissage, sa précision prédictive qui correspond au taux d'erreur de classement ou encore sa compréhensibilité, c'est-à-dire la capacité du classificateur à générer de la connaissance sémantiquement et structurellement similaire à celle d'un expert humain. Deux paramètres importants conditionnent donc la bonne performance d'un classificateur : les données disponibles et l'algorithme d'apprentissage utilisé.

### 3.2.4 Les algorithmes d'apprentissage supervisé

Dans notre démarche, nous ne nous intéressons qu'aux algorithmes d'apprentissage de concepts à partir d'exemples. Tsai [Tsa97] a classé en trois types les algorithmes de fouille de données et d'apprentissage supervisé. L'induction orientée attributs (Attribute-Oriented Induction), les algorithmes d'induction basés sur la logique et les arbres de décision.

- l'induction orientée attributs est une méthode qui généralise les sous-ensembles pertinents de données, attribut par attribut, jusqu'à l'obtention d'une relation générale. Méthode développée, pour extraire des règles caractéristiques et des règles de classification à partir de bases de données relationnelles. La caractéristique de cette méthode est l'emploi des concepts hiérarchisés dans le processus d'induction ;
- les algorithmes d'induction basés sur la logique sont des méthodes d'apprentissage utilisant un langage de logique de premier ordre. La logique de premier ordre se

base sur un sous-ensemble de logique de prédicat pour représenter les concepts, les règles et d'autres formes de connaissance ;

- les arbres de décision sont des systèmes d'apprentissage inductif à partir d'exemples. Ils permettent de distinguer les structures sous-jacentes qui régissent les données et de construire des règles capables de classer des objets à partir d'un ensemble d'apprentissage constitué d'objets dont les classes sont connues.

Les arbres de décision nous ont semblé particulièrement prometteurs pour leur capacité à retrouver immédiatement la classe en fonction d'attributs pertinents. Toutefois, ils présentent un inconvénient, qui n'affecte pas notre problème. A savoir leur non adaptabilité pour la plupart des logiciels d'arbres de décision, aux nouvelles instances de l'ensemble d'apprentissage. En effet, travaillant en prédictif, notre ensemble d'apprentissage n'est pas susceptible d'évoluer. Nous décrivons par la suite les arbres de décision et les principaux algorithmes développés dans ce domaine.

Un arbre de décision est constitué de trois éléments : les noeuds, les arcs et les feuilles. Chaque noeud est associé à un attribut et chaque arc issu de ce noeud est associé à l'une des caractérisations ou valeurs de cet attribut. Les feuilles, qui sont des noeuds sans arcs sortants, donnent des modalités de la classe associée à la branche suivie pour l'atteindre.

Pour construire des arbres de décision performants, un élagage est souvent utilisé pour enlever des parties de l'arbre qui ne contribuent pas à la précision de classification sur les instances non prises en compte lors de la construction de l'arbre. L'élagage permet de produire un arbre moins complexe et par conséquent plus compréhensible.

Depuis de nombreuses années, plusieurs méthodes de construction d'arbres de décision sont apparues. Ces méthodes s'intéressent à la fois aux attributs symboliques et attributs numériques qui prennent leurs valeurs dans un univers continu. Nous présentons par la suite les différentes méthodes existantes de construction d'arbres de décision.

#### 3.2.4.1 CART

Le système CART [BFOS84] est un outil d'induction d'arbres de décision et de régression dont les classes sont à valeurs continues. Il construit un arbre qu'il élague et obtient une série d'arbres imbriqués de coûts-complexités identiques parmi lesquels il choisit le meilleur. Pour sélectionner l'attribut test, une mesure d'impureté est utilisée. Le système CART traite des attributs discrets et des attributs ordonnés. Afin de résoudre le problème de la préférence aux attributs possédant un grand nombre de modalités, les auteurs proposent de binariser les attributs discrets. La binarisation des attributs discrets

et continus est réalisée lors de la phase de construction de l'arbre. A chaque noeud, le système CART étudie tous les attributs et cherche pour chaque attribut la meilleure binarisation. Il sélectionne alors le meilleur attribut binarisé. Constatant que le pré-élagage n'est pas une bonne solution, l'auteur de CART propose de descendre l'arbre jusqu'au profondeur, puis applique l'élagage en coût-complexité. Le résultat obtenu est une série d'arbres imbriqués. CART utilise la règle d'un écart-type pour choisir le meilleur arbre parmi la séquence d'arbres élagués.

#### 3.2.4.2 ID3, GID3 et GID3*

ID3 [Qui86] construit un arbre de décision utilisant une approche descendante, procédant comme suit : sélection d'un attribut et division de l'ensemble d'apprentissage en sous-ensembles caractérisés par les valeurs possibles de cet attribut. Cette procédure est appliquée récursivement à partir de chaque sous-ensemble jusqu'au moment où plus aucun sous-ensemble ne contient d'objets de classes différentes. Ces sous-ensembles de classe unique correspondent alors aux feuilles de l'arbre et peuvent être indexés par leur classe. ID3 ne traite qu'un type d'attribut : les attributs à valeurs discrètes. Certains auteurs abordent le problème des attributs continus en proposant un pré-traitement des attributs, d'autres discrétisent les attributs au fur et à mesure des besoins dans le déroulement de l'algorithme de création d'arbre.

L'algorithme GID3 [CFIQ88] est structurellement identique à ID3. Il diffère par le traitement des attributs discrets. GID3 a été créé afin de résoudre le problème des feuilles nulles, c'est à dire non étiquetées, et des valeurs non pertinentes d'attribut. A la différence de ID3 et C4, GID3 ne crée pas une branche par valeur de l'attribut sélectionné.

Fayyad [Fay94] a créé l'algorithme GID3* en s'appuyant sur GID3. Il diffère de son prédécesseur uniquement dans la phase de contrôle de la croissance de l'arbre en augmentant ou en diminuant la tendance de l'algorithme à effectuer des branchements sur certaines valeurs de l'attribut plutôt que sur d'autres.

#### 3.2.4.3 ASSISTANT86

ASSISTANT86 [CKB87] est un descendant d'ID3 qu'il vise à l'améliorer en traitant les données incomplètes et bruitées ainsi que les attributs multivalués et continus. A l'instar d'ID3, ASSISTANT86 utilise des heuristiques afin de déterminer l'attribut qui sera choisi comme noeud de l'arbre. Il fera appel à cette même technique pour décider de l'arrêt de la descente de l'arbre, à la différence d'ID3 qui construit l'arbre jusqu'à ce que les objets soient parfaitement classés. Une différence fondamentale avec ID3 est la

forme binaire de l'arbre obtenu. Pour cela, il suffit de binariser les attributs discrets. La binarisation des attributs discrets est obtenue via un algorithme heuristique si l'attribut possède plus de quatre modalités ou par le biais d'une recherche exhaustive sinon. Pour améliorer la précision de classification et décroître la complexité d'un arbre en présence de bruit, l'idée retenue par ASSISTANT86 est d'employer deux types d'élagage : le pré-élagage, ou arrêt du partitionnement à un noeud et puis élagage.

#### 3.2.4.4 C4 et C4.5

C4 [Qui90] utilise la même technique de fenêtrage et de création d'arbre que son prédécesseur ID3. La différence réside dans l'ajout d'une procédure d'élagage une fois que l'arbre classant correctement les objets de l'ensemble d'apprentissage est construit. L'ensemble de travail initial étant choisi aléatoirement, le même ensemble d'apprentissage peut donner lieu à différents arbres de décision. C4 construit donc plusieurs arbres de décision qu'il élague, puis choisit le plus intéressant en considérant sa taille et le nombre d'observations mal classées de la base d'apprentissage. La méthode d'élagage utilisée par C4 est appelée élagage pessimiste. Elle évalue chaque noeud à partir de la racine. Le sous-arbre considéré est élagué, c'est à dire réduit à une feuille, si le taux d'erreur de la nouvelle feuille diffère de moins d'un écart type du taux prédictif d'erreur du sous-arbre. L'arbre final élagué contient uniquement des sous-arbres qui ne peuvent pas être remplacés par des feuilles sans augmenter de manière significative le taux d'erreur prédictif de l'arbre.

C4.5 [Qui93] est une version évoluée de C4 et qui diffère dans la technique de fenêtrage utilisée. Ainsi, les instances composant la fenêtre initiale sont choisies afin que la distribution des classes à l'intérieur de la fenêtre soit la plus uniforme possible. Si toutes les instances ne sont pas classées, C4.5 choisit d'arrêter la croissance de l'arbre lorsque la séquence d'arbres créés ne devient pas plus précise. Il prévient ainsi l'inexorable croissance de la fenêtre lors de la présence de données bruitées ou indéterminées. A l'instar de C4, C4.5 construit plusieurs arbres différents et sélectionne l'arbre pour lequel le pourcentage d'erreur de classification est le plus faible. De plus, il génère toutes les règles issues des différents arbres à partir desquels il construit un seul classificateur. Chaque chemin de la racine d'un arbre non élagué à une feuille donne une règle initiale. La partie gauche de la règle contient toutes les conditions établies par le chemin et la partie droite spécifie la classe à la feuille. chaque règle est simplifiée en retirant les conditions qui n'aident pas à la discrimination de la classe concernée et en utilisant une estimation pessimiste de la précision de la règle.

#### 3.2.4.5 SLIQ

SLIQ [MAR96] est un classificateur basé sur la construction d'un arbre de décision. Il permet de manipuler à la fois les attributs numériques et symboliques. Pendant la phase de construction de l'arbre, SLIQ utilise une procédure de rangement pour réduire le coût d'évaluation des attributs numériques. Cette procédure de rangement est intégrée avec une stratégie de recherche en largeur d'abord. SLIQ utilise aussi une nouvelle technique d'élagage basée sur la mesure MDLP (Minimum Description Length Principle). Cette nouvelle technique n'est pas coûteuse en temps de calcul et fournit un arbre compact et précis. La combinaison de toutes ces techniques à l'intérieur de SLIQ limite sa capacité de traiter des grandes bases de données avec plusieurs classes, attributs et exemples d'apprentissage.

#### 3.2.4.6 SPRINT

SPRINT [SAM96] est une version améliorée de SLIQ. Il est plus rapide et permettant de traiter des bases de données plus larges. Des mécanismes de parallélisme sont introduites pour permettre d'effectuer plusieurs traitement en même temps. Comme la plupart des autres algorithmes, SPRINT est basé sur deux phases de traitement : la construction d'un arbre de décision, puis son élagage. A l'inverse des algorithmes de classification CART et C4.5 qui utilisent une exploration en profondeur d'abord des arbres,en répétant à chaque noeud le tri des données, SPRINT crée en temps réel des listes séparées pour chaque attribut.

#### 3.2.4.7 SEE5

See5 (`http://www.rulequest.com/`), développé par Quinlan, est la dernière version évoluée de l'algorithme C4.5. Il peut traiter des problèmes de classification avec n'importe quel nombre d'attributs. See5 permet de prédire la classe d'appartenance d'un individu en se basant sur ses valeurs d'attributs. A partir des données d'entrée, See5 construit un classificateur qui s'occupera de la prédiction. Le classificateur construit peut être exprimé sous forme d'un arbre de décision ou d'un ensemble de règles d'apprentissage. Chaque règle produite est associée d'un taux de bon classement. See5 se base essentiellement sur deux fichiers de données. Le premier, contient les noms des attributs ainsi que leurs instances. Le deuxième fichier contient les valeurs des attributs pour chaque exemple d'apprentissage. L'utilisation de See5 est facilité par une interface graphique permettant de lire les fichiers d'entrée est d'afficher les résultats de classification des individus.

La nouveauté de See5 par rapport aux anciennes versions d'algorithmes telles C4.5,

réside dans la génération de plusieurs classificateurs à la fois. Quand un nouveau cas se présente pour la classification, les classificateurs votent chacun de son coté. La classe est déterminée selon les votes obtenus.

Notre choix est porté sur le classificateur See5 pour la phase d'apprentissage et l'extraction des règles de priorité.

### 3.2.5 Pré-traitement : discrétisation des attributs

La plupart des algorithmes de pré-traitement manipulant des variables continues, cherchent à transformer ces dernières sous une autre forme simple et représentative. On parle ainsi de la discrétisation des variables continues en un nombre fini d'intervalles disjoints. La discrétisation a l'avantage de réduire le nombre de variables et par conséquent de réduire la complexité de la fouille de données. La discrétisation est souvent utilisée dans des contextes de classification [Bay01] dont l'objectif consiste à maximiser la précision de la prédiction des algorithmes qui ne peuvent pas manipuler des variables continues. En revanche, en extraction de connaissance, le but de l'analyse des données est la découverte des structures sous-jacentes et non pas l'augmentation de la précision de la prédiction. Pour cette raison, le critère de choix des intervalles doit être bien étudié. En effet, les intervalles induits par la discrétisation ne doivent pas cacher les structures pouvant exister au sein des données. Par example, si les intervalles sont trop larges, on risque d'ignorer des formes existantes dans des petites structures. En plus, une attention particulière doit être mise quand au choix de la sémantique des intervalles.

Afin d'élargir le champs des données prises en compte, c'est à dire de considérer les attributs continus, plusieurs algorithmes traitant spécifiquement des valeurs numériques entières ou réelles ont vu le jour. Une première voie à consisté à traiter sans distinction les valeurs numériques comme des valeurs symboliques [Qui86]. La première difficulté soulevée par le traitement homogène de tous les descripteurs, est la création d'un arbre de grande taille due aux nombreuses branches issues des noeuds associés aux attributs continus (une branche par valeur de l'attribut). La seconde difficulté est liée directement au critère de sélection qui privilégie les attributs possédant un grand nombre de modalités. Ainsi, les attributs sélectionnés ne sont pas forcément pertinents pour l'induction.

Une autre voie conduit à transformer les variables continues en variables compatibles avec les algorithmes d'apprentissage. Elle s'est attelée à découper l'ensemble des valeurs de l'attribut numérique en une série d'intervalles disjoints, c'est à dire à discrétiser ces valeurs numériques en intervalles identifiés par un symbole du nouvel attribut *symbolique* ainsi formé. Cette discrétisation peut s'intégrer dans l'algorithme d'apprentissage.

Par exemple, C4.5 [Qui93] estime dynamiquement le seuil de discrétisation durant le déroulement de l'algorithme. Elle peut également être réalisée avant l'utilisation d'un algorithme d'apprentissage comme pré-traitement visant à homogénéiser les variables descreptives.

Le choix de la méthode de discrétisation a des conséquences importantes sur le modèle d'induction à construire. Elle conditionne le choix des attributs discriminants lors de la construction du classificateur ainsi que dans la phase de pré-traitement. C'est la raison pour laquelle il nous a semblé indispensable d'aborder plus en détail ce sujet et de présenter les différentes méthodes de discrétisation existantes afin de choisir celle qui s'adaptera au mieux avec notre application.

La littérature de la discrétisation est riche mais la plupart des algorithmes sont statiques et traitent les variables les unes indépendamment des autres. Par example, Fayyad et Irani [FI93] effectuent la discrétisation récursive d'un attribut pour minimiser l'entropie de la classe. Ils ont utilisé le critère de la longueur descriptive minimale pour arrêter la discrétisation. D'autres algorithmes dans la même catégorie tels que Chi2 [LS95], Chi-Merge [Ker92] introduisent la discrétisation basée sur le taux d'erreur. Dougherty et al. [DKS95] et Zighed et al. [ZRRF99] ont fourni un bon état de l'art sur la discrétisation. Ils ont classé les travaux portant sur la discrétisation selon trois axes :

- le premier axe regroupant les méthodes supervisées dont on tient compte de la classe d'appartenance de chacun des objets et les méthodes non supervisées (aveugles), où on ne tient compte que de la similarité des objets sans préoccuper de leur classe d'appartenance respective ;
- le deuxième axe concernent les méthodes locales qui définissent localement les bornes et réalisent la discrétisation pendant la phase d'apprentissage en même temps avec l'utilisation de la variable. Cet axe contient également les méthodes globales réalisant la discrétisation en pré-traitement ;
- le dernier axe comporte les méthodes de discrétisation statiques et dynamiques. Statiques quand la discrétisation a lieu sur chaque variable indépendamment des autres. Dynamiques lorsque les variables sont appréhendées ensemble afin de tenir compte des éventuelles interactions.

Le problème de discrétisation se formalise de la manière suivante : soit un attribut $X$ qui prend ses valeurs sur la droite des réels $\mathbb{R}$. Pour tout exemple $w$ issu d'un échantillon d'apprentissage noté $\Omega$, $X(w)$ désigne la valeur prise par cet exemple pour l'attribut $X$

($X(w) \in \mathbb{R}$) et $Y(w)$ désigne la classe de l'exemple. Si l'exemple appartient à la classe $y_j$ ($j = 1, ..., l$), alors nous pouvons écrire $Y(w) = y_j$.

Soit $I$ l'space de définition de $X$. Discrétiser l'attribut $X$ revient à découper $I$ en $p$ intervalles $I_i$, $1 \leq i \leq p$ tels que :

$$\bigcup_{i=1}^{p} I_i = I \tag{3.1}$$

$$\bigcap_{i=1}^{p} I_i = \emptyset \quad \forall \ 1 \leq i \leq p \tag{3.2}$$

Concrètement parlant, cela revient à transformer un vecteur de données initialement numérique en un vecteur disjonctif. Les intervalles trouvés sont identifiés par des noms symboliques et significatifs. Supposons que l'espace de définition $I$ de la variable $X$ peut être assimilé à l'intervalle $[a, b]$. On veut partitionner $I$ en $p$ sous-intervalles.

$$I_1 = [a, d_1[, ..., I_j = [d_{j-1}, d_j[, ..., I_p = [d_{p-1}, b[ \tag{3.3}$$

Cela consiste aussi à déterminer les $p - 1$ points de discrétisation $d_j$. Une fois les points de discrétisation sont trouvés, l'attribut $X$ est remplacé par un attribut $\hat{X}$ qui prend ses valeurs dans l'ensemble $\{1, ..., p\}$ :

$$\hat{X}(w) = \begin{cases} 1 & si \quad X(w) < d_1 \\ i & si \quad d_{i-1} \leq X(w) < d_i \\ p & si \quad X(w) \geq d_{p-1} \end{cases} \tag{3.4}$$

Il s'agit alors de trouver une suite finie strictement croissante de points de coupures $d_1$ à $d_{p-1}$. Pour cela, il faut choisir le paramètre $p$ et ensuite déterminer les points $d_j$. Des méthodes de discrétisation simples, statiques et non supervisées sont apparues, mais elles demandent à l'utilisateur de spécifier le nombre d'intervalles $p$ résultant du partitionnement. La méthode la plus simple de discrétisation appelée *discrétisation à intervalles de dimension égale*, consiste à choisir arbitrairement $p$ et de diviser l'intervalle de départ $I$ en $p$ intervalles tels que $d_i = a + i\sigma$, $\forall \ 1 \leq i \leq p - 1$ avec $\sigma = \frac{b-a}{p}$. Une autre méthode plus significative que la première appelée discrétisation à intervalles d'effectif égal, consiste à construire $p$ intervalles de même effectif ($p$ étant choisi par l'utilisateur). Ces deux méthodes ne tiennent pas compte de la structure sous-jacente des données. Pour pallier ce découpage à priori, d'autres méthodes statiques non supervisées ont vu le jour. Nous citons à titre d'exemple les méthodes paramétriques et non paramétriques. Le premier type de méthodes estime les paramètres d'un modèle (méthode de Pearson,

estimation du maximum de vraisemblance, méthode bayésienne avec apprentissage) et permet d'adapter une règle de décision au choix de $p$ et des $d_j$. Le deuxième type de méthodes correspond à des algorithmes mono objectif pour l'estimation des paramètres de la discrétisation. Michaut [Mic99] estime que la lourdeur des calculs et la quasi impossibilité de vérifier les hypothèses émises, rendent obsolètes ce type de méthodes. En outre, ces dernières ne sont pas valables pour de petits échantillons.

Les méthodes citées auparavant qualifiées d'aveugles (ou non supervisées), ignorent les classes des objets. Par conséquent l'information essentielle est perdue et les méthodes sont, dans maintes situations, moins performantes que les méthodes supervisées. En effet, une bonne méthode de discrétisation doit prendre en compte toute la connaissance a priori dont elle dispose sur l'ensemble d'apprentissage, si non elle s'expose à des découpages qui ne correspondent pas aux données. Ainsi, le nombre $p$ d'intervalles qui résultent de la discrétisation ne doit pas être fixé à priori, mais doit se déduire de la structure des données. Il est donc impératif de tenir compte des classes d'appartenance des objets.

Pour chaque discrétisation en $p$ intervalles, une matrice $A$ de $l$ lignes et $p$ colonnes est généralement déterminée. Les lignes de cette matrice correspondent aux classes et les colonnes représentent les intervalles. Chaque élément $A_{ij}$ de la matrice $A$ représente le nombre d'exemples d'apprentissage appartenant à l'intervalle $I_i$ et à la $j^{th}$ classe.

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & ... & A_{1p} \\ A_{21} & A_{22} & ... & A_{2p} \\ ... & ... & ... & ... \\ A_{l1} & A_{l2} & ... & A_{lp} \end{pmatrix}$$

Le nombre d'exemples d'apprentissage contenus dans l'intervalle $I_i$ est égal à :

$$R_i = \sum_{j=1}^{l} A_{ij} \tag{3.5}$$

Le nombre d'exemples d'apprentissage appartenant à la $j^{th}$ classe est égal à :

$$C_j = \sum_{i=1}^{p} A_{ij} \tag{3.6}$$

Le nombre total d'exemples d'apprentissage est égal à :

$$N = \sum_{j=1}^{l} C_j \tag{3.7}$$

La fréquence moyenne de $A_{ij}$ est la suivante :

$$E_{ij} = \frac{R_i * C_j}{N} \tag{3.8}$$

Les algorithmes de discrétisation supervisée s'effectuent généralement en deux étapes. La première consiste à démarrer par une discrétisation initiale en déterminant des paliers par découpage a priori de la variable numérique en $p$ intervalles (on obtient ainsi la valeur de $p$) et en déterminant les points de coupure de ces intervalles. La deuxième étape fusionne ou partitionne les intervalles initiaux selon un critère à optimiser. A chaque étape, la dimension de la matrice $A$ est modifiée. L'arrêt de la discrétisation est déterminé par la satisfaction d'un critère donné.

Nous focalisons notre intérêt sur les algorithmes de discrétisation statiques et supervisées. Nous présentons aux paragraphes suivants les méthodes les plus connues dans ce domaine.

#### 3.2.5.1 Algorithme 1R : [Hol93]

Les objets sont triés par ordre croissant des valeurs prises par la variable numérique considérée. Les valeurs de la variable associée à des objets appartenant à la même classe sont fusionnées au sein d'un même intervalle. Holte préconise d'avoir, pour chaque intervalle, un nombre minimal d'objets. Un intervalle peut, par conséquent, contenir des objets appartenant à des classes autres que la classe majoritairement représentée dans cet intervalle. Les points de discrétisation sont choisis en bougeant les valeurs frontières : des valeurs particulières sont ajoutées aux intervalles dont la dimension est inférieure au seuil minimum requis. Cette méthode simple paraît discrétiser raisonnablement lorsqu'elle est utilisée avec l'algorithme d'induction 1R.

#### 3.2.5.2 Algorithme ChiMerge : [Ker92]

L'étape d'initialisation consiste à trier les examples d'apprentissage en ordre croissant de valeurs de l'attribut à discrétiser. Ensuite, construire autant d'intervalles que de nombre d'exemples (chaque exemple est mis dans un intervalle). Le processus de fusionnement des intervalles continu jusqu'au moment où tous les intervalles adjacents auront une valeur de $\chi^2$ excédant le seuil $\chi^2_{seuil}$ désiré (les intervalles adjacents sont alors significativement différents par le test d'indépendance). La valeur de $\chi^2_{seuil}$ qui dépend du nombre de classes -1, est choisie en fonction du niveau d'indépendance désiré. Pendant chaque itération du processus de fusion, la valeur $\chi^2$ de chaque pair d'intervalles adjacents est calculée. Le pair d'intervalles ayant la valeur minimale de $\chi^2$ est fusionné.

Pour calculer la valeur de $\chi^2$ de deux intervalles, on utilise la formule suivante :

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{l} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{3.9}$$

La lourdeur de cet algorithme apparaît nettement lorsque les instances sont nombreuses. En effet, il y a alors beaucoup d'intervalles à fusionner et la fusion n'est pas réalisée que deux à deux entre intervalles adjacents.

#### 3.2.5.3 Algorithme StatDisc : [RR95]

C'est un algorithme similaire au ChiMerge (dont la complexité est améliorée), qui utilise un autre critère statistique de fusionnement d'intervalles. C'est une méthode ascendante créant une hiérarchie d'intervalles de discrétisation. Les auteurs estiment que le critère statistique le mieux adapté pour comparer les fréquences relatives aux classes des intervalles adjacents est la mesure $\phi$. La mesure de Cramer est équivalente à $\sqrt{2}\phi$.

#### 3.2.5.4 Algorithme MDLPC : [FI93]

Cet algorithme proposé par Fayyad et Irani, utilise un critère d'information appelé MDLPC (Minimum Description Length Principal Cut). C'est une méthode récursive et binaire : l'intervalle $[a, b[$ est divisé en deux intervalles, qui de leur part seront divisés chacun en deux intervalles, jusqu'à la satisfaction d'une condition d'arrêt. Le critère de partitionnement MDLPC, calcule l'entropie de la classe induite par le point de coupure de l'ensemble d'apprentissage. Un premier point de coupure satisfaisant ce critère est recherché. Si un tel point existe, la population d'exemples est partitionnée en deux. La recherche d'un point de coupure qui répond au critère MDLPC dans la sous-population est réitérée. L'algorithme MDLPC se déroule de la manière suivante :

1. trier les exemples d'apprentissage selon un ordre croissant de leurs valeurs de l'attribut à discrétiser ;
2. regrouper les valeurs de l'attribut dont les exemples d'apprentissage appartiennent à la même classe dans le même intervalle ;
3. si plusieurs classes sont superposées sur une même valeur de l'attribut, alors former un intervalle contenant cette unique valeur. Au contraire des autres intervalles, ce dernier contiendra plusieurs classes ;
4. initialiser les points de discrétisation aux frontières $d_j$ des intervalles formés aux étapes 2 et 3 ;

5. parmi les $l$ points de discrétisation, on choisi celui menant à une bipartition satisfaisant au mieux le critère MDLPC ;
6. l'étape 5 est reprise pour chacune des deux sous-populations ;
7. le processus de discrétisation s'arrête lorsqu'aucune amélioration n'est possible.

#### 3.2.5.5 Algorithme CONTRAST : [Mer93]

Van de Merckt propose d'utiliser, pour discrétiser un attribut une heuristique basée sur la notion de la similarité entre les exemples en se servant d'un principe de l'apprentissage non supervisé. L'hypothèse de similarité utilisée en classification par apprentissage non supervisé suppose que deux exemples proches, dans l'espace de représentation $\Re$, appartiennent à la même classe. Cette hypothèse a conduit l'auteur à prendre en compte la proximité des exemples selon l'attribut à discrétiser. Par conséquent, Van de Merckt suggère de chercher le point de discrétisation binaire qui fournit le meilleur "contrast" entre les valeurs de l'attribut, même si les intervalles générés contiennent des exemples de classes différentes. Ce principe revient à trouver le point de discrétisation qui maximise la distance entre les exemples d'un même intervalle. A cette notion de contraste est rajoutée celle de l'entropie afin d'éviter de préférer une partition avec une forte entropie à une partition de même contraste mais avec une faible entropie.

L'algorithme de discrétisation se résume de la façon suivante :

1. initialisation en classant les exemples d'apprentissage par ordre croissant des valeurs de l'attribut $X$ à discrétiser ;
2. détermination des $p$ points frontières. Les points de discrétisation sont nécessairement des points frontières ;
3. sélectionner parmi les $p$ frontières, le point de discrétisation $d$ qui maximise la quantité $CE(X,d,S)$ :

$$CE(X,d,S) = \frac{Contrast(X,d,S)}{I_{X,d}(S)} = -\frac{R_1 R_2 (\bar{X}_{1.} - \bar{X}2.)^2}{\sum_{i=1}^{2}\sum_{j=1}^{l} A_{ij} \log_2 \frac{A_{ij}}{R_i}} \tag{3.10}$$

   Où $S$ représente l'échantillon au sommet considéré et $\bar{X}_i$ est la moyenne de l'attribut $X$ sur les sous-échantillons $S_1$ et $S_2$ : $S_1 = \{w \in S \ / X(w) \leq d\}$ et $S_2 = \{w \in S \ / X(w) > d\}$ ;
4. partitionner les exemples, selon le point de discrétisation, en deux sous-populations.

#### 3.2.5.6 Algorithme FUSINTER : [Rab96]

L'algorithme FUSINTER proposé par Rabaséda utilise un critère noté $\phi$ construit à partir d'une mesure d'incertitude sensible aux effectifs. L'objectif de cet algorithme est la recherche d'une discrétisation qui minimise $\phi$. Étant donnée la matrice $A$, le critère $\phi$ de discrétisation, basé sur l'entropie quadratique se calcule suivant l'équation 3.11.

$$\phi(A) = \sum_{i=1}^{p} \left[ \alpha \frac{R_i}{N} \sum_{j=1}^{l} \left( \left( \frac{N+\lambda}{R_i + l\lambda} \right) \left( 1 - \frac{N+\lambda}{R_i + l\lambda} \right) \right) + (1-\alpha) \frac{l\alpha}{R_i} \right] \tag{3.11}$$

$\alpha$ et $\lambda$ deux paramètres à choisir par l'utilisateur avec $0 \leq \alpha \leq 1$ et $\lambda > 0$. Rabaséda [Rab96] a fixé la valeur de $\alpha$ à 0.975. Le premier terme de la somme globale mesure l'hétérogénéité de la partition induite par la discrétisation alors que le deuxième terme permet de pénaliser les partitions qui ont beaucoup d'intervalles avec de faibles effectifs. L'algorithme de FUSINTER est le suivant :

1. initialisation en classant les exemples d'apprentissage par ordre croissant des valeurs de l'attribut $X$ à discrétiser ;
2. détermination des $p-1$ points frontières constituant la discrétisation initiale en $p$ intervalles ;
3. calcul de la quantité $\phi(\{I_i, I_{i+1}\}) - \phi\big(\{\underbrace{I_i + I_{i+1}}\}\big)$ associée au gain d'incertitude pour chaque paire d'intervalles adjacents ;
4. fusion de la paire d'intervalles adjacents dont le gain d'incertitude $\phi(\{I_i, I_{i+1}\}) - \phi\big(\{\underbrace{I_i + I_{i+1}}\}\big)$ est positif et maximal ;
5. $p \leftarrow p - 1$ ;
6. recommencer les étapes 3 à 5 pour les $p$ intervalles restants. Le processus s'arrête dès que plus aucune fusion n'est possible.

## 3.3 L'ECD appliqué au job shop

Nous avons utilisé la démarche complète de l'extraction de connaissance à partir de données pour déterminer une heuristique permettant la résolution du problème d'ordonnancement de job shop. La première étape consiste à générer une population de bonnes solutions de plusieurs problèmes de job shop en utilisant un algorithme génétique. Ses solutions appelées aussi séquences génétiques ou chromosomes, seront caractérisées par des attributs numériques liés au problème. Pour être traitées et bien fouillées, les solutions

sont ensuite transformées sous la forme de liste d'opérations devant chaque machine. Afin d'obtenir un modèle aussi général que possible, les attributs numériques sont discrétisés à l'aide d'un algorithme qui tient compte à la fois des caractéristiques des individus ainsi que des classes auxquelles ils appartiennent. L'étape la plus importante dans ce processus d'extraction de connaissance est celle de la fouille des solutions générées par l'algorithme génétique afin de trouver les liens entre la classe d'un individu et ses caractéristiques. L'étape de fouille de solutions permettra la découverte d'un ensemble de règles de priorité pour l'affectation des opérations sur les machines. Ces règles générées sur un ensemble test de problèmes de job shop sont ensuite transformées en une heuristique permettant de résoudre d'une manière optimale ou de trouver de bonnes solutions à plusieurs problèmes de job shop de tailles différentes. Nous détaillons par la suite le déroulement de chaque étape de l'ECD adapté à la résolution du job shop.

### 3.3.1 Collecte de données

La phase de collecte de données consiste dans notre cas à choisir un ensemble de problèmes types de job shop et de les résoudre en utilisant un algorithme génétique. Le choix est porté sur le benchmark de Muth et Thomson [MT63] de taille $6 \times 6$ dont on connait la valeur optimale du $C_{max}$ ($C^*_{max} = 55$). Ce problème dont les données sont détaillées au tableau 3.1, est en effet très utilisé dans la littérature pour valider les nouvelles approches de résolution. Pour avoir un ensemble de test assez significatif, nous avons ensuite généré aléatoirement dix autres problèmes de job shop de même taille. Les solutions obtenues par l'algorithme génétique sont caractérisées par des attributs numériques. Ces attributs liés au problème, permettent la description des opérations pour déterminer leur degré de priorité lors de la phase d'affectation sur les machines.

Nous allons par la suite introduire l'algorithme génétique utilisé ainsi que la caractérisation des solutions générées.

| Les jobs | $M(v)$,$p(v)$ | | | | | |
|---|---|---|---|---|---|---|
| $J_1$ | 3,1 | 1,3 | 2,6 | 4,7 | 6,3 | 5,6 |
| $J_2$ | 2,8 | 3,5 | 5,10 | 6,10 | 1,10 | 4,4 |
| $J_3$ | 3,5 | 4,4 | 6,8 | 1,9 | 2,1 | 5,7 |
| $J_4$ | 2,5 | 1,5 | 3,5 | 4,3 | 5,8 | 6,9 |
| $J_5$ | 3,9 | 2,3 | 5,5 | 6,4 | 1,3 | 4,1 |
| $J_6$ | 2,3 | 4,3 | 6,9 | 1,10 | 5,4 | 3,1 |

TAB. 3.1 – *Benchmark de Muth et Thomson de taille 6x6*

#### 3.3.1.1 Résolution par l'algorithme génétique

Pour mettre en oeuvre un algorithme génétique, plusieurs opérateurs doivent être mis au point. Le choix de ces opérateurs nous a demandé une bonne étude afin de générer de bons ordonnancements. La première étape consiste à choisir un codage adéquat des solutions génétiques. La représentation choisie est celle du Gen et al. [GTK94]. Cette représentation est un codage réel indirect. Un chromosome est ainsi représenté par une chaîne de $n \times m$ entiers, où $n$ est le nombre de jobs et $m$ est le nombre de machines. Chaque entier indique une opération spécifique d'un job. Pour le benchmark de Muth et Thomson formé de six jobs et six machines, un chromosome est composé de 36 gènes. La figure 3.3 montre un exemple d'un chromosome de ce benchmark de job shop.

L'algorithme génétique permet de générer des séquences (solutions) telles que montrées dans la figure 3.3. Ces séquences contiennent des ordonnancements implicites. Nous associons alors un ordonnanceur à la sortie de l'algorithme génétique pour l'interprétation des chromosomes afin de les transformer en ordonnancements effectifs dont on connaît les dates de début des tâches. La performance de l'algorithme génétique est affectée par le type d'ordonnanceur utilisé. Ainsi un ordonnanceur de type "décalage à gauche" (left-shit) intervient directement pour changer si nécessaire l'ordre de quelques gènes et améliore par conséquent la qualité des solutions. Ce type d'ordonnanceur permet d'obtenir des ordonnancements actifs. Un ordonnanceur de type "premier arrivé premier servi" (First-In-First-Out: FIFO), décode les chromosomes sans changer les ordres des gènes. Les ordonnancements générés sont alors semi-actifs. Afin d'accélérer la convergence de l'algorithme génétique et d'améliorer la qualité moyenne des ordonnancements fournis, nous utilisons un ordonnanceur de type left-shift.

| 1 | 4 | 5 | 1 | 3 | 6 | 3 | 2 | 5 | 5 | 1 | 5 | 3 | 4 | 2 | 6 | 1 | 5 | 3 | 4 | 2 | 1 | 6 | 4 | 3 | 5 | 6 | 4 | 1 | 6 | 3 | 2 | 6 | 2 | 4 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 1 | 1 | 2 | 1 | 2 | 3 | 3 | 4 | 3 | 2 | 2 | 2 | 4 | 5 | 4 | 3 | 3 | 5 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 5 | 6 | 4 | 6 | 5 | 6 | 6 |

Notations : Job / Opération

FIG. 3.3 – *Codage d'une solution du benchmark de Muth et Thomson*

Une fois le codage des solutions choisi, nous déterminons une population initiale. Cette population est générée aléatoirement. L'aspect aléatoire permet de visiter plusieurs zones de l'espace de recherche et par conséquent augmenter la probabilité de trouver la solution optimale du problème d'ordonnancement ou de trouver des bonnes solutions.

Pour mesurer la qualité des chromosomes, et ensuite sélectionner les meilleurs parmi

eux, une fonction d'évaluation, appelée aussi *fitness* est nécessaire. Souvent, les algorithmes génétiques traitant un problème d'ordonnancement et maximisant un critère donné, utilisent ce dernier comme fonction d'évaluation. Dans le cas d'une minimisation, il suffit de considérer le complémentaire du critère en question. Pour évaluer un chromosome $x$, nous avons défini, la fonction d'évaluation suivante :

$$F(x) = CM - C_{max}(x) + 1 \tag{3.12}$$

Où $CM$ est la plus grande valeur de $C_{max}$ observée dans la population courante ou depuis le début de l'exécution de l'algorithme génétique. On ajoute 1 pour éviter que l'individu le plus faible (ayant la plus grande valeur de $C_{max}$) ne soit jamais sélectionné pour la reproduction génétique. En effet, les chromosomes des individus faibles peuvent contenir des parties pouvant être utiles pour améliorer la force des individus forts. Dans notre cas, les solutions génétiques les plus adaptées sont celles qui possèdent les plus grandes valeurs de $F$. La fonction $F$ est calculée par l'ordonnanceur qui ajuste les opérations de chaque solution sur les machines en respectant les contraintes de précédence, et ensuite détermine les dates de début des opérations.

Les opérateurs génétiques nécessaires pour l'évolution des populations et la génération de nouvelles solutions sont la sélection, le croisement et la mutation. Pour la mise au point de notre algorithme génétique, nous avons choisi la procédure de sélection de Goldberg [Gol89] définie dans le chapitre 2. Cette sélection, basée sur le principe de la roue de loterie, favorise les individus qui possèdent une grande valeur de la fonction $F$. A l'inverse d'autres opérateurs de sélection élitiste, la sélection choisie maintient la diversité de la population en laissant une chance aux individus faibles pour être sélectionnés.

L'opérateur de croisement qui permet l'exploration de l'espace de recherche en créant de nouveaux individus est essentiel pour la convergence d'un algorithme génétique. Afin d'éviter la génération des solutions non admissibles, le croisement des solutions de notre problème de job shop nécessite une adaptation au type de codage utilisé. Nous avons utilisé le codage d'ordre maximal présenté au chapitre précédent. Ce type de croisement permet de modifier légèrement les positions et l'ordre des gènes. Cette propriété est très utile pour préserver les individus forts au fil des générations.

Afin d'introduire des petites modifications à certaines solutions de chaque population, la mutation est considérée dans le processus de recherche génétique avec une faible probabilité. Cet opérateur agit en général sur quelques éléments du chromosome. Pour l'algorithme génétique utilisé, nous avons adapté la mutation appelée insertion et décalage. Il s'agit de sélectionner aléatoirement deux gènes, d'insérer le premier gène à

la position du deuxième gène et ensuite décaler les gènes au milieu jusqu'à la première position. La figure 3.4 illustre le fonctionnement de cette mutation.

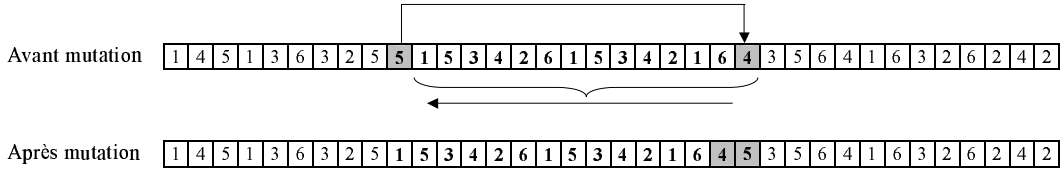



FIG. 3.4 – *Mutation d'un chromosome*

Pour mettre fin à l'exécution de l'algorithme génétique, un critère d'arrêt est nécessaire. Le choix de ce critère dépend de la complexité du problème traité, du nombre de variables qu'il présente ainsi que des informations connues à priori sur la solution optimale. Si la valeur du critère de la solution optimale est connue d'avance, le critère d'arrêt est évidemment trouver une solution réalisant cette valeur optimale. De même si on connaît une bonne borne inférieure de la solution optimale. L'algorithme s'arrête dès qu'il trouve une solution dont la valeur du critère soit la plus proche possible de la borne. Si aucune information n'est connue a priori, le critère d'arrêt peut être un nombre fixe d'itérations. Pour notre cas, nous avons utilisé deux conditions d'arrêt. La première est la valeur de solution optimale pour le cas du benchmark de Muth et Thomson. La deuxième condition est un nombre fixe d'itérations pour le cas des dix autres problèmes dont les données sont générés aléatoirement.

La dernière tâche pour la mise au point de l'algorithme génétique est la détermination des paramètres tels que la taille de la population, les probabilités de croisement et de mutation. Cette tâche, souvent empirique, est réalisée après plusieurs exécutions essais de l'algorithme avec différentes valeurs. Les valeurs des paramètres utilisés et générant les meilleurs résultats sont introduites dans le tableau 3.2.

| Paramètres | Valeurs |
|---|---|
| Taille de la population | 1000 |
| Nombre d'itérations | 1000 |
| Probabilité de croisement | 0.8 |
| Probabilité de mutation | 0.01 |

TAB. 3.2 – *Valeurs des paramètres de l'algorithme génétique*

Concernant le benchmark de Muth et Thomson, la population finale générée par l'algorithme génétique contient 92.7% de chromosomes optimisant le $C_{max}$.

En éliminant les redondances, nous avons obtenu 102 différents chromosomes. Lors de la transformation de ces solutions en ordonnancements effectifs, on a remarqué que plusieurs parmi eux sont identiques. En éliminant les solutions redondantes, nous avons obtenu 22 ordonnancements optimaux. Nous avons sélectionné une solution pour servir comme exemple d'apprentissage afin d'extraire des règles de priorité.

Pour les dix autres problèmes d'ordonnancement de job shop, générés aléatoirement, nous avons à chaque fois gardé la meilleure solution fournie par l'algorithme génétique. Afin d'assurer l'obtention d'une bonne solution, l'algorithme est exécuté plusieurs fois.

#### 3.3.1.2 Caractérisation des solutions

Comme nous l'avons mentionné dans la section de l'état de l'art du premier chapitre, la recherche des règles d'ordonnancement pour résoudre le problème du job shop a connu un grand succès. Néanmoins, les règles proposées se basent sur une seule caractéristique du problème (tel que le temps opératoire). Ces règles, malgré leur succès, ne peuvent pas être généralisées et donc ne permettent pas de résoudre tous les problèmes de job shop. En effet, plusieurs caractéristiques influencent en même temps la qualité de l'ordonnancement et coopèrent pour bien planifier l'affectation des opérations. Il est donc normal de considérer les différentes caractéristiques du problème afin de trouver les relations entre elles.

L'objectif de la phase d'apprentissage est la découverte des relations pouvant exister entre les caractéristiques d'une opération et son ordre d'exécution sur sa machine dans une bonne solution générée par l'algorithme génétique. La position de l'opération sur la liste d'attente devant la machine sur laquelle elle s'exécute est un facteur important dans notre tâche d'apprentissage. De plus, l'ordre de passage des opérations d'un même job constitue une contrainte fondamentale à prendre en compte et à respecter pour générer des ordonnancements admissibles. Il est donc normale que cette caractéristique soit considérée. Plusieurs chercheurs se sont focalisés sur la découverte et l'application des règles d'ordonnancement en fouillant les caractéristiques du job shop. Ces travaux se concentrent sur des caractéristiques statiques qui ne changent pas au cours du temps. Le temps opératoire est par conséquent une caractéristique très importante dans ces règles d'ordonnancement. Par exemple, la règle SPT (Shortest Processing Time) est considérée comme étant une règle performante pour la résolution du job shop dans l'absence des dates de fin au plus tard [BPH82]. Le temps opératoire restant, une autre information liée au temps opératoire, considère le reste de la durée globale d'un job en exécutant l'une de ses opérations. Le temps opératoire et le temps opératoire restant diffèrent dans

les éléments à comparer à chaque fois. Le temps opératoire permet la comparaison des opérations, tandis que le temps opératoire restant aide à comparer les jobs.

Le taux d'utilisation de la machine est un autre élément très important pouvant aussi être considéré pour le contrôle de l'ordonnancement. Pour cela, il suffit de calculer pour chaque machine la somme des temps opératoires de toutes les opérations qu'elle exécute. Cette somme est appelée la charge totale de la machine. Une telle information permet de donner une attention particulière aux machines ayant une charge lourde. En effet, retarder des opérations sur une machine chargée augmente la valeur du $C_{max}$.

Les caractéristiques sélectionnées pour la phase d'apprentissage sont les suivantes :

- Temps Opératoire (TO) : le temps nécessaire pour exécuter une opération ;
- Temps Opératoire Restant (TOR) : le reste de la durée globale d'un job en exécutant l'une de ses opérations ;
- Durée Totale d'un Job (DTJ) : la somme des temps opératoires des opérations d'un même job ;
- Charge Machine (CM) : la somme des temps opératoires des opérations devant s'exécuter sur la même machine ;
- Position de l'Opération dans son Job (POJ) : l'ordre d'exécution d'une opération donné par les contraintes de précédence du job contenant l'opération ;
- Ordre d'Affectation d'une opération sur sa Machine (OAM) : l'ordre de passage d'une opération sur la machine devant l'exécuter. Cet ordre est donné par le chromosome généré par l'algorithme génétique.

### 3.3.2 Pré-traitement

Pour préparer les données à la phase d'apprentissage et d'extraction de connaissances, deux traitements sont réalisés. Le premier, consiste à transformer des chromosomes générés par l'algorithme génétique afin de déterminer la classe d'appartenance de chaque opération. Le deuxième traitement le plus important, au cours duquel les différents attributs sont discrétisés afin d'extraire une connaissance plus générale et la plus représentative que possible. Par la suite une description détaillée de chaque traitement.

#### 3.3.2.1 Transformation des solutions

L'interprétation des chromosomes générés par l'algorithme génétique nécessite une modélisation et une représentation facile afin d'extraire les structures sous jacentes.

C'est pour cette raison nous utilisons le diagramme de Gantt pour la visualisation de l'ordonnancement. A partir de ce diagramme, nous pouvons déterminer les classes des opérations. L'apprentissage que nous effectuons est alors basé sur les ordonnancements et non pas sur les chromosomes comme a procédé Tsai [Tsa97] dans ses travaux. L'intérêt d'un tel apprentissage est l'extraction des règles de priorité qui permettent d'estimer l'ordre d'affectation des opérations sur les machines en fonction des valeurs d'attributs. En effet, il est plus intéressant de se focaliser sur la détermination des ordres d'affectation des opérations sur les machines plutôt que d'estimer les positions des opérations dans le chromosome. En plus, comme il a été indiqué dans la section de l'algorithme génétique, plusieurs chromosomes différents peuvent donner le même ordonnancement.

La transformation des chromosomes en ordonnancements peut se faire de deux manières. La première, respecte la séquence des opérations donnée par le chromosome en affectant les opérations sur les machines par un ordonnanceur de type "premier arrivé premier servi" (FIFO). Les ordonnancements obtenus sont par conséquent semi-actifs. La deuxième manière peut modifier le séquencement des opérations donné par le chromosome. L'ordonnanceur ainsi utilisé est de type "décalage à gauche". Il permet de remplir des intervalles de temps vides à gauche de l'opération en cours d'affectation. Ce type d'ordonnanceur améliore la qualité des solutions générées par l'algorithme génétique.

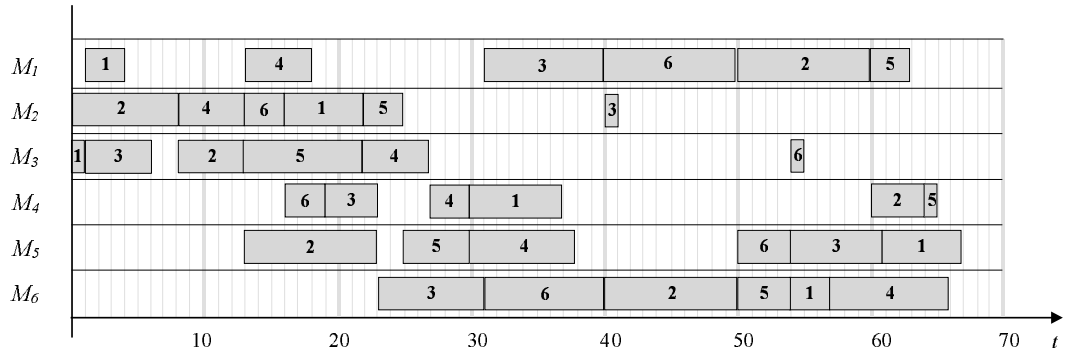
Pour illustrer ces deux ordonnanceurs, nous représentons par les figures 3.5 et 3.6 les résultats d'affectation des opérations du chromosome suivant du benchmark de Muth et Thomson :

(2,1,4,3,1,4,2,6,6,3,2,3,1,5,5,3,6,5,4,2,5,4,1,1,6,2,5,3,4,2,4,5,6,3,6,1).

L'ordonnanceur FIFO a généré un ordonnancement semi-actif dont la valeur du critère $C_{max}$ est égale à 67. Tandis que l'ordonnanceur "décalage à gauche" a construit un ordonnancement optimal avec $C_{max} = C^*_{max} = 55$. Le décalage de l'opération 32 (deuxième opération du troisième job) sur la machine $M_4$ a permis d'avancer l'exécution de plusieurs opérations sur d'autres machines et par conséquent d'améliorer la valeur de $C_{max}$.

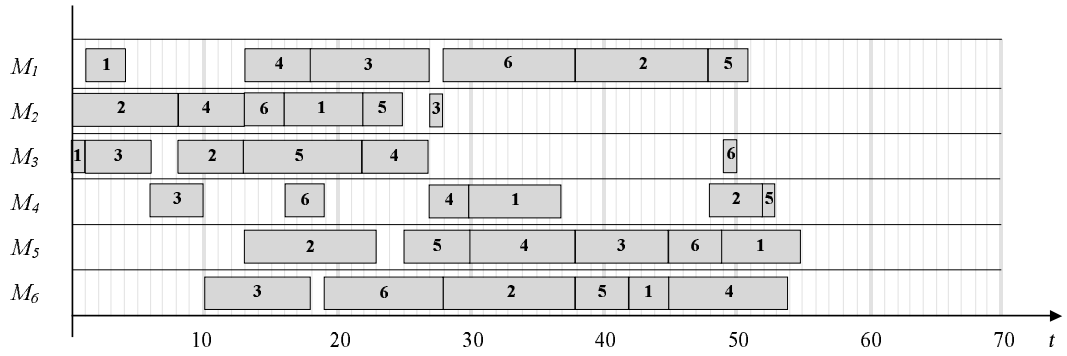
#### 3.3.2.2 Discrétisation réalisée

Pour discrétiser les attributs sélectionnés de notre problème d'ordonnancement de job shop, nous avons utilisé l'algorithme de ChiMerge. Un tel algorithme marche bien avec des ensembles de données de petite et moyenne taille. Pour illustrer la démarche de




| | | | | | | |
|---|---|---|---|---|---|---|
| $M_1$ : | 1:1 (1-4) | 4:2 (13-18) | 3:4 (31-40) | 6:4 (40-50) | 2:5 (50-60) | 5:5 (60-63) |
| $M_2$ : | 2:1 (0-8) | 4:1 (8-13) | 6:1 (13-16) | 1:3 (16-22) | 5:2 (22-25) | 3:5 (40-41) |
| $M_3$ : | 1:1 (0-1) | 31 (1-6) | 2:2 (8-13) | 5:1 (13-22) | 4:3 (22-27) | 6:6 (54-55) |
| $M_4$ : | 6:2 (16-19) | 3:2 (19-23) | 4:4 (27-30) | 1:4 (30-37) | 2:6 (60-64) | 5:6 (64-65) |
| $M_5$ : | 2:3 (13-23) | 5:3 (25-30) | 4:5 (30-38) | 6:5 (50-54) | 3:6 (54-61) | 1:6 (61-67) |
| $M_6$ : | 3:3 (23-31) | 6:3 (31-40) | 2:4 (40-50) | 5:4 (50-54) | 1:5 (54-57) | 4:6 (57-66) |

FIG. 3.5 – *Ordonnancement FIFO*




| | | | | | | |
|---|---|---|---|---|---|---|
| $M_1$ : | 1:1 (1-4) | 4:2 (13-18) | 3:4 (18-27) | 6:4 (28-38) | 2:5 (38-48) | 5:5 (48-51) |
| $M_2$ : | 2:1 (0-8) | 4:1 (8-13) | 6:1 (13-16) | 1:3 (16-22) | 5:2 (22-25) | 3:5 (27-28) |
| $M_3$ : | 1:1 (0-1) | 31 (1-6) | 2:2 (8-13) | 5:1 (13-22) | 4:3 (22-27) | 6:6 (42-43) |
| $M_4$ : | 3:2 (6-10) | 6:2 (16-19) | 4:4 (27-30) | 1:4 (30-37) | 2:6 (48-52) | 5:6 (52-53) |
| $M_5$ : | 2:3 (13-23) | 5:3 (25-30) | 4:5 (30-38) | 6:5 (38-42) | 3:6 (42-49) | 1:6 (49-55) |
| $M_6$ : | 3:3 (10-18) | 6:3 (19-28) | 2:4 (28-38) | 5:4 (38-42) | 1:5 (42-45) | 4:6 (45-54) |

FIG. 3.6 – *Ordonnancement par "décalage à gauche"*

la discrétisation réalisée, nous utilisons le benchmark de Muth et Thomson (tableau 3.1). Nous détaillons les étapes de la discrétisation de l'attribut $TO$ (temps Opératoire). Les valeurs de cet attribut appartiennent à l'ensemble {1,3,4,5,6,7,8,9,10}. L'étape d'initialisation de l'algorithme de ChiMerge consiste à mettre chaque valeur de l'attribut dans un intervalle. Nous obtenons alors 9 intervalles différents à la première étape de discrétisa-

tion de $TO$. La matrice $A$ est obtenue en déterminant pour chaque intervalle $I_i$, $1 \leq i \leq 9$ et pour chaque classe le nombre d'opérations ayant un temps opératoire dans l'intervalle en question. La matrice $A$ correspondante à l'attribut $PT$ est la suivante :

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 2 & 0 & 1 \\ 1 & 1 & 0 & 3 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 2 & 0 & 1 & 1 \\ 0 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

L'étape suivante consiste à calculer la valeur de $\chi^2$ de chaque deux intervalles adjacents (voir tableau 3.3). Les deux intervalles ayant la valeur minimale de $\chi^2$ sont fusionnés ; en l'occurrence dans notre exemple $I_5$ et $I_6$. L'étape de fusion continue jusqu'à atteindre une valeur seuil du critère $\chi^2$. Nous avons fixé la valeur de $\chi^2_{seuil}$ à 9.23. En pratique, la valeur de $\chi^2_{seuil}$ est choisi en déterminant un taux de bonne discrétisation voulu en fonction du nombre de classes auxquelles appartiennent les exemples d'apprentissages. La valeur $\chi^2_{seuil} = 9.23$ représente un taux égale à 95% pour trois classes. La dernière itération de l'algorithme ChiMerge pour la discrétisation de l'attribut $PT$ est présentée dans le tableau 3.4. Nous obtenons ainsi les trois intervalles suivants : $I_1 = [1, 4[$, $I_2 = [4, 8[$, *et* $I_3 = [8, 10]$. Nous associons à chaque intervalle un nom symbolique.

- $I_1 = [1, 4[ \longrightarrow$ Court ;
- $I_2 = [4, 8[ \longrightarrow$ Moyen ;
- $I_3 = [8, 10] \longrightarrow$ Long.

| $\chi^2(I_1,I_2)$ | $\chi^2(I_2,I_3)$ | $\chi^2(I_3,I_4)$ | $\chi^2(I_4,I_5)$ | $\chi^2(I_5,I_6)$ | $\chi^2(I_6,I_7)$ | $\chi^2(I_7,I_8)$ | $\chi^2(I_8,I_9)$ |
|---|---|---|---|---|---|---|---|
| 5.59 | 8.83 | 4.79 | 5.33 | **4.0** | 5.0 | 4.95 | **4.0** |

TAB. 3.3 – *Discrétisation de l'attribut TO : initialisation de ChiMerge*

| $\chi^2([1, 4[, [4, 8[)$ | $\chi^2([4, 8[, [8, 10])$ |
|---|---|
| 10.84 | 9.4 |

TAB. 3.4 – *Discrétisation de l'attribut TO : résultat*

Les autres attributs sont également discrétisés en utilisant l'algorithme de ChiMerge. Les valeurs prises par ces attributs sont détaillées par des histogrammes dans la figure

3.7 pour l'attribut $TOR$ et la figure 3.8 pour les attributs $CM$ et $DTJ$. Le résultat de la discrétisation des attributs liés au temps opératoire ainsi que l'attribution des noms symboliques à chaque intervalle est montré dans les tableaux 3.5 et 3.6.

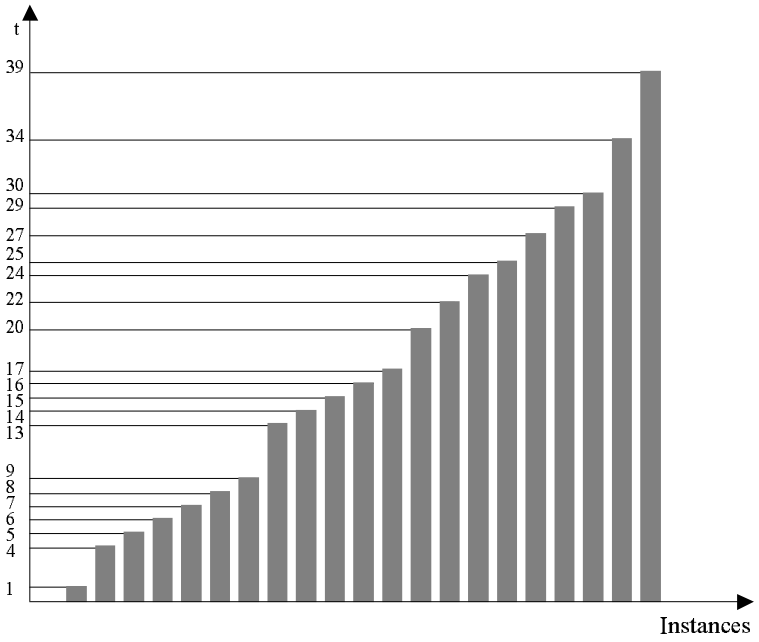



FIG. 3.7 – *Les valeurs prises par l'attribut TOR*

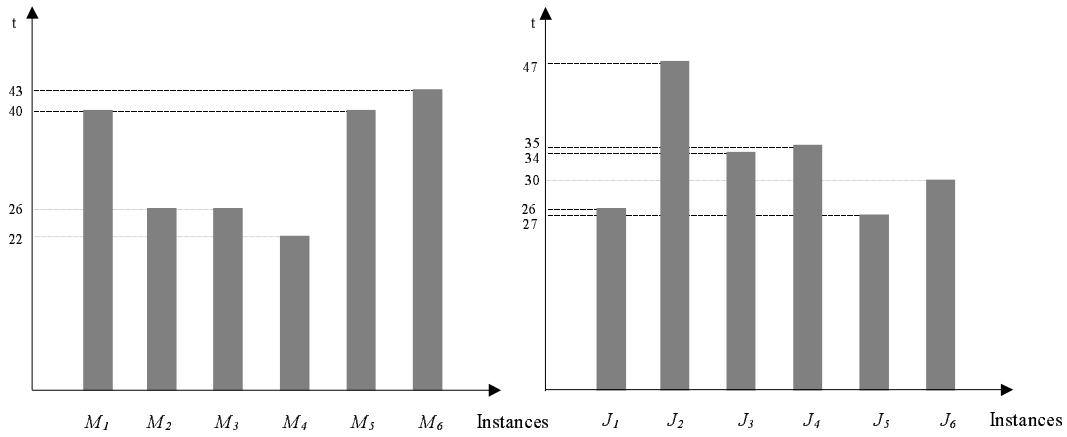



FIG. 3.8 – *Les valeurs prises par les attributs CM et DTJ*

| Attributs | Court(e) | Moyen(ne) | Long(ue) |
|---|---|---|---|
| $TO$ | [1, 4[ | [4, 8[ | [8, 10] |
| $TOR$ | [0, 14[ | [14, 27[ | [27, 39[ |
| $DTJ$ | [25, 30[ | [30, 47[ | [47, 47] |

TAB. 3.5 – *Discrétisation des attributs TO, TOR, DTJ*

| Attribut | Légère | Lourde |
|---|---|---|
| $CM$ | [22, 40[ | [40, 43[ |

TAB. 3.6 – *Discrétisation de l'attribut CM*

Le nombre de degrés de liberté est égal au nombre de classes - 1. Initialement, nous possédons six classes, à savoir les ordres d'affectation des opérations sur les machines représentées par l'attribut $OAM$. Nous avons testé plusieurs nombres de classes afin de trouver le nombre optimal qui permet un bon taux de classification. Le meilleur résultat obtenu est avec les trois classes suivantes :

- Première : cette classe contient toutes les opérations qui s'exécutent en premier ou en second ordre sur les machines ;
- Milieu : les opérations qui s'exécutent en troisième ou en quatrième position sur les machines appartiennent à la classe "Milieu" ;
- Dernière : cette dernière classe englobe les opérations s'exécutant en cinquième ou en sixième ordre.

L'attribut $POJ$ indiquant la position d'une opération donnée dans son job, est également discrétisé. Le résultat de la discrétisation de cet attribut ainsi que l'attribut $OAM$ est résumé dans le tableau 3.7.

| Attributs | Première | Milieu | Dernière |
|---|---|---|---|
| $POJ$ | 1, 2 | 3, 4 | 5, 6 |
| $OAM : Classe$ | 1, 2 | 3, 4 | 5, 6 |

TAB. 3.7 – *Discrétisation des attributs POJ et OAM*

### 3.3.3 Apprentissage à partir des solutions génétiques

Afin d'extraire des règles d'affectation des opérations sur les machines, nous avons utilisé le classificateur See5. Ce dernier permet à travers deux fichiers d'entrée, de déterminer les relations liant les caractéristiques des opérations à leurs classes d'appartenance. Le premier fichier d'entrée, ayant l'extension .names, contient les noms des attributs et leurs valeurs discrétisées ainsi que le nom de la variable classe est ses instances. Pour l'apprentissage à partir d'une solution optimale du problème de Muth et Thomson, on utilise le fichier Muth-Thomson.names suivant :

OAM.
Opération: label.
TO: Court, Moyen, Long.
TOR: Court, Moyen, Long.
DTJ: Courte, Moyenne, Longue.
CM: Légère, Lourde.
POJ: Première, Milieu, Dernière.
OAM: Première, Milieu, Dernière.

Le deuxième fichier utilisé par See5 ayant pour extension .data contient les données des exemples d'apprentissage (les opérations) relatives aux valeurs discrétisées des attributs. Les donnés que contient le fichier Muth-Thomson.data sont représentées par le tableau 3.8. Ces données sont classées par machines.

L'exécution de See5 sur ces données d'apprentissage du benchmark de Muth et Thomson a permis d'obtenir un ensemble de 6 règles d'ordonnancement de type « Si Alors ». Chaque règle est associée à un taux de classification généré avec. L'ensemble de ces règle est donné par le tableau 3.9.

Afin d'extraire le plus grand nombre de règles représentant plusieurs variantes de job shop, de même taille $6 \times 6$, nous avons généré aléatoirement 10 instances dont les données diffèrent largement. La même démarche d'extraction de règles d'ordonnancement, allant de la résolution par l'algorithme génétique à l'utilisation du See5, est appliquée. Nous avons constaté que plusieurs règles trouvées à partir du benchmark de Muth et Thomson sont aussi générées pour ces problèmes. En plus, il y a eu l'apparition de quatre autres règles supplémentaires (voir tableau 3.10).

Nous constatons que les 10 règles obtenues tiennent compte des quatre attributs suivants : TO, POJ, DTJ et TOR. L'attribut CM représentant la charge totale de chaque machine, n'est pas présent dans ces règles.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $M_1$ | 12, | Court, | Long, | Courte, | Lourde, | Première, | Première |
| | 42, | Moyen, | Long, | Moyenne, | Lourde, | Première, | Première |
| | 34, | Long, | Moyen, | Moyenne, | Lourde, | Milieu, | Milieu |
| | 64, | Long, | Court, | Moyenne, | Lourde, | Milieu, | Milieu |
| | 25, | Long, | Court, | Longue, | Lourde, | Dernière, | Dernière |
| | 55, | Court, | Court, | Courte, | Lourde, | Dernière, | Dernière |
| $M_2$ | 21, | Long, | Long, | Longue, | Légère, | Première, | Première |
| | 41, | Moyen, | Long, | Moyenne, | Légère, | Première, | Première |
| | 61, | Court, | Moyen, | Moyenne, | Légère, | Première, | Milieu |
| | 52, | Court, | Court, | Courte | Légère, | Milieu, | Milieu |
| | 13, | Moyen, | Moyen, | Courte, | Légère, | Milieu, | Dernière |
| | 35, | Court, | Court, | Moyenne, | Légère, | Dernière, | Dernière |
| $M_3$ | 31, | Moyen, | Long, | Moyenne, | Légère, | Première, | Première |
| | 11, | Court, | Moyen, | Courte, | Légère, | Première, | Première |
| | 22, | Moyen, | Long, | Longue, | Légère, | Milieu, | Milieu |
| | 51, | Long, | Court, | Courte, | Légère, | Première, | Milieu |
| | 43, | Moyen, | Moyen, | Moyenne, | Légère, | Milieu, | Dernière |
| | 66, | Court, | Court, | Moyenne, | Légère, | Dernière, | Dernière |
| $M_4$ | 32, | Moyen, | Long, | Moyenne, | Légère, | Première, | Première |
| | 62, | Moyen, | Long, | Moyenne, | Légère, | Première, | Première |
| | 44, | Moyen, | Moyen, | Moyenne, | Légère, | Milieu, | Milieu |
| | 14, | Long, | Court, | Courte, | Légère, | Milieu, | Milieu |
| | 26, | Moyen, | Court, | Longue, | Légère, | Dernière, | Dernière |
| | 56, | Court, | Court, | Courte, | Légère, | Dernière, | Dernière |
| $M_5$ | 23, | Long, | Long, | Longue, | Lourde, | Première, | Première |
| | 53, | Court, | Moyen, | Courte, | Lourde, | Première, | Première |
| | 45, | Long, | Moyen, | Moyenne, | Lourde, | Milieu, | Milieu |
| | 36, | Moyen, | Court, | Moyenne, | Lourde, | Dernière, | Milieu |
| | 65, | Court, | Court, | Moyenne, | Lourde, | Milieu, | Dernière |
| | 16, | Moyen, | Court, | Courte, | Lourde, | Dernière, | Dernière |
| $M_6$ | 33, | Long, | Long, | Moyenne, | Lourde, | Première, | Première |
| | 63, | Long, | Long, | Moyenne, | Lourde, | Première, | Première |
| | 24, | Long, | Long, | Longue, | Lourde, | Milieu, | Milieu |
| | 15, | Court, | Moyen, | Courte, | Lourde, | Milieu, | Milieu |
| | 54, | Court, | Court, | Courte, | Lourde, | Milieu, | Dernière |
| | 46, | Long, | Court, | Moyenne, | Lourde, | Dernière, | Dernière |

TAB. 3.8 – *Fichier Test.data*

Il n'est pas probablement discriminant et pertinent pour le processus de classification des opérations. Les autres attributs sont plus discriminants pour ce type particulier

| Règles $R_i$ | Expressions | Taux |
|---|---|---|
| $R_1$ | **Si** (POJ = Première) **et** (DTJ = Longue) **Alors** (Classe = Première) | 0.750 |
| $R_2$ | **Si** (POJ = Milieu) **Alors** (Classe = Milieu) | 0.933 |
| $R_3$ | **Si** (POJ = Dernière) **et** DTJ = Longue **Alors** (Classe = Milieu) | 0.750 |
| $R_4$ | **Si** (POJ = Première) **et** (DTJ = Courte) **Alors** (Classe = Milieu) | 0.667 |
| $R_5$ | **Si** (DTJ = Moyenne) **Alors** (Classe = Milieu) | 0.650 |
| $R_6$ | **Si** (POJ = Dernière) **et** ((DTJ = Courte) **ou** (DTJ = Moyenne)) **Alors** (Classe = Dernière) | 0.900 |

TAB. 3.9 – *Règles de priorité générées sur le Benchmark de Muth et Thomson*

d'application. Néanmoins, la charge machine peut être un élément déterminant pour affecter les opérations comme on va le voir dans la dernière partie de test à la fin de ce chapitre.

| Règles $R_i$ | Expressions | Taux |
|---|---|---|
| $R_7$ | **Si** (TOR = Long) **Alors** (Classe = Première) | 0.850 |
| $R_8$ | **Si** (TOR = Court) **Alors** (Classe = Dernière) | 0.900 |
| $R_9$ | **Si** (TO = Long) **et** DTJ = Longue **Alors** (Classe = Première) | 0.750 |
| $R_{10}$ | **Si** (DTJ = Longue) **et** TOR = Moyen **Alors** (Classe = Milieu) | 0.833 |

TAB. 3.10 – *Règles de priorité supplémentaires générées sur 10 problèmes* $6 \times 6$

### 3.3.4 Extraction d'une heuristique à partir des règles de priorité

Nous constatons que les règles générées sur les problèmes de taille $6 \times 6$ sont basées essentiellement sur l'attribut POJ. Ce résultat est tout à fait logique dans la mesure où l'affectation des opérations dans un ordre croissant de POJ sur une machine, permet de débloquer les opérations du même job sur les autres machines. Ce qui permet de finir le plutôt possible l'affectation de l'ensemble d'opérations. De même, l'attribut DTJ est pertinent lui aussi. En effet, si on donne une certaine priorité aux opérations des jobs longs, on peut contribuer à l'obtention d'un ordonnancement de bonne qualité. L'attribut TOR quant à lui permet d'éviter de retarder les opérations dont le temps opératoire restant de leurs jobs sont longs.

A partir de ces constatations, et en se basant sur les règles obtenues, on a voulu créer une heuristique facile à mettre en oeuvre et permettant de résoudre des problèmes de différentes tailles. Cette heuristique doit être capable d'ordonnancer les opérations sur

les machines de manière à optimiser le critère de $C_{max}$ et à résoudre les conflits pouvant exister entre des opérations possédant les même valeurs pour un attribut donné. Pour ce fait, on propose de déterminer les priorités d'affectation des opérations sur les machines selon trois critères : la position de l'opération dans son job d'abord, ensuite la durée totale d'un job et enfin le temps opératoire restant d'un job en exécutant l'une de ses opérations. L'heuristique proposée est la suivante :

***Étape 1*** : trier les opérations sur chaque machine dans un ordre croissant de l'attribut POJ. Les opérations ayant la même valeur de cet attribut sont triées dans l'étape 2 ;
***Étape 2*** : les opérations ayant la même valeur de l'attribut POJ sont triées en ordre décroissant de l'attribut DTJ. Les opérations ayant la même valeur de cet attribut sont triées dans l'étape 3 ;
***Étape 3*** : les opérations ayant la même valeur de l'attribut DTJ sont triées en ordre décroissant de l'attribut TOR ;
***Étape 4*** : si la solution fournie est admissible (respecte les contraintes de précédence), alors chercher un ordonnancement actif à partir des listes de priorité des opérations, si non aller à l'étape 5 ;
***Étape 5*** : avancer le minimum possible l'exécution des opérations qui causent un conflit. Aller à l'étape 4 pour finir.

## 3.4 Résultats expérimentaux

### 3.4.1 Validation de l'heuristique

Afin de valider cette heuristique, nous avons sélectionné un ensemble de benchmarks types de Muth et Thomson et de Lawrence. Ces problèmes de job shop possèdent des tailles différentes allant de $6 \times 6$ jusqu'à $10 \times 10$ et $20 \times 10$. Pour comparer les résultats obtenus par notre heuristique, nous citerons dans le tableau 3.11 les valeurs de $C_{max}$ obtenues pour ces mêmes problèmes par quatre approches de résolution. Ces approches sont : Recuit Simulé [LAL92], (colonne RS du tableau 3.11), Recherche Tabou [DT91] (colonne RT), Shifting Bottleneck [AED88] (colonne SB) et un algorithme génétique proposé par Della Croce [CTV95] (colonne AG).

Notre heuristique réussit à trouver une solution optimale pour le premier problème de taille $6 \times 6$. Pour les cinq autres problèmes, les valeurs de $C_{max}$ obtenues par notre heuristique sont très proches de l'optimal. Il s'avère que la dernière étape de cette démarche d'ECD, généralise, en effet la résolution du problème à des tailles supérieures,

mais ne garantit pas une solution optimale. Pour obtenir des résultats précis par cette démarche, on peut ne pas appliquer la dernière étape à savoir le post traitement consistant à extraire à partir des règles obtenues une heuristique. Dans ce cas, il est évident d'appliquer les règles uniquement à des problèmes de job shop de même taille que celui utilisé pour l'apprentissage. Donc l'utilisation ou non de la dernière étape de l'ECD doit répondre à un compromis entre la précision souhaitée et la généralisation.

| Benchmark | $n$ | $m$ | $OPT$ | $RS$ | $RT$ | $SB$ | $AG$ | Heuristique |
|---|---|---|---|---|---|---|---|---|
| $MT06$ | 6 | 6 | 55 | 55 | 55 | 55 | 55 | 55 |
| $MT10$ | 10 | 10 | 930 | 930 | 935 | 930 | 946 | 935 |
| $MT20$ | 20 | 5 | 1165 | 1165 | 1165 | 1178 | 1178 | 1175 |
| $La01$ | 10 | 5 | 666 | 666 | 666 | 666 | 666 | 673 |
| $La06$ | 15 | 5 | 926 | 926 | 926 | 926 | 926 | 936 |
| $La11$ | 20 | 5 | 1222 | 1222 | 1222 | 1222 | 1222 | 1230 |

TAB. 3.11 – *Comparaison entre les heuristiques*

### 3.4.2 Application de la démarche de l'ECD sans l'étape de post traitement

Pour résoudre des problèmes de tailles plus grandes que $6 \times 6$, nous avons appliqué la même démarche de l'ECD à l'exception de la dernière étape de post-traitement au benchmark de Lawrence connu par le nom La06. Ce problème contient 15 jobs et 5 machines. Après avoir déterminé une solution optimale de $C_{max}$ égale à 926 par l'algorithme génétique, nous avons procédé à la discrétisation des attributs. Cette étape nous a permis de déterminer trois classes d'opérations sur chaque machine. La première classe contient les opérations qui s'exécutent en cinq premières positions, la deuxième classe contient les opérations s'exécutant entre la sixième et la dixième position et les opérations restantes appartiennent à la dernière classe. Les résultats de la discrétisation sont résumés dans les tableaux 3.12, 3.13 et 3.14.

| Attributs | Court(e) | Moyen(ne) | Long(ue) |
|---|---|---|---|
| $TO$ | [7, 19] | ]19, 50] | ]50, 98] |
| $TOR$ | [0, 44] | ]44, 105] | ]105, 320] |
| $DTJ$ | [186, 233] | - | ]233, 413] |

TAB. 3.12 – *Discrétisation des attributs TO, TOR, DTJ : cas du benchmark La06*

| Attribut | Légère | Moyenne | Lourde |
|---|---|---|---|
| $CM$ | [726, 792] | ]792, 859] | ]859, 926[ |

TAB. 3.13 – *Discrétisation de l'attribut CM : cas du benchmark La06*

| Attributs | Première | Milieu | Dernière |
|---|---|---|---|
| $POJ$ | 1 | 2, 3 | 4, 5 |
| $OAM : Classe$ | 1, 2, 3, 4, 5 | 6, 7, 8, 9, 10 | 11, 12, 13, 14, 15 |

TAB. 3.14 – *Discrétisation des attributs POJ et OAM : cas du benchmark La06*

L'utilisation du See5 a permis de générer huit règles légèrement différentes à celles découvertes pour le benchmark de Muth et Thomson. Nous distinguons l'apparition de l'attribut CM représentant la charge totale de chaque machine. En revanche, l'attribut TOR n'apparaît dans aucune règle. Le tableau 3.15 résume les règles obtenues.

| $R_i$ | Expression | $\tau$ |
|---|---|---|
| $R_1$ | Si (POJ = Première) Alors (Classe = Première) | 0.824 |
| $R_2$ | Si (POJ = Dernière) Alors (Classe = Dernière) | 0.688 |
| $R_3$ | Si (POJ = Milieu) et (TO = Courte) Alors (Classe = Première) | 0.857 |
| $R_4$ | Si (TO = Longue) et (DTJ = Courte) Alors (Classe = Première) | 0.800 |
| $R_5$ | Si (POJ = Milieu) et (TO = Moyenne) et (DTJ = Courte) Alors (Classe = Milieu) | 0.857 |
| $R_6$ | Si (POJ = Milieu) et (TO = Longue) et (DTJ = Longue) Alors (Classe = Milieu) | 0.643 |
| $R_7$ | Si (POJ = Milieu) et (TO = Moyenne) et (CM = Moyenne) Alors (Classe = Milieu) | 0.714 |
| $R_8$ | Si (POJ = Milieu) et (TO = Moyenne) et (DTJ = Longue) et (CM = Légère) Alors (Classe = Première) | 0.750 |

TAB. 3.15 – *Règles générées à partir du benchmark La06*

Pour résoudre des problèmes de job shop de même taille que La06 on utilise les huit règles générées. Ces règles seules, ne peuvent pas résoudre les conflits entre les opérations devant chaque machine. En effet, le taux de classification n'est pas très élevé pour pouvoir généraliser ces règles à d'autres problèmes. Pour éviter ce problème nous procédons comme suit :

- ***Étape 1*** : classer les opérations de chaque machine en trois classes en utilisant les règles du tableau 3.15. Les classes peuvent ne pas contenir le même nombre d'opérations ;
- ***Étape 2*** : classer les opérations à l'intérieure de chaque classe suivant les valeurs croissantes de POJ. Les opérations ayant la même valeur de POJ sont classées en fonction de TO croissant ;
- ***Étape 3*** : si la solution obtenue est admissible (respecte les contraintes de précédence), alors chercher un ordonnancement actif à partir des listes de priorité des opérations, si non aller à l'étape 4 ;
- ***Étape 4*** : avancer le plus tôt possible l'exécution des opérations qui causent un conflit. Aller à l'étape 3 pour finir.

Pour valider cette deuxième heuristique, nous avons choisi de tester quatre benchmark de Lawrence tous de même taille $15 \times 5$ : La07, La08, La09 et La10. Nous avons obtenu pour les quatre problèmes des solutions optimales avec les valeurs respectives suivantes de $C_{max}$ : 890, 863, 941 et 958.

## 3.5 Conclusion

Dans ce chapitre, nous avons présenté le processus d'Extraction de Connaissance à partir des Données (l'ECD) composé d'une suite de traitements allant de l'acquisition des données jusqu'à l'extraction des structures sous jacentes. Nous avons adapté ce processus étape par étape, à la résolution de problèmes d'ordonnancement dans un atelier de type job shop. La première étape a permis d'obtenir, grâce à un algorithme génétique, un ensemble de bonnes solutions, résolvant le problème de job shop. Ces solutions ont été représentées par trois classes (chaque classe correspond à un ensemble de positions des opérations sur la machine). Lors de la deuxième étape, celle du pré-traitement, nous avons transformé les solutions fournies par l'algorithme génétique sous forme de chromosomes, en un ensemble d'ordres de passage sur les machines. Dans le but d'expliquer les classes, plusieurs caractéristiques telles le temps opératoire et la charge machine, ont été choisies. Ces caractéristiques ont été discrétisées afin de généraliser l'application des résultats. L'étape de fouille de données a permis de générer un ensemble de règles d'ordonnancement à partir des solutions fournies par l'algorithme génétique. Ces règles ont été la base d'une heuristique élaborée lors de la phase de post-traitement. L'heuristique a permis de générer de bonnes solutions pour un ensemble de test composé de problèmes types de job shop.

Si l'on désire avoir des résultats plus précis, on utilisera les règles d'ordonnancement issues directement de la troisième étape de l'ECD, mais sur des problèmes de job shop de même taille, travail fait sur un problème de taille $15 \times 5$ et testé sur quatre benchmarks. Les résultats expérimentaux ont montré l'efficacité de cette deuxième méthode de résolution. Par contre, si l'on désire avoir une solution approchée, rapide, sur des problèmes de tailles différentes, on applique l'heuristique construite à la quatrième étape.

Dans ce chapitre, nous avons cherché à résoudre le problème d'ordonnancement dans un atelier de type job shop, en considérant les machines tout le temps disponibles. Cette hypothèse ne colle pas souvent à la réalité. En effet, les machines sont sujettes à des arrêts fréquents à cause des pannes aléatoires ou des interventions d'entretien et de maintenance préventive. Nous prenons en compte l'aspect de la maintenance dans l'atelier de type job shop dans le chapitre suivant, en proposant un algorithme génétique multiobjectif.

# Chapitre 4

# L'ordonnancement conjoint de la production et de la maintenance dans un atelier job shop

***Résumé :*** *Une grande partie de la littérature dédiée aux problèmes d'ordonnancement se place dans le contexte de disponibilité totale des ressources. Cette hypothèse n'est pourtant pas fidèle à la réalité des ateliers de production. En effet, les différentes ressources qu'elles soient humaines ou matérielles peuvent, pour diverses raisons, être indisponibles et l'on peut même planifier leurs périodes d'indisponibilité. Les dates et les durées d'indisponibilité sont déterministes dans certains cas, comme les congés de personnel, les opérations de maintenance préventive sur les machines de l'atelier. Ces même données, dans d'autres cas, peuvent être imprévisibles et aléatoires, comme par exemple les pannes de machines, la maladie d'une ressource humaine etc. La présence de ces intervalles d'indisponibilité des ressources perturbe l'ordonnancement de la production et influe d'une manière significative sur les valeurs des critères d'optimisation. Dans ce chapitre, nous tenons compte des contraintes d'indisponibilité des machines au sein du job shop. Ceci nous amène à étudier la maintenance et ses différentes formes et niveaux d'interventions ainsi que l'ordonnancement conjoint. Un état de l'art de cet aspect sera présenté. Vue la complexité du job shop, nous avons opté pour l'étude de la maintenance préventive systématique. Nous proposons ensuite deux méthodes de choix de périodes systématiques. Nous avons recours à l'optimisation multiobjectif et aux algorithmes génétiques pour proposer des solutions pareto optimales. Ces solutions sont validées par des bornes inférieures.*

## 4.1 Introduction

La maintenance et la production sont deux fonctions qui agissent sur les mêmes ressources. Cependant l'ordonnancement de leurs activités respectives se fait en pratique d'une manière indépendante. En effet, la production oeuvrant quotidiennement sur les équipements, tient compte pour planifier ses tâches de contraintes de délais, de stocks d'approvisionnement etc, et néglige les contraintes liées à la maintenance. Généralement, il existe un ordonnancement de la production, destiné à satisfaire les contraintes de coût, qualités et délais. D'autre part, un plan de maintenance, visant à ordonnancer et à affecter les ressources aux tâches de maintenance, pour garantir le bon état de marche de la ressource, doit aussi être effectué. Ces deux éléments ayant été établis séparément, leurs intégrations dans le fonctionnement de l'atelier posent un problème qui est souvent résolu par négociation entre les responsables respectifs des deux services et ceci de manière séquentielle.

L'ordonnancement de production et de maintenance est abordé de plusieurs façons. Trois politiques d'ordonnancement ont été recensées, l'ordonnancement séparé, le séquentiel et l'intégré. Chaque politique a pour objectif le partage de ces ressources communes, et la garantie du bon état de fonctionnement des équipements, et enfin l'évitement de tout conflit provenant des interactions entre la production et la maintenance [LC00].

- **ordonnancement séparé** : actuellement la maintenance et la production sont le plus souvent traitées de manière indépendante au sein de l'entreprise [Bem02]. Les ordonnancements correspondants à ces deux activités sont donc réalisés de manière séparée et interfèrent bien souvent l'un avec l'autre entraînant des retards de la production ou de la maintenance. Cette méthode implique la mise en place d'une communication accrue entre les services de production et de maintenance pour limiter les conflits dans l'immobilisation des ressources aussi bien humaines que matérielles ;
- **ordonnancement séquentiel** : cette politique consiste à planifier l'une des deux activités, production ou maintenance, et à utiliser cet ordonnancement comme une contrainte supplémentaire d'indisponibilité des ressources dans la résolution du problème d'ordonnancement de l'ensemble des deux types de tâches. De manière générale, la maintenance est planifiée en premier, ensuite l'ordonnancement de la production est réalisé en prenant les opérations de maintenance comme des contraintes fortes d'indisponibilité des ressources [Agg02] ;
- **ordonnancement intégré** [SS00], [BFP96] : cette politique consiste à créer un or-

donnancement conjoint et simultané des tâches de production et de maintenance. Une telle politique de planification limite les risques d'interférence entre la production et la maintenance et permet ainsi d'optimiser la qualité des ordonnancements. Cependant, cette politique n'est actuellement qu'au stade de recherche et de test, vue la différence de caractérisation des tâches de production et de maintenance. Néanmoins, elle offre un bon espoir de voir un jour disparaître les conflits d'utilisation des ressources, en impliquant une bonne coordination entre les services de production et de maintenance.

Des études récentes sur l'efficacité de la gestion en maintenance [Bem02], [LC00] ont montré qu'un tiers des coûts de maintenance provient d'opérations inutiles ou mal effectuées. Cette inefficacité a pour raison principale, l'absence d'informations réelles qui permettraient de développer un modèle de maintenance préventive capable de réduire ou d'éliminer les interventions inutiles et d'éviter les risque de pannes les plus graves des machines.

Notre travail s'inscrit dans ce souci d'optimiser l'ordonnancement de production et de maintenance en anticipant tout conflit pouvant se présenter entre ces deux services. Nous proposons dans ce chapitre une étude sur l'ordonnancement conjoint de la production et de la maintenance au sein du job shop. Cette étude passe par un compromis entre les objectifs, parfois antagonistes de ces deux fonctions. Nous proposons donc une méthode d'ordonnancement s'appuyant sur l'optimisation multiobjectif comportant une fonction liée à la production et une autre liée à la maintenance. Le job shop est pratiquement l'atelier de production le plus proche de la réalité du monde industriel. La complexité de ce problème a obligé les chercheurs à simplifier le plus possible les contraintes réelles qui peuvent exister. La contrainte de disponibilité des machines a été, la plupart de temps relaxée comme dans l'étude que nous avons faite au chapitre 2 pour pouvoir comparer notre apport par rapport aux travaux existants. Cette contrainte a une influence considérable sur la qualité de l'ordonnancement et sur sa réalisation. La prise en compte de l'état de disponibilité ou de non disponibilité des machines pendant des intervalles réguliers de temps permet de se rapprocher encore plus de la réalité du terrain. Ce que nous nous proposons de faire dans ce chapitre, qui débute par un rappel sur les notions de maintenance, suivi d'un état de l'art sur l'ordonnancement conjoint de la production et la maintenance. Notre méthode de résolution sera présentée au paragraphe 4.3.2, puis sera validée sur des benchmarks de job shop que l'on a modifié pour tenir compte de la maintenance.

## 4.2 Généralités sur la maintenance

Les entreprises industrielles, face à la concurrence, sont de plus en plus confrontées aux impératifs de la modernisation des appareils productifs, de la qualité des produits et de la réduction de leurs coûts. Bien que la modernisation ait plutôt une dimension technique et technologique, il semble primordial actuellement de réaliser des investissements revêtant des aspects économiques, humains et organisationnels, afin d'assurer la rentabilité des investissements matériels (machines, équipements de production). Un des axes majeurs permettant d'atteindre ces objectifs, réside dans la conservation de l'état des moyens matériels investis. En effet, la robotisation des systèmes de production, de plus en plus grande, oblige à accorder une forte importance à leur fiabilité. Ainsi un arrêt pour cause de panne de l'équipement de production, peut engendrer des coûts énormes à l'entreprise (les taux de pannes pénalisent la rentabilité). En particulier, pour des raisons de sécurité, il est primordial de s'intéresser aux problèmes de fiabilité et de disponibilité. On constate ainsi que la maintenance connaît de plus en plus une promotion accrue (embauche de personnel qualifié, mise en place de moyens au niveau des services de maintenance, etc).

**Définition 4.1.** *L'Association Française de Normalisation : L'AFNOR, a défini sous la norme "NF X 60-010" la maintenance comme étant l'ensemble des activités destinées à maintenir ou à rétablir un bien dans un état ou dans des conditions données de sûreté de fonctionnement, pour accomplir une fonction requise. Ces activités sont ainsi une combinaison d'activités techniques, administratives et de management».*

Le terme *maintenir* contient la notion de surveillance et de prévention sur un bien en fonctionnement normal. Tandis que le terme *rétablir* contient la notion de correction (remise à niveau) après perte de fonction. La maintenance comporte aussi en plus de la gestion et de la maîtrise de la disponibilité des moyens de production d'une entreprise, l'ensemble des actions intégrant dès la conception des produits et des équipements, des notions de fiabilité, de maintenabilité de disponibilité et de sécurité. Ces quatre derniers paramètres définissent la notion de *sûreté de fonctionnement.* La sûreté de fonctionnement est l'ensemble des aptitudes d'un bien qui lui permettent de remplir sa fonction au moment voulu, pendant la durée prévue, sans dommage pour lui même et pour son environnement.

**Définition 4.2.** *La fiabilité est l'aptitude d'une entité à accomplir une fonction requise, dans les conditions données, pendant un intervalle de temps donné. (AFNOR NF X 60-500)*

**Définition 4.3.** *La maintenabilité est l'aptitude d'un dispositif à être maintenu ou rétabli dans un état dans lequel il peut accomplir sa fonction requise lorsque la maintenance est accomplie dans des conditions données avec des procédures et des moyens prescrits.(AFNOR NF X 60-010)*

### 4.2.1 Les politiques de maintenance

Selon l'objectif et le moment d'intervention, on peut distinguer deux grandes classes de politiques de maintenance : la maintenance préventive et la maintenance corrective (figure 4.2).

#### 4.2.1.1 Maintenance préventive

La maintenance préventive a pour objectif de réduire la probabilité de défaillance ou de dégradation d'un bien ou d'un service rendu. Les activités correspondantes sont déclenchées selon un échéancier établi à partir d'un nombre prédéterminé d'usage (maintenance *systématique*), et/ou des critères prédéterminés significatifs de l'état de dégradation du bien ou du service (maintenance *conditionnelle*) [Gil96]. Cette politique de maintenance s'adresse aux éléments provoquant une perte de production ou des coûts d'arrêts imprévisibles classés comme importants pour l'entreprise. Lyonnet [Lyo92] a indiqué que ce type de maintenance s'applique sur des matériels appartenant à la catégorie A d'une courbe ABC[1] (coût/nombre de pannes). Il convient donc d'organiser un système de maintenance minimisant ces arrêts tout en ne devenant pas trop onéreux.

La maintenance préventive agit sur plusieurs éléments dans l'entreprise. En effet, la planification des tâches dans le service maintenance est prédéterminée, ce qui diminue les arrêts imprévus et rend la charge de travail plus régulière. Au niveau de la sécurité, les risques de pannes ayant des conséquences catastrophiques diminuent. La fiabilité s'améliore et le taux de défaillance est artificiellement réduit, ce qui permet une meilleure disponibilité des équipements. Ce type de maintenance permet de générer une base de données pour la prise en charge ultérieure, préventive systématique ou conditionnelle ou les deux.

1. La méthode ABC, ou Loi de Paréto ou encore Règle 20/80 est une analyse basée sur l'étude d'une période écoulée permettant d'une façon simple et objective de mettre en évidence en fonction d'un critère déterminé les individus les plus marquants d'une population. Au début du siècle, un sociologue et économiste italien, Vilfredo Pareto, démontra le principe d'une inégalité de la répartition des richesses et des revenus dans une population quelconque et en déduisit une loi qui peut s'énoncer ainsi: "Quand il y a un grand nombre de variantes possibles, très souvent moins de 20% de ces variantes représentent plus de 80% des cas qui peuvent se présenter dans la réalité"

On distingue plusieurs type de maintenance préventive :

- **maintenance systématique** : ce type de maintenance comprend l'ensemble des actions destinées à restaurer, en totalité ou partiellement, la marge de résistance des matériels non défaillants, lorsque ces tâches sont décidées en fonction du temps ou de la production, sans considération de l'état des matériels à cet instant. Les tâches de maintenance préventive systématique sont alors effectuées avec une périodicité régulière, soit à intervalles fixes dans le temps, soit au bout d'un certain nombre d'heures de fonctionnement ou de kilométrage parcourus. Ce type de maintenance, comprend le remplacement systématique de certains composants critiques en limite d'expiration de leur durée de vie, le remplacement de composants peu coûteux pour éviter les dépenses d'évaluation de leur état et l'essentiel des opérations de service (remplacement d'huile, de pièces d'usure, réglage de pression, contrôle du niveau d'huile, etc). Ce type de maintenance est représenté par la figure 4.1 où $T$ représente la période d'intervention prédéterminée et $I_{PS}$ chaque intervention préventive systématique ;

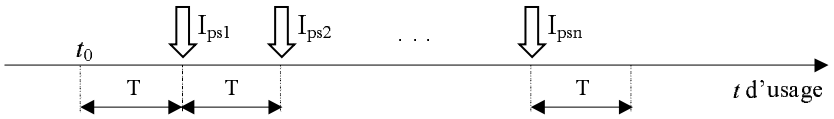



FIG. 4.1 – *Maintenance systématique*

- **maintenance conditionnelle** : il s'agit d'une maintenance préventive subordonnée à un type d'événement prédéterminé (information d'un capteur, mesure d'une usure, mesure des vibrations et du bruit, mesure de température, analyse des huiles, etc) pour restaurer du matériels ou de composants non défaillants. Les remplacements ou les remises en état des pièces, les remplacements ou les appoints des fluides auront lieu après une analyse de leur état de dégradation. Il apparaît immédiatement que ce type de maintenance préventive requiert des tâches additionnelles pour évaluer le niveau de dégradation. Selon Gilles [Gil96], Ces tâches sont considérées comme appartenant à la maintenance conditionnelle, car elles sont au coeur de la procédure de décision, bien que la plupart d'entre elles soient effectuées selon une programmation régulière ;
- **maintenance prédictive** : appelée aussi maintenance prévisionnelle, elle prédit la possibilité d'une défaillance à une certaine échéance, à partir d'une analyse permanente ou périodique de l'état de chaque équipement obtenue par les méthodes

de la maintenance conditionnelle. Ainsi les tâches de maintenance peuvent être planifiées au moins dans les limites du délai prédit.

D'autres types de maintenance préventive apparaissent et deviennent de plus en plus intéressants. Nous citons à titre d'exemple la maintenance basée sur la fiabilité qui utilise un ensemble de méthodes structurées et formelles pour sélectionner les tâches efficaces et applicables de maintenance préventive pour atteindre un niveau de fiabilité accepté. La maintenance proactive utilisée essentiellement par les industriels et prestataires de service en maintenance aux États Unis, est une autre forme avancée de maintenance prédictive. Elle permet de déterminer les causes initiales des défaillances à partir de l'état de défaillance potentielle. La mise en oeuvre de cette maintenance implique l'accumulation d'un retour d'expérience de haute qualité.

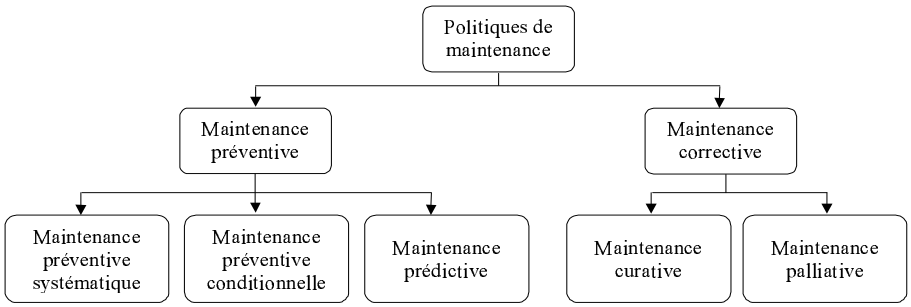



FIG. 4.2 – *Les différentes politiques de maintenance selon AFNOR*

#### 4.2.1.2 Maintenance corrective

Avant que l'AFNOR ne définisse la maintenance corrective comme étant «l'ensemble des activités réalisées après la défaillance», les termes de maintenance subite, fortuite, réparatrice, palliative, curative étaient utilisés. Ce type de maintenance agit sur des équipements défaillants ou après dégradation de leurs fonctions pour leurs permettre de reprendre leurs activités, au moins provisoirement. La maintenance corrective comporte notamment la localisation de la défaillance et son diagnostic, la remise en état avec ou sans modification, le contrôle du bon fonctionnement.

La définition précédente nous conduit à distinguer deux types de maintenance corrective : la maintenance palliative et la maintenance curative.

– **maintenance palliative** : activités de maintenance corrective destinées à permettre à un bien d'accomplir provisoirement toute ou partie d'une fonction requise.

Appelée couramment dépannage, cette maintenance palliative est principalement constituée d'actions (interventions) à caractère provisoire qui devront être suivies d'actions curatives ;

- **maintenance curative** : activités de maintenance corrective ayant pour objectif de rétablir un bien dans un état spécifié ou de lui permettre d'accomplir une fonction requise. Le résultat des activités réalisées doit présenter un caractère permanent. Ces activités peuvent être des réparations, des modifications ou aménagements ayant pour objet de supprimer les défaillances.

### 4.2.2 Les niveaux de maintenance

Plusieurs classifications des niveaux de maintenance existent selon les secteurs industriels. Gilles a cité dans [Gil96] les trois niveaux de maintenance dans le milieu aéronautique militaire. Le premier niveau, appelé aussi la maintenance en ligne, concerne les visites journalières avant et après le vol, les tests en piste etc. Le deuxième niveau s'agit d'une maintenance hors ligne et regroupe les opérations d'entretien de certains équipements en atelier, le contrôle et la remise en état des systèmes. Le dernier niveau, appelé aussi la maintenance industrielle, est réalisé dans des ateliers industriels spécialisés. Il concerne les visites d'entretien majeures de l'avion, la remise en état des avions accidentés, la réparation et la révision générale des équipements.

Une autre classification normalisée par niveaux de maintenance est donnée par l'AFNOR, pour servir comme un guide entre les partenaires selon le type de bien à maintenir. Une politique de maintenance bien définie, doit selon la même source clairement identifier cinq niveaux de maintenance réalisés à l'intérieur de l'entreprise et ceux confiés à des entreprises de sous-traitance ou à des constructeurs. Il permettent, en outre, d'identifier le niveau de diagnostic auquel on s'intéresse : systèmes, sous-systèmes, matériels, composants élémentaires.

Le niveau 1 regroupe les réglages simples prévus par le constructeur au moyen d'éléments accessibles sans aucun démontage ou ouverture de l'équipement, ou échanges d'éléments consommables accessibles en toute sécurité, tels que voyants ou certains fusibles, etc. Ce type d'intervention peut être confié à l'exploitant du bien, sur place, sans outillage et à l'aide des instructions d'utilisation.

Le niveau 2 correspond à des dépannages par échange standard des éléments prévus à cet effet et à des opérations mineures de maintenance préventive, telles que graissage ou contrôle de bon fonctionnement. Ce type d'intervention peut être effectué par un

technicien habilité de qualification moyenne, sur place, avec l'outillage portable défini par les constructions. Des pièces de rechange nécessaires peuvent être utilisées à condition qu'elles soient transportables, sans délai et à proximité du lieu d'exploitation.

Le niveau 3 réunit l'identification et le diagnostic des pannes, les réparations par échange de composants ou d'éléments fonctionnels, les réparations mécaniques mineures et toute opération courante de maintenance préventive telles que réglage général ou réalignement des appareils de mesure. Ce niveau de maintenance un peu élevé, est confié à des techniciens spécialisés, sur place ou dans le local de la maintenance, à l'aide de l'outillage prévu dans les instructions de maintenance ainsi que des appareils de mesure et de réglage, des bancs d'essai et de contrôle des équipements et utilisant l'ensemble de la documentation nécessaire à la maintenance du bien, ainsi que les pièces approvisionnées par le magasin.

Le niveau 4 englobe les travaux importants de maintenance corrective ou préventive à l'exception de la rénovation et de la reconstruction. Ce niveau comprend aussi les réglages des appareils de mesure utilisées pour la maintenance, et éventuellement de bancs de mesure et étalons de travail par les organismes spécialisés. Ce type d'intervention peut être effectué par une équipe comprenant un encadrement technique très spécialisé, dans un atelier spécialisé doté d'un outillage général (moyens mécaniques, des câblages, de nettoyage, etc.) et à l'aide de toute documentation générale ou particulière.

Le niveau 5 correspond à la rénovation, la reconstruction ou exécution des réparations importantes confiées à un atelier central ou à une unité extérieure. Généralement, c'est le constructeur qui effectue ce genre de maintenance par ses propres moyens.

| Niveau | Activités |
|---|---|
| Niveau 1 | Ronde, petit entretien, Graissage |
| Niveau 2 | Échange standard, contrôle de bon fonctionnement |
| Niveau 3 | Diagnostic, petites réparations, opérations mineures préventives |
| Niveau 4 | Travaux de maintenance préventive et corrective, réglage des moyens de mesure |
| Niveau 5 | Rénovation, reconstruction et réparations importantes |

Coûts croissants ↓

FIG. 4.3 – *Niveaux de maintenance AFNOR*

En règle générale, les qualifications des personnels et les coûts associés s'accroissent avec le niveau de maintenance comme le montre la figure 4.3.

Dans notre travail, nous focalisons notre intérêt sur le troisième niveau de maintenance, et d'un degré moins élevé sur le quatrième niveau. En effet, nous cherchons à planifier des tâches de maintenance préventives de courtes durées.

## 4.3 Ordonnancement conjoint de la production et de la maintenance

### 4.3.1 État de l'art

Depuis une dizaine d'années plusieurs auteurs se sont intéressés à l'ordonnancement conjoint de la maintenance et de la production. Les travaux dans ce domaine varient en fonction du type de maintenance considéré et du modèle de production étudié. Les premiers travaux considèrent une seule machine. Puis, au fur et à mesure que les travaux avancent, les modèles se compliquent pour aller jusqu'aux ateliers de type flow shop ou job shop. Nous présentons ici un état de l'art des principaux travaux sur l'ordonnancement conjoint de la production et de la maintenance classés en fonction du nombre de machines.

- **cas d'une seule machine** : Sloan et al. [SS00] ont étudié le problème d'ordonnancement conjoint sur une seule machine et plusieurs types de produits. La machine subit souvent des pannes ce qui engendre un effet négatif sur la production de plusieurs produits. Ce problème a été modélisé par le processus de décision Markovien et résolu par la programmation linéaire.

  Graves et al. [GL99] ont traité le même problème mais en donnant plus de flexibilité quant aux périodes d'intervention de maintenance. Ainsi, deux scénarios sont étudiés concernant l'horizon de production. Dans le cas où l'horizon planifié est long par rapport à la période de maintenance $T$, le problème étudié devient $\mathcal{NP}$-*complet*. Les auteurs ont développé un algorithme pseudo-polynomial basé sur la programmation dynamique. En revanche, si l'horizon de production est assez court, il est parfois impossible de continuer une tâche de maintenance. Il faut donc la terminer pendant l'horizon suivant. Ce scénario est lui aussi $\mathcal{NP}$-*complet*. Néanmoins, les règles SPT (Shortest Processing Time) et EDD (Earliest Due Date) permettent de le résoudre d'une manière exacte dans le cas de la minimisation de la somme des dates de fin ou la minimisation du maximum d'avance.

  Liao et al. [LC02], ont étudié également le cas d'une seule machine en tenant compte de la possibilité des pannes pouvant se produire sur la machine. Ainsi, des périodes d'indisponibilité de la machine sont déterminées avant de commencer l'ordonnan-

cement. Le critère à optimiser est le retard maximum. Pour résoudre d'une manière exacte ce type de problème, un algorithme de type branch and bound est proposé pour des instances de taille réduite. Afin de proposer des solutions approchées pour des instances de grande taille, les auteurs ont développé une heuristique.

Asano et al. [AO99], ont apporté un aspect économique dans leur étude de ce problème. Ces derniers ont proposé une heuristique qui permet la manipulation des dates de début et de fin des périodes d'indisponibilité des machines pour minimiser le coût total.

Un autre modèle stochastique d'une seule machine est étudié par Stadje [Sta95]. Les moments des pannes de la machine sont aléatoires et dépendent du job en cours de traitement. A chaque job est associé un gain en fonction de sa date de fin et de sa date d'échéance. Le problème consiste à choisir parmi un ensemble donné de jobs, ceux qui maximisent le gain total avant qu'une panne surgisse sur la machine ;

- **cas de deux machines** : une étude récente de Lee [Lee99], a considéré le cas de deux machines avec des périodes d'indisponibilité sur une seule machine puis sur les deux machines. Le problème étudié est un flow shop déterministe. L'objectif est de trouver un ordonnancement qui minimise le makespan et qui respecte les périodes d'indisponibilité des machines (durant ces périodes, les machines subissent des opérations de maintenance). Lee a développé un algorithme pseudo-polynomial se basant sur la programmation dynamique pour résoudre ce problème.

  Le même problème de flow shop à deux machines sous des contraintes d'indisponibilité est étudié par Allahverdi et al. [AM98]. A la différence de Lee, Allahverdi et al. considèrent le cas dynamique des pannes machines. Les périodes d'indisponibilité sont alors inconnues au début de l'ordonnancement. En plus, deux critères sont à minimiser à la fois, le makespan et le maximum des retards. Les auteurs citent deux types différents de modèles étudiés dans la littérature. Le premier, avec mémoire, permet de reprendre les tâches interrompues lors des pannes machines une fois ces dernières réparées. Le deuxième, sans mémoire, réinitialise la réalisation des tâches interrompues par une panne. Les auteurs ont gardé le premier modèle. Pour résoudre ce problème d'une manière exacte, une technique d'énumération complète est utilisée ;

- **cas de plusieurs machines** : le cas de plusieurs machines est considéré par Lee et al. [LC00]. Les machines sont parallèles et identiques et subissent chacune une intervention de maintenance durant un intervalle de temps donné. Le but est de

chercher un ordonnancement conjoint des tâches de production et de maintenance qui minimise la somme pondérée des dates de fin. Dans ce problème, les auteurs ont considéré deux cas de maintenance. Le premier, quand l'atelier dispose de ressources suffisantes pour maintenir plusieurs machines à la fois si nécessaire. Le deuxième, se base sur l'hypothèse de limitation de ressources de maintenance. Dans ce cas, une seule machine peut être maintenue en un temps donné. Les problèmes considérés sont résolus d'une manière exacte à l'aide des algorithmes de type évaluation et séparation (branch and bound). Les problèmes résolus sont de taille moyenne allant de 20 à 30 opération et de 2 à 8 machines. Les données de production et de maintenance sont générées par une distribution uniforme.

Un autre cas d'ordonnancement conjoint, mais cette fois-ci industriel, a été traité par Deniaud et al. [DNMM99]. Il s'agit de définir une planification des interventions de maintenance préventive réalisées sur une des lignes de fabrication d'Alstom. L'objectif est de minimiser l'influence des interventions de maintenance préventive sur le flux de production. Pour en juger, deux critères sont utilisés, le temps de cycle moyen et le temps de cycle maximum de fabrication des lots. La démarche adoptée passe tout d'abord par la modélisation et l'identification de la ligne de fabrication. Ensuite, le problème est étudié en deux étapes à l'aide de la simulation. La première étape consiste à déterminer la planification de la maintenance préventive de chaque îlot de machines séparément. La seconde étape consiste à étudier globalement la ligne de fabrication en intégrant les résultats issus de la première étape afin de définir la planification de la maintenance préventive.

Une modélisation du problème d'ordonnancement conjoint de la production et de la maintenance sous forme d'un programme linéaire, est proposée par Ashayeri et al. [ATS96]. Ce modèle détermine, à chaque fois qu'un nouveau job doit être réalisé, s'il vaut mieux débuter son traitement ou réaliser d'abord une intervention de maintenance préventive. Cette décision est prise en fonction des coûts de préparation induits si l'on retarde le job, du coût de la maintenance préventive et corrective, du risque de défaillance (estimé à partir du cumul des durées opératoires réalisé depuis la dernière intervention de maintenance), du niveau et des coûts de stock, etc. Le modèle proposé précise s'il faut ou non produire un certain produit sur une certaine ligne pendant une certaine période, mais n'a pas (comme beaucoup d'autres systèmes) la quantité à produire comme principale variable de décision. Pour valider ce modèle, les auteurs ont testé quatre scénarios de maintenance sur un système de production composé de deux lignes de même vitesse.

Le temps de calcul de la solution optimale est d'environ 24h, ce qui a poussé les auteurs à proposer une heuristique qui permet de trouver une bonne solution en un temps acceptable pour d'autres problèmes de taille plus grande.

D'autres études intéressantes de la planification intégrée de la production et de la maintenance sont proposées dans la littérature. Ces études font intervenir l'aspect économique de cette problématique [BFP96] en testant plusieurs types de maintenance. Par exemple, Weinstein et al. [WC99] ont étudié deux types de maintenance : la maintenance "à période fixe" effectuée à des intervalles de temps réguliers et la maintenance "à fonctionnement cumulé fixe" se basant sur l'utilisation cumulée de chaque équipement. Cette étude avait comme objectif de trouver la politique de maintenance qui minimise au mieux les défaillances des équipements et maximise leur disponibilité. Les auteurs proposent un modèle d'organisation pour mettre en oeuvre une politique de maintenance de manière à minimiser son coût global (coûts de main-d'oeuvre de maintenance, de stock, de production, etc).

Une évaluation de l'impact de l'intégration de la maintenance préventive et corrective dans le MRP (Material Requirements Planning) est présentée par Rishel et al. [RC96]. Le but du MRP est d'ordonnancer des tâches à partir de la date de fin souhaitée en considérant séparément les tâches de production et les opérations de maintenance sans se soucier des conflits qui risquent de se poser au niveau des machines. Cette évaluation est réalisée à travers quatre indicateurs : le nombre de jobs traités, le nombre d'interventions de maintenance programmées effectivement réalisées, le nombre de pannes et le coût total de la maintenance. Cette étude est validée sur trois machines parallèles ;

– **cas du job shop** : la maintenance est peu étudiée dans le cas du job shop. La complexité de ce problème rend la tâche encore plus difficile si l'on tient compte des contraintes de disponibilité des machines. Banerjee et al. [BB90] sont probablement les premiers à avoir étudié l'ordonnancement conjoint des tâches de production et de maintenance dans ce type particulier d'atelier. Les auteurs ont proposé un modèle de simulation pour résoudre un job shop dynamique constitué de quatre groupes différents de machines. Chaque groupe est composé de trois machines semblables mais pas identiques. L'atelier tourne 24h/24h sans arrêt sauf s'il y a un problème de pannes des machines. Les jobs arrivent selon la loi de poisson avec un nombre d'opérations déterminé aléatoirement par la loi uniforme. Dans ce modèle, les pannes machines sont introduites par une loi de probabilité. La maintenance préventive est aussi considérée en testant plusieurs périodes systématiques.

Récemment, Aggoune [Agg02] a étudié le problème d'ordonnancement de type job shop à deux jobs, avec la prise en compte de contraintes de disponibilité des machines. Dans son modèle, il considère que les machines peuvent être indisponibles durant certaines périodes, dont les dates et durées sont fixes et connues a priori. Il suppose en effet, que les machines sont soumises à des activités de maintenance préventive et chacune d'elles peut subir plusieurs tâches de maintenance. L'objectif d'une telle étude est de déterminer une séquence d'entrée des opérations sur les machines minimisant le makespan. Pour résoudre ce problème, l'auteur a développé une méthode exacte et polynomiale basée sur l'approche géométrique et qui a permis de transformer le problème initial en recherche de plus court chemin.

Dans notre cas d'étude, nous avons choisi d'appliquer la maintenance préventive systématique au cas du job shop. En effet, la maintenance préventive si elle est bien planifiée, permet d'augmenter la disponibilité des ressources et par conséquent d'améliorer la productivité. Afin d'évaluer la qualité de la maintenance, les entreprises utilisent souvent le critère de coût. Ce critère est corroboré par la littérature, en effet, plus que la moitié des articles étudiés utilisent le critère de coût. Quant au choix des périodes de maintenance préventive, la littérature propose souvent des périodes fixes pré-déterminées. Pour notre cas d'étude, nous avons opté pour la maintenance préventive systématique avec une génération déterministe des périodes avec une tolérance d'affectation permise avant ou après la période. Pendant ces intervalles de tolérance, nous considérons que le coût de maintenance est nul. Ces intervalles permettront plus de flexibilité à la planification de la maintenance en cas de besoin. Souvent, la maintenance préventive est réalisée pendant des intervalles périodiques indépendants de la charge réellement effectuée par la machine en question. Afin de comparer l'impact de chaque type de période de maintenance, nous allons les comparer à travers des benchmarks types de job shop. Enfin, concernant les critères d'optimisation utilisés pour l'ordonnancement conjoint, la plupart des travaux cités aux paragraphes précédents ne considèrent à la fois que des critères de production ou de maintenance. Nous voulons dans notre étude considérer deux types de critère, l'un concerne la production et l'autre la maintenance. Pour cela, nous continuons à travailler avec le critère $C_{max}$ qui est très utilisé dans la littérature. Concernant la maintenance, le critère choisi est le coût total des retards et/ou des avances des tâches de maintenance.

### 4.3.2 La méthode de résolution proposée

Le problème étudié dans cette section est l'ordonnancement conjoint de la production et de la maintenance au sein du job shop. C'est le même problème traité au chapitre précédent auquel on rajoute les contraintes de maintenance. Nous cherchons à optimiser

| $N^{bre}$ de machines | Auteurs | Type d'ordonnancement | Critère(s) | Méthode de résolution |
|---|---|---|---|---|
| 1 | Sloan et al. 00 | intégré | gain moyen dû au bon fonctionnement | processus de Markov + programmation dynamique |
| 1 | Graves et al. 99 | séquentiel | $\sum C_i$ et $E_{max}$ | programmation dynamique + règles SPT et EDD |
| 1 | Liao et al. 02 | séquentiel | $T_{max}$ | B & B + heuristique |
| 1 | Asano et al. 99 | séquentiel | coût total | heuristique |
| 1 | Stadje 95 | intégré | gain total | chaînes de Markov |
| 2 machines (flow shop) | Lee 99 | séquentiel | $C_{max}$ | programmation dynamique |
| 2 machines (flow shop) | Allahverdi et al. 98 | intégré | $C_{max} + T_{max}$ | énumération complète |
| machines parallèles flexibles | Brandolese 96 | intégré | respect des $r_j$, des $d_j$, coût total, temps total | système expert, B & B |
| machines parallèles | Lee et al. 00 | intégré | $\sum w_i C_i$ | B & B |
| plusieurs machines | Deniaud et al. 99 | séquentiel | temps de cycle | Simulation |
| plusieurs machines | Ashayeri et al. 96 | intégré | coûts (maintenance, production, stock) | programmation linéaire |
| plusieurs machines | Rishel 96 | séparé | nbre de pannes, coût total de maintenance | MRP |
| job shop | Banerjee 90 | intégré | $\overline{F}$, nbre de pannes, nbre de tâches de maintenance, ... | simulation |
| job shop | Aggoune 02 | séquentiel | $C_{max}$ | approche géométrique |

TAB. 4.1 – *État de l'art de l'ordonnancement de la production et de la maintenance*

deux critères simultanément : le $C_{max}$ et le coût total de la maintenance noté $CM$. L'association des opérations de maintenance aux opérations de production nous amène à utiliser une nouvelle notation du problème à savoir : $J//C_{max}$,$CM$.

Pour résoudre ce problème, nous proposons une approche basée sur l'algorithme génétique développé dans le chapitre 3. Des modifications sont apportées à cet algorithme pour tenir compte des opérations de maintenance à planifier. Le choix des algorithmes génétiques est justifié par leur capacité de génération de plusieurs solutions Pareto optimales en une seule exécution. Cette capacité est due au parallélisme inhérent qui ca-

ractérise les algorithmes génétiques ainsi qu'à leur pouvoir d'exploiter les similarités des solutions par l'opérateur de croisement. En plus, les algorithmes génétiques sont très bien adaptés au traitement d'un problème d'optimisation multiobjectif.

#### 4.3.2.1 Algorithme génétique multiobjectif

Nous développons dans cette section un algorithme génétique multiobjectif utilisé pour générer un ensemble de solutions Pareto optimales du problème d'ordonnancement conjoint de la production et de la maintenance. Pour ce fait, nous avons adapté l'algorithme génétique développé au troisième chapitre à la résolution de ce problème. Plusieurs modifications ont été introduites. Ces modifications concernent le codage, la fonction objectif, la sélection et les valeurs des paramètres génétiques. Nous décrivons par la suite les opérateurs utilisés par cet algorithme génétique.

- **codage** : le même codage réel basé sur les opérations du chapitre 3 est utilisé pour générer des chromosomes valides. Un chromosome est ainsi une chaîne de nombres entiers représentant une combinaison des opérations de production et de maintenance. Les opérations de production d'un même job sont représentées par le même numéro. Les tâches de maintenance d'une même machine $M_k$ reçoivent toutes un numéro spécial, par exemple $k+n$ où $n$ est le nombre de jobs. A titre d'exemple, pour un problème de job shop de taille $3 \times 3$ (3 jobs et 3 machines) dont chaque machine doit subir deux opérations de maintenance, un chromosome peut être identique à (1,2,1,3,4,6,5,1,3,2,6,4,2,3,5). Les nombres 1, 2 et 3 représentent les opérations des jobs et les nombres 4, 5 et 6 représentent respectivement les opérations de maintenance des machines $M_1$, $M_2$ et $M_3$. La répétition d'un même nombre plusieurs fois indique l'ordre d'affectation des opérations d'un même job ou l'ordre d'affectation des opérations de maintenance sur la même machine ;
- **population initiale** : une fois le codage choisi, une population initiale formée de solutions admissibles du problème doit être déterminée. Nous utilisons une génération aléatoire d'une population non homogène de 800 individus ;
- **fonctions d'évaluation** : afin de mesurer la performance d'un individu vis à vis de chaque critère, on utilise directement les critères comme fonction d'évaluation. Ainsi, $f_1 = C_{max}$ et $f_2 = CM$ seront les fonctions *fitness* de notre algorithme. Notons que ces deux fonctions sont liées à un ordonnanceur actif permettant d'ajuster les opérations des séquences et de retourner les valeurs de $f_1$ et de $f_2$ associées ;
- **sélection** : Nous avons combiné deux manières de sélection des chromosomes à reproduire : la première consiste à trier les individus en utilisant l'approche MOGA

afin de retenir les 250 meilleurs. La deuxième permet de tirer aléatoirement 250 individus en se basant sur le principe de la roulette de Goldberg [Gol89]. Nous allons expliquer le principe de la sélection appliquée dans notre cas, à travers un exemple d'une population composée de 10 individus dont les valeurs des fonctions $f_1$ et $f_2$ sont données aux deux premières colonnes du tableau 4.2.

Nous déterminons tout d'abord en utilisant l'approche MOGA, le rang $r_i$ de chaque individu $i$. Ce rang $r_i$ est obtenu en calculant le nombre de solutions dominant l'individu $i$ (voir la dernière colonne du tableau 4.2). Nous trions les individus en ordre croissant des valeurs $r_i$ afin de garder les meilleurs pour la phase de croisement et de mutation.

Nous avons introduit l'aspect aléatoire afin de sélectionner une partie de la population courante tout en donnant une chance aux individus faibles d'être sélectionnés. La roue de loterie constitue un moyen très efficace pour réaliser cette tâche. Nous déterminons pour cela la probabilité de sélection de chaque individu en fonction de sa valeur de $r_i$. On commence tout d'abord par définir la variable $\bar{r}_i$ :

$$\bar{r}_i = Max_i r_i - r_i + 1 \tag{4.1}$$

On ajoute 1 à la fin de l'équation pour éviter qu'aucun individu n'aura une probabilité nulle. La probabilité de sélection d'un individu $i$ sera alors égale à :

$$s_i = \frac{\bar{r}_i}{\sum_i \bar{r}_i} \tag{4.2}$$

La roue de loterie résultante est représentée par la figure 4.4. Il est clair, que les individus ont une chance décroissante de sélection en fonction de leur apparition dans la roue. Ainsi, l'individu 1 a la plus grande chance d'apparaître même plusieurs fois dans la population futur. Par contre, l'individu 10 risque de disparaître.

Voici en pratique comment programmer en language C la fonction de sélection.

```
int select (int sum)
{
int s = 0, i, rnd, pop-size, si;
rnd = rand() % sum ; // un nombre aléatoire entre 1 et sum
for(i = 0 ; i < pop − size , s <= rnd ; i++)
s = s + si;
return i; // retourner l'individu sélectionné
}
```

| $i$ | $f_1(i)$ | $f_2(i)$ | $r_i$ | $\bar{r}_i$ | $100 \times s_i$ |
|---|---|---|---|---|---|
| 1 | 52 | 111 | 0 | 9 | 18 |
| 2 | 56 | 115 | 1 | 8 | 16 |
| 3 | 61 | 116 | 2 | 7 | 14 |
| 4 | 64 | 117 | 3 | 6 | 12 |
| 5 | 65 | 119 | 4 | 5 | 10 |
| 6 | 68 | 120 | 5 | 4 | 8 |
| 7 | 66 | 135 | 5 | 4 | 8 |
| 8 | 67 | 147 | 6 | 3 | 6 |
| 9 | 70 | 125 | 6 | 3 | 6 |
| 10 | 79 | 139 | 8 | 1 | 2 |

TAB. 4.2 – *Population test*

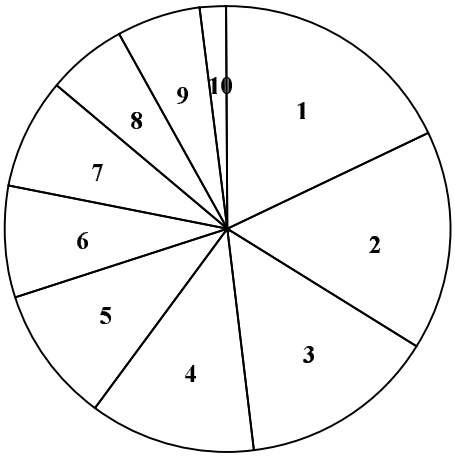



FIG. 4.4 – *Roulette de loterie*

- **croisement et mutation** : nous utilisons les mêmes opérateurs de croisement et de mutation utilisés dans le chapitre précédent. Seules les probabilités de croisement et de mutation ont changé ;
- **paramètres de l'algorithme génétique** : le tableau 4.3 résume les valeurs des paramètres utilisés par l'algorithme génétique afin de générer de bonnes solutions. Ces paramètres sont : la taille de la population, le nombre d'itérations (condition d'arrêt), la probabilité du croisement et de la mutation. Notons que les valeurs de ces paramètres sont obtenues après plusieurs exécutions de l'algorithme génétique.

| Taille de la population | nombre d'itérations | probabilité de croisement | probabilité de mutation |
|---|---|---|---|
| 800 | 600 | 0.5 | 0.09 |

TAB. 4.3 – *Valeurs des paramètres génétiques*

#### 4.3.2.2 Formalisation des critères d'optimisation

Pour qualifier l'ordonnancement conjoint du job shop, nous avons choisi deux critères d'optimisation. Le premier, lié à la production est décrit par l'équation 1.9 du chapitre 1. Il permet de calculer la longueur de l'ordonnancement. Le deuxième, lié à la maintenance, calcule le coût global de la maintenance dans l'atelier. Ce critère ne tient pas compte du coût de main d'oeuvre qui est supposé constant. Il intègre uniquement le coût de retard ou d'avance de l'exécution de l'ensemble des tâches de maintenance. Pour formuler le problème d'ordonnancement conjoint de la production et de la maintenance dans un atelier de type job shop, nous allons utiliser les notations données au tableau 4.4. Ces notations nous permettront de définir et formuler le critère d'optimisation lié à la maintenance. Nous supposons que le nombre de tâches de maintenance à effectuer sur chaque machine est connu d'avance. En effet si on connaît la longueur approximative de l'horizon de production et les périodes de maintenance systématiques, on peut déterminer le nombre de tâches de maintenance de chaque machine. En général les périodes de maintenance systématique sont constantes (ce qui est le cas dans notre étude), mais ceci n'empêche pas que ces périodes peuvent être variables en fonction de l'âge des machines, de leurs charges effectives etc.

Pour formuler le critère de maintenance noté $CM$, nous devons calculer le coût total dû aux retards et/ou avances de maintenance de chaque machine $M_k$. Nous détaillons par la suite la formalisation de ce critère en fonction de chaque type de période de maintenance. On notera $CM^1$ et $CM^2$ le coût lié à chaque type de période.

- **cas où les périodes sont indépendantes des charges machines** : comme le montre la figure 4.5, l'exécution de la tâche $m_{jk}$ peut se dérouler selon trois situations : en avance par rapport à sa période de maintenance, à temps ou en retard. Par conséquent, le coût d'exécution de cette même tâche noté $CM^1_{jk}$, peut se formuler comme montré dans l'équation 4.3.

$$CM^1_{jk} = \begin{cases} W_{ak}(j \times L_k - \Delta L_k - T_{jk}) \quad si \quad T_{jk} < j \times L_k - \Delta L_k \\ 0 \quad si \quad j \times L_k - \Delta L_k \leq T_{jk} \leq j \times L_k + \Delta L_k \\ W_{rk}(T_{jk} - j \times L_k - \Delta L_k) \quad si \quad T_{jk} > j \times L_k + \Delta L_k \end{cases} \tag{4.3}$$

| Symbole | Signification |
|---|---|
| $M_k$, $1 \leq k \leq m$ | L'ensemble des machines de l'atelier |
| $l_k$ | Nombre de tâches de maintenance de la machine $M_k$ (constante) |
| $m_{jk}$ | La $j^{me}$ tâche de maintenance de la machine $Mk$ |
| $d_{jk}$ | durée de la tâche de maintenance $m_{jk}$ |
| $T_{jk}$ | date d'affectation de l'opération $m_{jk}$ |
| $L_k$ | Période de maintenance systématique de la machine $M_k$ (constante) |
| $L_{jk}$ | Charge totale de la machine $M_k$ entre l'exécution des deux tâches $m_{(j-1)k}$ et $m_{jk}$ |
| $\Delta L_k$ | Retard toléré (ou avance) de maintenance sur la machine $M_k$ (constante) |
| $Wak$ | Coût d'avance par une unité de temps d'une tâche de maintenance sur la machine $M_k$ |
| $W_{rk}$ | Coût de retard par une unité de temps d'une tâche de maintenance sur la machine $M_k$ |

TAB. 4.4 – *Notations utilisées pour la formulation des critères d'optimisation*

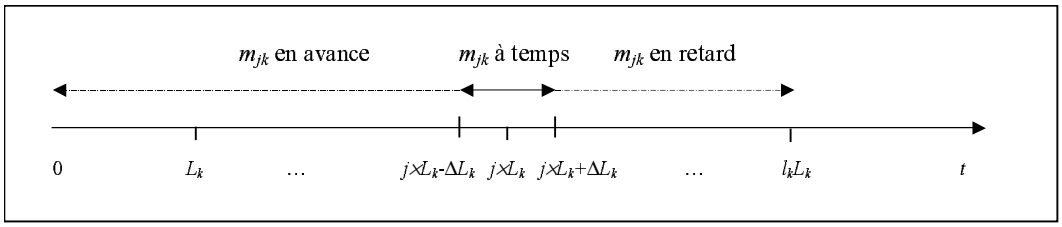



FIG. 4.5 – *Planification de la maintenance systématique : période indépendante de la charge machine*

Selon l'équation 4.3, le coût de maintenance de l'opération $m_{jk}$ est égal à :

$$CM^1_{jk} = \max[0, W_{ak}(j \times L_k - \Delta L_k - T_{jk}), W_{rk}(T_{jk} - j \times L_k - \Delta L_k)] \quad (4.4)$$

Le coût total de maintenance de la machine $M_k$ est le suivant :

$$CM^1_k = \sum_{j=1}^{l_k} CM^1_{jk} \quad (4.5)$$

D'après l'équation 4.5, on peut déduire le coût total de la maintenance dans l'atelier :

$$CM^1 = \sum_{k=1}^{m} CM_k^1 \tag{4.6}$$

– **cas où les périodes sont basées sur les charges machines** : pour calculer le coût de maintenance de la tâche $m_{jk}$ sur la machine $M_k$, nous prenons en compte la charge réelle de la machine après avoir exécuté la tâche $m_{(j-1)k}$. La figure 4.6 montre trois cas de figure quant à l'exécution de la tâche $m_{jk}$.

$$CM_{jk}^2 = \begin{cases} W_{ak}(L_{jk} - \Delta L_k - T_{jk}) \quad si \quad T_{jk} < L_{jk} - \Delta L_k \\ 0 \quad si \quad L_{jk} - \Delta L_k \leq T_{jk} \leq L_{jk} + \Delta L_k \\ W_{rk}(T_{jk} - L_{jk} - \Delta L_k) \quad si \quad T_{jk} > L_{jk} + \Delta L_k \end{cases} \tag{4.7}$$

D'après l'équation 4.7, le coût de maintenance de la tâche $m_{jk}$ est le suivant :

$$CM_{jk}^2 = \max[0,\, W_{ak}(L_{jk} - \Delta L_k - T_{jk}),\, W_{rk}(T_{jk} - L_{jk} - \Delta L_k)] \tag{4.8}$$

Le coût total de toutes les tâches de maintenance de la machine $M_k$ est donné par l'équation suivante :

$$CM_k^2 = \sum_{j=1}^{l_k} CM_{jk}^2 \tag{4.9}$$

L'équation 4.9, permet de déduire le coût total de maintenance dans l'atelier :

$$CM^2 = \sum_{k=1}^{m} CM_k^2 \tag{4.10}$$

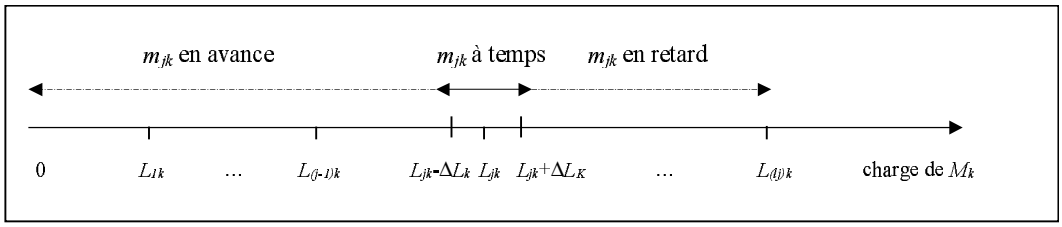



FIG. 4.6 – *Planification de la maintenance systématique : période dépendante de la charge machine*

#### 4.3.2.3 Bornes inférieurs

Dans le cas où l'on ne connaît pas la solution optimale d'un problème (par une méthode analytique par exemple), ou que la méthode de résolution utilisée n'est pas exacte, la détermination des bornes de la solution optimale est souvent utile. Ces bornes permettent dans des cas d'accélérer la convergence de l'algorithme en réduisant l'espace de recherche (algorithmes de type branch and bound par exemple), et dans d'autres cas de mesurer la qualité des solutions obtenues.

La problématique des bornes inférieures a été traitée dans la littérature pour plusieurs problèmes d'ordonnancement notamment les problèmes à une machine [JBC02], à machines parallèles [Car87], les problèmes de flow shop hybride [BCN01] et l'ordonnancement des job shop [CP89] et [KHB02]. Généralement, les méthodes proposées se basent sur la relaxation d'une ou de plusieurs contraintes (préemption des tâches, contraintes disjonctive sur les ressources, etc) pour minorer le makespan de l'ordonnancement optimal. La démarche suivie dans ce chapitre, généralise certaines bornes proposées dans la littérature en ajoutant les données de maintenance.

**Lemme 1.** *$B_1 = \max_{i=1}^{n}(\sum_{j=1}^{m} p_{ij})$ est une borne inférieure du critère de production $C_{max}$.*

**Preuve 1.** *Evident, à cause des contraintes de précédence entre les opérations d'un même job.*

$$B_2 = \max_{k=1}^{m}(\sum_{j=1}^{l_k} d_{jk} + \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}/O_{ij} \in M_k) \tag{4.11}$$

**Lemme 2.** *$B_2$ donnée par l'équation 4.11 est une borne inférieure du critère de production $C_{max}$.*

**Preuve 2.** *Soit $P_k$ une variable représentant la charge totale de la $k^{me}$ machine, en d'autre termes, $P_k$ est la somme des temps opératoires de toutes les tâches devant s'exécuter sur la machine $M_k$. Notons que chaque machine $M_k$ doit être entretenue (ou éventuellement réparée) durant une période de temps donnée par $D_k = \sum_{j=1}^{l_k} d_{jk}$. Par conséquent, la dernière opération de la machine $M_k$ ne peut pas finir avant la date $P_k + D_k$. Ce qui permet de montrer que $C_{max} \geq B_2$.*

La borne $B_2$ peut être améliorée en considérant les dates effectives à partir desquelles on peut commencer à affecter les opérations sur chaque machine $M_k$.

Soit $R_{ik}$ une variable représentant la date de début au plus tôt du $i^{me}$ job sur la machine $M_k$, alors, $R_{ik} = \sum_{s=1}^{j-1} p_{is}/O_{ij} \in M_k$. La date de début au plus tôt $R_k$ de la machine $M_k$ peut se formuler alors comme suit $R_k = \min_{i=1}^{n} R_{ik}$.

$$B_3 = \max_{k=1}^{m}(R_k + P_k + D_k) \tag{4.12}$$

**Lemme 3.** *$B_3$ est une borne inférieure du critère de production $C_{max}$ et on a en plus $B_3 \geq B_2$.*

**Preuve 3.** *Chaque machine $M_k$ peut exécuter ses opérations à partir de sa date de début au plus tôt $R_k$, et ceci durant une période de temps effective égale à $P_k$. En plus, chaque machine $M_k$ doit être maintenue pendant la durée $D_k$. Par conséquent, $C_{max} \geq R_k + P_k + D_k$. Ce qui prouve que $C_{max} \geq B_3$.*

*On a $\forall 1 \leq k \leq m$, $R_k + P_k + D_k \geq P_k + D_k$ car $\forall 1 \leq k \leq m$, $R_k \geq 0$. Ce qui montre que $\max_{k=1}^{m}(R_k + P_k + D_k) \geq \max_{k=1}^{m}(P_k + D_k)$, et finalement $B_3 \geq B_2$.*

### 4.3.3 Résultats expérimentaux

Le problème du job shop est souvent traité sans contraintes de disponibilité ou de maintenance. Les benchmarks disponibles dans la littérature concernent uniquement des critères de production. Pour obtenir des problèmes tests, nous avons alors sélectionné quelques benchmarks connus auxquels on a rajouté des données de maintenance.

Pour tester notre méthode d'ordonnancement conjoint de la production et de la maintenance, nous avons sélectionné à partir du web (la bibliothèque OR : `http://mscmga.ms.ic.ac.uk/info.html`) trois benchmarks de job shop dont nous donnons quelques détails dans le tableau 4.5. Nous avons choisi deux benchmarks de Lawrence avec différentes tailles, La01 et La06 et le benchmark connu de Muth et de Thomson MT06 avec six jobs et six machines (dans le tableau, $n$ indique le nombre de jobs et $m$ le nombre de machines). Pour chaque benchmark, nous avons généré des données de maintenance telles que la période de maintenance systématique $L_k$, les durées des tâches de maintenance $d_k$, les coût d'avance ou de retard de la maintenance $W_{ak}$ et $W_{rk}$ etc. Les valeurs de $C_{max}$ optimales sans considération de la maintenance sont données dans le tableau 4.5. Afin de valider les solutions obtenues par l'algorithme génétique, nous utilisons les bornes inférieures pour le critère de production. Dans le tableau 4.5, la dernière colonne contient la borne $LB$ qui est égale au minimum des bornes $B_1$, $B_2$ et $B_3$ citées auparavant. Il est clair que la valeur de $LB$ est supérieure à $C^*_{max}$ à cause des durées de maintenance supplémentaires sur les machines.

| Problème | $n$ | $m$ | $C^*_{max}$ (sans maintenance) | $LB$ (avec la maintenance) |
|---|---|---|---|---|
| MT06 | 6 | 6 | 55 | 58 |
| La01 | 10 | 5 | 666 | 691 |
| La06 | 15 | 5 | 926 | 961 |

TAB. 4.5 – *Benchmarks choisis pour les tests*

Nous supposons que pour chaque machine $M_k$, la charge machine séparant deux tâches de maintenance est une constante notée $L_k$. Cette constante est déterminée en fonction de la charge totale de la machine en question. Nous supposons aussi que le coût de retard est plus grand que celui de l'avance. Pour les expériences réalisées avec les différents benchmarks, nous avons utilisé la relation suivante entre ces deux coûts :

$$W_{ak} = \frac{1}{2} W_{rk} \tag{4.13}$$

Afin d'obtenir une bonne population Pareto optimale dont la frontière Pareto est la plus basse possible (il s'agit d'un problème de minimisation), nous exécutons l'algorithme génétique plusieurs fois. Dans chaque exécution $i$, nous gardons seulement la population de solutions optimales au sens de Pareto notée $S_i$. Finalement, nous considérons la population $S = \bigcup_{i=1}^{n} S_i$. Nous avons gardé de nouveau uniquement les solutions Pareto optimales contenues dans $S$ ($n$ étant le nombre d'exécutions indépendantes de l'algorithme génétique).

Nous allons par la suite développer les expériences réalisées sur les trois benchmarks sélectionnés. Pour chaque benchmark, les données de maintenance seront détaillées dans des tableaux. Les résultats de chaque type de période de maintenance seront résumés avec des courbes. Enfin, une interprétation sera donnée à la fin de chaque cas.

#### 4.3.3.1 Benchmark de Muth et Thomson : MT06

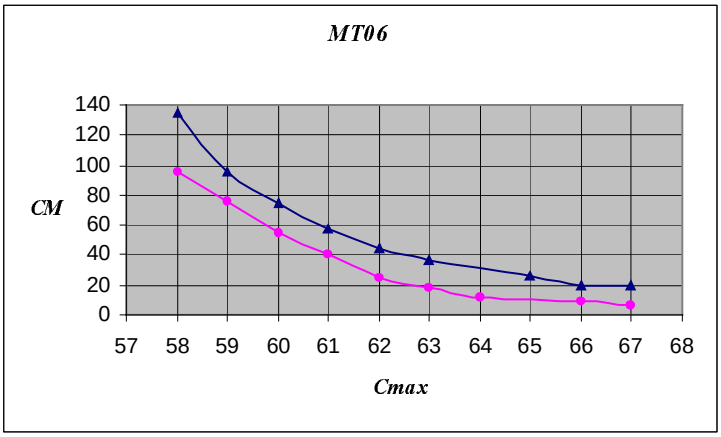
Le benchmark de Muth et Thomson noté MT06, est constitué de six machines, $M_1$ jusqu'à $M_6$. Pour diversifier nos tests, nous avons varié le nombre de tâches de maintenance pour chaque machine $M_k$. En l'occurrence, les machines $M_2$, $M_3$ et $M_4$ doivent être maintenues une fois. Les machines $M_1$ et $M_5$ doivent subir deux tâches de maintenance. Finalement, la machine $M_6$ doit être maintenue trois fois. Les données de maintenance relatives au benchmark MT06, sont détaillées dans le tableau 4.6.

L'ensemble $S$ des solutions générées par l'algorithme génétique contient 206 solutions non redondantes dont 9 sont Pareto optimales. La figure 4.7 résume les résultats obtenus

| Machines $M_k$ | $d_{jk}$ | | | Charge totale | $L_k$ | $\Delta L_k$ | $W_{ak}$ | $W_{rk}$ |
|---|---|---|---|---|---|---|---|---|
| $M_1$ | 4 | 3 | - | 40 | 18 | 2 | 4 | 8 |
| $M_2$ | 2 | - | - | 26 | 13 | 1 | 6 | 12 |
| $M_3$ | 3 | - | - | 26 | 14 | 1 | 7 | 14 |
| $M_4$ | 1 | - | - | 22 | 11 | 1 | 5 | 10 |
| $M_5$ | 3 | 3 | - | 40 | 17 | 4 | 2 | 4 |
| $M_6$ | 2 | 4 | 3 | 43 | 14 | 2 | 3 | 6 |

TAB. 4.6 – *Données de maintenance : MT06*

pour les deux types de périodes de maintenance. Afin de faciliter l'interpretation des courbes de cette figure, nous avons gardé uniquement les solutions Pareto optimales.




▲ : *Solutions Pareto Optimales dans le cas où les périodes de maintenance sont indépendantes des charges machines*

• : *Solutions Pareto Optimales dans le cas où les périodes de maintenance dépendent des charges machines*

FIG. 4.7 – *Comparaison des solutions Pareto optimales : Cas du benchmark MT06*

Les courbes de la figure 4.7 montrent bien l'antagonisme existant entre les deux critères étudiés. En effet, le critère de la production $C_{max}$ et celui de la maintenance $CM$ évoluent dans deux directions opposées. Par exemple, sur la courbe des solutions Pareto optimales dans le cas des périodes indépendantes des charges des machines, la solution réalisant la valeur minimale du critère de production $C^*_{max} = 58$ engendre le coût maximal de maintenance $CM = 135$. Pareil pour la solution minimisant le coût de maintenance $CM = 19$, elle retarde beaucoup la production $C_{max} = 67$. Comme le montrent les deux courbes, l'algorithme génétique a réussi à trouver une solution optimisant le critère $C_{max}$. En effet la valeur 58 est donnée par la borne inférieure $LB$.

On remarque aussi, que dans le cas des périodes de maintenance basées sur la charge des machines, les solutions Pareto optimales sont meilleures par rapport aux solutions obtenues en choisissant des périodes indépendantes de la charge machine. Ceci est vrai pour les deux critères.

#### 4.3.3.2 Benchmark de Lawrence : La01

Le premier benchmark de Lawrence noté La01, est constitué de cinq machines, $M_1$ jusqu'à $M_5$. Vues les charges des machines, on a choisi des différentes périodes de maintenance. En effet, les machines $M_1$ et $M_5$ ont les plus grandes charges et sont les plus utilisées pour la production. Il est normal alors qu'elles soient les plus maintenues. Ces deux machines doivent être entretenues deux fois. Tandis que les machines $M_2$, $M_3$ et $M_4$ possèdent une seule tâche de maintenance. Les données de maintenance relatives au benchmark La01, sont détaillées dans le tableau 4.7.

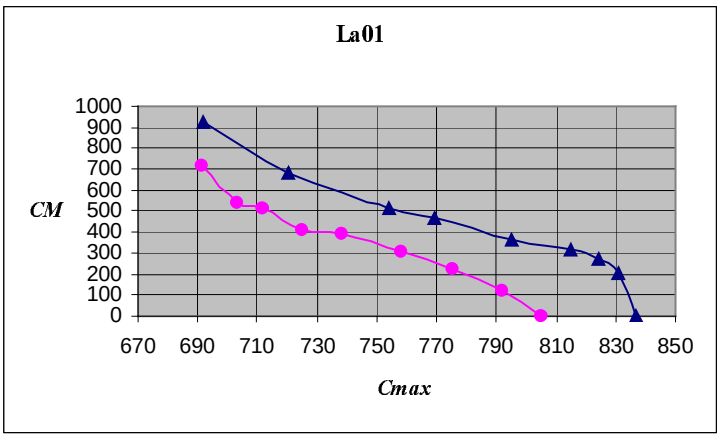
| Machines $M_k$ | $d_{jk}$ | | Charge Totale | $L_k$ | $\Delta L_k$ | $W_{ak}$ | $W_{rk}$ |
|---|---|---|---|---|---|---|---|
| $M_1$ | 10 | 15 | 609 | 330 | 30 | 5 | 10 |
| $M_2$ | 20 | - | 536 | 270 | 25 | 4 | 8 |
| $M_3$ | 15 | - | 530 | 270 | 25 | 3 | 6 |
| $M_4$ | 17 | - | 508 | 270 | 20 | 7 | 14 |
| $M_5$ | 12 | 13 | 666 | 330 | 35 | 6 | 12 |

TAB. 4.7 – *Données de maintenance : La01*

Les résultats expérimentaux de ce benchmark sont affichés dans la figure 4.8. Les mêmes constatations pour le benchmark précédent sont vraies pour le benchmark La01. En revanche, l'algorithme génétique cette fois ci n'a réussi à trouver une solution optimisant le critère de production que dans le cas des périodes de maintenance basées sur la charge machine. On remarque aussi que ces courbes contiennent des solutions avec un coût de maintenance nul. Ceci est dû au respect des périodes de maintenance par les ordonnancements donnés par ces solutions. Malheureusement, ce respect des délais de maintenance a retardé trop la production, ce qui a engendré les plus grandes valeurs de $C_{max}$ (805 et 837).

#### 4.3.3.3 Benchmark de Lawrence : La06

Le deuxième benchmark de Lawrence choisi pour les tests est noté La06. Il est constitué de cinq machines, $M_1$ jusqu'à $M_5$. Vues les charges des machines, on a choisi des




▲ : *Solutions Pareto Optimales dans le cas où les périodes de maintenance sont indépendantes des charges machines*

• : *Solutions Pareto Optimales dans le cas où les périodes de maintenance dépendent des charges machines*

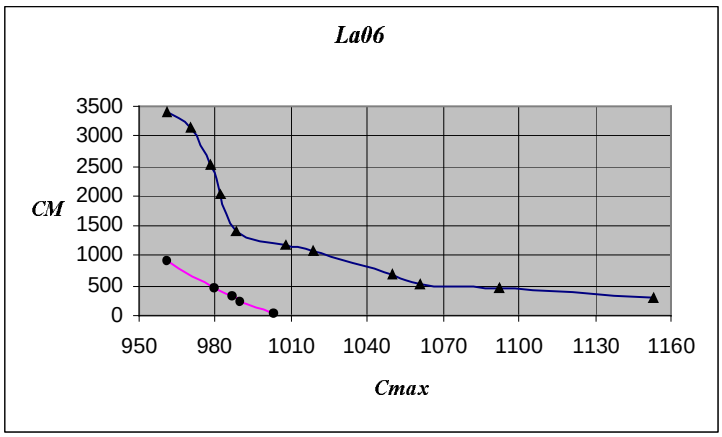
FIG. 4.8 – *Comparaison des solutions Pareto optimales : Cas du benchmark La01*

différentes périodes de maintenance. En effet, les machines $M_1$ et $M_5$ ont les plus grandes chargent et sont les plus utilisées pour la production. Il est normal alors qu'elles soient les plus maintenues. Ces deux machines doivent être entretenues deux fois. Tandis que les machines $M_2$, $M_3$ et $M_4$ possèdent une seule tâche de maintenance. Les données de maintenance relatives au benchmark La06, sont détaillées dans le tableau 4.8.

| Machines $M_k$ | $d_{jk}$ | | Charge Totale | $L_k$ | $\Delta L_k$ | $W_{ak}$ | $W_{rk}$ |
|---|---|---|---|---|---|---|---|
| $M_1$ | 20 | 15 | 926 | 450 | 15 | 5 | 10 |
| $M_2$ | 22 | - | 740 | 400 | 25 | 4 | 8 |
| $M_3$ | 10 | - | 785 | 400 | 15 | 3 | 6 |
| $M_4$ | 15 | - | 726 | 400 | 18 | 7 | 14 |
| $M_5$ | 17 | 10 | 815 | 350 | 25 | 6 | 12 |

TAB. 4.8 – *Données de maintenance : La06*

Les résultats expérimentaux de ce benchmark sont affichés dans la figure 4.9. Les mêmes constatations pour le benchmark MT06 sont vraies pour le benchmark La01. En effet, l'algorithme génétique a réussi à trouver une solution optimisant le critère de production dans chacun des cas des périodes de maintenance. Pour une valeur de $C_{max}$ optimale (égale à 961), le coût de la maintenance est égale à 3396 dans le cas des périodes de maintenance indépendantes des charges des machines, et égale à 913 dans l'autre cas.




▲ : *Solutions Pareto Optimales dans le cas où les périodes de maintenance sont indépendantes des charges machines*

• : *Solutions Pareto Optimales dans le cas où les périodes de maintenance dépendent des charges machines*

FIG. 4.9 – *Comparaison des solutions Pareto optimales : Cas du benchmark La06*

#### 4.3.3.4 Interpretation

Les résultats expérimentaux sur les trois problèmes types de job shop de tailles différentes, montrent bien l'existence d'un antagonisme entre les deux critères étudiés. Ces derniers évoluent dans deux sens opposés. La minimisation de l'un entraîne l'augmentation de l'autre. Cet antagonisme rend la tâche très difficile à l'utilisateur pour choisir une solution adéquate à son problème. Dans ce cas, l'assistance d'un expert pour choisir une solution réalisant un compromis entre les critères d'optimisation devient primordiale.

D'autre part, les test réalisés sur le choix des deux types de période de maintenance a montré que la prise en compte de la charge réelle effectuée par la machine est plus économique pour la maintenance. En effet, ceci permet de maintenir les machines en fonction de leur quantité de travail et non pas toutes les périodes de temps même si les machines n'ont pas été suffisamment utilisées pour la production. De plus, en fonction de la charge effectuée réellement, on détermine le moment de la maintenance de la machine. Les courbes des trois figures 4.7, 4.8 et 4.9 confirment ces aspects. Le gain est aussi bien au niveau de la production qu'au niveau de la maintenance.

Le bon choix de la période de maintenance préventive permet d'améliorer la qualité, la productivité et la maîtrise des coûts au sein des entreprises. En effet, le grand souci de la maintenance préventive reste son coût élevé. Une bonne période de maintenance réduit les tâches inutiles et permet une planification adéquate de la maintenance. Par conséquent, les coûts seront maîtrisés.

## 4.4 Conclusion

Dans le milieu industriel, l'optimisation des délais et des coûts sont des facteurs clés pour le développement. La proposition des méthodes pour l'ordonnancement conjoint de la production et de la maintenance représente actuellement une voie possible pour atteindre cet objectif.

Nous avons pu constater dans ce chapitre l'intérêt d'une approche génétique multiobjectif pour la minimisation des conflits sur les ressources entre la production et la maintenance préventive systématique au sein d'un atelier de type job shop. En effet, l'ordonnancement conjoint de ces deux types de tâches a permis d'avoir un bon compromis en minimisant au mieux les critères de production et de maintenance.

Dans ce chapitre, nous avons pu atteindre deux objectifs majeurs. Le premier, consiste à étudier le job shop avec les contraintes de maintenance des machines. Dans ce volet, nous avons proposé une approche génétique Pareto optimale pour l'ordonnancement conjoint de la production et de la maintenance. Cette approche testée sur des benchmarks types, a démontré son efficacité en générant des solutions avec des valeurs très proches de l'optimum (et parfois optimales) aussi bien pour le critère de la production que de maintenance. Le deuxième objectif, consiste à comparer deux types de périodes de maintenance. Le premier consiste à effectuer la maintenance périodiquement et indépendamment des charges réellement effectuées par les machines. Tandis que le deuxième, plus raisonnable, tient compte de la charge de travail réalisée par chaque machine pour décider la date de sa maintenance. Les tests réalisés sur les benchmarks que nous avions modifié, ont permis d'élaborer une meilleure planification des tâches de production et de maintenance dans le cas des périodes dépendant des charges machines.

# Conclusion générale et perspectives

Le contexte de notre travail concerne l'ordonnancement d'un atelier de production de type job shop. Nous avons pour cela élaboré une méthode de résolution aussi bien dans le cas classique d'un ordonnancement relatif à la production que dans le cas beaucoup moins étudié touchant l'ordonnancement conjoint de la production et de la maintenance. Les algorithmes génétiques ayant fait leur preuve dans le domaine aussi bien mono objectif que multiobjectif ont été à la base de notre étude. Etude faite tout d'abord sur un problème classique de Job shop noté $J//C_{max}$, en considérant les machines disponibles tout le temps, puis en introduisant dans un deuxième temps l'aspect de maintenance préventive ayant des objectifs parfois antagonistes avec la production et qui nécessite une résolution multiobjectif.

Notre contribution comporte deux volets :

- le premier volet prend appui sur les solutions générées par un algorithme génétique qui sont étudiées par la suite par des méthodes d'apprentissage. Méthodes resituées dans le processus d'Extraction de Connaissance à partir des Données (ECD). Dans un souci de validation et de comparaison par rapport aux travaux faits dans la communauté, la démarche proposée a été élaborée sur un problème classique de type $J//C_{max}$ et sur des benchmarks connus ;
- le deuxième volet propose un algorithme génétique Pareto optimal résolvant le problème d'ordonnancement conjoint de la production et de la maintenance au sein du job shop.

Après avoir resitué, au premier chapitre le contexte de travail, à savoir l'ordonnancement dans un atelier de production de type job shop, nous avons fait un état de l'art des méthodes exactes et approchées pour la résolution de ce problème. Les algorithmes génétiques qui constituent un élément de cet état de l'art, ont été choisis pour l'élaboration de notre démarche de résolution. Nous avons pour ce faire introduit au chapitre 2

ces algorithmes et leur principe de fonctionnement ainsi que leur adaptation à l'ordonnancement.

La démarche de résolution du problème d'ordonnancement dans un atelier de type job shop, proposée au troisième chapitre, comporte deux phases cruciales :

- lors de la première phase, nous avons travaillé dans l'espace de solutions d'un problème de job shop, pour sélectionner une population de bonnes solutions. Ceci à l'aide des algorithmes génétiques ;
- lors de la deuxième phase, nous avons cherché à expliquer cet espace de solutions, par les caractéristiques du problème se trouvant dans l'espace de définition à l'aide de l'ECD. Les données étant les ordonnancements solutionnant le problème de job shop. La connaissance, étant les règles d'ordonnancement obtenues à partir de ces solutions.

L'Extraction de Connaissance à partir de Données (ECD) ou Knowledge Discovery in Databases (KDD) utilisée pour élaborer une heuristique de résolution du job shop, est une démarche complète d'exploitation des données que l'on peut résumer en quatre phases distinctes : l'acquisition des données, le pré-traitement, la fouille de données et le post-traitement. Nous avons adapté ce processus étape par étape de la manière suivante :

- lors de la première étape, nous avons généré une population de bonnes solutions de plusieurs problèmes de job shop en utilisant un algorithme génétique ;
- dans l'étape de pré-traitement, les solutions générées par l'algorithme génétique sous forme de chromosomes sont transformées sous la forme d'une liste d'opérations devant chaque machine. Puis, ces solutions sont caractérisées par des attributs numériques liés au problème, tel que les temps opératoires, les charges des machines, la durée totale d'un job etc. Ces attributs numériques ont été transformés en attributs symboliques par un algorithme de discrétisation supervisée, le ChiMerge. Cette étape de discrétisation nous a permis d'obtenir un modèle général, applicable à la résolution d'un ensemble vaste de problèmes. L'une des caractéristiques les plus importantes de l'algorithme de ChiMerge est sa considération à la fois des caractéristiques des individus ainsi que des classes dont ils appartiennent ;
- lors de l'étape importante de fouille de données, nous avons cherché à expliquer les positions des opérations sur les machines en fonction des leurs attributs symboliques par des règles d'ordonnancement. Ces règles ont été extraites à partir d'un problème type de Muth et Thomson et un ensemble de dix autres problèmes générés aléatoirement ;

– lors de l'étape de post-traitement, nous avons transformé les règles d'ordonnancement trouvées en une heuristique permettant de généraliser le résolution à des problèmes de tailles différentes. Cette heuristique a permis de générer de bonnes solutions pour un ensemble de test composé de problèmes types de job shop (Benchmark de Muth et Thomson et Lawrence).

Par contre, si l'on désire avoir des résultats plus précis, on utilisera les règles d'ordonnancement issues directement de la troisième étape de l'ECD, mais sur des problèmes de job shop de même taille. Travail validé sur un problème de taille $15 \times 5$ et testé sur quatre benchmarks. Les résultats expérimentaux ont montré l'efficacité de cette méthode de résolution. En résumé, suivant l'objectif que l'on se donne :

– si l'on désire avoir rapidement une solution approchée sur des problèmes de tailles différentes que celui traité, on applique l'heuristique élaborée à la quatrième étape.

– en revanche, pour des problèmes de même taille que celui traité, on applique les règles obtenues.

Une grande partie de la littérature dédiée aux problèmes d'ordonnancement se place dans le contexte de disponibilité totale des ressources. Cette hypothèse n'est pourtant pas fidèle à la réalité des ateliers de production. En effet, les différentes ressources qu'elles soient humaines ou matérielles peuvent, pour diverses raisons, être indisponibles. Dans la deuxième partie de la thèse nous tenons compte de l'aspect maintenance au sein du job shop. Après un état de l'art sur les méthodes multiobjectif élaboré au deuxième chapitre, nous avons développé un algorithme génétique pour la résolution du problème d'ordonnancement conjoint de la production et de la maintenance au sein du job shop. Vue la complexité de ce problème, nous avons opté pour la maintenance préventive systématique. Deux méthodes de choix de périodes systématiques ont été proposées. La première consiste à fixer des périodes de maintenance préventive indépendamment des charges de travail des machines, comme il est communément fait en entreprises. La deuxième prend en compte ce dernier paramètre. Il s'avère que le deuxième type de périodes est naturellement plus efficace que le premier, mais nécessite une forte dépendance entre les deux services production et maintenance. Nous avons eu recours à l'optimisation multiobjectif et les algorithmes génétiques pour générer des solutions Pareto optimales. Solutions que nous avons validées par des bornes inférieures.

Après la mise au point, par un algorithme génétique des solutions Pareto optimales, nous envisageons d'étudier ce nouvel espace solution par le processus d'ECD. Nous proposons donc d'étendre la démarche décrite au troisième chapitre, à l'ordonnancement

conjoint de la production et de la maintenance. Ceci en introduisant d'autres caractéristiques liées à la maintenance telle que les durées des périodes systématiques ou le coût dû à un retard ou à une avance de l'intervention de maintenance. Le développement d'une heuristique permettant l'ordonnancement conjoint de la production et de la maintenance au sein du job shop sera très utile. En effet, cette heuristique nous permettra d'éviter d'une part la recherche des paramètres génétiques adéquats à chaque problème à résoudre, et d'autre part l'exécution multiple de l'algorithme génétique pour trouver des solutions Pareto optimales.

Nous proposons aussi l'élaboration des bornes inférieures du coût total de la maintenance dû aux retards ou aux avances des interventions sur les machines. Ces bornes inférieures avec les bornes proposées dans la thèse et l'algorithme génétique multiobjectif du quatrième chapitre, constitueront un moyen très utile pour la validation de l'heuristique de l'ordonnancement conjoint.

Le cas des problèmes d'ordonnancements statiques reste loin de la réalité industrielle. Ainsi, la considération des données dynamiques et des événements aléatoires (tels que les pannes non envisagées des équipements, l'arrivée de nouvelles commandes, etc) permet de s'approcher encore plus des problèmes réels. Nous proposons dans ce cas d'étendre notre méthode d'extraction de règles d'ordonnancement et l'algorithme génétique multiobjectif au cas de la maintenance corrective. Cet objectif sera réalisé en considérant des tâches de maintenance corrective qui apparaissent aléatoirement lors de l'ordonnancement des tâches de production et de maintenance systématique. Dans ce cas, il faut élaborer un nouveau critère de maintenance. Ce critère prendra en compte le coût supplémentaire dû à la maintenance corrective.

# Bibliographie


[AED88] J. Adams, B. Egon, and Z. Daniel. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[AF55] S. B. JR. Akers and J. Freidman. A non numerical approach to scheduling problems. *Operations Research*, 3:429–442, 1955.

[Agg02] R. Aggoune. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. PhD thesis, Université de Metz, France, 2002.

[Ake56] S. B. JR. Akers. A graphical approach to production scheduling problems. *Operations Research*, 4:244–245, 1956.

[Alt95] H. Altay. A genetic algorithm for multicriteria inventory classification. In Ales, editor, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 6–9, Avril 1995.

[AM98] A. Allahverdi and J. Mittenthal. Dual criteria scheduling on a two-machine flowshop subject to random breakdowns. *International Transaction of Operational Research*, 5(4):317–324, 1998.

[AO99] M. Asano and H. Ohta. Single machine scheduling to meet due times under ahutdown constraints. *International Journal of Production Economics*, 60-61:537–547, 1999.

[ATS96] J. Ashayeri, A. Teelen, and W. Selen. A production and maintenance planning model for the process industry. *International Journal of production Research*, 34(12):3311–3326, 1996.

[Bak85] J. E. Baker. Adaptive selection methods for genetic algorithms. Pittsburgh, PA, 1985. International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum associates.

[Bal69] E. Balas. Machine scheduling via disjunctive graphs: An implicit enumeration algorithm. *Operations Research*, 17:941–957, 1969.

[Bay01] S. D. Bay. Multivariate discretization for set mining. *Knowledge and Information Systems*, 3(4):491–512, 2001.

[BB90] A. Banerjee and J. S. Burton. Equipement utilization based maintenance task scheduling in a job shop. *European Journal of Operational Research*, 45:191–202, 1990.

[BCN01] J. C. Billaut, J. Carlier, and E. Néron. *Ordonnancement de la production*, chapter Ordonnancement d'ateliers à ressources multiples. Hermès, france edition, 2001.

[BCS97] E. Burke, J. Clark, and A. Smith. Four methods for maintenance scheduling. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 264–269. Springer, 1997.

[BD74] R. Belmann and S. E. Dreyfus. *Applied dynamic programming.* Princeton University Press, 1974.

[Bem02] M. Bembla. Ordonnancement conjoint production et mainten ance : Critère et heuristique de résolution. Mémoire de dea, U.F.R des Sciences et Techniques de l'Université de Franche Comté, 2002.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees.* CA: Wadsworth International Group, Belmont, 1984.

[BFP96] M. Brandolese, M. Franci, and A. Pozzetti. Production and maintenance integrated planning. *International Journal of production Research*, 34(7):2059–2075, 1996.

[Bie95] C. Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithm. *OR Spectrum*, 17:87–92, 1995.

[BLK83] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling projects subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[BPH82] J. H. Jr Blackstone, D. T. Philips, and G. L. Hoggs. A state of the art survey of dispatching rules for manufacturing job-shop operations. *International Journal of Production Research*, 20:27–45, 1982.

[BS98] E. K. Burke and A. J. Smith. Hybrid evolutionary techniques for the maintenance scheduling problem. *IEEE Transactions on Power Systems*, 1998.

[Car87] J. Carlier. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, 29:298–306, 1987.

[Cav00] G. Cavory. *Une approche génétique pour la résolution d'ordonnancements cycliques.* PhD thesis, Université d'Artois, Décembre 2000.

[CFIQ88] J. Cheng, U. M. Fayyad, K. B. Irani, and Z. Qian. Improved decision trees: A generalized version of id3. In *Proceedings of the fifth International Conference on Machine Learning*, volume 1, pages 100–108, 1988.

[CKB87] B. Cestnik, L. Kononenko, and I. Bratko. Assistant86: a knowledge-elicitation tool for sophisticated users. *In Progress in machine learning*, pages 31–45, 1987.

[ClPP95] C. Caux, H. l. Pierreval, and M. C. Portmann. Les algorithmes génétiques et leurs applications aux problèmes d'ordonnancement. *APII*, 29(4-5):409–443, 1995.

[CMHZ03] B. Chebel-Morello, Y. Harrath, and N. Zerhouni. Application de la maintenance systématique au job shop : Approche génétique pareto optimale. In *Proceedings of the International Conference PENTOM*, pages 85–96. Valenciennes, France, mars 2003.

[Col02] Y. Collett. *Optimisation multiobjectif*. Eyrolles edition, 2002.

[CP89] J. Carlier and E. Pinson. An algorithm for solving the job shop problem. *Management Science*, 35(2):164–176, 1989.

[CP90] J. Carlier and E. Pinson. A practical use of jackson's pre-emptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[CS96] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In P. Smyth R. Uthurusamy U. M. Fayyad, G. Piatetsky-Shapiro, editor, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. Te AAAI Press, Menlo Park, 1996.

[CTV95] F. Delle Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. *Computers and Operations Research*, 22(1):15–24, 1995.

[Dav85] L. Davis. Job shop scheduling with genetic algorithms. In J. Grefenstette, editor, *Proceedings of the first International Conference on Genetic Algorithms*, pages 136–140. Hillsdale, 1985.

[DKS95] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning ICML*, pages 194–202, 1995.

[DMU97] E. Demirkol, S. Mehta, and R. Uzsov. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Operational Research*, 109(1):137–141, 1997.

[DNMM99] S. Deniaud, N. Noureddine, A. AL Moudni, and F. Morel. Planification de la maintenance préventive d'une ligne de fabrication d'alstom. *Journal Européen des Systèmes Automatisés*, 33(3):227–250, 1999.

[DP95] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, 22:25–40, 1995.

[DT91] M. Dell'Amico and M. Trubian. Applying tabu search to the job shop scheduling problem. Mimeo, Dipartimento di Elettronica,, Politecnico di Milano, 1991.

[EHM99] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141, 1999.

[EL01] P. Esquirol and P. Lopez. *Concepts et méthodes de base en ordonnancement*, chapter 2, pages 25–53. hermes edition, 2001.

[Esp98] M. L. Espinouse. *Flowshop et extensions : chevauchement des ttâches, indisponibilité des machines et systèmes de transport.* PhD thesis, Université Joseph Fourier Grenoble 1 Sciences et Géographie, 1998.

[Fay94] U. M. Fayyad. Branching on attribute values in decision tree generation. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pages 601–606, 1994.

[FB91] E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proceedings 1991 IEEE International Conference on Robotics and Automation*, volume 1, pages 824–829, Sacramento, CA, , 9-11 1991. IEEE Computer Society Press, Los Alamitos, CA.

[FI93] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

[Fin99] G. Finke. Ordonnancement. Rapport de cours, INPG, 1999.

[Fis87] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[FP95] C. M. Fonseca and J. Peter. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computaion*, 3(1):1–16, 1995.

[FPSS96] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data minind: Towards an unifying framework. In *Proceedings of the the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 82–88, 1996.

[FRC93] H. L. Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job-shop sheduling, rescheduling, and open-shop scheduling problems. In *Proceedings of the the International Conference on Genetic Algorithms*, pages 375–382, San Mateo, 1993.

[FT63] H. Fisher and G. L. Thomson. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*, pages 225–251, 1963. Prentice Hall, Englewood Cliffs, New Jersey.

[Gil96] Z. Gilles. *La maintenance basée sur la fiabilité.* Hermès edition, 1996.

[GL99] G. H. Graves and C. Y. Lee. Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics*, 46:845–863, 1999.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic machine scheduling: a survey. *Annals of Discrete Mathematics*, 1979.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computer & Operations Research*, 13:533–549, 1986.

[Glo89] F. Glover. Tabu search: Part i. *ORSA Journal on Computing*, 1, 1989.

[Glo90] F. Glover. Tabu search: Part ii. *ORSA Journal on Computing*, 2, 1990.

[Gol89] D. Goldberg. *Genetic Algorithm in Search, Optimisation and Machine Learning.* Addison Wesley, 1989.

[GPS00] C. Guéret, C. Prins, and M. Sevaux. *Programmation linéaire.* Eyrolles edition, 2000.

[GRG01] C. Guillaume, D. Rémy, and G. Gilles. Une approche basée sur la simulation pour résoudre le problème du job shop périodique à contraintes linéaires. In *Proceedings of the 3td International Conference on Modelisation and simuling Mosim*, pages 199–203, Troyes, France, Avril 2001.

[GTK94] M. Gen, J. Tsujimura, and E. Kubota. Solving job-shop scheduling problem using genetic algorithm. In M. Gen and S. Koboyashi, editors, *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, pages 576–579. Ashikaga, Japan, 1994.

[HA82] N. Hefetz and I. Adiri. An efficient optimal algorithm for the two machines unit-time job-shop schedule-length problem. *Mathematics of operations Research*, 7:354–360, 1982.

[HL92] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.

[HM94] P. Husbands and F. Mill. Scheduling with genetic algorithms, 1994.

[HMCZ01] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. A genetic algorithm and data mining to resolve a job shop schedule. In *Proceeding of the 8th IEEE international Conference ETFA*, volume 2, pages 727–728, Antibes - Juan Les Pins, France, Octobre 2001.

[HMCZ02a] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. Algorithme génétique et fouille de données pour la résolution réactive d'un job shop. In *Proceedings of the 2nd international conference JTEA*, volume 1, pages 349–355, Sousse, Tunisie, Avril 2002.

[HMCZ02b] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. A genetic algorithm and data mining based meta-heuristic for job shop scheduling problem. In *CD-ROM of IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisie, october 2002.

[HMCZ03a] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. Bornes inférieures et optimisation génétique multiobjective pour l'ordonnancement conjoint de la maintenance et de la production d'un job shop. *JESA*, 2003. à paraître.

[HMCZ03b] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. Lower bounds and multiobjective evolutionary optimization for combined maintenance and production scheduling in job shop. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal, 16-19 september 2003.

[HMCZ03c] Y. Harrath, B. Morello-Chebel, and N. Zerhouni. Using learning to find patterns in genetic algorithm solutions to solve job-shop scheduling. In *CD-ROM of International Conference IEPM*, Porto, Portugal, 26-28 may 2003.

[HMS96] E. B. Hunt, J. Marin, and P. J. Stone. *Experiment in induction.* academic press, New York, 1996.

[Hol75] J. Holland. *Adaption in Natural and Artificial Systems.* 1975.

[Hol93] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. In *Proceedings of the 11th International Conference on Machine Learning*, pages 63–90, 1993.

[HVPZ93] C. W. Holsapple, S. Varghese, J. Ramakrishna Pakath, and J. S. Zaveri. A genetics based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 1993.

[Jac56] J. R. Jackson. An extension of johnson's result on job-lot scheduling. *Naval Research Logistics Quarterly*, 3, 1956.

[JBC02] A. Jouglet, P. Baptiste, and J. Carlier. Exact procedures for single machine total cost scheduling. In *Proceedings of the International IEEE/SMC'02 Conference*, Hammamet, Tunisia, 6-9 october 2002.

[Jen01] T. Mikkel Jensen. *Robust and flexible scheduling with evolutionary computation.* PhD thesis, Faculty of science of the university of Aarhus, 2001.

[Jho54] S. M. Jhonson. Optimal two-and three stage production schedules with set-up times included. *Nav. Res. Logist. Quart.*, 1:61–68, 1954.

[JM98a] A. Jain and S. Meeran. Job-shop scheduling using neural networks. *International Journal of Production Research*, 36(5):1249–1272, May 1998.

[JM98b] A. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

[JM98c] A. S. Jain and S. Meeran. Deterministic job-shop scheduling: past, present and future. http://citeseer.nj.nec.com/jain98deterministic.html, 1998.

[Jon75] K.A De Jong. *An Analysis of the behaviour of a class of genetic algorithms.* PhD thesis, University of Michigan, 1975.

[Jon97] H. Jonsson. Neural network approaches to job shop problems. Master's thesis, Department of Theoretical Physics Lund University, 1997.

[Kac03] I. Kacem. *Ordonnancement multicritère des jobs-shops flexibles : formulation, bornes inférieures et approche évolutionniste coopérative.* PhD thesis, l'Ecole Centrale de Lille et l'Université de Lille 1, 2003.

[Ker92] R. Kerber. Chimerge: Discretization of numeric attributes. pages 123–128. California, USA, 1992.

[KHB02] I. Kacem, S. Hammadi, and P. Borne. Pareto-optimality approach for flexible job-shop scheduling problems: Hybridisation of evolutionary algorithms and fuzzy logic. *Journal of Mathematics and Computers in Simulation*, 2002.

[KJV83] S. Kirkpatrick, C. D. Gelattand Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, Mai 1983.

[Kod99] Y. Kodratoff. Apprentissage automatique. chapter L'Extraction de connaissance à partir des données : un nouveau sujet pour la recherche scientifique, pages 9–39. Hermes Science Publications, Paris, 1999.

[Koz] J. R. Koza. *Genetic Programming : on the programming of computers by means of natural selection.* a Bradford Book the Mit Press Cambridge, Massachusettes, London, England.

[KS96] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 114–119, 1996.

[KT00] D. A. Koonce and S. C. Tsai. Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. *Computers Industrial Engineering*, 38:361–374, 2000.

[LAL92] P. J. M. Van Laarhoven, E. H. L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[LC00] C. Y. Lee and Z. L. Chen. Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics*, 47:145–165, 2000.

[LC02] C. J. Liao and W. J. Chen. Single-machine scheduling with periodic maintenance and nonresumable jobs. *Computers and Operations Research*, 2002.

[Lee99] C. Y. Lee. Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114:420–429, 1999.

[Ler00] E. Lereno. *Apprentissage des problèmes d'ordonnancement : apllication des méthodes de filtrage de données.* PhD thesis, L'UFR des Sciences et Techniques de l'Université de Franche Comté, 2000.

[LLKS93] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *Sequencing and scheduling: algorithms and complexity.* Handbook in Operations Research and Management Science 4: Logistics of production and Inventory. 1993.

[LM98] H. Liu and H. Motoda. *Feature Extraction, Construction and selection: A Data Mining Perspective.* Kluwer Academic Publishers, 1998.

[LR01] P. Lopez and F. Roubellat. *Ordonnancement de la production.* hermès edition, 2001.

[LS95] H. Liu and R. Setiono. Chie2: Feature selection and discretization on numeric attributes. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391, 1995.

[LT94] A. Mickael Lee and H. Takagi. A framework for studying the effects of dynamic crossover, mutation and population sizing in genetic algorithms. *Artificial Intelligence*, page 1011, 1994. lecture note.

[Lyo92] P. Lyonnet. *La maintenance : mathématiques & méthodes.* Lavoisier - Paris, technique & documentation edition, 1992.

[MAR96] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of the 5th International Conference on Extending Database Technology*, Avignon, France, March 1996.

[Mat96] D. C. Mattfeld. *Evolutionary Search and the Job Shop: Investigations on Genetic Algorithms for production Scheduling.* PhD thesis, Heidelberg, Germany: Physica-Verlag, 1996.

[Mer93] T. Van De Merckt. Decision trees in numerical attribute spaces. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1016–1021, 1993.

[Mic93] R. S. Michalski. *Apprentissage Symbolique : Une approche de l'Intelligence Artificielle*, volume 1, chapter 2 : Une théorie et une méthodologie pour l'apprentissage inductif, pages 41–88. 1993.

[Mic99] D. Michaut. *Filtrage et sélection d'attributs en apprentissage.* PhD thesis, L'UFR des Sciences et Techniques de l'Université de Franche Comté, 1999.

[MK90] R. S. Michalski and Y. Kodratoff. *Evolving Research in Machine Learning*, chapter 1. San Mateo 3, 1990.

[Mon00] F. Monchy. *Maintenance : Méthodes et organisation.* Dunod edition, 2000.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1953.

[MRTD01] M. H. Mabed, M. Rahoual, E. G. Talbi, and C. Dhaenens. Algorithmes génétiques multicritères pour les problèmes de flowshop. In *Proceedings of 3ème Conférence Francophone de Modélisation et SImulation : MOSIM'01*, volume 2, pages 843–849. Troyes - France, avril 2001.

[MT63] J. Muth and G. Thomson. Industrial scheduling (prentice-hall). *Englewood Cliffs, NJ*, pages 225–251, 1963.

[NY91] R. Nakano and T. Yamada. Conventional genetic algorithms for job-shop problems. In Belew and Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 474–479. San Mateo: Morgan Kaufmann, 1991.

[PI77] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[Pin95] M. Pinedo. *Scheduling Theory, Algorithms, and Systems.* Prentice-Hall, Inc., New Jersey, 1995.

[PRJ01a] S. G. Ponnambalam, V. Ramkumar, and N. Jawahar. Comparative evaluation of genetic algorithms for job shop scheduling. *PRODUCTION PLANNING AND CONTROL*, 12(6):560–574, 2001.

[PRJ01b] S. G. Ponnambalam, V. Ramkumar, and N. Jawahar. A multiobjective genetic algorithm for job shop scheduling. *PRODUCTION PLANNING AND CONTROL*, 12(8):764–774, 2001.

[PV01] M. C. Portmann and A. Vignier. *ordonnancement de la production*, chapter Algorithmes génétiques et ordonnancement, pages 95–130. HERMES Sciences Publications, 2001.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[Qui90] J. R. Quinlan. Probabilistics decision trees. *Machine Learning*, 3:140–152, 1990.

[Qui93] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufman, 1993.

[Rab96] S. Rabaséda. *Contributions à l'extraction automatique de connaissances : application à l'analyse clinique de la marche*. PhD thesis, Université Claude Bernard - Lyon I -, 1996.

[RC96] T. Rishel and D. Christy. Incorporating maintenance activities into production planning at the master schedule versus material requirements level. *International Journal Of Production Research*, 34(2):421–446, 1996.

[RR95] M. Richeldi and M. Rossotto. Class driven statistical discretization of continuous attributes. In *Proceedings of the European Conference on Machine Learning*, pages 335–338, 1995.

[RR96] S. Rabaséda and R. Rakotomala. A comparison of some contextual discretization methods. *Information sciences*, pages 137–157, 1996.

[RS64] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Note D. S. 9 bis, SEMA, Paris, France, 1964.

[SAM96] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22th International Conference on Very Large Databases*, pages 544–555, Bombay, India, September 1996.

[Sch85] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In John J. Grefestette, editor, *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pages 93–100, 1985.

[SD94] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[SK95] M. R. Sikonja and I. Kononenko. *Discretization of continuous attributes using ReliefF*, pages 149–152. ERK, 1995.

[SS00] T. W. Sloan and J. G. Shanthikumar. Combined production and maintenance scheduling for a multiple product, single machine production system. *Production and Operation Management*, 9(4):379–399, 2000.

[Sta95] W. Stadje. Selecting jobs for scheduling on a machine subject to failure. *Discrete Applied Mathematics*, 63:257–265, 1995.

[Tai94] E. Taillard. Parallel tabu search techniques for the job-shop scheduling problem. *ORSA Journal on Computing*, 16(2):108–117, 1994.

[Tal99] E-G. Talbi. Métaheuristiques pour l'optimisation combinatoire multi-objectifs. Technical report, CNET (France Telecom), 1999.

[TN92] H. Tamaki and Y. Nishikawa. A parallel genetic algorithm based on a neighbourhood model and its applications to the job shop scheduling. In R. Manner and B. Manderick, editors, *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature*, pages 573–582, North Holland, 1992. Elsevier Science.

[Tsa97] S. C. Tsai. *Using Data Mining to Explore the Regularity of Genetic Algorithms in Job Shop Schedule Problems.* PhD thesis, Ohio University, November 1997.

[VL94] R. Vaessens and J. K. Lenstra. A computational study of local search algorithms fo job shop scheduling. *ORSA Journal on Computing*, 6:118–125, 1994.

[Vla96] E. C. Vladimir. The role of selection in genetic algorithms. Technical Report FIT-TR-1996-04, 1996.

[WC99] L. Weinstein and C. H. Chung. Integrating maintenance and production decisions in a hierarchical production planning environment. *Computers and Operations research*, 26:1059–1074, 1999.

[WHC01] M. Widmer, A. Hertz, and D. Costa. *Les métaheuristiques*, chapter 3, pages 55–93. hermes edition, 2001.

[Woo97] G. W. Woods. *A Hybrid Genetic Algorithm that Adapts to Binary and Real Coded Operators.* PhD thesis, Departement of Computer Science RMIT, 1997.

[YN92] T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, pages 281–290, 1992.

[YN95] T. Yamada and R. Nakano. Job shop scheduling by simulated annealing combined with deterministic local search. In *Metaheuristics International Conference*, pages 344–349. Hilton, Breckenridge, Colorado, USA, 1995.

[YN97] T. Yamada and R. Nakano. Genetic algorithms for job-shop scheduling problems. Technical report, Modern Heuristic for Decision Support, London, England, 1997.

[ZM99] S. V. D. Zwaan and C. Marques. Ant colony optimisation fort job shop scheduling. In *Proccedings of the third workshop on Genetic Algorithms and Artificial life GAAL99*, 1999.

[ZRF97] D. A. Zighed, R. Rakotomalala, and F. Feschet. Optimal multiple intervals discretization of continuous attributes for supervised learning. In *Proceedings of the 3td International Conference in Knowledge Discovery in Databases*, pages 295–298, 1997.

[ZRRF99] D. A. Zighed, S. Rabaseda, R. Rakotomalala, and F. Feschet. Discretization methods in supervised learning. *Encyclopedia of computer science and technology*, 40:35–50, 1999.

[ZT98a] E. Zitzler and L. Thiele. An evolutionary algorithm for multiobjective optimization: the strength pareto approach. Technical Report 43, Computer Ingineering and Communication Networks LAB (TIK), Swissfederal Institute of technology, mai 1998.

[ZT98b] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Proceedings of the 5th International Conference in Parallel Problem Solving from Nature, PPSN5*, pages 292–301, 1998.