Contribution à l'ordonnancement des activités de maintenance dans les systèmes de production.

Jihène Kaabi-Harrath

N° 1019

Année : **2004**

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 DES ACTIVITES DE MAINTENANCE DANS LES SYSTEMES DE PRODUCTION

par

Jihène KAABI-HARRATH

Soutenance prévue le 20 septembre devant la Commission d'examen :

Rapporteurs	PORTMANN M.-C. Professeur, Ecole des Mines, Nancy
	SIMEU-ABAZI Z. H.D.R., Maître de Conférences, LAG-Grenoble
Examinateurs	CHU Chengbin. Professeur, Université de Technologie de Troyes
	LESAGE J.-J. Professeur, ENS-Cachan
Directeur de thèse	ZERHOUNI N. Professeur, ENSMM, Besançon
Co-encadrant	VARNIER Ch. Maître de Conférences, ENSMM, Besançon
Invité	REBOUL D. Chef de Projet, CEGELEC, Belfort

Sommaire

Table des figures

Liste des tableaux

Table des algorithmes

A mes parents, auxquels j'exprime un profond amour et respect

à mon cher époux, l'homme de ma vie

à ma famille

Remerciements

Cette section traduit sans aucun doute l'instant le plus agréable dans la rédaction du mémoire. C'est également le moment privilégié pour mettre en avant toutes les personnes qui, de près ou de loin, directement ou indirectement, ont contribué à l'avancement de mon travail.

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 Monsieur Christophe Varnier mon co-encadreur, Maître de conférences à l'École Nationale Supérieure de Mécanique et de Microtechniques de Besançon, 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 Mademoiselle Marie-Claude Portmann, Professeur à l'institut National polytechnique de Lorraine et Madame Zineb Simeu-Abazi, HDR, maître de conférences à l'université Joseph Fourier de Grenoble 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 Chengbin Chu, Professeur à l'Université de Technologie de Troyes et monsieur Jean-Jaques Lesage, Professeur à l'ENS de Cachan, qui ont bien voulu examiner ce travail.

Je remercie également monsieur Denis Reboul, Chef de Projet à CEGELEC Belfort d'avoir accepté l'invitation pour assister à ma thèse.

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

Introduction générale

Dans le domaine de la production industrielle, les tendances actuelles indiquent que les systèmes manufacturiers performants doivent s'adapter rapidement aux fluctuations du marché (demandes aléatoires) et aux perturbations internes (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 machines, et d'une règle d'ordonnancement et d'affectation des produits aux machines qui permet de minimiser les coûts d'exploitation de ces systèmes est de nos jours un problème préoccupant dans le domaine de l'optimisation des systèmes de production.

Nos travaux présentés dans cette thèse concernent la résolution du problème d'ordonnancement conjoint des tâches de production et des activités de maintenance préventive. De par leur nature, les tâches de maintenance 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 un critère lié à la production et 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 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. Notre intérêt se focalise ensuite sur les méthodes de résolution développées dans la littérature. Ces méthodes sont classées en deux catégories : les méthodes exactes et les méthodes approchées. En particulier, nous donnons une description détaillée des règles de priorité et des algorithmes génétiques.

Le deuxième chapitre s'intéresse à la nécessité de la mise en oeuvre d'une relation de coopération entre les services de production et de maintenance. Nous présentons en premier lieu une description de la fonction maintenance : sa typologie, ses concepts ainsi que sa structure. Nous montrons en deuxième lieu les spécificités de l'ordonnancement de la maintenance et décrivons les méthodes de résolution développées dans la littérature. L'ordonnancement conjoint de la production et de la maintenance, sujet de cette thèse, n'a attiré que récemment l'attention des chercheurs. Un état de l'art sur ces travaux, dans le cas déterministe et stochastique, sera présenté.

Notre troisième chapitre s'intéresse à l'ordonnancement conjoint de la production et de la maintenance sur une machine. Le critère d'optimisation étudié est une agrégation d'un critère lié à la production et d'un autre représentant la maintenance. Nous résolvons le problème de deux façons différentes : la première ordonnance séquentiellement la production et la maintenance et la deuxième les ordonnance d'une manière intégrée. Nous proposons dans les deux cas des méthodes de résolution approchées basées sur une règle de priorité et une règle de dominance. En effet, les règles de priorité, souvent aisées à implémenter, permettent d'obtenir des bonnes solutions en un temps raisonnable. Ces solutions seront validées par une borne inférieure. Afin de tester la performance des heuristiques proposées, nous générons plusieurs problèmes de tailles différentes.

Le quatrième chapitre, traite l'ordonnancement dans un flow shop avec prise en compte de la maintenance. Ce chapitre est composé de deux parties.

Dans la première partie, nous nous intéressons au cas d'un flow shop à deux machines. Ce problème est $\mathcal{NP}$-*difficile* au sens fort. Afin de réduire la combinatoire de l'espace de recherche, nous nous restreignons au cas d'un flow shop de permutation. Nous proposons ainsi une borne inférieure à ce problème et un algorithme de type "séparation et évaluation" (PSE).

Dans la deuxième partie, nous nous intéressons au cas d'un flow shop à plusieurs machines. Pour résoudre ce problème, nous développons un algorithme génétique. Le choix des algorithmes génétiques s'est imposée tout naturellement à nous pour la recherche de bonnes solutions. En effet, leurs aspects se basant sur l'aléatoire et l'évolution à l'aide des opérateurs spécifiques, leur permet d'explorer toutes les zones de l'espace de recherche. Nous proposons un nouveau codage approprié des solutions génétiques. La population

initiale est formée de solutions générées à l'aide d'un algorithme de descente stochastique couplé avec une adaptation d'une heuristique proposée pour le cas d'une machines. Dans le cas de flow shop à deux machines, une comparaison avec l'algorithme de séparation et évaluation est élaborée. Pour le flow shop à plusieurs machines, nous testons l'efficacité de l'algorithme génétique par une variation de quelques paramètres du problème tels que les degrés d'importance de la production et de la maintenance (figurant dans la fonction objectif).

Le rapport se termine par des conclusions ainsi que des perspectives de recherche envisagées.

Chapitre 1

L'ordonnancement dans les ateliers de production

Résumé : *Dans le cadre d'une compétition internationale de plus en plus forte entre les entreprises, la réduction des coûts est un objectif crucial. Celle-ci peut être obtenue par une meilleure utilisation des ressources. En particulier, la productivité peut être fortement affectée par l'ordonnancement des opérations sur les machines. Les problèmes d'ordonnancement apparaissent dans des domaines aussi variés que l'organisation opérationnelle du travail dans les usines, la planification de grands projets. L'étude des problèmes d'ordonnancement est également d'un intérêt théorique toujours renouvelé pour les chercheurs surtout avec la prise en compte des disponibilités des machines. Ce problème a commencé à attirer l'attention de nombreux chercheurs durant ces vingt dernières années. Dans ce chapitre, nous présentons quelques notions de base concernant les problèmes d'ordonnancement dans les ateliers de production. Les méthodes de résolution exactes et approchées utilisées dans cette thèse sont ensuite plus développées, parmi ces méthodes on trouve les algorithmes génétiques et les règles de priorité.*

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. Contrairement aux systèmes de production mono produit ou aux ateliers gérés en Kanban où les flux de produit s'écoulent naturellement et doivent en principe s'auto-réguler, l'ordonnancement trouve sa place essentiellement dans tous les ateliers ayant une production diversifiée en flux poussé, gérée par des ordres de fabrication.

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.

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).

Nous aborderons dans ce premier chapitre le thème de l'ordonnancement en général.

L'ordonnancement est en effet 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 limitons 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. Ce chapitre est divisé en deux parties. Dans la première partie , nous présentons quelques généralités sur les problèmes d'ordonnancement dans les ateliers de production, à savoir les types d'ateliers, les critères d'optimisation ainsi qu'une brève description de la notion de complexité des problèmes d'optimisation combinatoire. La deuxième partie, est consacrée à la description des principales méthodes de résolution des problèmes d'ordonnancement proposées dans la littérature et plus précisément les règles de priorité et les algorithmes génétiques.

1.2 L'ordonnancement dans les ateliers de production

1.2.1 Généralités

Nous tirons de la littérature les deux définitions ci-dessous de l'ordonnancement :

Définition 1.1. [CC88]
Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. Les problèmes d'ordonnancement, apparaissent dans tous les domaines de l'économie: l'informatique, la construction (suivi de projet), l'industrie (problèmes d'ateliers, gestion de production), l'administration (emploi du temps). [...]. Les tâches sont le dénominateur commun des problèmes d'ordonnancement, leur définition n'est ni toujours immédiate, ni toujours triviale [...]. Enfin, il faut programmer les tâches de façon à optimiser un certain objectif qui sera, suivant le cas, la minimisation de la durée totale (c'est le critère le plus fréquemment employé) ou le respect des dates de commande ou de lissage des courbes de main d'oeuvre ou encore la minimisation d'un coût. D'une manière générale, trois types d'objectifs sont essentiels dans la résolution des problèmes d'ordonnancement : l'utilisation efficace des ressources, un délai d'exécution des tâches aussi faible que possible et le respect des dates d'achèvement prescrites à l'avance.

Définition 1.2. [GOT93]
Une tâche est un travail élémentaire nécessitant un certain nombre d'unités de temps et de ressources. Ordonnancer un ensemble de tâches, c'est programmer leur exécution en leur allouant les ressources requises et en fixant leur date de début.

Ces deux définitions nous permettent de déterminer les éléments caractérisant un problème d'ordonnancement. Un problème d'ordonnancement est composé de façon générale d'un **ensemble de tâches** soumises à certaines **contraintes**, et dont l'exécution nécessite des **ressources**. Résoudre un problème d'ordonnancement, consiste à organiser ces tâches, c'est à dire à déterminer leur dates de démarrage et à leur attribuer des ressources, de telles sorte que les contraintes soient respectées.

Une *tâche* est un travail (ou job) dont la réalisation nécessite un nombre d'opérations élémentaires. Chaque opération élémentaire nécessite un certain nombre d'unités de temps (sa durée) et d'unités de ressources.

Une *ressource* est un moyen, technique ou humain, permettant la réalisation des tâches et dont la disponibilité limitée ou non est connue à priori. On distingue deux types de ressources: les ressources renouvelables et les ressources consommables. Une ressource est renouvelable si après avoir été allouée à une tâche, elle redevient disponible pour les autres. Une ressource est consommable si après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches restant à exécuter.

Une *contrainte* exprime des restrictions sur les valeurs que peuvent prendre conjointement les variables représentant les relations reliant les tâches et les ressources. On distingue les contraintes temporelles et les contraintes de ressources.

Les contraintes temporelles intègrent:

- les contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches (délais de livraison, disponibilité des approvisionnements) ou à la durée totale d'un projet ;
- les contraintes d'antériorité et plus généralement les contraintes de cohérence technologique, qui décrivent le positionnement relatif de certaines tâches par rapport à d'autres ;
- les contraintes de calendrier liées au respect d'horaires de travail, etc.

Les contraintes de ressources traduisent le fait que les ressources sont disponibles en quantité limitée. On distingue deux types de contraintes de ressources, liées à la nature disjonctive ou cumulative des ressources. Une ressource disjonctive ne peut être utilisée que par une tâche à la fois. Par contre dans une ressource cumulative les ensembles de tâches non réalisables simultanément sont de cardinalité quelconque.

1.2.2 Description d'un problème d'ordonnancement

Les données d'un problème d'ordonnancement sont les tâches et leurs caractéristiques, les contraintes potentielles, les ressources et le(s) critère(s) d'optimisation. Souvent un travail J_i ne peut commencer son exécution avant une date de disponibilité et doit être achevé avant une date échue. Les travaux sont souvent liés entre eux par des relations d'antériorité ; si ce n'est pas le cas, on dit qu'ils sont indépendants.

1.2.2.1 Notations

Nous utilisons les notations présentées dans la plupart des ouvrages et articles sur le sujet. Soient $J = \{J_i/i = 1,2,\ldots,n\}$ l'ensemble des n travaux à ordonnancer et $M = \{M_j/j = 1,2,\ldots,m\}$ l'ensemble des m machines.

i indice de l'opération

j indice de machine

$p_{i,j}$ durée de la $j^{ème}$ opération du travail J_i sur la machine M_j (processing time)

r_i date minimale de début du travail J_i (release date)

d_i date d'achèvement souhaitée du travail J_i (due date)

w_i poids du travail J_i (weight)

t_i date de début d'exécution du travail J_i

C_i date de fin d'exécution du travail J_i (completion time)

F_i temps de présence dans l'atelier ou durée de flot (flow time) du travail J_i : $F_i = C_i - r_i$

L_i écart par rapport à la fin souhaitée ou retard algébrique (lateness) du travail J_i : $L_i = C_i - d_i$

E_i avance du travail J_i (earliness) : $E_i = max(d_i - C_i,0)$

T_i retard vrai du travail J_i (tardiness) : $T_i = max(C_i - d_i,0)$

U_i indicateur de retard du travail J_i : $U_i = 1$ si $T_i > 0$, $U_i = 0$ sinon

La résolution des problèmes d'ordonnancement nécessite le plus souvent le calcul ou l'évaluation de critères de performance (quantitatifs ou qualitatifs). Ces critères seront décrits dans la section suivante.

1.2.2.2 Critères d'optimisation

Le choix d'un ordonnancement parmi les ordonnancements réalisables se fait en fonction d'un ou plusieurs critère(s) que l'on cherche à optimiser. Généralement, les critères considérés sont des fonctions des dates de fin C_i des produits. Ces critères peuvent être de type *Max* ou de type *Somme*. Il s'agit de considérer le maximum ou la somme (éven-

tuellement pondérée) sur tous les travaux d'une certaine mesure ou d'une combinaison de plusieurs mesures. Les mesures les plus intéressantes, considérées le plus souvent dans la littérature, pour un travail J_i, sont données dans le tableau 1.1

Critères	Maximum	Somme	Somme pondérée
Fin de traitement	$C_{max} = max_{i=1,n}(C_i)$	$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$	$\bar{C}_w = \sum_{i=1}^{n} w_i C_i$
Décalage	$L_{max} = \max_{i=1,n} L_i$	$\bar{L} = \frac{1}{n}\sum_{i=1}^{n} L_i$	$\bar{L}_w = \sum_{i=1}^{n} w_i L_i$
Retard	$T_{max} = max_{i=1,n} T_i$	$\bar{T} = \frac{1}{n}\sum_{i=1}^{n} T_i$	$\bar{T}_w = \sum_{i=1}^{n} w_i T_i$
Avance	$E_{max} = max_{i=1,n} E_i$	$\bar{E} = \frac{1}{n}\sum_{i=1}^{n} E_i$	$\bar{E}_w = \sum_{i=1}^{n} w_i E_i$
Durée de flot	$F_{max} = max_{i=1,n} F_i$	$\bar{F} = \frac{1}{n}\sum_{i=1}^{n} F_i$	$\bar{F}_w = \sum_{i=1}^{n} w_i F_i$
Nombre de retards		$\bar{U} = \frac{1}{n}\sum_{i=1}^{n} U_i$	$\bar{U}_w = \sum_{i=1}^{n} w_i U_i$

TAB. 1.1 – *Critères d'optimisation*

Une caractéristique intéressante pour un critère est sa régularité. Un critère est dit *régulier* si pour deux ordonnancements S_1 et S_2, le fait que chaque produit de S_1 se termine au plus tard ou en même temps que le produit correspondant dans S_2, entraîne que la valeur du critère de S_1 est inférieure ou égale à la valeur du critère de S_2.

Dans un ordonnancement *semi-actif*, aucun travail ne peut se terminer plus tôt sans changer l'ordre d'exécution sur les machines. Dans un ordonnancement *actif*, aucun travail ne peut se terminer plus tôt sans retarder au moins un autre travail [Bak74]. Il a été démontré, en effet, que pour tout critère régulier, les ensembles des ordonnancements semi-actifs et actifs sont *dominants*, c'est à dire qu'ils contiennent au moins une solution optimale [Chu95].

Les problèmes d'ordonnancement sont très différents d'un atelier à l'autre, et il n'existe pas de méthode universelle permettant de résoudre efficacement tous les cas. Dans ce qui suit, nous décrivons les problèmes classiques d'atelier.

1.2.3 Les problèmes d'ordonnancement

Une classification des problèmes d'ordonnancement peut s'opérer selon le nombre de machines et leur ordre d'utilisation pour fabriquer un produit (gamme[1] de fabrication) qui dépend de la nature de l'atelier. Un atelier se définit par le nombre de machines qu'il contient et par son type. Les différents types possibles sont les suivants :

– une machine ($\emptyset$) : chaque travail est constitué d'une seule opération. Dans ce cas, on appelle indifféremment travail et tâche ;

1. La gamme donne l'enchaînement logique des opérations

– machines parallèles : elles remplissent, a priori, toutes les mêmes fonctions. Selon leur vitesse d'exécution, on distingue :

  – **machines identiques** (P) : la vitesse d'exécution est la même pour toutes les machines M_j et pour tous les travaux J_i ;

  – **machines uniformes** (Q) : chaque machine M_j a une vitesse d'exécution propre et constante. La vitesse d'exécution est la même pour tous les travaux J_i d'une même machine M_j ;

  – **machines indépendantes** (R) : la vitesse d'exécution est différente pour chaque machine M_j et pour chaque travail J_i.

– machines dédiées : elles sont spécialisées à l'exécution de certaines opérations. Dans cette catégorie, chaque travail est constitué de plusieurs opérations. En fonction du mode de passage des opérations sur les différentes machines, trois ateliers spécialisés sont différenciés, à savoir :

  – **flow Shop** (F) (figure 1.1) : c'est un cas particulier du problème d'ordonnancement d'atelier pour lequel le cheminement des travaux est unique : les n travaux utilisent les m machines dans l'ordre 1,2,...,m (ligne de production) ;

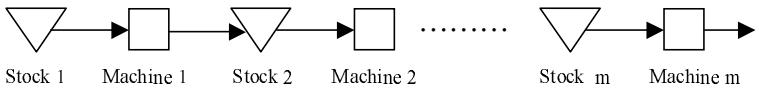


FIG. 1.1 – *Représentation d'un atelier flow Shop*

  – **job Shop** (J) (figure 1.2) : les n travaux doivent être exécutés sur les m machines, sous des hypothèses identiques à celles du flow shop, la seule différence est que les séquences opératoires relatives aux différents travaux peuvent être distinctes et sont propres à chaque travail ;

  – **Open Shop** (O) : c'est un modèle d'atelier moins contraint que le flow shop et le job shop, car l'ordre des opérations n'est pas fixé à priori. Le problème d'ordonnancement consiste d'une part à déterminer le cheminement de chaque travail et d'autre part à ordonnancer les travaux en tenant compte des gammes trouvées. Notons que ces deux problèmes peuvent être résolus simultanément. Comparé aux autres modèles d'ateliers multi-machines, l'open shop qui n'est pas non plus courant dans les entreprises, n'est pas très étudié dans la litté-

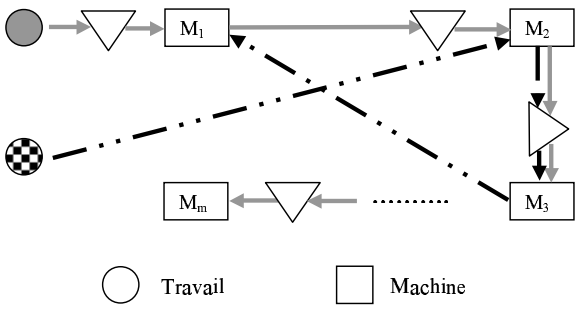


FIG. 1.2 – *Représentation d'un atelier job shop*

rature. Un exemple d'application de l'open shop est le cas des télécommunications numériques par satellite où les travaux sont l'ensemble de paquets à émettre par une station terrienne, une opération serait le paquet d'une station émettrice vers un récepteur et enfin les machines seront les répéteurs (canaux) du satellite.

Il existe une très grande variété de problèmes d'ordonnancement. Pour leur identification et leur classification nous adoptons la notation proposée par Graham et *al.* [GLLK79] et par Blazewicz [BLK83]. Cette notation est constituée de trois champs $\alpha|\beta|\gamma$.

Le premier champ α est constitué de deux éléments : $\alpha = \alpha_1\alpha_2$ et décrit l'environnement des machines utilisées :

- le paramètre $\alpha_1 \in \{\emptyset,P,Q,R,F,J,O\}$ décrit le type des machines utilisées ;
- le paramètre α_2 indique le nombre de machines utilisées. Lorsque α_2 n'est pas précisé, le nombre de machines est quelconque.

Le deuxième champ $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7$ décrit les caractéristiques des travaux et des machines. Il indique si des éléments spécifiques sont à prendre en compte telles que des dates au plus tôt, des précédences entre les travaux etc.

Le dernier champ γ indique le critère d'optimisation, il peut donc prendre de nombreuses valeurs et peut être une combinaison de plusieurs critères.

La réalisation d'un problème d'ordonnancement doit concilier plusieurs objectifs. *L'aspect statique* (ou off-line, on connaît par avance la liste des travaux à effectuer et leurs caractéristiques) consiste à générer un plan de réalisation des travaux sur la base de données prévisionnelles. *L'aspect dynamique* (ou on-line, les travaux peuvent arriver à des instants quelconques et viennent s'ajouter à l'ordonnancement en cours) consiste

à prendre des décisions en temps réel compte tenu de l'état des ressources et de l'avancement dans le temps des différents travaux.

1.2.4 Modélisation et représentation des ordonnancements

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

1.2.4.1 Modélisation d'un problème d'ordonnancement

Le graphe disjonctif est un outil de modélisation abondamment utilisé pour résoudre les problèmes d'atelier multi-machines [RS64].

Un graphe disjonctif est constitué d'un ensemble de noeuds et de deux types d'arcs. Les noeuds du graphe représentent les opérations composant les travaux à réaliser, auxquels s'ajoutent deux sommets fictifs appelés source et puits correspondant respectivement au début et à la fin des jobs.

Deux types d'arc reliant les noeuds du graphe : les arcs conjonctifs et les arcs disjonctifs. Un *arc conjonctif* connecte deux opérations consécutives d'un même travail, décrivant ainsi la contrainte de précédence liant ces opérations. Un *arc disjonctif* connecte deux opérations appartenant à des travaux différents qui utilisent la même machine. La source est reliée via des arcs conjonctifs aux premières opérations de chaque travail, et les dernières opérations de chaque travail sont reliées par des arcs conjonctifs au puits. Chaque arc est pondéré par la durée de l'opération de l'extrême droite, excepté les arcs de la source qui sont pondérés par 0. La figure 1.3 représente le graphe disjonctif d'un problème de flow shop à trois machines et trois travaux.

Pour définir un ordonnancement admissible il va falloir *arbitrer* chaque arc disjonctif, une orientation telle que le graphe résultant soit sans circuit. On dit alors que le graphe disjonctif est arbitré [CC88].

Exemple 1.1. *Considérons le problème* $F3|n = 3,d_i|\sum T_i$ *avec les données fournies dans le tableau 1.2.*

Une solution à ce problème serait : $\begin{cases} M1 : J_1 \rightarrow J_2 \rightarrow J_3 \\ M2 : J_1 \rightarrow J_3 \rightarrow J_2 \\ M3 : J_1 \rightarrow J_3 \rightarrow J_2 \end{cases}$

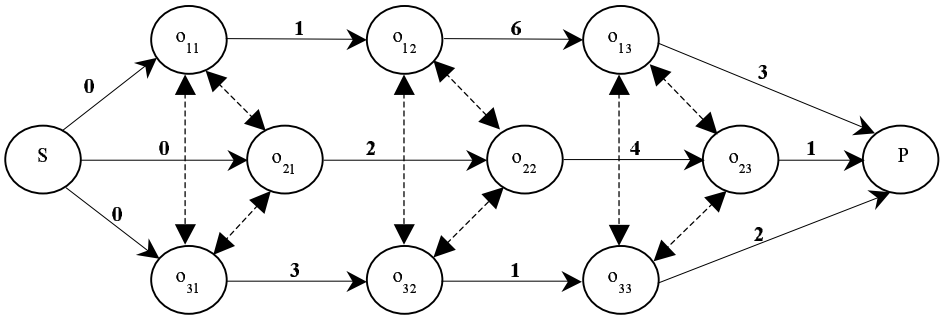


FIG. 1.3 – *Graphe disjonctif d'un flow shop à trois machines et trois travaux*

J_i	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	d_i
J_1	1	6	3	10
J_2	2	4	1	11
J_3	3	1	2	12

TAB. 1.2 – *Données du problème* $F3|n = 3,d_i|\sum T_i$

La figure 1.4 représente le graphe disjonctif arbitré de cette solution où o_{ij} est l'opération j du travail i.

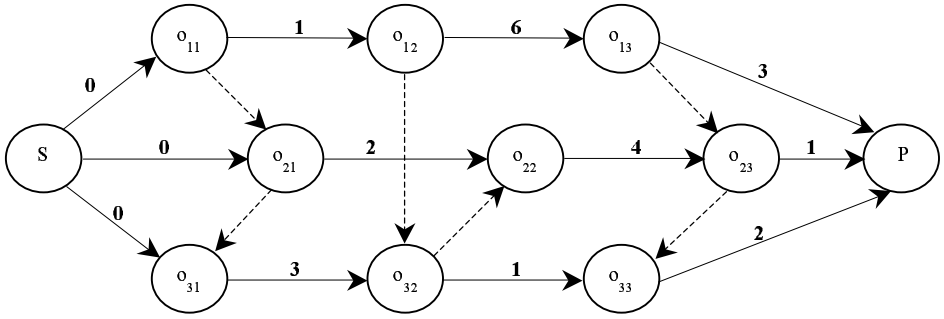


FIG. 1.4 – *Graphe disjonctif arbitré de l'exemple précédant*

1.2.4.2 Représentation des ordonnancements

Le diagramme de Gantt, du nom de son développeur Henry Gantt (1861-1919), est le moyen le plus simple et le plus utilisé pour représenter un ordonnancement, à la fois dans la littérature et dans les entreprises. Dans le cas de problèmes d'atelier, ce diagramme se compose de plusieurs lignes horizontales, chacune d'entre elles désignant une machine. Les opérations exécutées sur une machine donnée sont représentées sous forme de barres ayant

des longueurs proportionnelles à leurs temps opératoires. L'axe des abscisses représente le temps, les barres sont positionnées aux dates de début d'exécution des opérations. Ainsi, le diagramme de Gantt permet de montrer l'occupation des machines dans le temps, les séquences de traitement sur chaque machine et les dates de fin des travaux.

La figure 1.5 représente la solution du problème considéré dans le paragraphe précédent dont la somme des retards est égale à 1.

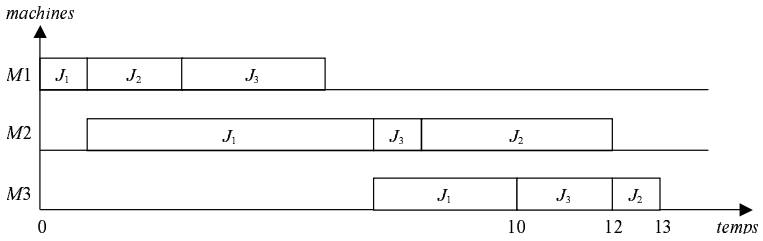


FIG. 1.5 – *Diagramme de Gantt de la solution du problème* $F3|n = 3,d_i|\sum T_i$

Les problèmes d'ordonnancement d'ateliers sont des problèmes combinatoires extrêmement difficiles et il n'existe pas de méthodes universelles permettant de résoudre efficacement tous les cas [GJ79]. Une classification de ces problèmes selon leur difficulté sera donnée dans la section suivante.

1.2.5 Complexité des problèmes

La théorie de la complexité [Coo71] a pour but d'analyser les coûts de résolution, notamment en terme de temps de calcul, des problèmes d'optimisation combinatoire. Elle permet d'établir une classification des problèmes en plusieurs niveaux de difficulté. Elle fait 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 $P||C_{max}$ avec n travaux et y un entier positif. Un problème de décision associé à ce problème d'optimisation peut être le suivant : existe-t-il un ordonnancement avec $C_{max} \leq y$.

On appellera :

– *algorithme polynomial*, un algorithme dont le nombre d'opérations est un polynôme de n : par exemple $O(n^{\alpha})$ avec α une constante ;

– *algorithme non polynomial*, un algorithme dont le nombre d'opérations n'est pas borné par un polynôme de n : par exemple $\Omega(a^n)$ avec $a > 1$ une constante.

La classe des problèmes $\mathcal{NP}$ ($\mathcal{NP}$ pour Non déterministe Polynomial) est la classe des problèmes de décision pouvant être résolus par un algorithme polynomial non déterministe. Parmi la classe des problèmes $\mathcal{NP}$, on distingue deux grandes sous-classes : la classe des problèmes polynômiaux (la classe $\mathcal{P}$) et la classe des problèmes $\mathcal{NP}$-*complet* (la classe $\mathcal{NPC}$).

La classe P est la classe des problèmes résolvables par un algorithme polynomial déterministe. C'est la classe des problèmes les plus "faciles" de $\mathcal{NP}$. Malheureusement, la classe $\mathcal{P}$ des problèmes que l'on peut résoudre en un temps polynomial est assez limitée.

La classe $\mathcal{NPC}$ est la classe des problèmes dont on n'a pas trouvé d'algorithme polynomial pour les résoudre, sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$. Cependant, si on pouvait résoudre un problème de $\mathcal{NPC}$ en un temps polynomial, on obtiendrait automatiquement des algorithmes polynômiaux pour tous les problèmes de $\mathcal{NPC}$.

Quand un problème est $\mathcal{NP}$-*complet*, il n'est pas raisonnable d'espérer construire un algorithme polynomial le résolvant. Mais dans certains cas, on peut construire des algorithmes très efficaces appelés algorithmes *pseudo-polynomiaux*, le problème étudié est alors dit $\mathcal{NP}$-*complet au sens faible*. Dans le cas contraire, il est dit $\mathcal{NP}$-*complet au sens fort*.

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

Pour résoudre un problème d'ordonnancement de manière efficace, il faut prendre en compte la particularité de chaque problème. C'est pourquoi il y a autant d'approches que de problèmes d'ordonnancement. Néanmoins, il existe quelques approches qui fournissent des solutions approchées de bonne qualité pour un certain nombre de problèmes. Dans ce qui suit, nous passons en revue différentes approches exactes ou approchées pour la résolution des problèmes d'ordonnancement.

1.3 Les méthodes de résolution

Les méthodes de résolution des problèmes d'ordonnancement puisent dans toutes les techniques de l'optimisation combinatoire (programmation mathématique, programmation dynamique, procédure par séparation et évaluation, théorie des graphes...). Ces

méthodes garantissent en général l'optimalité de la solution fournie. Mais les algorithmes dont la complexité n'est pas polynomiale ne peuvent pas être utilisés pour des problèmes de grandes tailles, d'où la nécessité de construire des méthodes de résolution approchées, qu'on appelle méthodes heuristiques, efficaces pour ces problèmes souvent $\mathcal{NP}$-*difficiles*. Dans la suite, nous présentons les méthodes de résolution les plus connues. On commencera par les méthodes exactes, ensuite on présentera les méthodes heuristiques.

1.3.1 Les méthodes exactes

Généralement, il n'est pas possible de construire un algorithme polynomial résolvant un problème d'optimisation $\mathcal{NP}$-*difficile*. Toutefois, on peut mettre au point des méthodes efficaces pour une grande partie des énoncés. Pour certains problèmes, on appliquera une Procédure par *Séparation et Evaluation (PSE)*, pour d'autres, la *programmation dynamique* ou encore la *programmation linéaire*. Le but des méthodes exactes est de trouver, en un temps de calcul le plus court possible, la solution optimale du problème. nous détaillons dans ce qui suit la Procédure par Séparation et Évaluation.

La PSE est utilisée pour résoudre d'une façon exacte des problèmes d'optimisation combinatoire. Cette méthode consiste à construire une arborescence dont la racine correspond à l'espace des solutions du problème initial. Dans le cas d'un problème de minimisation, un majorant (BS) de la valeur de la fonction objectif pour une solution réalisable est avant tout calculé en utilisant une méthode approchée. La méthode utilise deux concepts : *le branchement* ou encore *la séparation* qui consiste à diviser un ensemble de solutions en sous-ensembles et *l'évaluation* qui consiste à minorer ou majorer les solutions.

Le branchement consiste à décomposer un sommet de l'arborescence représentant une sous-classe de solutions d'un problème en une partition de sous-sous-classes. L'utilisation seule du branchement revient à effectuer une énumération complète de l'espace des solutions. Aussi l'évaluation des sommets permet-elle d'éliminer les branches qui ne contiennent pas de solutions optimales. Plus précisément, avant l'exploration d'un sommet, une borne inférieure (toujours pour un problème de minimisation) est calculée. Cette borne est un minorant de la valeur de la fonction objectif pour le problème associé au sommet. Si cette valeur dépasse ou est égale à celle de la borne supérieure BS (la sous-classe ne donne pas de meilleure solution), le sommet est éliminé. Par ailleurs, l'exploration d'un sommet est arrêtée s'il s'agit d'une feuille (solution réalisable), c'est à dire si on ne peut plus effectuer de séparation ou si on connaît une solution optimale appartenant au sommet. La borne supérieure est mise à jour à chaque fois qu'une

solution réalisable donnant une meilleure valeur pour la fonction objectif est trouvée. L'algorithme 1.1 suivant décrit le schéma général d'une PSE.

Algorithme 1.1 Schéma général d'une procédure par séparation et évaluation

Début
  Calculer une borne supérieure BS
  Tant que il existe des sommets non explorés **faire**
    Sélectionner le sommet à séparer et créer ses fils
    Pour tous les sommets S créés **faire**

      Si S représente une solution complète ou si on connaît une solution optimale de S **alors**
        calculer sa valeur du critère d'optimisation et mettre à jour BS
      Sinon
        calculer la borne inférieure BI
        Si $BI < BS$ **alors**
          placer S dans la liste des sommets à explorer
        Fin si
      Fin si
    Fin pour
    Éliminer tout sommet tel que $BI > BS$
  Fin tant que
Fin

Les performances de la méthode dépendent évidemment de la qualité des bornes inférieures et supérieures mais aussi de la rapidité, en terme de temps de calcul, à obtenir ces bornes. En outre il existe plusieurs stratégies pour la sélection des sommets à séparer ; dont les plus connues sont la profondeur d'abord, la largeur d'abord et la manière progressive. De nombreuses PSE ont été proposées dans la littérature de l'ordonnancement. Une référence dans le domaine est celle développée par Carlier et Pinson [CP89a] pour résoudre le problème du job shop.

1.3.2 Les méthodes heuristiques

Le principe d'une méthode heuristique est de trouver en un temps polynomial une solution réalisable la meilleure possible.

1.3.2.1 Les méthodes constructives

Le principe de ces méthodes est de construire une solution à partir des données initiales. Ce sont des méthodes itératives où, à chaque itération, on complète une solution partielle. En ordonnancement, on peut développer de nombreuses méthodes constructives. Par exemple celles qui à chaque itération placent successivement toutes les opérations d'un travail, celles qui au contraire à chaque itération placent une opération au plus sur une des machines. Les méthodes constructives se distinguent par leur rapidité et leur grande simplicité. Le principal défaut de ces méthodes réside malheureusement dans la qualité des solutions obtenues. Le fait de vouloir opérer à tout prix le meilleur choix à chaque étape est une stratégie dont les effets peuvent être catastrophiques à long terme. La méthode constructive la plus connue, développée dans le cadre de la recherche d'une séquence de durée minimale sur un atelier de type flow shop est l'algorithme de Nawaz et *al.* [NEH83].

Parmi les méthodes constructives on peut citer les règles de priorité. Nous allons dans le paragraphe suivant détailler ces méthodes.

Le problème de la détermination de règles de priorité performantes a suscité et suscite toujours une littérature très abondante. Un très grand nombre de règles ou combinaisons de règles ont été définies et testées. L'analyse de cette littérature montre que, à part quelques résultats généraux bien connus, il est extrêmement délicat de vouloir définir le domaine d'application (type d'atelier, critère d'évaluation) d'une règle donnée.

Les règles de priorité étant souvent aisées à implémenter, cette méthode est très répandue et permet d'obtenir de bons ordonnancements en un temps raisonnable. Chaque règle appliquée séparément sur la file d'attente d'une ressource est généralement performante pour un critère particulier [Gen95]. Cependant, des approches permettant de combiner, par agrégation, différentes règles élémentaires afin de réaliser des compromis pour tenter de satisfaire des objectifs multiples ont également été développées.

La principale difficulté dans cette approche est de faire une corrélation entre les règles utilisées et leur impact sur les indicateurs de performances employés. De plus, les performances de chaque règle, ou combinaison de règles, sont différentes d'un contexte

de production à un autre et il est difficile de tirer des conclusions sur l'efficacité générale de l'une ou l'autre de ces règles.

Classification des règles de priorité Les règles de priorité peuvent être classées selon plusieurs critères [BPH82]:

• Elles peuvent être **locales** ou **globales**. Une règle est *locale* si elle ne prend en compte que les informations locales à la file d'attente d'une ressource. Tandis que elle est *globale* si elle prend en compte, par exemple, la charge sur la machine suivante.

• Elles peuvent être **statiques** ou **dynamiques**. Une règle est *statique* si la valeur de la priorité reste invariante pendant le séjour du travail dans la file d'attente. Une règle *dynamique* peut conserver l'ordre relatif entre deux travaux, mais la valeur de la priorité doit être remise à jour dès qu'un nouveau travail arrive.

• Elles peuvent être classées selon les **informations prises en compte** par celles ci. On peut distinguer les règles qui prennent en compte les durées opératoires, les dates de fin au plus tard, combinent les deux, n'utilisent ni l'un ni l'autre, etc.

• Elles peuvent être classées selon leur **complexité**. Cette complexité peut être due à la combinaison pondérée de règles, utilisation de paramètres à régler, prise en compte des spécificités de l'atelier, etc.

Ces classifications peuvent être intéressantes pour mettre en évidence les difficultés d'implantation des règles, soit par un simulateur, soit directement dans l'atelier.

Nous allons dans la section suivante donner une description des règles de priorité les plus connues dans la littérature ainsi que leurs limites d'utilisation.

Les règles de priorité statiques

Pour la résolution du problème $1|\ |\sum C_i$, Smith [Smi56] a proposé la règle optimale SPT (Shortest Processing Time) qui consiste à séquencer les tâches dans l'ordre croissant des durées opératoires. Cette règle peut être très mauvaise pour des variantes de ce même critère du fait des temps d'attente des opérations de grande durée qui peuvent être très importants. Concernant le problème de minimisation du temps moyen pondéré de séjour ou encore le stock d'en-cours $(1|\ |\sum w_i C_i)$, la règle $WSPT$ (Weighted Shortest Processing Time) est optimale [Smi56]. Cette règle consiste à séquencer les tâches dans l'ordre croissant des quotients durées/poids (p_i/w_i).

En ce qui concerne le respect des dates de fin au plus tard, la règle SPT reste performante pour le problème $(1|\ |\bar{T})$, notamment quand les dates de fin au plus tard

sont relativement serrées. Les règles prenant en compte explicitement les dates de fin au plus tard sont par contre bien meilleures pour la variance du retard vrai. La règle EDD (Earliest Due Date) proposée par Jackson [Jac55] consiste à séquencer les tâches dans l'ordre croissant des dates de fin au plus tard. Cette règle permet de résoudre d'une manière optimale les problèmes $1|\ |L_{max}$ et $1|\ |T_{max}$. Le nombre moyen de tâches en retard ($1|\ |\frac{1}{n}\sum U_i$) est minimisé par l'algorithme de $Hodgson - Moore$ [Moo68]. Celui-ci consiste à construire progressivement un ordonnancement guidé par la règle EDD. Dès qu'un retard apparaît dans l'ordonnancement partiel, la tâche de plus grande durée déjà placée est reportée en fin d'ordonnancement.

Les règles de priorité dynamiques

La règle ATC (Apparent Tardiness Cost) proposée par Vepsalainen et Morton [VM87] pour résoudre le problème $1|\ |\sum w_i T_i$ consiste à calculer, à chaque instant t et pour toute les tâches i non encore placées (A forme l'ensemble de ces tâches), un indice de classement $\Pi_i(t)$ puis sélectionner la tâche avec le plus grand indice. Cet indice est défini comme suit :

$$\Pi_i(t) = \frac{w_i}{p_i} \exp\big(-\max(0,d_i - t - p_i)/(k.\bar{p})\big) \quad avec\ k = 2\ et\ \bar{p} = \frac{1}{|A|}\sum_{\{i\in A\}} p_i$$

La règle $COVERT$ (Weighted Cost $OVER$ Time) constitue un cas particulier en ce sens qu'elle peut être très performante en ce qui concerne le retard vrai mais elle dépend de la qualité de l'estimation des temps d'attente devant les machines. Plusieurs méthodes d'estimation de ces temps ont été proposées : simulation préalable avec la règle $FIFO$ (First In First Out), répartition de la marge de départ sur chaque tâche, estimation dynamique dans l'atelier. La règle COVERT est définie comme suit :

$$\max_{\{i\in A\}} \left\{ \frac{w_i}{p_i} \max\left[0{,}1 - \frac{\max(0,d_i - t - p_i)}{k.p_i}\right] \right\}$$

Russel et *al.* [RDET86] a étudié ces méthodes d'estimation, avec différentes valeurs du paramètre k, et les a comparé aux règles classiques les plus performantes. Les auteurs ont montré que, pour chaque critère, COVERT est aussi performante que la meilleure règle classique pour le critère considéré, même avec des estimations rudimentaires des temps d'attente.

La règle $X - RM$ proposée par Morton et Ramnath [MR95] est une adaptation de la règle ATC pour prendre en compte les temps morts. Le principe de cette règle consiste à calculer pour chaque travail i, sa priorité Π_i avec la règle ATC puis effectuer une

correction sur ces priorités pour réduire celles des travaux critiques qui sont en retard. La règles $X - RM$ est définie comme suit :

$$\max_{\{i\in A\}}\left\{\Pi_i\Big[1 - B.\frac{\max(0,r_i - t)}{\hat{p}}\Big]\right\} \text{ avec } B \in \{1.6,\ 2\} \text{ et } \hat{p} = \frac{1}{|A|}\sum_{\{i\in A\}} p_i \text{ ou } \hat{p} = \min_{\{i\in A\}} p_i$$

Concernant le problème $1|r_i|\sum T_i$, Chu et Portmann [CP89b] ont proposée une règle dynamique appelée $PRTT$ (Priority Rule for Total Tardiness). Cette règle est définie à un instant, pour un travail J_i, t donné par : $\max(r_i,\ \Delta) + \max(\max(r_i,\ \Delta) + p_i,\ d_i)$. Cette règle sera mieux décrite dans le chapitre 3.

Le tableau 1.3 résume toutes les règles citées ci-dessus.

Règle	Type	Problème(s)	Référence
SPT	statique	$1\|\ \|\sum C_i, \frac{1}{n}\sum T_i, \sum U_i$	Smith [Smi56]
WSPT	statique	$1\|\ \|\sum w_i C_i$	Smith [Smi56]
EDD	statique	$1\|\ \|T_{max}, L_{max}$	Jackson [Jac55]
ATC	dynamique	$1\|\ \|\sum w_i T_i$	Vepsalainen et Morton [VM87]
COVERT	dynamique	$1\|\ \|\sum w_i T_i$	Russel et *al.* [RDET86]
X-RM	dynamique	$1\|r_i\|\sum w_i T_i$	Morton et Ramnath [MR95]
PRTT	dynamique	$1\|r_i\|\sum T_i$	Chu et Portmann [CP89b]

TAB. 1.3 – *Principales règles de priorité*

1.3.2.2 Les méthodes amélioratrices

Le principe de ces méthodes n'est plus de construire un ordonnancement à partir des données initiales du problème, mais de modifier le résultat d'une solution admissible en vue d'améliorer la valeur de la fonction objectif. La plupart de ces méthodes utilisent la notion de voisinage de solution, on parle aussi de méthodes par voisinage.

A partir d'une solution initiale (générée aléatoirement ou avec une heuristique), une méthode par voisinage ou encore méthode de recherche locale réalise un processus itératif qui consiste à remplacer la solution courante par l'un de ses voisins en tenant compte de la fonction objectif. La méthode s'arrête et retourne la meilleure solution trouvée. Le critère d'arrêt peut être un nombre d'itérations atteint ou une valeur de la fonction objectif atteinte.

Les méthodes de recherche locale diffèrent essentiellement par le système de voisinage utilisé et la stratégie de parcours du système de voisinage. Parmi les méthodes qui ont

prouvé leur efficacité dans la résolution de problème d'optimisation combinatoire, nous pouvons citer le recuit simulé [KGV83], la recherche tabou [Glo86], la recherche dispersée [Glo77] et la colonie des fourmis [CDM91]. Le lecteur intéressé trouvera une comparaison de ces différentes méthodes dans le récent état de l'art de Taillard et *al.*[TGGP01].

Parmi les méthodes amélioratrices, on peut citer également les *algorithmes génétiques* qui sont inspirés de la théorie de l'évolution. Ce sont des algorithmes aléatoires, mais au lieu de chercher à améliorer progressivement une seule solution (comme les algorithmes de recherche locale : le recuit simulé et le tabou), ils font évoluer une population de solutions qui s'améliore globalement par des techniques de reproduction, de sélection et de mutation. L'idée de base est que travailler avec une population permet d'identifier et d'explorer les propriétés qu'ont en commun les bonnes solution. Nous distinguons et présentons dans la section 1.3.3 en détail le principe des algorithmes génétiques ainsi que leur application aux problèmes d'ordonnancement.

1.3.2.3 Méthodes basées sur l'intelligence artificielle

Il s'agit de méthodes qui utilisent des techniques de représentation des connaissances et de résolution de problèmes issus de l'intelligence artificielle. De nombreux travaux ont abordé les problèmes d'ordonnancement à l'aide de ces outils. Citons par exemple la propagation par contrainte qui alliée à des règles de production permet de résoudre des problèmes d'ordonnancement ; les systèmes experts et les systèmes à base de connaissances utilisant de l'apprentissage, comme par exemple la mémoire artificielle[GOT93].

1.3.3 Les Algorithmes Génétiques

Les algorithmes génétiques (AG) sont certainement la branche des algorithmes évolutionnistes la plus connue et la plus utilisée. Ils ont été crées par analogie avec des phénomènes naturels. Dans ce cas, il s'agit de simuler l'évolution naturelle d'organismes (individus), génération après génération, en respectant des phénomènes d'hérédité et une loi de survie. Dans une population d'individus, ce sont en général les plus forts , c'est à dire les mieux adaptés au milieu, qui survivent et donnent une descendance. Par ailleurs, on suppose que les qualités et les défauts peuvent être hérités des parents de manière stochastique. De tels algorithmes furent développés dès 1950 par des biologistes qui utilisaient des ordinateurs pour simuler l'évolution des organismes. Vers la fin des années 60, John Holland [Hol75] et son équipe, relayés plus tard par d'autres chercheurs [Gol89] et [Dav91], ont adapté ces algorithmes pour la recherche de solutions de problèmes d'optimisation, en développant une analogie entre un individu dans une population et une

solution d'un problème dans un ensemble de solutions.

Les éléments constituant ces algorithmes ainsi que le principe de leur fonctionnement seront abordés dans ce paragraphe. Une étude des applications des algorithmes génétiques à la résolution des problèmes d'ordonnancement en général sera ensuite présentée.

1.3.3.1 Terminologie

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

Définition 1.3. *(Gène) Un gène est une suite de bases azotées qui contient le code d'une protéine donnée. On appellera gène la suite de symboles qui codent la valeur d'une variable.*

Dans le cas général, un gène correspond à un seul symbole (0 ou 1 dans le cas binaire). Une mutation changera donc systématiquement l'expression du gène muté.

Définition 1.4. *(Chromosome) Un chromosome est constitué d'une séquence finie de gènes qui peuvent prendre des valeurs appelées allèles qui sont prises dans un alphabet qui doit être judicieusement choisi pour convenir du problème étudié.*

Définition 1.5. *(Individu) On appellera individu une des solutions potentielles. Dans la plupart des cas un individu sera représenté par un seul chromosome, dans ce cas, par abus de langage, on utilisera indifféremment individu et chromosome.*

Définition 1.6. *(Population) On appellera population l'ensemble des solutions potentielles qu'utilise l'AG.*

Définition 1.7. *(Génération) On appellera génération l'ensemble des opérations qui permettent de passer d'une population Π_i à une population Π_j. Ces opérations seront généralement : sélection des individus de la population courante, application des opérateurs génétiques, évaluation des individus de la nouvelle population.*

Par abus de langage, on pourra aussi appeler $i^{ème}$ *génération* l'ensemble des individus après i itérations de l'algorithme.

1.3.3.2 Mise en Oeuvre d'un Algorithme Génétique

Les algorithmes génétiques font partie de la classe des algorithmes dits stochastiques. En effet, une grande partie de leur fonctionnement est basée sur le hasard. Cependant, ce hasard est dirigé grâce à la fonction d'évaluation qui permet d'introduire dans les opérateurs génétiques une quantité de déterminisme utile afin d'obtenir une solution de bonne qualité. Le schéma global d'un algorithme génétique est constitué de 6 éléments principaux.

1. Une population initiale d'individus ;
2. Une fonction de codage/décodage des chromosomes ;
3. Des opérateurs génétiques (Croisement, Mutation, ...) ;
4. Une fonction d'évaluation ;
5. Un algorithme de sélection (Tournoi, Roulette, ...) ;
6. Des paramètres.

La mise en oeuvre d'un algorithme génétique sur un ordinateur nécessite de régler certains paramètres. Comme tout algorithme itératif, il faut définir un critère d'arrêt comme par exemple une valeur seuil ou un nombre maximum de générations. Les probabilités de croisement et de mutations sont à définir aussi. Elles peuvent varier en fonction du numéro de la génération. Enfin, la taille de la population gérée par l'algorithme doit être dimensionnée. Très souvent, ces paramètres sont réglés de manière empirique. Ce réglage a un impact très important sur la convergence de l'algorithme et sur les résultats obtenus. Les paragraphes suivants détaillent chaque étape de cet algorithme.

Codage des solutions C'est une modélisation d'une solution d'un problème quelconque en un chromosome : c'est à dire une séquence des gènes. L'algorithme de base s'appuie sur un codage sous forme de chaînes 0/1 de longueur fixe (voir Figure 1.6). 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], [CPP95]. 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.

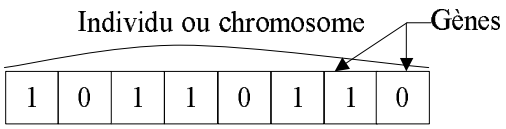


FIG. 1.6 – *Codage Binaire*

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 les rend plus efficaces et plus adaptés.

Deux sortes de codages couramment utilisés pour la représentation des solutions ; ce sont le codage direct et indirect [GRG01]. Dans un *codage direct*, un chromosome représente explicitement une solution alors que dans un *codage indirect*, un chromosome comporte uniquement des caractéristiques permettant de construire la solution à laquelle il correspond. La difficulté principale consiste en la conception du contenu de ces gènes de manière à décrire toutes les données du problème et respecter ses contraintes. Chaque gène représente donc une partie ou une variable élémentaire du problème.

Construction de la population Généralement, la population initiale est choisie aléatoirement, chacun des bits de chacun des chromosomes étant choisi au hasard. Cependant, rien n'interdit d'utiliser des *méthodes par construction simples*, de fournir des solutions déjà connues que l'on cherche à améliorer ou d'autres *algorithmes de recherche* qui fourniront à l'AG des solutions ayant subi une première phase d'optimisation. Le choix de l'initialisation se fera en fonction des connaissances que l'utilisateur a sur le problème.

Evaluation L'opérateur d'évaluation n'est pas anodin. Il est utilisé par l'opérateur de sélection pour faire son choix des individus à conserver. Il va donc influencer la convergence de l'algorithme. En effet, il va falloir définir la fonction de *fitness* de telle sorte qu'elle ne favorise pas trop les meilleurs individus, mais aussi qu'elle permette de repérer les individus manifestement mauvais et inutiles pour la recherche. L'évaluation est aussi

l'opérateur le plus dépendant du problème à traiter. On doit généralement définir un opérateur d'évaluation par type de problème, voire par problème.

Si le problème étudié consiste à maximiser un critère donné, ce dernier peut servir directement pour l'évaluation des individus.

Dans le cas où le critère d'optimisation est à minimiser, il est nécessaire de chercher son complémentaire pour se ramener au cas de la 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 :

$$F(x) = \begin{cases} C_{max} - C(x) & si \;\; C(x) > 0 \\ 0 & sinon \end{cases} \tag{1.1}$$

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.

Sélection L'opérateur de sélection est l'opérateur qui va devoir choisir dans la population courante les individus qui auront le « droit » de survivre et de se reproduire. Généralement, la sélection s'effectue sur toute la population, il existe cependant des cas où on peut préférer ne prendre en compte qu'une partie de la population, et générer ainsi un fossé de génération [Gol89]. On aurait ainsi moins de chance de perdre les bons individus, et on permet ainsi le croisement entre individus issus de générations différentes. Plusieurs méthodes sont possibles pour choisir les individus. Nous allons évoquer ici les plus courantes.

La sélection *par roulette* place sur une roue des cases de tailles différentes proportionnelles au rapport *meilleure fitness/fitness moyenne* de chaque individu. La figure 1.7 montre une roulette de sélection pour une population de cinq individus dont les évaluations figurent à droite de la roulette (l'objectif ici est de maximiser la fonction d'évaluation). On lance ensuite n fois la roue, et pour chaque case désignée, on fait une copie de l'individu correspondant dans la population.

Ils existent d'autres méthodes de sélection à savoir la sélection *par tournoi* (k-tournament en anglais) qui consiste à prendre au hasard k individus de la population, et leur fait faire un tournoi. Celui qui a la meilleure *fitness* est copié dans la nouvelle population. Nous pouvons citer aussi la sélection *par classement* (Ranking en anglais)

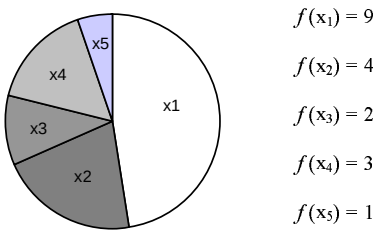


FIG. 1.7 – *Roulette de sélection*

qui consiste à classer les individus de la population selon la fonction d'évaluation et à retenir uniquement un nombre fixé de génotypes[2] représentant les meilleurs individus pour la phase de reproduction. L'inconvénient de cette technique est de fixer un seuil à la sélection ce qui prive des individus relativement bons, mais contenant de bonnes configurations, de participer dans la construction de la nouvelle population.

croisement L'opérateur de croisement est généralement considéré comme l'opérateur d'exploitation. On entend par là qu'il va permettre de découvrir de meilleures solutions en combinant les avantages de solutions déjà découvertes. Dans sa version classique, on va choisir deux individus de la population, et leur appliquer l'opérateur de croisement ou bien les recopier dans la population ; tels quels, en fonction de la probabilité de croisement.

Nous distinguons plusieurs modes de croisements binaires dont les plus importants sont le croisement à un point, le croisement multi-points et le croisement uniforme.

– **Croisement à un point** : il consiste à choisir une position de croisement au hasard et à échanger deux parties à partir de la position choisie (Figure 1.8).

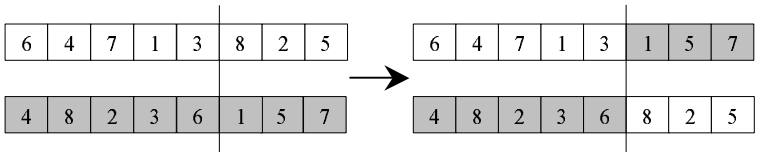


FIG. 1.8 – *Croisement à un point*

– **Croisement multi-points** : il consiste à choisir plusieurs points de coupe au hasard. Ces points de coupe vont ensuite représenter les limites des zones à échanger entre les deux parents (Figure 1.9).

2. Un génotype représente l'ensemble des valeurs des gènes du chromosome

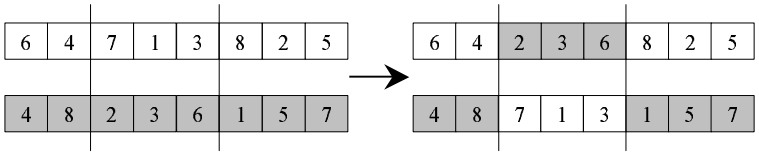


FIG. 1.9 – *Croisement multi-points*

– **Croisement uniforme** : il constitue la généralisation du principe d'échange introduit par le croisement à un point. Il procède en effet à l'échange de chaque élément selon une probabilité fixée (Figure 1.10). Le jeu de paramètres se réduit donc à la donnée de cette probabilité.

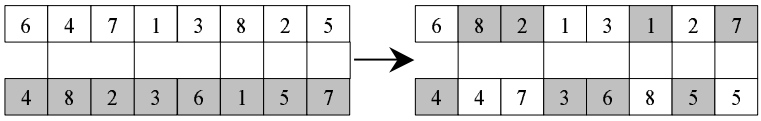


FIG. 1.10 – *Croisement uniforme*

Mutation C'est un changement aléatoire de faible probabilité de la valeur d'un gène. La mutation n'apporte pas forcément d'améliorations néanmoins, elle permet de faire évoluer les générations des individus et de les diversifier. Cette diversification a pour but d'éviter la stagnation de la population et la convergence prématurée vers un minimum local.

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 :

– transposition de deux gènes consécutifs (échange de deux gènes consécutifs dans un chromosome) ;

– Échange de deux gènes distants. Il constitue donc une extension de l'opérateur précédent. Le degré d'altération dépend de la distance entre les deux éléments impliqués. Si le premier se situe à la position i et le second à la position j, alors l'échange implique $2|j-i|-1$ permutations élémentaires ;

– Insertion décalage : il s'agit de sélectionner deux gènes, de placer le premier dans la position du deuxième et de décaler tous les gènes se trouvant après celui ci.

Après avoir défini tous les opérateurs d'un AG, nous donnons ci-dessous le principe d'un algorithme génétique.

Algorithme 1.2 Le principe des algorithmes génétiques

Entrée : P_0 *{une population initiale de n individus}*

Début

$i := 0$

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

$i := i + 1$

Soit P_i une population vide,

Tant que P_i ne contient pas n individus **faire**

Choisir aléatoirement I_1 et I_2 deux individus de P_{i-1}.

Appliquer l'opérateur de croisement sur I_1 et I_2 pour obtenir les enfants E_1 et E_2.

Ajouter E_1 et E_2 à P_i.

Fin tant que

Appliquer l'opérateur de mutation avec la probabilité p_m sur certains individus de P_i

$P_i := P_i \cup P_{i-1}$

sélectionner p individus de P_i et supprimer les n autres.

Fin tant que

Fin

Convergence et réglage des paramètres Les algorithmes génétiques présentent l'avantage d'être modulaires. En effet, leur application ne nécessite pas une connaissance de la particularité du problème étudié. Néanmoins, leur efficacité dépend fortement du réglage des différents paramètres caractérisant ces algorithmes. De tels paramètres sont la taille de la population, le nombre maximal des générations, la probabilité de mutation p_m, et la probabilité de croisement p_c.

Les deux premiers paramètres vont directement dépendre de la complexité du problème. Si la taille de la population est trop grande le temps de calcul de l'algorithme peut s'avérer très important, et si elle est trop petite, l'algorithme peut converger trop rapidement vers un mauvais chromosome. Cette importance de la taille est essentiellement due à la notion de *parallélisme implicite* qui implique que le nombre d'individus traité

par l'algorithme est au moins proportionnel au cube du nombre d'individus. Il s'ensuit que le bon choix se fait par la détermination d'un bon compromis. De même le nombre maximal des générations doit représenter un compromis de la qualité des solutions et du temps de calcul.

La probabilité de croisement p_c dépend de la forme de la fonction d'évaluation (appelée *fitness*). Son choix est en général heuristique (tout comme pour p_m). Plus elle est élevée, plus la population subit de changements importants.

La probabilité de mutation p_m est généralement faible puisqu'un taux élevé risque de conduire à une solution sous-optimale. En effet, l'étude de l'influence de ce taux montre qu'avec une probabilité de mutation très élevée, cet opérateur perturbe la convergence en induisant une oscillation de la valeur moyenne du critère optimisé [Mes99]. En revanche, un faible taux de mutation permet d'assurer une bonne exploration de l'espace de recherche sans d'autant perturber la convergence.

Plutôt que réduire p_m, une autre façon d'éviter que les meilleurs individus soient altérés est d'utiliser la reconduite explicite de l'élite dans une certaine proportion. Ainsi, bien souvent, les meilleurs 5% , par exemple, de la population sont directement reproduits à l'identique, l'opérateur de reproduction ne jouant alors que sur les 95% restants. Cela est appelée une "*stratégie élitiste*".

Après avoir exposé les grands principes des algorithmes génétiques, nous nous intéressons dans la suite à la résolution des problèmes d'ordonnancement par de tels algorithmes dans la section suivante.

1.3.3.3 Application des algorithmes génétiques à l'ordonnancement

La puissance croissante des ordinateurs permet depuis plusieurs années de remplacer les heuristiques simples utilisées jusqu'à présent dans les progiciels commercialisés d'ordonnancement par des méthodes plus sophistiquées. La plupart de ces méthodes appartiennent à la grande famille des métaheuristiques (une métaheuristique est un ensemble de concepts applicables à un large ensemble de problèmes d'optimisation combinatoire pour créer de nouvelles heuristiques). Parmi ces métaheuristiques on trouve les algorithmes génétiques qui feront l'objet de cette section. Nous considérons essentiellement leur application à des problèmes d'ordonnancement.

De nombreuses approches génétiques ont été proposées dans la littérature pour résoudre les problèmes d'ordonnancement. Les paragraphes suivants présentent un survol de celles les plus connues classées selon le type du problème traité.

Problème à une machine Ce problème consiste à rechercher la séquence optimale de n tâches sur une seule machine. Une solution est décrite par l'ordre de passage des tâches sur la machine.

Codage des solutions Il existe de nombreux codages qui peuvent être utilisés pour représenter une permutation de n éléments correspondant à l'ordre des n tâches sur la machine. On recense de la littérature trois types de codages.

Le premier codage est appelé « permutation ». Il consiste simplement à ranger dans un vecteur le numéro des tâches dans l'ordre où on les place sur la machine.

Le deuxième codage est appelé « rang ». Il consiste à donner dans un vecteur pour chaque tâche sa position relative sur la machine.

Exemple 1.2. *L'ordonnancement considéré consiste à séquencer six tâches selon l'ordre suivant : tâche* 5*, tâche* 1*, tâche* 3*, tâche* 2*, tâche* 4 *et enfin tâche* 6*. Un tel exemple est représenté selon le codage de permutation (resp. le codage de rang) dans la figure 1.11 (resp. la figure 1.12)*

Position dans la séquence	1	2	3	4	5	6
Permutation	5	1	3	2	4	6

FIG. 1.11 – *Le vecteur permutation*

Numéro de la tâche	1	2	3	4	5	6
Rang	2	4	3	5	1	6

FIG. 1.12 – *Le vecteur rang*

Le troisième codage, proposé par Portmann [Por96], consiste à représenter le séquencement par une « matrice de permutation » MT. Cette matrice de permutation est définie par :

$$\begin{cases} MT(i,i) = 0 \\ MT(i,j) = 1 & \text{si i précède j dans la permutation} \\ MT(i,j) = -1 & \text{sinon} \end{cases}$$

Exemple 1.3. *L'ordonnancement décrit dans la figure 1.11 et la figure 1.12 selon le codage de permutation et le codage de rang est représenté ici sous la forme matricielle dans la figure 1.13.*

	1	2	3	4	5	6
1	0	1	1	1	-1	1
2	-1	0	-1	1	-1	1
3	-1	1	0	1	-1	1
4	-1	-1	-1	0	-1	1
5	1	1	1	1	0	1
6	-1	-1	-1	-1	-1	0

FIG. 1.13 – *La matrice de permutation MT*

Génération de la population initiale Pour générer un individu, il faut déterminer l'ordre de passage des opérations puis transcrire cet ordre, soit sous la forme du codage permutation, soit sous la forme du codage rang ou encore sous la forme de codage matriciel.

Plusieurs façons permettent d'obtenir la population initiale. La première consiste à générer le codage de manière totalement aléatoire. Cette approche permet d'obtenir une sous-population très diversifiée. Une autre façon serait d'utiliser des méthodes de résolution approchées avec ou sans introduction de décisions aléatoires dans les méthodes. Ceci permet d'obtenir une sous-population d'assez bonne qualité. Toutefois, pour obtenir de bons individus initiaux pour le critère somme des retards pondérés, on peut utiliser des heuristiques spécifiques de ce problème, par exemple l'ordre croissant des dates de fin au plus tard (l'algorithme EDD) et à égalité, l'ordre décroissant des pénalités, éventuellement amélioré par une procédure simple d'amélioration par voisinage.

Opérateurs de croisement Concernant les opérateurs de croisement, la littérature propose plusieurs variantes selon le codage par liste de permutation. Par exemple, dans le cas d'un codage de permutation ou de rang, nous pouvons appliquer le **croisement 1.X.** proposé par Davis [Dav85]. Le principe de cet opérateur est décrit ci-dessous :

Soient $E1$ et $E2$ deux enfants obtenus en croisant deux père $P1$ et $P2$ et p un point de croisement choisi aléatoirement.

• L'enfant $E1$ reçoit les mêmes gènes que $P1$ entre la position 1 et la position p ; le reste sera complété selon l'ordre d'apparition des gènes manquants dans $P2$.

• L'enfant $E2$ reçoit les mêmes gènes que $P2$ entre la position 1 et la position p ; le reste sera complété selon l'ordre d'apparition des gènes manquants dans $P1$.

La figure 1.14 représente un exemple d'application de l'opérateur de croisement 1.X

sur un problème à une machine comportant six taches.

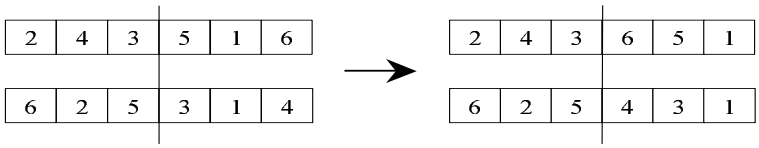


FIG. 1.14 – *Croisement 1.X*

Un autre type de croisement, appelé **croisement MPX** (Maximal Preservative X), a été proposé par Mülhenbein [Mül93] pour le problème du voyageur de commerce. L'idée de cet opérateur est d'insérer une partie du chromosome d'un parent dans le chromosome de l'autre parent de telle façon que le chromosome résultant soit le plus proche possible de ses parents. C'est un croisement à deux points. Les deux fils sont obtenus de manière symétrique. Le principe de cet opérateur est donné dans l'algorithme 1.3.

Algorithme 1.3 Croisement MPX

Début

Choisir aléatoirement deux individus $I1$ et $I2$;

Choisir aléatoirement deux positions $p1$ et $p2$;

Étape 1. L'enfant $E1$ reçoit les mêmes gènes que $I1$ entre les positions $p1$ et $p2$;

Étape 2. Le $i^{ème}$ gène de l'enfant $E1$ reçoit le $i^{ème}$ gène de $I2$. Si Cette recopie ne respecte pas les contraintes, aller à l'étape 3. Sinon $i = i + 1$, aller à l'étape 2.

Étape 3. Le $i^{ème}$ gène de l'enfant $E1$ reçoit le $i^{ème}$ gène de $I1$. Si cette recopie crée des doublons, aller à l'étape 4.

Étape 4. Le $i^{ème}$ gène de l'enfant $E1$ reçoit un gène de la zone de croisement de $I2$ qui respecte les contraintes (premier non compris). Si $i < n$ aller à l'étape 2, sinon Fin.

Fin

La figure 1.15 représente un exemple d'application de l'opérateur de croisement MPX sur un problème à une machine comportant six taches.

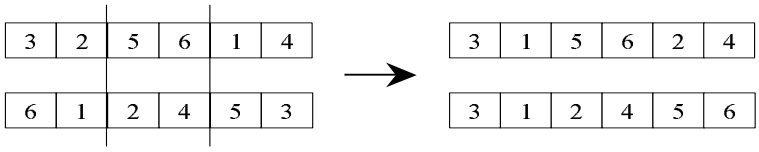


FIG. 1.15 – *Croisement MPX*

Opérateurs de mutation Les opérateurs de permutation décrits dans la section 1.3.3.2 peuvent être utilisés pour créer de la diversité dans la population. Dans le cas d'addition de contraintes de précédence, de tels opérateurs ne sont plus valables. Ils doivent être suivis d'un processus de correction pour respecter les contraintes imposées. Cela pourra induire un coût élevé en terme du temps de calcul. Pour remédier à ce problème Lee et Kawa [LK96] ont proposé un opérateur de mutation appelé **PPS** (**P**recedence **P**reserving **S**hift mutation). Cet opérateur sera décrit à travers l'exemple ci-dessous.

Exemple 1.4. *Considérons un problème à six tâches avec les contraintes de précédence suivantes : {1 avant 6}, {2 avant 4} et {5 avant 3 avant 6}. Les étapes nécessaires à cet opérateur sont :*

1. Choisir un gène de l'individu sélectionné :

2	*4*	*5*	$\bar{3}$	*1*	*6*

2. Déterminer l'ensemble des gènes en relation de précédence avec le gène sélectionné. Ici, il s'agit de {2, 3, 5, 6}.

3. Parmi les gènes trouvés, localiser dans le chromosome le gène le plus à droite et le gène le plus à gauche :

$\hat{2}$	*4*	*5*	$\bar{3}$	*1*	$\hat{6}$

4. Choisir un gène entre les deux gènes localisés et le permuter avec le gène choisi dans la première étape :

$\hat{2}$	*4*	*5*	*1*	$\bar{3}$	$\hat{6}$

Problème de job shop Il s'agit d'un atelier contenant plusieurs machines, chaque machine ayant des fonctionnalités différentes. Un travail i est composé de n_i opérations, chacune nécessitant une machine particulière dans l'atelier.

Codage des solutions Nous allons présenter deux approches pour coder une solution du problème de job shop.

La première est un **codage de permutation généralisée** proposé pour la première fois par Whitley, Starkweather et Shaner dans [Dav91]. Il consiste à utiliser un seul vecteur qui contient autant de fois un numéro de travail que le travail comporte d'opérations.

La première occurrence du travail dans le vecteur correspond à la première opération de ce travail, la deuxième occurrence à la deuxième opération du travail, etc. L'exemple 1.4 est le résultat de codage d'une solution au problème du job shop à quatre travaux et trois machines.

1	2	1	3	1	3	4	2	3	4	2	4

TAB. 1.4 – *Le codage de permutation généralisé*

Le deuxième est le **codage par matrice ternaire avec blocs diagonaux de permutation** proposé par Potmann [Por96]. Cette matrice MT est décrit de la manière suivante :

$$\begin{cases} MT(i,j) = 1 & \text{si i précède j dans le graphe de précédence} \\ MT(i,j) = -1 & \text{si MT(j,i)=1} \\ MT(i,j) = 0 & \text{si il n'y a pas de relation de précédence entre i et j} \end{cases}$$

Opérateurs de croisement et de mutation Plusieurs opérateurs de croisement on été construits dans la littérature pour le cas du job shop. A titre d'exemple, nous présentons ci-dessous le croisement GPX proposé par Mattfeld [Mat96] pour le codage de type « permutation généralisée ». Cet opérateur sera décrit à travers l'exemple suivant :

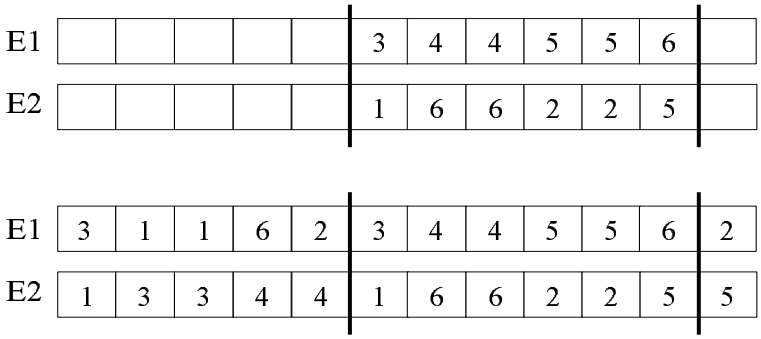
Exemple 1.5. *Considérons un problème à six travaux et deux machines. Les étapes nécessaires à cet opérateur sont :*

1. Choisir deux points de croisement qui séparent le chromosome en trois parties : partie gauche, partie centrale et partie droite.

P1	1	6	2	2	3	3	4	4	5	5	6	1
P2	3	3	4	4	1	1	6	6	2	2	5	5

2. Recopier la partie centrale de $P1$ (resp. $P2$) dans la partie correspondante de l'enfant $E1$ (resp. $E2$).
3. Les cases non remplies de l'enfant $E1$ (resp. $E2$) sont complétées à partir de $P2$ (resp. $P1$) en allant de gauche à droite et en respectant la cardinalité[3] de chaque opération.

3. la cardinalité d'une opération est égale au nombre total de machines



Pour le codage indirect, les opérateurs de mutations classiques pour les permutations peuvent être utilisés. Pour le codage direct au moyen d'une matrice ternaire, une mutation possible serait de choisir une case hors diagonale dans une des sous-matrices correspondant à une des machines. Pour cette case, on prend l'inverse de la valeur de la matrice correspondant à l'individu à muter. Ensuite, on continu comme lors d'un croisement (à un seul individu). Afin d'éliminer les zéros se trouvant hors diagonale, on remplace ce dernier par la valeur se trouvant dans la matrice à muter. Le processus est réitéré jusqu'à ce qu'il n'y ait plus de 0.

Problème de flow shop La plupart des approches considèrent un codage de liste des travaux proposé par Syswarda [Sys89]. Ainsi cette liste représentera l'ordre de passage des différents travaux dans l'atelier.

Le codage de permutation généralisé présenté dans le cas du job shop peut aussi être appliqué pour les cas du flow shop.

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

Nous commençons par le cas du flow shop Hybride à un seul étage qui n'est autre qu'un problème à machines parallèles. Dans ce cas, un travail est composé d'une seule opération. On se place dans le cas où toutes les machines sont identiques.

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

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

Exemple 1.6. *Considérons un problème de cinq travaux à exécuter dans un flow shop hybride à un étage et trois machines parallèles. Un codage possible d'une solution au problème est le suivant :* ***(1, 2, 3, 2, 1)****. Cela signifie qu'il faut affecter les travaux* 1 *et* 5 *sur la machine* 1*, les travaux* 2 *et* 4 *sur la machine* 2 *et le travail* 3 *sur la machine* 3*.*

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

$$\begin{cases} M(i,j) = 1 & \text{si les travaux i et j sont sur la même machine} \\ M(i,j) = 0 & \text{sinon} \end{cases}$$

Exemple 1.7. *Considérons le même problème que l'exemple 1.6. Un codage possible de la même solution que l'exemple 1.6 est donné par la matrice suivante :*

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	1	0	1	0
5	1	0	0	0	1

TAB. 1.5 – *La matrice M*

Le troisième codage de type « indirect », appelé **CI3-AFF** (**C**odage **I**ndirect pour l'**AFF**ectation), consiste à établir une liste de priorités pour l'ensemble des travaux fournis, par exemple, par un vecteur permutation. Cette solution nécessite l'emploi d'un générateur de solutions. Il aura pour objet de déterminer une solution à partir d'un codage donné.

Génération de la population initiale Concernant le codage direct, une population initiale peut être générée en utilisant une méthode qui fabrique de manière aléatoire plusieurs individus et qui respecte la condition de dominance « pas de machine inac-

tive ». Pour le codage indirect, les méthodes décrites dans le paragraphe 1.3.3.3 peuvent être appliquées sur l'ensemble des travaux.

Opérateurs de croisement et de mutation Pour le codage CI3-AFF, les opérateurs de croisement et de mutation décris dans les paragraphes 1.3.3.3 et 1.3.3.3. Pour ce type de problème et pour le codage CD1-AFF, il suffit d'appliquer l'opérateur de croisement simple en un point. Néanmoins, cet opérateur ne garantit pas l'obtention de deux enfants valides. En effet, il est possible que ce type de croisement génère une solution dans laquelle une machine serait laissée inactive. Dans ce cas une phase de « réparation » est nécessaire. Cette phase consiste à déplacer un travail vers toute machine laissée libre. concernant la mutation, une simple probabilité de modifier un gène dans les chromosomes est suffisante, suivie si nécessaire de l'opérateur correctif.

Exemple 1.8. *Nous reprenons les mêmes données de l'exemple précédent. La figure 1.16 représente le croisement en un point de deux individus.*

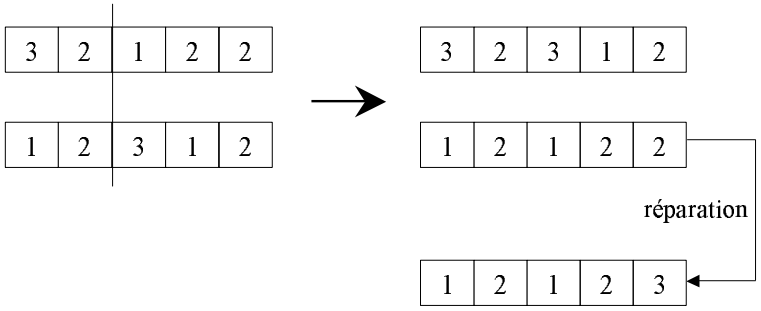


FIG. 1.16 – *Opérateur de croisement pour le flow shop hybride à un étage*

Concernant le flow shop hybride à plusieurs étages, une adaptation des différents opérateurs cités ci-dessus est nécessaire.

1.4 Conclusion

A travers ce chapitre, nous avons présenté quelques notions de base concernant les problèmes d'ordonnancement dans les ateliers de production. Nous avons ensuite donné une brève description des méthodes de résolution exactes et approchées les plus utilisées dans la littérature.

La deuxième partie montre les algorithmes génétiques et les règles de priorité comme deux outils efficacement appliqués dans la résolution des problèmes d'ordonnancement

de la production et justifie la littérature abondante concernant leur application à de tels problèmes. Dans la suite, nous nous intéressons à montrer et à justifier l'adaptation de ces outils pour la résolution des problèmes qui introduiront les contraintes d'indisponibilité des machines.

Dans la littérature actuelle portant sur la planification de la production, les ressources (machines) sont toujours considérées comme disponibles à tout moment ou éventuellement durant certaines fenêtres de temps. Dès lors le service maintenance d'une entreprise n'est jamais prioritaire sur la production, pour effectuer des interventions préventives. Nous montrons dans le chapitre suivant l'importance de la prise en compte de l'aspect maintenance dans le planning de production pour satisfaire dans les délais les clients.

Chapitre 2

L'ordonnancement de la maintenance et de la production : nécessité de la coopération

Résumé : *La majeure 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. Les dates et durées d'indisponibilité sont déterministes dans certain cas : congés de personnel, opérations de maintenance préventive sur les machines de l'atelier. Ces mêmes données sont parfois imprévisibles et aléatoires, comme par exemple les pannes de machines. 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 donnons une justification de la nécessité de mettre en oeuvre une coopération entre la production et la maintenance. Ce qui nous amène à donner une description de la maintenance ainsi que la spécificité de son ordonnancement. Nous présenterons ensuite un état de l'art sur les travaux réalisés dans le cas où les machines sont sujettes à des périodes d'indisponibilité qui peuvent être dues à des interventions de maintenance ou autre.

2.1 Introduction

Dans la littérature actuelle portant sur la planification de la production, les ressources (machines) sont toujours considérées comme disponibles à tout moment ou éventuellement durant certaines fenêtres de temps. Dès lors le service maintenance d'une entreprise n'est jamais prioritaire sur la production, pour effectuer des interventions préventives. Lorsque le responsable de la maintenance demande à immobiliser une ressource pour effectuer une opération de maintenance préventive, cela est généralement ressenti par les responsables de production comme une perturbation supplémentaire et forcément malvenue. Cet arrêt de la machine a généralement été planifiée séparément des tâches de production et ne s'insère donc que très rarement dans un créneau horaire où elle ne nuirait pas à la productivité. Cependant, les interventions de maintenance préventive ont un rôle prépondérant, puisqu'elles permettent au système de production de fonctionner de façon nominale. En effet, le coût engendré par une panne (impliquant un arrêt non programmé de la production, des retards conséquents de livraison, etc.) est largement supérieur à celui d'un arrêt prévu de la production. Les besoins du service ordonnancement sont donc quelque peu antagonistes : pour satisfaire dans les délais les clients, il doit utiliser de façon optimale l'ensemble des installations ; mais il doit également prévoir les interventions de maintenance dans le planning de production.

Ce chapitre a pour objectif de montrer la nécessité de mettre en place des relations de coopération entre la production et la maintenance pour traiter le problème d'ordonnancement conjoint de ces deux aspects. Il est composé de deux parties.

La maintenance étant une fonction complexe, nous présentons dans la première partie les définitions des différentes formes de maintenance ainsi qu'une description de la fonction maintenance. Nous décrirons aussi l'activité d'ordonnancement de la maintenance ainsi que ses spécificités. Nous justifierons ensuite la nécessité de l'ordonnancement conjoint de la production et de la maintenance dans l'industrie.

La deuxième partie sera consacrée à l'étude des problèmes d'ordonnancement avec contraintes d'indisponibilité rencontrés dans la littérature. Nous donnerons un état de l'art des différents travaux concernant ce problème.

2.2 Description de la fonction maintenance

Toute entreprise industrielle poursuit des objectifs de productivité, de rentabilité et de croissance. Pour y répondre, elle doit notamment maîtriser ses équipements de pro-

duction dans les dimensions technique, économique et sociale ; rôle qui incombe à la fonction maintenance ([BO87]) ([Ogu88]). Ainsi, selon la norme AFNOR NF X 60-010, la maintenance est « l'ensemble des activités destinées à rétablir un bien [équipement] dans un état ou dans des conditions données de sûreté de fonctionnement, pour accomplir une fonction requise. Ces activités sont une combinaison d'activités techniques, administratives et de management ». À cette définition, la norme AFNOR NF X 60-000 ajoute que « bien maintenir, c'est assurer ces opérations au coût global optimum ».

Nous avons défini la maintenance de façon générale. Comme elle prend des formes variées, il convient de compléter cette définition par une typologie. Celle-ci comprend la maintenance préventive et la maintenance corrective. Le critère de distinction entre ces deux types de maintenance est le moment de l'intervention par rapport à la panne. Comme leur nom l'indique, le premier type est appliqué avant la panne et le second après. Leurs définitions sont présentées ci-dessous.

2.2.1 Typologie de la maintenance

Depuis son essor, la maintenance préventive n'a cessé d'évoluer pour répondre aux exigences des entreprises. Le besoin d'accroître la disponibilité et de maîtriser le coût global des équipements, conjointement à l'évolution des technologies, conduit à distinguer quatre types de maintenance préventive : la maintenance préventive systématique, préventive conditionnelle, prévisionnelle et proactive.

D'après la norme AFNOR NF X 60-010, la **maintenance préventive systématique** « 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 ». Ces actions sont effectuées suivant une période qui repose sur l'âge de l'équipement ou à des dates fixes. La première solution de mesure de la période est plus efficace et moins coûteuse que la seconde, mais elle est plus difficile à gérer ([Cha87]). La maintenance préventive systématique se traduit donc par deux types d'actions :

- des interventions planifiées ([Mon87]) qui consistent à nettoyer, réparer ou remplacer certains matériels tels que des composants ou sous-ensembles d'équipements ;
- des inspections périodiques ([Mon87]), dont les buts sont de contrôler ces mêmes composants et sous-ensembles, qui suivent des dispositions réglementaires ou non ([Zwi96]), d'effectuer des révisions, mineures ou majeures, d'équipements, voire d'ateliers entiers lors d'arrêts généraux.

D'après la norme AFNOR NF X 60-010, la **maintenance préventive conditionnelle** « comprend toutes les tâches de restauration de matériels ou de composants non défaillants, entreprises en application d'une évaluation d'état et de la comparaison avec un critère d'acceptation préétabli (défaillance potentielle) ». La maintenance préventive conditionnelle est à appliquer sur les équipements critiques qui impliquent des coûts d'arrêt de production prohibitifs, et pour les équipements dangereux pouvant mettre en cause la sécurité des personnes et des biens ([Bou88]).

Afin d'introduire de la prévision dans la maintenance préventive conditionnelle, celle-ci a évolué vers la **maintenance prévisionnelle**. D'après la norme AFNOR NF X 60-010, celle-ci est une « maintenance préventive subordonnée à l'analyse de l'évolution surveillée de paramètres significatifs de la dégradation du bien, permettant de retarder et de planifier les interventions ». Cette maintenance répond au besoin de planification des interventions, pour autant que la prévision de l'évolution d'une dégradation soit réalisable.

La **maintenance proactive** est une « forme avancée de maintenance prévisionnelle consistant à déterminer les causes initiales de défaillances à partir de l'état de défaillance potentielle » ([Zwi96]).

La **maintenance corrective** regroupe l'« ensemble des activités réalisées après la défaillance du bien ou la dégradation de sa fonction pour lui permettre d'accomplir une fonction requise, au moins provisoirement : ces activités comportent notamment la localisation de la défaillance et son diagnostic, la remise en état avec ou sans modification, le contrôle du bon fonctionnement » (NF X 60-010). Cette définition conduit à distinguer deux types de maintenance corrective :

- la **maintenance palliative**, si l'intervention a un caractère provisoire ;
- la **maintenance curative**, si elle a un caractère définitif.

2.2.2 Concepts de maintenance

Les concepts modernes de la maintenance sont imprégnés des notions « FMDS » (*F*iabilité, *M*aintenabilité, *D*isponibilité, *S*écurité) dont le but est l'optimisation d'une politique de maintenance. Ces notions prennent en particulier toute leur importance dans la mise en pratique « scientifique » de la maintenance préventive [Lyo92]. Parmi ces concepts, il existe trois grandes orientations en matière de politique de maintenance :

- une orientation « coût » dans son principe, souvent dénommée « LCC » (*L*ife *C*ycle *C*ost), prônant l'optimisation du coût global de possession qui comprend

tous les postes de dépenses nécessaires pendant le cycle de vie d'un produit. Développée initialement au Royaume-Uni dans les années 1970, cette stratégie cherche à considérer l'impératif de maintenance sur l'ensemble du cycle de vie de l'équipement. La défaillance d'un équipement ou une opération de maintenance préventive induisant l'arrêt d'un outil de production, engendrent des dépensent directes liées à la maintenance de l'équipement et indirectes liées à cet arrêt [FG92]. L'utilisation du concept LCC dans une politique décisionnelle, pour choisir une politique de maintenance sous la forme d'un budget maximum admissible de maintenance et l'âge de remplacement de l'équipement compte tenu de sa valeur de revente éventuelle, se heurte aux difficultés habituelles de valorisation des coûts cumulés de maintenance sur la durée de vie probable du matériel. Cette approche strictement économique et locale, c'est à dire consacrée aux équipements particuliers, est évidemment difficilement décomposable en objectifs techniques de maintenance ;

– une orientation centrée sur la « fiabilité » et la « maintenabilité » des matériels, désignée par *M*aintenance *B*asée sur la *F*iabilité (*MBF*), est également connue sous l'appellation militaire *RCM*(*R*eliability *C*entred *M*aintenance), ou en France sous le nom d'*OMF* (*O*ptimisation de la *M*aintenance par la *F*iabilité). Elle constitue principalement une méthode de maintenance préventive applicable au matériel jugé critique par les différents acteurs de l'entreprise. Cette méthode est largement exploitée par plusieurs constructeurs automobiles tels que Peugeot et Renault. La MBF est fondée sur la compréhension des fonctions de chaque pièce d'un équipement et de l'impact d'une défaillance sur ces fonctions [Zwi96]. Elle implique un changement radical de la culture d'entreprise en matière de politique de maintenance. Cette approche, strictement technique dans ses fondements, même si l'aspect économique n'est jamais totalement absent, est née à la fin des années 1960 du besoin des entreprises de transport aérien de développer des programmes d'entretiens préventifs plus rationnels et techniquement fondés ;

– une orientation « productivité » généralement désignée par *TPM* (*T*otal *P*roductive *M*aintenance) fondée sur le respect des facultés humaines et la volonté participative de l'ensemble du personnel pour rentabiliser les installations [Nak89]. La TPM, née au Japan à la fin des années 1970,est fondamentalement un élargissement du rôle de la maintenance à la totalité du personnel. Principalement utilisée pour la fabrication de biens d'équipements à vie courte sur un marché très concurrentiel, la TPM est destinée à d'adapter très rapidement à l'évolution des besoins, des marchés, des progrès technologiques et de l'évolution des comportements sociologiques. La TPM repose sur la participation de tous les acteurs en suivant une démarche

organisée, structurée et professionnelle. Elle vise à retenir une politique de maintenance plus proche de l'équipement et réalisée par le personnel de production. Contrairement à la maintenance productive à « l'américaine » où les opérations de maintenance sont confiées à des services spécialisés, la TPM est caractérisée par l'auto-maintenance des outils de production par leurs opérateurs qui deviennent de ce fait des opérateurs régleurs [Zwi96].

2.2.3 Structure de la fonction maintenance

La fonction maintenance a fortement évoluée depuis une décennie sous l'effet des contraintes de productivité, d'optimisation des coûts et sous l'influence des modèles industriels japonais. Si le terme fonction est employé à la place de service, c'est parce que la maintenance n'est plus réservée à l'activité d'un groupe d'hommes sur lesquels on se déchargerait de tout ce qui n'est pas production, finance ou commercial. À l'inverse, maintenir n'est plus de la seule responsabilité d'un service maintenance.

Gérer la structure matérielle de la fonction maintenance revient à gérer les équipements à maintenir et les ressources propres de maintenance [Cou00].

Les ressources propres de maintenance sont de deux types : les ressources matérielles caractérisant les moyens physiques du service maintenance et les historiques. La gestion des ressources matérielles concerne les personnels, les outillages, le magasin de pièces de rechange et de consommables ainsi que les documents. La gestion de ces ressources est définie par la stratégie choisie. Les historiques sont obtenus par collecte des informations en temps réel et retracent le passé des équipements. Ces données nécessitent un traitement de manière à fournir des indicateurs pertinents qui permettent de vérifier si la stratégie choisie est respectée et si elle ne l'est pas, de guider les choix pour sa modification.[Cou00]

2.2.4 La GMAO

Selon Gabriel et Pimor [GP85], « un système informatique de management de la maintenance (GMAO) est un progiciel organisé autour d'une base de données permettant de programmer et de suivre sous les trois aspects techniques, budgétaire et organisationnel, toutes les activités d'un service de maintenance et les objets de cette activité (services, lignes d'atelier, machines, équipements, sous-ensembles, pièces, etc) à partir de terminaux disséminés dans les bureaux techniques, les ateliers, les magasins et les bureaux d'approvisionnement. »

La GMAO doit alléger la charge administrative, et améliore la qualité et la rapidité de l'information. Le système va agir sur l'efficacité de la maintenance : diminution des temps d'attente et des temps perdus, augmentation du taux de disponibilité de l'outil de production, diminution des pièces de rechange en stock. La bonne pièce de rechange est tout de suite identifiée par la nomenclature, le fichier des stocks consulté renseigne sur la disponibilité et permet de faire des réservations. Les interventions sont mieux préparées, plus précises avec la mise à disposition de la bonne information reportée sur le bon de travail, les consignes de sécurité et l'accès rapide aux documents. Les outils de diagnostic et la capitalisation de l'expérience inhérente à la GMAO amènent une réduction des durées des arrêts par suite de pannes, ce qui corrobore l'augmentation de la disponibilité des équipements par suite d'un meilleur préventif.

2.3 L'ordonnancement de la maintenance

L'ordonnancement représente la fonction « chef d'orchestre ». Dans un service maintenance caractérisé par l'extrême variété des tâches en nature, en durée, en urgence et en criticité, l'absence du chef d'orchestre débouche vite sur la cacophonie quel que soit le brio des solistes. L'ordonnancement se situe entre la *fonction méthode*, chargée de la définition des tâches à effectuer et des moyens à mettre en oeuvre, et la *fonction réalisation* chargée de leur exécution [Mon03].

La notion d'ordre de travail de maintenance peut être rapprochée de celle d'opération de production et la notion de ressource de maintenance peut être rapprochée de celle de ressource de production. Le problème d'ordonnancement de la maintenance revient donc à un problème d'affectation de ressources de maintenance à la réalisation d'ordre de travail de maintenance sur des machines du système de production, les ressources de maintenance étant constituées des opérateurs de maintenance [Cou00] [PL94].

Ayant la responsabilité de la conduite et de la synchronisation des actions de maintenance internes ou externalisées, la fonction ordonnancement a pour mission de prévoir la chronologie du déroulement des différentes tâches, optimiser les moyens nécessaires en fonction des délais et des chemins critiques, etc.

Cette fonction stratégique est peu visible (effectif dédié faible) et souvent peu étudiée, mais elle repose sur des méthodes à connaître. Son absence ou son insuffisance est par contre fort visible : tâches préventives négligées, gaspillage de temps en recherche de moyens indispensables, améliorations toujours reportées à plus tard, techniciens parfois inoccupés associés à des heures supplémentaires évitables, etc.

L'ordonnancement de la maintenance peut être réalisé à long terme (cas de gros travaux de maintenance), à moyen terme (cas de révisions générales) ou à court terme (cas d'interventions correctives).

Parmi les méthodes utilisées pour l'ordonnancement de la maintenance à long terme, nous citons la méthode P.E.R.T. [BPS86] ou encore la méthode potentiel-tâche. Il s'agit de méthodes de gestion de projet qui conviennent bien à ce type de tâches non répétitives.

Concernant la planification à moyen terme, il s'agit de partir d'un plan de production et d'un plan de maintenance préventive pour fournir un planning des opérations de maintenance (pour une ou plusieurs équipes de maintenance) respectant à la fois les contraintes de production et les impératifs de remise en état des installations.

A plus court terme, l'intégration des tâches de maintenance dans un plan de production pose également des problèmes. Leur réalisation est souvent retardée ou compromise, même si une planification à moyen terme a été établie au préalable. En effet, face aux objectifs et impératifs à court terme associés à la production, le souci d'intervenir sur les équipements suffisamment tôt pour éviter les défaillances et les conséquences coûteuses qu'elles engendrent paraît souvent dérisoire. Cette situation est source de conflit entre les services de maintenance et de production et nuit à la mise en oeuvre d'une politique de maintenance efficace. Celle-ci ne réussit à s'imposer que lorsque les relations humaines entre responsables sont d'une qualité suffisante, et cela se fait généralement au détriment d'une baisse de productivité péniblement tolérée.

2.3.1 Spécificité de l'ordonnancement de la maintenance

Il est évident que l'ordonnancement des tâches de maintenance possède un caractère spécifique par rapport à l'ordonnancement des travaux de production. En effet, le problème majeur à prendre en compte est sa dépendance à la production. Une bonne synchronisation est indispensable entre les services production et maintenance afin de profiter des arrêts de production (formatage, chômage technique, vacances, etc.) pour réaliser des opérations de maintenance.

La spécificité de l'ordonnancement de la maintenance vient de [PL94]:

- la variété et l'incertitude des travaux de maintenance : la durée d'une intervention ne peut pas être connue de manière sûre tant que l'on n'a pas examiné l'équipement ;

– la date de réalisation d'une tâche de maintenance préventive peut être différée mais également être avancée par rapport à la date optimale ou à l'intervalle optimal de visite [Git94].

2.4 Ordonnancement conjoint de la production et de la maintenance

2.4.1 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 produit 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 la pratique, la mise en place de liens entre les fonctions production et maintenance a toujours été une source de conflits dans les entreprises [WC99]. Pendant de nombreuses années, la maintenance a bien souvent été considérée comme un « mal nécessaire » par la production et n'a pas réellement été intégrée dans ses prises de décisions du niveau opérationnel. Grâce à un niveau d'encours élevé, l'occurrence de défaillances n'induisant pas de coûts de retards trop importants, la maintenance n'intervenait que pour les travaux correctifs. En conséquence, la coordination des décisions prises par la maintenance avec celles prises par la production n'a reçu qu'une attention limitée. Une situation de conflit s'est instaurée entre elles lorsqu'on a voulu implanter de nouvelles méthodes de gestion de production telles que le juste-à-temps. En effet, des liens beaucoup plus étroits devraient être mis en place et la maintenance prenait une place beaucoup plus importante dans le système. La coopération de la maintenance avec la production doit permettre d'assurer que l'ordonnancement de la production laisse du temps à la maintenance pour intervenir, que le service de maintenance intervienne aux bons moments pour l'entretient préventif et réagisse rapidement aux défaillances des machines.

L'ordonnancement conjoint de la production et de la maintenance n'a attiré que récemment l'attention des chercheurs. Un état de l'art sur ces travaux sera présenté dans la section suivante.

2.4.2 État de l'art

La majeure partie de la littérature dédiée aux problèmes d'ordonnancement se place dans le contexte où les ressources nécessaires à l'exécution des tâches sont disponibles en permanence. Cette hypothèse n'est pourtant pas fidèle à la réalité des entreprises. En effet, les différentes ressources qu'elles soient humaines ou matérielles peuvent, pour diverses raisons, être indisponibles. Les indisponibilités des ressources peuvent être dues à une opération de maintenance sur les machines de l'atelier ou à des emplois du temps du personnel. Nous supposons dans cette étude qu'une indisponibilité est due à une opération de maintenance préventive systématique. Dans certains modèles, un travail interrompu par une période d'indisponibilité sur la machine peut finir son exécution dès que la machine est redevenue disponible, dans ce cas on dira que les travaux sont *sécables*. Par contre dans d'autres problèmes, un travail ne peut commencer que s'il peut finir sans être gêné par une période d'indisponibilité dans ce cas on dira que les travaux sont *non-sécables*. Dans certains cas, un travail interrompu par une période d'indisponibilité reprendra son exécution partiellement dès que la machine redevient disponible, dans ce cas on dira que les travaux sont *semi-sécables*.

Afin de mesurer la performance des heuristiques, deux notions sont principalement utilisées : l'erreur relative, la garantie de performance et la borne au pire cas. Nous donnons dans ce qui suit leurs définitions.

Définition 2.1. *Soit $C_H(i)$ la valeur de la fonction objectif obtenue par l'heuristique H pour une instance i du problème de minimisation P, et soit $C_{opt}(i)$ la valeur optimale. L'erreur relative de H est donnée par $\varepsilon = \frac{C_H(i) - C_{opt}(i)}{C_{opt}(i)}$.*

Définition 2.2. *Soit $C_H(i)$ la valeur de la fonction objectif obtenue par l'heuristique H pour une instance i du problème de minimisation P, et soit $C_{opt}(i)$ la valeur optimale. On dit que l'heuristique H a une performance garantie λ ($\lambda \geq 1$), si pour toute instance i du problème P, nous avons $\frac{C_H(i)}{C_{opt}(i)} \leq \lambda$.*

Définition 2.3. *Soit $C_H(i)$ la valeur de la fonction objectif obtenue par l'heuristique H pour une instance i dans le cas le plus défavorable du problème de minimisation P, et soit $C_{opt}(i)$ la valeur optimale. La borne au pire cas de H est donnée par $B_p = \frac{C_H(i)}{C_{opt}(i)}$.*

Nous présentons dans ce qui suit les principaux résultats de complexité concernant l'ordonnancement conjoint de la production avec prise en compte des périodes d'indisponibilité. Nous présentons par la suite un état de l'art sur le contexte déterministe où les périodes d'indisponibilité sont connues à l'avance ainsi que sur le contexte stochastique

où les périodes d'indisponibilité arrivent d'une manière aléatoire.

La disponibilité des machines a d'abord et surtout été prise en compte dans le cadre de problèmes à une machine et à machines parallèles [SS97]. Par la suite, plusieurs études ont été consacrées aux ateliers de type Flow Shop et open shop, essentiellement à deux machines. Quelques un de ces problèmes peuvent être résolus d'une manière optimale en adaptant les algorithmes classiques proposés dans le cas où les machines sont disponibles d'une manière continue. La plupart de ces problèmes sont $\mathcal{NP}$-*difficiles*.

2.4.2.1 Cas déterministe

Dans ce cas, on suppose qu'on connaît à l'avance toutes les données concernant les périodes d'indisponibilité sur les machines, à savoir leurs dates d'arrivée ainsi que leurs durées.

Problèmes à une machine Les problèmes à une machine ont un caractère fondamental. Ils peuvent former la base de problèmes plus complexes. Dans ce qui suit nous donnons un état de l'art sur les travaux faits sur ce type de problèmes.

a. Cas sécable

Lee [Lee96] a étudié le problème à une machine avec différentes mesures de performances. Il a montré que le problème de minimisation de C_{max} avec des tâches sécables est résolu avec une séquence arbitraire. Avec les mêmes hypothèses, l'auteur a résolu le problème de minimisation de $\sum C_i$ avec l'algorithme SPT (Shortest Processing Time). Avec SPT, les tâches sont placées suivant l'ordre croissant des durées opératoires. Il a aussi montré que le critère L_{max} est minimisé avec l'algorithme EDD où les tâches sont placées suivant l'ordre croissant des dates de fin au plus tard. Enfin, l'auteur a prouvé que la minimisation de $\sum U_i$ peut être résolue avec l'algorithme de Moore-Hodgson's.

Le problème de minimisation de $\sum w_i C_i$ avec prise en compte des indisponibilités de la machine a été démontré NP-difficile, même si $w_i = p_i$, par Lee [Lee96]. Dans le cas où on a une seule période d'indisponibilité sur l'horizon de planification, l'auteur a proposé un algorithme de programmation dynamique.

Lorigeon et *al.* [LBB02] se sont intéressés au problème à une machine, avec une seule période d'indisponibilité et en considérant des dates de début au plus tôt et des durées de latence sur les tâches de production. Les auteurs ont proposé une borne inférieure, deux bornes supérieures ainsi qu'un algorithme Branch and Bound.

b. cas non-sécable

Le problème de minimisation de C_{max} avec des tâches non-sécables a été prouvé $\mathcal{NP}$-*difficile*, par Lee [Lee96], avec une ou plusieurs périodes d'indisponibilité. L'auteur a aussi montré que l'algorithme LPT (Longest Processing Time) a une erreur relative égale à 1/3. Avec LPT, les tâches sont placées suivant l'ordre décroissant des durées opératoires.

Adiri et *al.* [ABFK89], Lee et Liman [LL92] ainsi que Qi et *al.* [QCT99] se sont intéressés à la minimisation de $\sum C_i$ et ils ont prouvé que ce problème est $\mathcal{NP}$-*difficile*. Lee et Liman [LL92] ont montré que l'algorithme SPT peut résoudre ce problème avec une faible erreur relative égale à 2/7. Par ailleurs, Qi et *al.* [QCT99] ont proposé pour la résolution trois heuristiques et un algorithme Branch and Bound.

Sadfi [Sad02] a étudié le même problème que Lee et Liman [LL92]. Pour la résolution, il a développé un algorithme de programmation dynamique et une heuristique ayant une garantie de performance égale à 19/17.

Lee [Lee96] a montré que la minimisation du plus grand retard, du nombre de travaux en retard et de la somme des dates de fin pondérées des tâches sont $\mathcal{NP}$-*difficiles* au sens faible. En outre, l'algorithme EDD (Earliest Due Date) permet de résoudre le premier problème avec une erreur relative égale à p_{max} (plus grande durée opératoire). La minimisation du nombre de travaux en retard est résolue par l'algorithme de Moore-Hodgson avec une erreur relative égale à 1. Pour le troisième problème, l'auteur a prouvé que le ratio de performance de l'algorithme SWPT pouvait être arbitrairement grand, même si tous les coefficients de pondération étaient égaux aux durées opératoires.

Liao et al. [LC02], ont étudié également le cas d'une seule machine qui subit des périodes d'indisponibilité dues à des interventions de maintenance périodiques. Leur critère d'optimisation étant 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 tailles réduites. Les instances de grandes tailles, ont été résolues à l'aide d'une heuristique basée sur la règle de priorité EDD.

c. cas semi-sécable

Dans le cas où les tâches sont semi-sécables, Graves et *al.* [GL99] ont considéré des périodes d'indisponibilité dues à des interventions de maintenance préventive et se sont intéressés à l'étude de deux critères d'optimisation : la somme des dates de fin pondérées et le retard max. Les auteurs ont donné 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 $\mathcal{NP}$-*difficile*. 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 $\mathcal{NP}$-*difficile*. Néanmoins, les règles SPT (resp. EDD) permettent de le résoudre d'une manière exacte dans le cas de la minimisation de la somme des dates de fin (resp. la minimisation du maximum d'avance).

En résumé, la majorité des travaux étudiés dans la littérature ne s'intéressent qu'à des critères de production même pour les cas où les périodes d'indisponibilité sont dues à des interventions de maintenance préventive. Le tableau 2.1 résume les différents travaux cités dans le paragraphe 2.4.2.1.

Problème	Hypothèse(s)	Référence	Méthode(s) de résolution
$1\|sec\|C_{max}$	Nb_{indisp} quelconque	[Lee96]	séquence arbitraire
$1\|sec\|\sum C_i$	Nb_{indisp} quelconque	[Lee96]	l'alg. SPT
$1\|sec\|\sum w_iC_i$	$Nb_{indisp} = 1$	[Lee96]	alg. de progr. dynamique
$1\|sec\|\sum U_i$	Nb_{indisp} quelconque	[Lee96]	l'alg. de Moore-Hodgson's
$1\|sec\|L_{max}$	Nb_{indisp} quelconque	[Lee96]	l'alg. EDD
$1\|sec,r_i,q_i\|C_{max}$	$Nb_{indisp} = 1$	[LBB02]	alg. Branch & Bound
1 |non-sec|$\sum C_i$	Nb_{indisp} quelconque	[LL92]	l'alg. SPT avec $\varepsilon = 2/7$
		[QCT99]	3 heuristiques + alg. Branch & Bound
		[Sad02]	alg. prog. dynamique + heurist. avec $\lambda = 19/17$
1 |non-sec|L_{max}	Nb_{indisp} quelconque	[Lee96]	l'alg. EDD avec $\varepsilon = p_{max}$
		[LC02]	alg. Branch & Bound + heurist. basée sur EDD
1 |non-sec|$\sum U_i$	Nb_{indisp} quelconque	[Lee96]	l'alg. de Moore-Hodgson's avec $\varepsilon = 1$
1 |sem-sec|$\sum w_iC_i$	Nb_{indisp} quelconque	[GL99]	alg. de progr. dynamique
1|sem-sec|L_{max}			

TAB. 2.1 – *Résumé des différents problèmes à une machine étudiés dans la littérature*

Problèmes à machines parallèles Plusieurs auteurs se sont intéressés à l'étude des problèmes à machines parallèles. Ces machines peuvent être identiques ou pas. Dans ce

qui suit, nous présentons un état de l'art non exhaustif sur ces travaux.

a. cas sécable

Lee [Lee91] a étudié le problème de minimisation de C_{max} sur m machines parallèles où les périodes d'indisponibilité sont effectuées au début de l'horizon de planification et il y en a au plus une période d'indisponibilité. L'auteur a montré que le problème peut être résolu à l'aide de l'algorithme LPT (Longest Processing Time) avec une erreur relative égale à 1/2 ou bien à l'aide de l'algorithme LPT modifié avec une erreur relative égale à 1/3.

Le même problème a été étudié par Lee [Lee96] et il a proposé deux variantes de l'algorithme LPT. La première variante, appelée LPT1, consiste à affecter les tâches à la machine la moins chargée. La deuxième, appelée LPT2, affecte une tâche à la machine telle que sa date de fin soit minimisée. Toujours avec les mêmes hypothèses, Lin et *al.* [LYH98] ont étudié le problème de maximisation du minimum des dates de fin. Les auteurs ont montré que l'algorithme LPT a une borne au pire cas égale à $(2m-1)/(3m-2)$.

Kaspi et Montreuil [KM88] d'une part et Liman [Lim91] d'autre part ont étudié le problème de minimisation de C_{max} sur m machines parallèles identiques où chaque machine subit une seule période d'indisponibilité et les machines sont disponibles à des dates distinctes. Ils ont montré que pour ce problème, l'ordonnancement des tâches de production selon la règle SPT est optimal.

Schmidt [Shm84] a étudié le problème de minimisation de C_{max} sur m machines parallèles et a donné les conditions d'existence d'un ordonnancement préemptif dans le cas où les machines sont disponibles durant un nombre arbitraire de périodes. Un tel ordonnancement peut être construit avec un algorithme de complexité $O(n+m\ log m)$. Ce problème a été par la suite généralisé par Schmidt [Shm88] en considérant des différentes dates de début au plutôt et de fin au plus tard. L'auteur a proposé un algorithme de complexité $O(nmlogm)$. Lorsque les dates de début au plus tôt ne sont pas considérées, Schmidt [Shm88] a proposé un algorithme de complexité $O(nmlogn)$ pour la minimisation de l'avance maximum.

Liu et Sanlaville [LS95] ont étudié différents schémas de disponibilité sur des machines parallèles avec des contraintes de précédence sur les tâches. Ils ont prouvé que la minimisation de C_{max} pour des schémas arbitraires d'indisponibilité et des contraintes de précédence de type chaînes peut être polynômialement résolue avec la règle LRP (Longest Remaining Path). Dans le cas de contraintes de précédence arbitraires, le résultat

est vrai seulement pour deux machines. Le même problème a attiré l'attention de Blażewicz et *al.* [BDF+00]. Ils ont montré que pour des schémas d'indisponibilité en escalier et des contraintes de précédence de type chaînes le problème de minimisation de C_{max} peut être résolu en un temps polynomial. Pour les critères C_{max} et L_{max}, ils ont développé un algorithme basé sur les réseaux de neurones dans le cas de machines uniformes. La programmation dynamique a été utilisée pour résoudre le cas où les machines sont indépendantes. Dans les deux cas les schémas d'indisponibilité sont arbitraires.

b. cas non-sécable

Lee [Lee96] a démontré que le problème de minimisation de C_{max} sur m machines parallèles est $\mathcal{NP}$-*difficile*. Il a prouvé que l'algorithme d'ordonnancement de liste (LS) a une erreur relative égale à m et que la règle LPT a une faible erreur relative égale à $(m+1)/2$. Mosheiov [Mos94] a étudié le même problème avec $\sum C_i$ comme critère d'optimisation et il a montré que la règle SPT devient asymptotiquement optimale quand le nombre de tâches tend vers l'infini.

Dans le cas où chaque machine ne doit subir qu'une seule intervention de maintenance, Chung [CC00] s'est intéressé à la minimisation de $\sum w_i C_i$. Deux cas de figures ont été étudiés. Dans le premier, plusieurs machines peuvent être entretenues en même temps si nécessaire. Le deuxième considère des ressources de maintenance limitées et qu'une seule machine peut être entretenue en même temps. Pour la résolution de ces deux cas, l'auteur a proposé un algorithme Branch and Bound basé sur la méthode de génération de colonnes.

Le problème à deux machines parallèles minimisant le critère $\sum w_i C_i$ a été prouvé $\mathcal{NP}$-*difficile* par Lee [Lee96]. Un algorithme de programmation dynamique a été proposé pour résoudre optimalement le cas où $w_i = 1\ \forall i$ et la première machine ne subit pas d'interventions de maintenance. Le même problème a attiré l'attention de Lee et Liman [LL93] auquel ils ont ajouté l'hypothèse que la deuxième machine n'est disponible qu'à partir de la date zéro jusqu'à une date fixée à priori. Pour la résolution ils ont proposé un algorithme de programmation dynamique et une heuristique basée sur la règle SPT. Cette heuristique admet une borne au pire cas égale à 1/2.

La majorité des problèmes traitant des machines parallèles sont démontré $\mathcal{NP}$-*difficile*. Ce qui a incité la majorité des chercheurs a proposer des algorithmes exactes pour les problèmes relaxés et des heuristiques pour résoudre des problèmes généralisés. Le tableau 2.2 résume les travaux cités dans le paragraphe 2.4.2.1.

Problème	Hypothèse(s)	Référence	Méthode(s) de résolution
$Pm\|sec\|C_{max}$	$Nb_{indisp} \leq 1$	[Lee91]	l'alg. LPT avec $\varepsilon = 1/2$ ou l'alg. LPT modifié avec $\varepsilon = 1/3$
		[Lee96]	deux variantes de LPT
	Nb_{indisp} quelconque	[Shm84]	alg. de complexité $O(n + m \log m)$
$Pm\|sec\|minC_i$	$Nb_{indisp} \leq 1$	[LYH98]	l'alg. LPT avec $B_p = (2m-1)/(3m-2)$
$Pm\|sec\|C_{max}$	$Nb_{indisp} \leq 1$	[KM88] [Lim91]	l'alg. SPT est optimal
$Pm\|sec,d_i\|E_{max}$	Nb_{indisp} quelconque	[Shm88]	alg. de complexité $O(nm \log n)$
$Pm\|sec,prec\|C_{max}$	Nb_{indisp} quelconque	[LS95]	alg. LRP
Pm |non-sec$\|C_{max}$	Nb_{indisp} quelconque	[Lee96]	alg. de liste avec $\varepsilon = m$ LPT avec $\varepsilon = (m+1)/2$
Pm |non-sec$\|\sum C_i$	Nb_{indisp} quelconque	[Mos94]	alg. SPT est asymptotiquement optimal quand $n \to \infty$
Pm |non-sec$\|\sum w_i C_i$	$Nb_{indisp} = 1$	[CC00]	Branch & bound basé sur méth. de génération de colonnes
$P2$ |non-sec$\|\sum w_i C_i$ $w_i = 1 \forall i$	Nb_{indisp} quelconque et M_1 ne subit pas de périodes d'indisp.	[Lee96]	alg. de prog. dynamique
	M_2 disponible de 0 à t (fixé)	[LL93]	alg. prog. dynamique + heurist. basée sur SPT ($\varepsilon = 1/2$)

TAB. 2.2 – *Résumé des différents problèmes à machines parallèles étudiés dans la littérature*

Problèmes d'ateliers flow shop Les études portant sur les problèmes de type flow shop avec contraintes de disponibilité des machines sont relativement récentes. Par ailleurs, la majorité des travaux dédiés à ce type d'ateliers sont limités à deux machines. Néanmoins, quelques résultats ont été publiés très récemment dans le cas de plusieurs machines.

a. cas sécable

Lee [Lee97] a étudié le problème de minimisation de C_{max} dans un Flow Shop à deux machines subissant chacune une seule période d'indisponibilité. L'auteur a montré que le problème est $\mathcal{NP}$-*difficile* au sens faible quelle que soit la machine concernée par l'indisponibilité et a proposé des algorithmes pseudo-polynômiaux de programmation dynamique. Il a également développé une heuristique avec une performance garantie de 3/2 (resp. 4/3) dans le cas où seule la première machine subit une période d'indisponibilité (resp. la deuxième), l'algorithme de Johnson admet une erreur relative égale à 1 (resp. 3/2).

Lee [Lee99] a étudié le problème où les deux machines peuvent subir des périodes d'indisponibilité. Dans le cas où les indisponibilités se produisent à des dates identiques, l'auteur a prouvé que l'algorithme de Johnson résout le problème d'une manière optimale. Par ailleurs, si les deux machines sont indisponibles à des dates différentes et même si les durées des indisponibilités sont égales, le problèmes est prouvé $\mathcal{NP}$-*difficile* au sens faible.

Cheng et Wang [CW00] ont traité le problème de la minimisation de C_{max} en supposant que la première machine subit une seule période d'indisponibilité. Les auteurs ont montré que la borne au pire cas égale à 1/2 trouvée par Lee [Lee97] était serrée, puis ont développé une heuristique ayant une performance garantie égale à 4/3.

Błażewicz et *al.* [BBF$^+$01] ont étudiés le problème de flow shop à deux machines minimisant C_{max} et où les deux machines peuvent subir des périodes d'indisponibilité. Pour la résolution, ils proposent deux heuristiques constructives ainsi qu'un recuit simulé. La première consiste à ordonnancer les jobs entre deux périodes d'indisponibilité consécutives selon la règle de Johnson, tandis que la seconde est basée sur une optimisation locale. Le même problème a été étudié par Kubiak et *al.* [KBF$^+$02] et résolu avec un algorithme Branch and Bound basé sur des propriétés d'optimalité des ordonnancements. Dans le cas où les périodes d'indisponibilité ne concernent que la première machine, une heuristique à performance garantie égale à 2 de complexité O(n log n) est proposée.

Allaoui et *al.* [AARE03] se sont intéressés à la minimisation de C_{max} dans un flow shop à deux machines dont la première subit une seule période d'indisponibilité. Ils ont considérés deux scénarios pour les travaux : sécables et non-sécables. Pour les deux scénarios, ils ont proposé un algorithme de programmation dynamique. De plus ils se sont focalisés sur l'étude de la performance de l'algorithme de Johnson (JO) comme une heuristique. Ils ont établi la condition d'optimalité de JO et ont démontré que dans les autres cas sa performance est majorée par 2.

b. cas semi-sécable

Lee [Lee99] a montré que la minimisation de C_{max} est $\mathcal{NP}$-*difficile* au sens faible si une période d'indisponibilité était considérée sur l'une ou l'autre des machines. Dans le cas où seule la première machine est non disponible, l'auteur a développé un algorithme de programmation dynamique, et a prouvé que l'algorithme de Johnson a une erreur relative égale à 1. Si seule la deuxième machine subit des périodes d'indisponibilité, l'algorithme de Johnson a une erreur relative égale à $max\{1/2,\mu\}$, où μ est la portion à ré-exécuter de la tâche semi-sécable interrompue par la période d'indisponibilité.

Lee [Lee99] a aussi montré que si les deux machines peuvent subir des périodes d'indisponibilité, le problème est $\mathcal{NP}$-*difficile* au sens faible, même si les dates de début et de fin des périodes d'indisponibilité sont identiques pour les deux machines. Dans ce cas, l'algorithme de Johnson a une erreur relative égale à μ.

c. cas non sécable

Dans le cas non-sécable, Lee [Lee99] a développé une heuristique ayant une erreur relative égale à 1, lorsque seule la première machine subit des périodes d'indisponibilité. Si la période d'indisponibilité est imposée sur la deuxième machine, l'algorithme de Johnson a une erreur relative égale à 1.

Wang et Xie [WX02] ont étudié le problème de minimisation de C_{max} dans un atelier de flow shop hybride à deux étages. Les auteurs ont montré que ce problème est $\mathcal{NP}$-*difficile* au sens fort même s'il y a qu'une seule machine par étage et que cette machine ne subit qu'une seule période d'indisponibilité. Pour la résolution, ils ont proposé deux algorithmes avec une faible borne au pire cas dans le cas où il n'y a qu'une seule machine dans le premier étage et que cette machine ne subit qu'une seule intervention de maintenance. Ce problème a été généralisé par Wang et Xie [WX03] en considérant un nombre quelconque d'étages. Les auteurs ont proposé deux bornes inférieures l'une liée aux machines et l'autre aux tâches et ont développé un algorithme Branch and Bound qui est prouvé efficace.

Pour le problème de flow shop à m machines ($m > 2$) et pour des travaux non-préemptifs, quelques études ont été menées très récemment. Citons les travaux de Aggoune [Agg01] qui a étudié le problème de minimisation de C_{max} avec deux scénarios pour les périodes d'indisponibilité. Dans le premier, il considère que les périodes d'indisponibilité sont fixes et définies à priori. Tandis que dans le deuxième ces périodes varient dans un intervalle. Pour la résolution, l'auteur propose une hybridation d'un algorithme génétique et une méthode de recherche taboue.

Le problème d'ordonnancement dans un atelier flow shop avec prise en compte des contraintes d'indisponibilité a attiré l'attention de beaucoup de chercheurs. Néanmoins, la majorité des travaux se sont restreints au cas de deux machines vu la difficulté du problème général. Dans ce cas d'étude, un seul aspect est pris en compte, à savoir la production. Le tableau 2.3 les différents travaux cités dans le paragraphe 2.4.2.1.

Problèmes d'ateliers job shop Le cas des ateliers de type job shop avec prise en compte des périodes d'indisponibilité est peu étudié. La complexité de ce problème rend la tâche encore plus difficile si l'on tient compte des contraintes de disponibilité des

Problème	Hypothèse(s)	Référence	Méthode(s) de résolution
$F2\|sec\|C_{max}$	$Nb_{indisp} = 1$ pour $M1$ et $M2$ seule $M1$ subit une période d'indisp. seule $M2$ subit une période d'indisp.	[Lee97]	alg. de prog. dynamique heurist. avec $\lambda = 3/2$ heurist. avec $\lambda = 3/4$
	Nb_{indisp} quelconque et des dates d'indisponibilité identiques	[Lee99]	l'alg. de Johnson est optimal
	$Nb_{indisp} = 1$ pour $M1$	[CW00]	heurist. avec $\lambda = 3/4$
	Nb_{indisp} quelconque	[BBF+01]	2 heurist. constructives et un recuit simulé
		[KBF+02]	alg. Branch & Bound
	seule $M1$ subit des périodes d'indisponibilité	[KBF+02]	heurist. de complexité $O(n \log n)$ avec $\lambda = 2$
$F2$ |sem-sec$\|C_{max}$	seule $M1$ subit une période d'indisp.	[Lee99]	alg. de prog. dynamique
	seule $M2$ subit une période d'indisp.	[Lee99]	alg. de Johnson avec $\varepsilon = \max\{1/2,\mu\}$
$F2$|non-sec$\|C_{max}$	seule $M1$ subit une période d'indisp.	[Lee99]	heurist. avec $\varepsilon = 1$
	seule $M1$ subit une période d'indisp.	[Lee99]	alg. de Johnson avec $\varepsilon = 1$
FHm|non-sec$\|C_{max}$	Nb_{indisp} quelconque	[WX03]	2 bornes inférieures + alg. Branch & Bound
Fm|non-sec$\|C_{max}$	Nb_{indisp} quelconque	[Agg01]	hybridation d'un AG et d'une méthode taboue

TAB. 2.3 – *Résumé des différents problèmes de flow shop étudiés dans la littérature*

machines. Banerjee et *al.* [BB90] 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 subissant chacune plusieurs périodes d'indisponibilité dues à des interventions de maintenance préventive.

Aggoune [Agg02] a étudié le problème à deux jobs, avec la prise en compte de contraintes de disponibilité des machines. Dans son modèle, il considère que les dates et durées des périodes d'indisponibilité sont fixes et connues à priori. Son objectif était de déterminer une séquence d'entrée des opérations des tâches sur les machines minimisant C_{max}. Pour résoudre ce problème, l'auteur a développé une extension de l'approche géométrique qui permet de transformer le problème initial en recherche de plus court chemin. La méthode ainsi développée est exacte et polynômiale.

Harrath [Har03] quant à lui s'est intéressé à l'étude d'un atelier de type job shop à m machines optimisant deux critères : le premier, C_{max}, concerne la production, le deuxième est la somme des coûts d'avance et de retard des interventions de maintenance. Pour la résolution, l'auteur a développé un algorithme génétique multicritère.

Problèmes d'ateliers open shop Le cas des ateliers de type open shop avec prise en compte des indisponibilités des machines est également peu étudié. Nous pouvons citer

les travaux de Lu et Posner [LP93] qui ont proposé un algorithme polynomial pour la minimisation de C_{max} dans un open shop à deux machines, l'une d'entre elles n'étant pas disponible à l'instant zéro. Le même problème a intéressé Breit [Bre00]. L'auteur a prouvé qu'il n'existait pas d'heuristique à performance garantie pour la minimisation de C_{max} dans le cas de tâches sécables, si une machine subit une période d'indisponibilité et l'autre deux.

Breit [BSS01] a étudié le problème de minimisation de C_{max} dans un open shop à deux machines dont l'une subit une seule intervention de maintenance. Ils ont montré que le problème avec des tâches sécable est $\mathcal{NP}$-*difficile* au sens faible. Pour la résolution, les auteurs ont développé une heuristique avec une borne au pire égal à 4/3. Le cas de travaux non-sécables, Breit [BSS02] ont supposé qu'une machine peut subir plusieurs périodes d'indisponibilité et ont montré qu'il n'existe pas d'heuristiques à performance garantie. Dans le cas d'une seule période d'indisponibilité sur chaque machine (resp. sur une seule machine), les auteurs ont développé une heuristique ayant une borne au pire cas égal à 2 (resp. 4/3).

Les problèmes d'ordonnancement avec contraintes d'indisponibilité dans des ateliers job shop et open shop sont assez peu étudiés. Nous présentons dans le tableau 2.4 les quelques références trouvées dans la littérature.

Problème	Hypothèse(s)	Référence	Méthode(s) de résolution
$Jm\|\|C_{max}$	Nb_{indisp} quelconque	[BB90]	modèle de simulation
$Jm\|n=2\|C_{max}$	Nb_{indisp} quelconque	[Agg02]	extension de l'approche géométrique
$Jm\|\|C_{max}$,C_{maint}	Nb_{indisp} quelconque	[Har03]	alg. génétique multicritère
$O2\|sec\|C_{max}$	Nb_{indisp} quelconque	[LP93]	alg. polynômial
	1 machine subit 1 seule période d'indisponibilité	[BSS01]	heurist. avec $B_p = 4/3$
$O2\|non-sec\|C_{max}$	$Nb_{indisp} = 1$ sur $M1$ et $M2$	[BSS02]	heurist. avec $B_p = 2$

TAB. 2.4 – *Résumé des différents problèmes de job shop et open shop étudiés dans la littérature*

De la littérature on a recensé d'autres travaux traitant l'ordonnancement de la production et de la maintenance dans des ateliers comportant plusieurs machines. Ces travaux tiennent en compte les spécificité des tâches de maintenance. Le paragraphe suivant donne quelque une de ces travaux.

Problèmes à plusieurs machines 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

travail 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 induit si l'on retarde le travail, 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 scénario 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, ...).

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 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.

2.4.2.2 Cas stochastique

Les interventions de maintenance préventive permettent de diminuer le risque de pannes sur la machine mais pas de l'annuler. Pour cette raison et pour être réaliste il

faut étudier le cas où la machine subit des pannes aléatoires. Nous présentons par la suite quelques travaux réalisés dans le cas de tâches sécables et non-sécables.

a. cas sécable

Qi et *al.* [QYB02] ont étudié le cas d'une seule machine subissant des pannes aléatoires et traitant des tâches de durées opératoires comprimables (à cause de l'addition d'un nouvel équipement ou une nouvelle technologie). Les auteurs ont étudié deux scénarios ; le premier considère que les dates de fin au plus tard sont fixées à priori et il faut déterminer la séquence optimale des tâches. Tandis que dans le deuxième scénario la séquence et les dates de fin au plus tard sont des variables de décision. Pour les deux problèmes, ils ont montré qu'un ordonnancement optimal est V-shaped.

Albers et Schmidt [AS01] se sont intéressés au problème à m machines identiques. Ils ont montré qu'un algorithme online peut trouver la valeur optimale de C_{max} si on connaît la prochaine date à laquelle l'ensemble des machines disponibles change.

Le cas du flow shop a attiré l'attention de Allahverdi et Mittenthal [AM98] qui ont étudié le cas de deux machines sujettes à des pannes aléatoires optimisant deux critères. Les deux critères sont C_{max} et L_{max}. Les auteurs ont montré que les règles LPT et SPT sont optimales si l'une des deux machines subit des pannes aléatoires. Dans le cas où les deux machines peuvent tomber en panne, ils ont développé un critère d'élimination. Afin de trouver la solution optimale, il ont utilisé des techniques d'énumération implicite. Le cas à plusieurs machines a intéressé Akturk et Gorgulu [AG99]. Pour la résolution, les auteurs ont déterminé une méthode pour réordonnancer les tâches quand une machine tombe en panne ainsi qu'une procédure de détermination du point de début du nouveau ordonnancement utilisant un mécanisme de retour arrière.

b. cas non-sécable

Adiri et *al.* [ABFK89] ont étudié le problème de minimisation de $\sum C_i$ sur une machine sujette à des pannes. Les auteurs ont montré que lorsque la fonction de distribution des temps de bon fonctionnement est concave, la règle SPT minimise asymptotiquement le critère $\sum C_i$. Cette règle minimise le critère considéré lorsque la fonction de distribution des temps de bon fonctionnement suit la loi exponentielle.

Li et Cao [LC95] ont étudié une version plus générale du problème. En effet, la machine est sujette à différents types de pannes et avec différentes probabilités. Les auteurs considèrent deux critères d'optimisation : $E[\sum W_i C_i]$ et $E[\sum W_i U_i]$.

2.5 Conclusion

Nous avons présenté dans la première partie de ce chapitre une description de la fonction maintenance, à savoir ses différentes formes, sa spécificité ainsi que son importance au sein de l'entreprise.

La majorité des travaux traitant l'ordonnancement avec prise en compte des indisponibilités des machines ne considèrent pas que ces indisponibilités sont dues à des interventions de maintenance. Peu de travaux se sont intéressés à la maintenance. Dans la deuxièmes partie de ce chapitre nous avons donné un état de l'art sur les différents travaux réalisés dans ce domaine dans le cas déterministe et stochastique.

Cette étude bibliographique nous a permis de choisir les variables de décision liées à la maintenance. De plus, l'aspect économique nous a semblé intéressant dans la mesure où c'est l'indicateur le plus important pour les entreprises. C'est pourquoi dans cette étude nous considérons un critère d'optimisation tenant en compte les deux aspects production et maintenance. Quand au choix des périodes de maintenance préventive, la littérature propose deux types : des périodes fixes prédéterminées, ou des périodes aléatoires générées lors de l'ordonnancement. Pour notre cas d'étude, nous avons travaillé avec la maintenance préventive déterministe dont les périodes sont données d'avance. Néanmoins, nous rajoutons des intervalles de tolérance autour de chaque période de maintenance, durant lesquels le coût de maintenance est faible. Ces intervalles donneront plus de flexibilité à la planification de la maintenance en cas de besoin.

Chapitre 3

Ordonnancement de la production et de la maintenance sur une machine

Résumé : *Dans ce chapitre, nous étudions le problème d'ordonnancement conjoint de la production en tenant compte des contraintes d'indisponibilité de la machine et qui sont dues à des interventions de maintenance. Vue la complexité du problème, nous nous limitons à l'étude de la maintenance préventive systématique. Nous proposons ensuite des méthodes de résolution dans les deux cas séquentiel et intégré ainsi qu'une borne inférieure. Ces heuristiques ont été testées sur des données générées avec une méthode qui tient compte des deux aspects production et maintenance.*

3.1 Introduction

La nécessité de faire partager les ressources entre les fonctions production et maintenance constitue une source de conflits et traduit l'existence d'un lien fort entre ces fonctions. En effet, trop souvent, les séquences d'opérations de maintenance et de production sont perçues comme antagonistes. Les périodes d'immobilisation des équipements nécessaires aux interventions des agents de maintenance, sont considérées comme perturbant réduisant les périodes d'utilisation des ressources et non comme un facteur favorisant leur bon déroulement. Ce type de conflit entraîne naturellement des querelles qui nuisent à la productivité globale de l'entreprise. La planification intégrée apporte une solution à ce conflit et entraînera sans aucun doute un changement de mentalité en déracinant l'idée préconçue : «*Moi je fabrique, toi tu répares*» et en mobilisant l'ensemble des compétences disponibles.

Nous sommes donc face à un problème multicritère, d'une part ordonnancer la production sous les contraintes de respect des délais, coût et qualité des produits et d'autre part planifier la maintenance sous les contraintes de sûreté de fonctionnement des équipements qui assurent la pérennité de l'outil de production. Ainsi la planification des activités de maintenance sur les machines, à intervalles donnés, peut gêner l'ordonnancement des opérations de production, mais elle est nécessaire pour garantir la disponibilité de l'outil de production. Des études récentes sur l'efficacité de la gestion en maintenance [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 risques 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 et ceci peut être réalisé en créant une certaine coopération entre les deux services production et maintenance. Cette problématique est décrite dans la figure 3.1.

Nous proposons dans ce chapitre une étude de l'ordonnancement conjoint de la production et de la maintenance sur une machine et nous nous intéressons à la maintenance préventive systématique.

Ce chapitre est composé de deux parties : nous nous intéresserons lors de la première partie à l'ordonnancement séquentiel de production et de maintenance et nous proposeront dans la deuxième des méthodes de résolution du problème intégré.

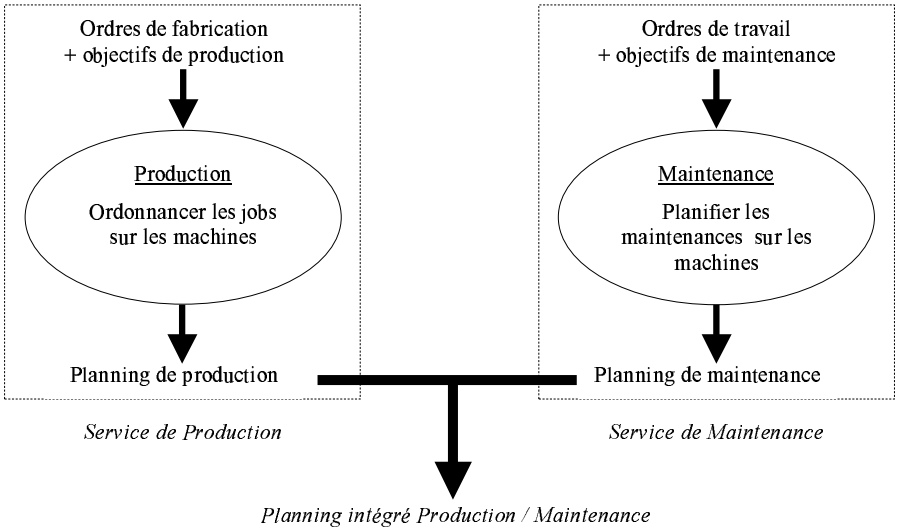


FIG. 3.1 – *La problématique générale*

Nous commençons ce chapitre par la description des différentes politiques d'ordonnancement conjoint de la production et de la maintenance recensées de la littérature. Par la suite, nous décrivons en détail le problème étudié. Un bref état de l'art des problèmes d'ordonnancement sur une machine sans prise en compte de l'aspect maintenance sera donné. Les méthodes de résolution, dans les deux cas séquentiel et intégré, seront ensuite décrites. Enfin, nous présenterons les différents résultats expérimentaux.

3.2 Politiques d'ordonnancement conjoint de la production et de la maintenance

Le rapprochement entre les deux fonctions production et maintenance est naturel vu que de plus en plus les petites tâches d'entretien sont intégrées dans les temps de production. L'objectif étant de planifier l'exécution des autres tâches de maintenance, en altérant le moins possible le plan de production, tout en respectant au mieux la périodicité de maintenance des équipements. Dans le cas de la maintenance préventive, le but est de réaliser l'ordonnancement des tâches de maintenance. Pour ce faire, différentes politiques de planification peuvent être mises en place. Certains auteurs proposent de réaliser les tâches de maintenance au cours d'arrêts des machines programmés pour d'autres activités (inspections de contrôle de qualité par exemple). D'autres se positionnement au niveau de la planification et déterminent un planning des opérations de maintenance et de production, sans se préoccuper des conflits qui risquent d'apparaître. Les derniers,

enfin, traitent des problèmes d'ordonnancement au sens propre relatif à une machine, à des machines parallèles et au flow-shop. Ils construisent un ordonnancement respectant toutes les contraintes et optimisant un critère donné. Par contre, ils ne s'intéressent pas forcément au même niveau de décision ni aux décideurs et encore moins à l'interactivité qui existe entre eux. On recense dans la littérature plusieurs stratégies d'ordonnancement qui vont être décrites ci-dessous et qui visent à résoudre ces conflits le plus efficacement possible. Trois politiques d'ordonnancement ont été recensées, l'ordonnancement séparé, le séquentiel et l'intégré [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 ordonnancement 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.

3.3 Description du problème

Nous étudions dans ce chapitre un problème d'ordonnancement sur une machine avec des tâches strictement non-préemptives, en supposant, comme dans la majorité des ouvrages traitant des contraintes de disponibilité dues à une activité de maintenance préventive, que les tâches de maintenance sont connues à l'avance. L'objectif est la minimisation d'une fonction objectif f tenant compte de la production et de la maintenance en même temps.

3.3.1 Notations et hypothèses

Pour formuler le problème d'ordonnancement conjoint de la production et de la maintenance sur une machine, nous allons utiliser les notations données au tableau 3.1. Ces notations nous permettront de définir et formuler le critère d'optimisation lié à la production et à la maintenance.

Symbole	Signification
$J = \{J_i/i = 1,2,\ldots,n\}$	ensemble des n travaux à ordonnancer
p_i	temps opératoire du travail J_i
r_i	date de début au plus tôt du travail J_i
d_i	date de fin au plus tard du travail J_i
C_i	date d'achèvement du travail J_i
p_m	durée d'une tâche de maintenance
$m = \{m_j/j = 1,2,\ldots,N_m\}$	ensemble des N_m tâches de maintenance à ordonnancer
S_{m_j}	date de début de la $j^{ème}$ tâche de maintenance m_j
C_{m_j}	date d'achèvement de la $j^{ème}$ tâche de maintenance m_j
T^*	période optimale de maintenance
ΔT	retard/avance toléré pour une tâche de maintenance
$T_{min} = T^* - \Delta T$	durée minimale séparant deux tâches de maintenance consécutives
$T_{max} = T^* + \Delta T$	durée maximale séparant deux tâches de maintenances consécutives

TAB. 3.1 – *Notations utilisées*

Afin d'optimiser le placement des tâches de maintenance nous supposons qu'elles sont "flexibles" et que leurs dates de début doivent être déterminées durant la procédure d'ordonnancement. Chaque tâche de maintenance préventive est caractérisée par une gamme de maintenance préétablie par le service maintenance ou par le constructeur de l'équipement considéré. Elle consiste en une suite d'opérations élémentaires dont la durée est évaluée avec plus ou moins de certitude (Par la suite, nous la supposerons fixe).

Nous supposons que la période optimale de maintenance T^* varie dans un intervalle de tolérance noté $[T_{min},T_{max}]$ (figure 3.2).

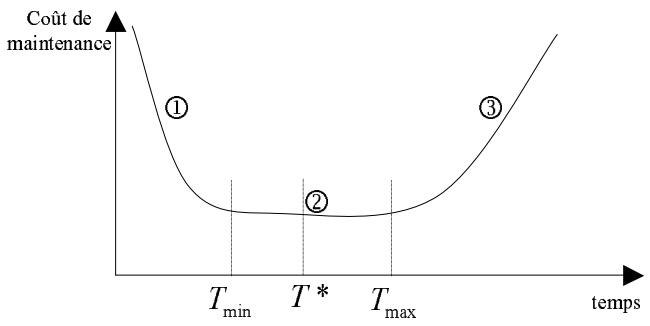


FIG. 3.2 – *Variation du coût de la maintenance*

Cet intervalle représente un compromis entre le coût total de maintenance et le risque de perte de disponibilité de la machine. Si la durée séparant deux opérations de maintenance consécutives est inférieure à T_{min} (partie (1) de la figure 3.2), les interventions sont trop fréquentes par rapport aux besoins réels ce qui rend la disponibilité des machines plus faibles et par conséquent le coût total de la maintenance trop élevé. Dans le cas contraire, le fait que la durée séparant deux tâches de maintenance consécutives soit supérieure à T_{max} (partie (3) de la figure 3.2) augmente la probabilité de panne et par la même le nombre d'interventions de maintenance correctives, ce qui induit un accroissement du coût total de la maintenance. Le cas idéal (partie (2) de la figure 3.2) serait de planifier la maintenance dans l'intervalle $[T_{min},T_{max}]$ car dans cet intervalle le coût de maintenance est relativement constant. La période optimale $T^* \in [T_{min},T_{max}]$ peut être déterminée à l'aide de modèles technico-économiques de coût [Den99].

Étant données les hypothèses et notations, nous donnerons dans ce qui suit une formulation du critère d'optimisation considéré dans cette étude. Ce critère tiendra compte des deux aspects production et maintenance.

3.3.2 Critère d'optimisation

3.3.2.1 Aspect maintenance

Le but de cette étude est de pouvoir effectuer les interventions de maintenance avec un minimum de retard ou d'avance et dans le cas optimal dans les délais. Ces interventions sont liées entre elles. En effet, l'emplacement de toute tâche de maintenance dépend de celui de la précédente.

La $j^{ème}$ tâche de maintenance m_j effectuée dans les délais vérifie la condition suivante :

$$C_{m_{j-1}} + T_{min} \leq S_{m_j} \leq C_{m_{j-1}} + T_{max}$$

La figure 3.3 montre les trois cas de figure pour le placement de la $j^{ème}$ tâche de maintenance qui peut être en avance, à temps ou en retard. L'hypothèse de la dépendance des placements des différentes tâches de maintenance préserve leur périodicité.

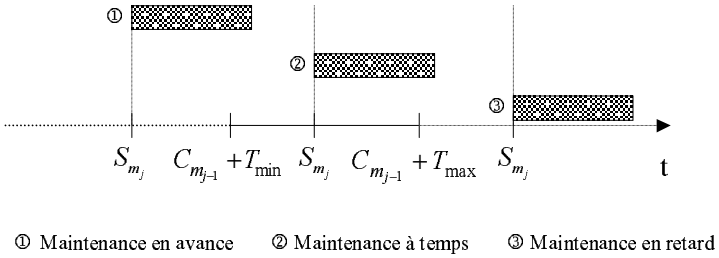


FIG. 3.3 – *Les différents cas d'affectation de la $j^{ème}$ maintenance*

Soit $\sigma = (J_1, J_2, \ldots, J_i, m_1, J_{i+1}, \ldots, m_j, \ldots)$ un ordonnancement contenant une sequence des travaux et des tâches de maintenance.

Soient

$E_{m_j} = max(0, C_{m_{j-1}} + T_{min} + p_m - C_{m_j})$ l'avance de la $j^{ème}$ tâche de maintenance.

$T_{m_j} = max(0, C_{m_j} - T_{max} - p_m - C_{m_{j-1}})$ le retard de la $j^{ème}$ tâche de maintenance.

Du point de vue du fournisseur, le respect des contraintes impose un bon fonctionnement de son système de production. Ce ci passe par le respect des périodes de maintenance prévues. Soit f_m la fonction objectif concernant la maintenance. Elle est définie par :

$$f_m = \sum_{j=1}^{N_m} (E_{m_j} + T_{m_j}) \tag{3.1}$$

3.3.2.2 Aspect Production

Les contraintes imposées par les clients à leurs fournisseurs s'expriment souvent en terme de délai, ce qui nous incite à considérer la somme des retards comme critère de production. Soit f_p cette fonction objectif définie par :

$$f_p = \sum_{i=1}^{n} \max(0, C_i - d_i) \tag{3.2}$$

3.3.2.3 Aspects Production & Maintenance

Pour optimiser les deux aspects production et maintenance, on effectue une agrégation des deux critères f_p et f_m. Un ordonnancement conjoint de la production et de la maintenance doit minimiser :

$$f(\sigma) = \phi_1 * f_p + \phi_2 * f_m \tag{3.3}$$

où ϕ_1 et ϕ_2 sont fixées par l'utilisateur suivant le degré d'importance qu'il donne à la production et à la maintenance. Elles peuvent éventuellement dépendre du nombre de tâches (production et/ou maintenance).

D'après la notation concernant les contraintes de disponibilité des machines introduite par Schmidt [Shm00], le problème d'ordonnancement sur une machine avec maintenance peut être noté $1,NC_{win-flex} \mid r_i \mid f$, où $NC_{win-flex}$ signifie que les périodes de maintenance sont arbitrairement distribuées sur la machine et sont flexibles.

Le problème d'ordonnancement classique sur une machine, c'est-à-dire sans prise en compte des contraintes d'indisponibilité a été démontré $\mathcal{NP}$-*difficile* au sens fort pour la somme des retards par Du et Leung [DL90]. L'introduction des périodes d'indisponibilité sur la machine rendra le problème plus difficile. Il est impossible de trouver une méthode optimale qui résout le problème en un temps polynomial. C'est pour cette raison qu'on est amené à proposer des méthodes approchées.

Avant de décrire les méthodes de résolution proposées pour les deux cas séquentiel et intégré, nous présenterons dans la section suivante un bref état de l'art sur les travaux réalisés concernant la minimisation du retard sur une machine.

3.4 État de l'art des problèmes classiques d'ordonnancement sur une machine

Plusieurs recherches ont été menées sur le problème $1 \mid \mid \sum T_i$. Smith [Smi56] a proposé une condition suffisante d'optimalité locale, qui plus tard a été examinée par Baker et Bertrand [BB82] en la considérant comme une règle de priorité dynamique. Une règle de dominance performante a été introduite par Emmons [Emm69].

Rinnooy Kan et *al.* [KLL75] et Rachamadugu [Rac87] ont étendu la règle d'Emmons [Emm69] au problème du retard pondéré. Rachamadugu [Rac87] a identifié une propriété liant les travaux adjacents dans une séquence optimale du problème $1 \mid \mid \sum W_i T_i$. Abdul-Razacq et *al.* [ARPW90] ont proposé deux méthodes exactes pour la résolution

du problème pondéré : un algorithme de programmation dynamique et un algorithme Branch and Bound. La plupart du temps la règle d'Emmons est utilisée pour former le graphe de précédence afin de trouver les bornes inférieure et supérieure.

Chu et Portmann [CP92] ont proposé une condition suffisante pour l'optimalité locale d'une solution du problème $1 \mid r_i \mid \sum T_i$ et qui peut être considérée comme une règle de priorité dynamique. Cette règle leur a permis de définir un sous-ensemble dominant d'ordonnancements et de construire des heuristiques efficaces. Pour ce problème, Chu [Chu92] a fourni une règle de dominance efficace et une borne inférieure.

Puisque le problème $1 \mid \mid \sum W_i T_i$ est $\mathcal{NP}$-*difficile* au sens fort [Law77], le problème avec des dates de disponibilité $1 \mid r_i \mid \sum W_i T_i$ est aussi $\mathcal{NP}$-*difficile* au sens fort. Pour ce problème, Akturk et Ozdemir [AO01] ont proposé une condition suffisante d'optimalité locale qui améliore les heuristiques. Cette règle est ensuite utilisée avec une généralisation de la règle de Chu dans un algorithme Branch and Bound [AO00]. Cet algorithme peut résoudre les problèmes de taille maximum égale à vingt travaux.

A partir de cet état de l'art, nous pouvons conclure que les règles de dominance et les règles de priorité sont utilisées dans les heuristiques pour s'approcher au mieux de la solution optimale. Ce qui nous a amené à proposer des heuristiques utilisant ces notions pour la résolution de notre problème. La section suivante sera consacrée à la description de ces différentes heuristiques.

3.5 Méthodes de résolution proposées

Nous proposons dans cette section des méthodes heuristiques dans le cas séquentiel et intégré. Ces heuristiques sont basées sur une règle de priorité pour le cas séquentiel et sur une règle de dominance pour le cas intégré.

3.5.1 Approche séquentielle

Le principe de cette approche est de planifier en premier lieu les tâches de maintenance, puis ordonnancer les travaux sans tenir compte de la maintenance. Enfin, unifier les travaux ordonnancés et les tâches de maintenance planifiées.

3.5.1.1 Planification de la maintenance

La maintenance effectuée est une maintenance préventive systématique où chaque tâche est dépendante de celle qui la précède (la date de début de chaque tâche de main-

tenance est dépendante de la date de fin de celle qui la précède). Pour chaque tâche de maintenance, on suppose qu'on connaît l'intervalle pendant lequel celle ci doit être exécutée. Une manière de placer la maintenance est de la planifier à chaque $(T_{min}+T_{max})/2$. De cette façon on pourra avancer ou reculer une tâche de maintenance sans qu'elle soit trop en retard ou trop en avance. La figure 3.4 montre ce principe de planification de la maintenance.

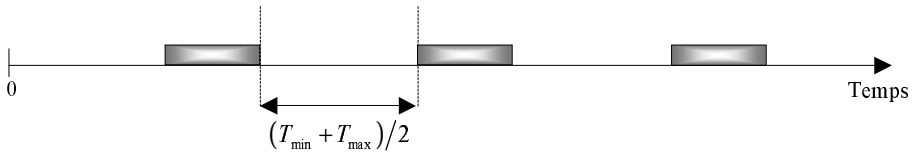


FIG. 3.4 – *Planification de la maintenance*

3.5.1.2 Ordonnancement de la production

L'ordonnancement des travaux est réalisé à l'aide d'un algorithme basé sur une règle de priorité appelée **PRTT** (Priority Rule for Total Tardiness) proposée par Chu et Portmann [CP89b]. Cette règle est définie comme suit :

Définition 3.1. *Une fonction $PRTT(J_i,\Delta)$ du travail J_i à l'instant Δ est définie comme*

$$PRTT(J_i,\ \Delta) = \max(r_i,\ \Delta) + \max(\max(r_i,\ \Delta) + p_i,\ d_i)$$

$PRTT(J_i,\ \Delta)$ est la somme de la date de début au plus tôt "dynamique" $(\max(r_i,\ \Delta))$ et de la date de fin au plus tard "dynamique" modifiée $(\max(\max(r_i,\ \Delta) + p_i,\ d_i))$.

Soit (J_i,J_k) un ordonnancement partiel obtenu en plaçant le travail J_i immédiatement avant J_k sur une machine disponible à l'instant Δ, et T_{jk} la somme des retards des travaux J_i et J_k. En se basant sur la fonction définie ci-dessus , la condition suffisante d'optimalité locale est donnée par le théorème suivant.

Théorème 3.1. *[CP89b] Considérons deux travaux J_i et J_k à ordonnancer sur une machine disponible à partir de l'instant Δ. Si $PRTT(J_i,\ \Delta) \leq PRTT(J_k,\ \Delta)$ alors $T_{ik} \leq T_{ki}$.*

Selon ce théorème, la fonction $PRTT$ peut être considérée comme une mesure de priorité des travaux : plus la valeur de cette fonction est petite, plus le travail correspondant est prioritaire. En utilisant cette règle un algorithme approché performant a été construit par Chu et Portmann [CP92] appelé $IPRTT$. Cet algorithme comporte

deux phases, une phase d'affectation et une phase d'insertion. Dans la première phase, à chaque itération, on ordonnance un travail J_i qui est disponible le plus tôt parmi les travaux non ordonnancés les plus prioritaires selon la règle $PRTT$. Il se peut que l'ordonnancement de ce travail laisse la machine inoccupée, c'est pourquoi une deuxième phase d'insertion est définie pour remplir ce vide. Cette phase consiste à placer le travail le plus prioritaire parmi ceux qui peuvent commencer plus tôt et finir avant le début du travail J_i. Le processus continu jusqu'à ce qu'aucun travail ne puisse être inséré devant le travail J_i, auquel cas la phase d'insertion se termine et on passe à l'itération suivante jusqu'à ce que tous les travaux soient placés.

Nous allons maintenant présenter les deux heuristiques proposées dans le cas séquentiel et qui utilisent l'algorithme $IPRTT$ pour l'ordonnancement de la production. Le principe de la première est simple. Il consiste à décaler la maintenance si cette dernière laisse la machine inoccupée. Tandis que la deuxième est une procédure de branchement. Ces deux heuristiques seront bien détaillées dans la section suivante.

3.5.1.3 Ordonnancement conjoint Production/Maintenance

Heuristique séquentielle HS1

Le principe général de cette heuristique est d'unifier les tâches de maintenance planifiées et les travaux ordonnancés avec l'algorithme $IPRTT$. Si le placement d'une tâche de maintenance crée un temps mort sur la machine, alors deux cas de figures sont possibles. On décale cette tâche de maintenance à gauche si cette action ne la rend pas en avance. Sinon, si ce décalage n'est pas possible, on fait une insertion décalage à gauche de cette tâche de maintenance jusqu'à minimisation maximum du temps mort. L'algorithme 3.1 explique en détail le principe de cette heuristique.

Après avoir ordonnancé les travaux, le nombre de tâches de maintenance est calculé à l'aide de la formule suivante : $N_m = \lfloor 2 * (C_{max} - p_{[n]})/(T_{min} + T_{max}) \rfloor$ où $p_{[n]}$ est la durée opératoire du dernier travail ordonnancé. Avec cette formule on évite la mise d'une maintenance inutile à la fin de l'horizon de planification.

Exemple : considérons un problème de six travaux, les durées opératoires p_i, les dates de disponibilité r_i et les dates de fin au plus tard sont données dans le tableau 3.2, $\phi_1 = 1$ et $\phi_2 = 1$. Concernant la maintenance, $T_{min} = 10$, $T_{max} = 15$ et $p_m = 8$. Le diagramme de Gantt de la solution obtenue est donné dans la figure 3.5. En appliquant cette heuristique, nous obtenons un ordonnancement de valeur du égale à 43.

Cette heuristique présente l'inconvénient de ne pas donner trop de flexibilité aux

J_i	1	2	3	4	5	6
p_i	5	10	4	8	7	6
r_i	0	3	5	18	20	30
d_i	6	45	50	30	27	40

TAB. 3.2 – *Données choisies pour les tests*

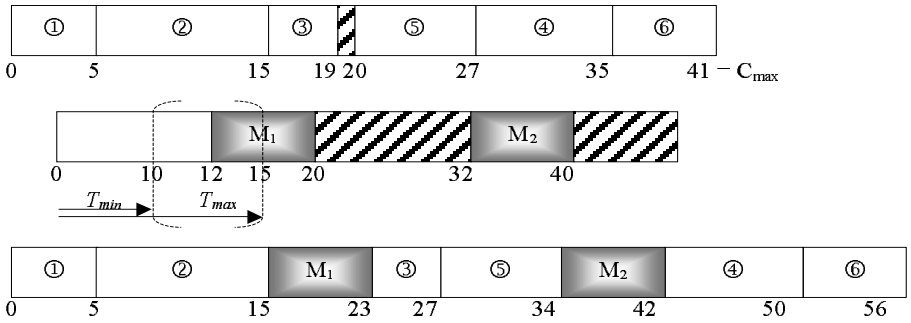


FIG. 3.5 – *Ordonnancement obtenu par l'heuristique HS1*

tâches de maintenance. C'est pour cette raison qu'on a proposé l'heuristique $HS2$ qui énumère à chaque instant tous les emplacements possibles d'une tâche de maintenance. Cette heuristique sera détaillée dans le paragraphe suivant.

Heuristique séquentielle HS2 [KVZ02]

Cette heuristique tient compte de l'intervalle de tolérance attribué à chaque tâche de maintenance. Une tâche de maintenance peut être insérée sur tous les emplacements de son intervalle de tolérance. Deux emplacements sont possibles : à la fin de chaque lot qui se trouve à l'intérieur de l'intervalle ou bien à l'instant t date début de l'intervalle s'il n'existe aucun travail qui s'exécute à cet instant. Pour chaque emplacement on évalue l'ordonnancement obtenu (critère d'optimisation f). Le choix du meilleur emplacement est celui qui minimise la fonction f. Dans le but d'essayer d'optimiser la solution finale, une tâche de maintenance pourra être insérée avant sa date de début au plus tôt (maintenance inutile sur la figure 3.2) ou après sa date de fin au plus tard (risque de panne sur la figure 3.2) à condition que l'évaluation de l'ordonnancement résultant soit meilleure. La recherche d'une solution est faite en profondeur d'abord sans remise en cause des choix opérés.

Chaque noeud est défini par une séquence partielle K contenant les travaux ordonnancés et les tâches de maintenance insérées. Chaque noeud descendant est obtenu par l'insertion, dans la séquence partielle K, d'une tâche de maintenance. La solution initiale

Algorithme 3.1 Algorithme de l'heuristique HS1

Entrée : $J = \{J_i / i = 1, \ldots, n\}$ *{l'ensemble des travaux à ordonnancer}*

Début

Ordonnancer les travaux avec l'algorithme $IPRTT$;
Planifier les N_m maintenances toutes les $(T_{min} + T_{max})/2$ unités de temps ;
$i := 0 \quad t := 0 \quad j := 0$;
Tant que $i < n$ **faire**

Tant que $C_i \leq \max(r_i,t) + (T_{min} + T_{max})/2$ **faire**
$i = i + 1$;
Fin tant que
Si $C_i \geq \max(r_i,t) + T_{min}$ **alors**
placer la maintenance m_j à la date C_i ;
$\max(r_i,t) = C_i + p_m$;
$j = j + 1$;
Sinon
$i = i + 1$;
Tant que $C_{i-1} + p_i \leq \max(r_i,t) + T_{max}$ **faire**
$C_i = C_{i-1} + p_i$;
$i = i + 1$;
Fin tant que
placer la maintenance m_j à la date C_{i-1} ;
$t = C_{i-1} + p_m$;
Fin si
Fin tant que
Fin

est celle donnée par l'algorithme $IPRTT$. Dans le but d'expliquer en détail l'algorithme de l'heuristique, nous introduisons les notations suivantes :
$T^j_{min} = C_{m_{j-1}} + T_{min}$ la date de début au plus tôt de la tâche m_j ;
$T^j_{max} = C_{m_{j-1}} + T_{max}$ la date de début au plus tard de la tâche m_j ;
S_D un ordonnancement partiel ;
$l_{ij} = \sum_{k=1}^{n} \max(0,\ C_k - d_k) + \max(0,\ T^j_{min} - S_{m_j}) + \max(0,\ S_{m_j} - T^j_{max})$ la borne inférieure ;
$\Gamma_j = \{J_i,\ J_{i+1}, \ldots,\ J_k\}$ la séquence des travaux non définitivement ordonnancés dans l'intervalle $I_j = [T^j_{min},\ T^j_{max}]$ où i est le plus petit indice avec $C_i \geq T^j_{min}$ et k le plus grand indice avec $C_k \leq T^j_{max}$.

En suivant ces notations, l'algorithme 3.2 donne en détail le principe de cette heuristique. Cette dernière sera mieux expliquée par les figures 3.6 et 3.7.

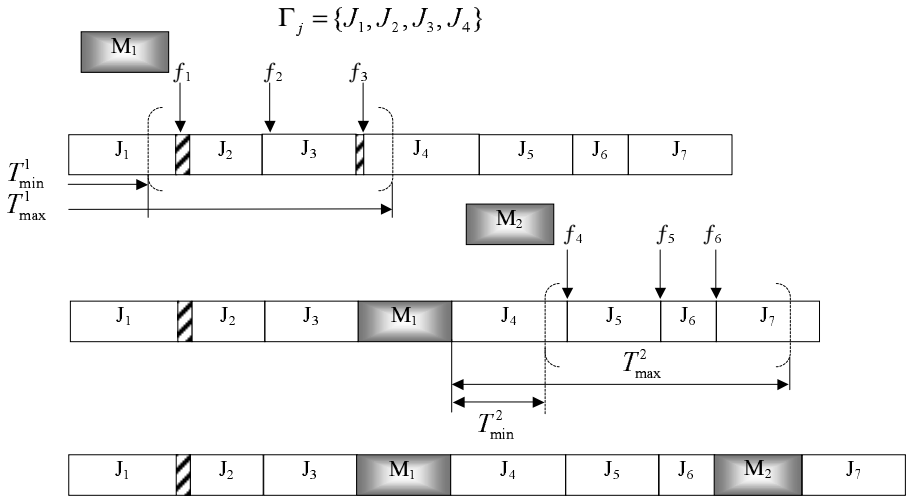


FIG. 3.6 – *Emplacements possibles des tâches de maintenance*

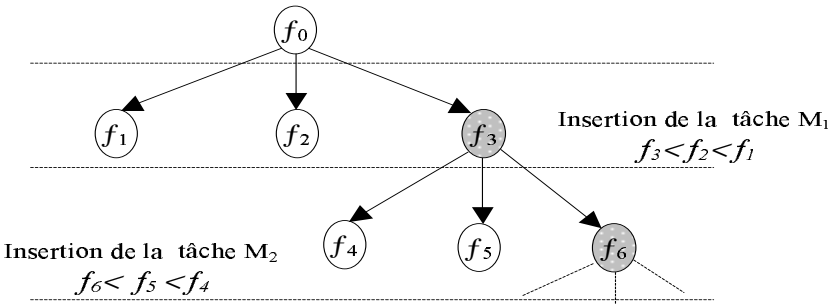


FIG. 3.7 – *Recherche en profondeur*

Algorithme 3.2 Algorithme de l'heuristique HS2

Entrée : $J = \{J_i / i = 1, \ldots, n\}$ *{l'ensemble des travaux à ordonnancer}*

Début

Ordonnancer les travaux avec l'algorithme $IPRTT$. Soit π l'ordonnancement obtenu.
Utiliser la valeur du critère donnée par HS1 comme une borne supérieure BS.
$S_D = \pi \quad j = 1 \quad \Gamma_j = \emptyset$;
Tant que $j <= n$ **faire**
calculer Γ_j ;
Pour chaque travail J_i de Γ_j **faire**
générer un nouveau noeud en insérant dans S_D avant J_i une tâche de maintenance m_j ;
calculer l_{ij} ;
Si $l_{ij} \geq BS$ **alors**
éliminer le noeud correspondant ;
Sinon
conserver le noeud correspondant ;
Fin si
Fin pour
conserver la séquence S_D ayant la valeur minimale ;
j=j+1 ;
Fin tant que

Fin

Exemple 3.1. *Nous reprenons les mêmes données figurant dans le tableau 3.2. L'application de l'heuristique HS2 nous fourni l'ordonnancement donné sur la figure 3.8. La valeur du critère obtenue est égale à* 39*. Nous remarquons bien que ce résultat est meilleur que celui donné par HS1.*

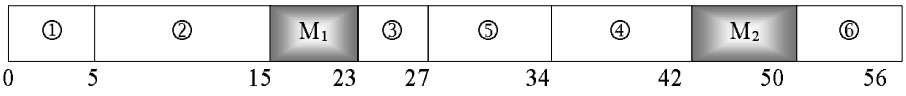


FIG. 3.8 – *Diagramme de Gantt du résultat de l'heuristique HS2*

On remarque bien que l'heuristique $HS2$ améliore $HS1$ à cause de la flexibilité qu'elle donne aux tâches de maintenance. Toutefois, la stratégie séquentielle n'est pas une stra-

tégie idéale pour tenir compte des deux aspects production et maintenance. Une stratégie intégrée sera donc plus adaptée au problème d'ordonnancement conjoint de la production et de la maintenance. Cette approche fera l'objet de la section suivante.

3.5.2 Approche intégrée

Nous étudions dans cette section le problème d'ordonnancement conjoint de la production et de la maintenance sur une machine en suivant une approche intégrée ([KVZ03b]) qui nous permettra de mieux tenir compte de l'aspect maintenance. Deux méthodes heuristiques ont été proposées pour la résolution de ce problème. La première considère une tâche de maintenance comme un travail et ordonnance tout avec le même algorithme. Par contre, la deuxième heuristique utilise une règle de dominance reliant un travail et une tâche de maintenance. Nous donnerons ci-dessous les détails de ces deux heuristiques.

Heuristique HI1

L'idée générale de cette heuristique est de considérer une tâche de maintenance comme un travail et ce dans le but de pouvoir utiliser la règle de priorité $PRTT$. On suppose aussi qu'une tâche de maintenance ne peut pas être en avance. Au début on crée la première maintenance et on l'ajoute à l'ensemble des travaux. Les travaux seront ensuite triés à l'aide de $PRTT$. Le premier travail est ainsi placé au début de l'ordonnancement, si ce dernier est une maintenance alors on crée dynamiquement la maintenance suivante et on refait le même procédé.

Afin de détailler l'algorithme (algorithme 3.3) de cette heuristique, nous introduisons les notations suivantes pour une tâche de maintenance m_j (en supposant que $(j-1)$ tâches de maintenance sont déjà placées) :

$r_j = C_{m_{j-1}} + T_{min}$ la date de début au plus tôt de m_j ;
$d_j = C_{m_{j-1}} + p_m + T_{max}$ la date de fin au plus tard de m_j.

Algorithme 3.3 Algorithme de l'heuristique HI1

Entrée : $TS = \{J_i / i = 1, \ldots, n\}$ *{l'ensemble des travaux à ordonnancer}*

Début

$N_m = 1, \quad t = 0, \quad f = 0$;
créer le travail J_{n+N_m} avec $r_{n+N_m} = T_{min}$ et $d_{n+N_m} = T_{max} + p_m$;
$TS = TS \cup \{J_{n+N_m}\}$;
$N_m = N_m + 1$;
Tant que $TS \neq \emptyset$ **faire**
sélectionner le travail J_i ayant la plus petite valeur de $PRTT$;
$t = max(r_i, t) + p_i$;
Si $i \leq n$ **alors**
$f = f + \phi_1 * max(0, t - d_i)$;
Sinon
$f = f + \phi_2 * max(0, t - d_i)$;
Fin si
$TS = TS \setminus \{J_i\}$;
Si $i > n$ (le travail sélectionné est une maintenance) **alors**
créer le travail J_{n+N_m} avec $r_{n+N_m} = t + T_{min}$ et $d_{n+N_m} = t + T_{max} + p_m$;
$TS = TS \cup \{J_{n+N_m}\}$;
$N_m = N_m + 1$;
Fin si
Fin tant que

Fin

Exemple 3.2. *Nous reprenons les mêmes données figurant dans le tableau 3.2. Concernant la maintenance,* $T_{min} = 10$*,* $T_{max} = 15$ *et* $p_m = 8$*. Nous donnons dans ce qui suit l'exécution de deux itérations de l'heuristique HI1.*

$t = 0$*, on crée une maintenance* m_1 *(*J_7*) avec* $r_7 = 10$ *et* $d_7 = 23$*.*

$$\left.\begin{array}{l} PRTT(J_1, 0) = 6 \\ PRTT(J_2, 0) = 50 \\ PRTT(J_3, 0) = 55 \\ PRTT(J_4, 0) = 48 \\ PRTT(J_5, 0) = 47 \\ PRTT(J_6, 0) = 70 \\ PRTT(J_7, 0) = 33 \end{array}\right\} \Rightarrow \textit{on sélectionne le travail } J_1 \textit{ (}PRTT(J_1, 0) \textit{ est la plus petite)}$$

$t = 5$

$$\left.\begin{array}{l} PRTT(J_2,\,5) = 50 \\ PRTT(J_3,\,5) = 55 \\ PRTT(J_4,\,5) = 48 \\ PRTT(J_5,\,5) = 47 \\ PRTT(J_6,\,5) = 70 \\ PRTT(J_7,\,5) = 33 \end{array}\right\} \Rightarrow \textit{on sélectionne le travail } J_7 \textit{ (c'est une tâche de maintenance)}$$

on crée alors une maintenance m_2 *(*J_8*) avec* $r_8 = 28$ *et* $d_8 = 41$*. On réitère le même procédé pour les autres travaux.*

La séquence obtenue finalement est $S = \{J_1,\, m_1,\, J_5,\, J_4,\, J_6,\, m_2,\, J_3,\, J_2\}$ *avec une valeur du critère égale à* 35*. La figure 3.9 représente le diagramme de Gantt de la solution trouvée.*

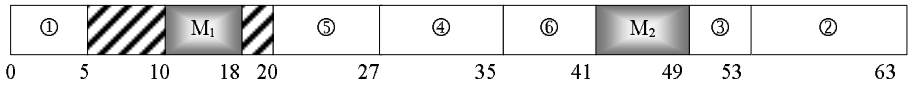


FIG. 3.9 – *Diagramme de Gantt du résultat de l'heuristique HI1*

D'après les tests faits sur cette heuristique, nous avons remarqué que le fait de considérer une tâche de maintenance comme un travail n'est pas réaliste. En effet, le principe de la règle $PRTT$ est de donner la priorité aux travaux ayant des durées opératoires courtes et de retarder ceux avec des durées assez longues. De ce fait, les tâches de maintenance de grandes durées se trouvent toujours retardées.

Afin de préserver les spécificités des tâches de maintenance, nous allons maintenant développer une deuxième heuristique basée sur une règle de dominance reliant les travaux et les tâches de maintenance. Cette règle s'est montrée expérimentalement efficace pour construire de bonnes solutions. Elle sera présentée ci-dessous.

Règle de dominance locale

Du fait de la prise en compte de la maintenance dans le critère d'optimisation, il n'est pas possible d'obtenir une fonction de priorité pour chaque travail et qui ne dépend que du temps. En se basant sur ces idées, on définit une condition d'optimalité locale pour le critère considéré.

D'après l'étude expérimentale faite sur l'heuristique HI1, on a remarqué que le fait de ne pas permettre à une maintenance d'être en avance risque parfois de laisser la machine inoccupée en attente d'une tâche de maintenance préventive. On a alors ajouté l'hypothèse qu'une tâche de maintenance m_j peut être, soit en avance soit en retard et

on lui attribue une date au plus tôt pour l'avance ρ_j. Concernant les travaux, on pose $\rho_i = -\infty \ \forall \ i = 1, \ldots, n$; c'est à dire qu'un travail ne peut pas être en avance.

Supposons qu'à l'instant t on a à ordonnancer un travail J_i et une maintenance m_k. Soit Ψ_{ik} la somme des retards/avances de J_i et m_k obtenue en plaçant J_i avant m_k à l'instant t. Ψ_{ik} est donnée par la formule suivante :

$$\Psi_{ik}(t) = \phi_1 * \Big(\max(0,\ \rho_i - R_i - p_i) + \max(0,\ R_i + p_i - d_i) \Big) +$$

$$\phi_2 * \Big(\max\big(0,\ \rho_k - \max(R_i + p_i,\ R_k) - p_k\big) + \max\big(0,\ \max(R_i + p_i,\ R_k) + p_k - d_k\big) \Big)$$

avec $R_i = \max(r_i,\ t)$.

On définit la fonction $PRTTE$ (**P**riority **R**ule for **T**otal **T**ardiness **E**arliness) pour un travail J_i et une tâche de maintenance m_j comme suit :

$$PRTTE(J_i,\ m_j, t) = \Psi_{ij}(t) - \Psi_{ji}(t)$$

En utilisant cette règle de dominance locale, nous avons proposé le théorème suivant :

Théorème 3.2. *Considérons un travail J_i et une tâche de maintenance m_j à ordonnancer sur une machine disponible à partir de l'instant t. Placer J_i avant m_j est optimal ssi $PRTTE(J_i,\ m_j, t) < 0$.*

Preuve. Soit S un ordonnancement optimal contenant deux tâches J_i et m_j tel que J_i est placée avant m_j et $PRTTE(J_i,\ m_j,\ t) >= 0$.
Soit S' l'ordonnancement obtenu à partir de S en interchangeant J_i et m_j. $f(S) - f(S') = \Psi_{ij}(t) - \Psi_{ji}(t) = PRTTE(J_i,\ m_j,\ t) >= 0$, contradiction avec S optimal.

Cette règle de dominance locale nous a permis de proposer une heuristique qui tient compte des deux aspects production et maintenance séparément. L'idée générale de cette heuristique est de déterminer, avec la règle $PRTTE$ et à chaque itération, l'ensemble des travaux qu'on peut placer avant une maintenance fixée. Si un tel ensemble est vide, cela signifie qu'il n'existe plus de travaux à insérer avant cette tâche de maintenance. Dans ce cas on place la maintenance et on passe à l'itération suivante où on crée la prochaine tâche de maintenance. Nous présentons ci-dessous le détail de cette heuristique appelée HI2 et nous utilisons les mêmes notations que l'heuristique HI1.

Algorithme 3.4 Algorithme de l'heuristique HI2

Entrée : $TS = \{J_i / i = 1, \ldots, n\}$ *{l'ensemble des travaux à ordonnancer}*

Début

$N_m = 1, \quad t = 0, \quad f = 0$, $\rho_i = -\infty \ \forall J_i \in TS$;

créer le travail J_{n+N_m} avec $r_{n+N_m} = 0$, $\rho_{n+N_m} = T_{min} + p_m$ et $d_{n+N_m} = T_{max} + p_m$;

$TS = TS \cup \{J_{n+N_m}\}$;

Tant que $TS \neq \emptyset$ **faire**

déterminer $TB = \{J_i \in TS / PRTTE(J_i, J_{n+N_m}, t) < 0\}$;

Si $TB \neq \emptyset$ **alors**

sélectionner le travail $J_i \in TB$ avec la plus petite valeur de $PRTT$;

$t = \max(r_i, t) + p_i$;

$f = f + \phi_1 * \max(0, t - d_i)$;

$TS = TS \setminus \{J_i\}$

Sinon

$f = f + \phi_2 * \big(\max(0, \rho_{n+N_m} - t - p_m) + \max(0, t + p_m - d_{n+N_m})\big)$;

$t = t + p_m$;

$N_m = N_m + 1$;

créer le travail J_{n+N_m} avec $r_{n+N_m} = t$, $\rho_{n+N_m} = t + T_{min} + p_m$ et $d_{n+N_m} = t + T_{max} + p_m$;

$TS = TS \cup \{J_{n+N_m}\}$;

Fin si

Fin tant que

Fin

Exemple 3.3. *Nous reprenons les mêmes données figurant dans le tableau 3.2 ainsi que celles Concernant la maintenance. Nous donnons dans ce qui suit l'exécution de trois itérations de l'heuristique HI2.*

$t = 0$, *on crée une maintenance* m_1 *(*J_7*) avec* $r_7 = 0$, $\rho_7 = 18$ *et* $d_7 = 23$.

$$\left.\begin{array}{l} PRTTE(J_1, J_7, t) = -12 \\ PRTTE(J_2, J_7, t) = -10 \\ PRTTE(J_3, J_7, t) = -9 \\ PRTTE(J_4, J_7, t) = 1 \\ PRTTE(J_5, J_7, t) = 2 \\ PRTTE(J_6, J_7, t) = 11 \end{array}\right\} \Rightarrow TB = \{J_1, J_2, J_3\}$$

$\Rightarrow$ *on sélectionne le travail* J_1 *(*$PRTT(J_1, t)$ *est la plus petite).*

$t = 5$

$$\left.\begin{array}{l} PRTTE(J_2,\, J_7,\, t) = -5 \\ PRTTE(J_3,\, J_7,\, t) = -4 \\ PRTTE(J_4,\, J_7,\, t) = 6 \\ PRTTE(J_5,\, J_7,\, t) = 7 \\ PRTTE(J_6,\, J_7,\, t) = 16 \end{array}\right\} \Rightarrow TB = \{J_2,\, J_3\}$$

$\Rightarrow$ *on sélectionne le travail* J_2.

$t = 15$

$$\left.\begin{array}{l} PRTTE(J_3,\, J_7,\, t) = 4 \\ PRTTE(J_4,\, J_7,\, t) = 10 \\ PRTTE(J_5,\, J_7,\, t) = 3 \\ PRTTE(J_6,\, J_7,\, t) = 21 \end{array}\right\} \Rightarrow TB = \emptyset$$

$\Rightarrow$ *on place la maintenance* m_1 *(*J_7*) et on crée alors une deuxième tâche* m_2 *(*J_8*) avec* $r_8 = 23$, $\rho_8 = 41$ *et* $d_8 = 46$. *On continue de la même manière pour les autres travaux.*

La séquence obtenue finalement est $S = \{J_1,\, J_2,\, m_1,\, J_5,\, J_4,\, J_6,\, J_3,\, m_2\}$ *avec une valeur du critère égale à* 25. *La figure 3.10 représente le diagramme de Gantt de la solution trouvée.*

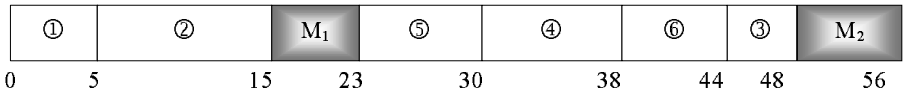


FIG. 3.10 – *Diagramme de Gantt du résultat de l'heuristique HI2*

Il est clair que cette heuristique conserve bien les aspects production et maintenance. Néanmoins, pour quelques cas, elle engendre un grand retard pour la maintenance ce qui contredit la réalité industrielle. En effet, on ne peut pas retarder indéfiniment une tâche de maintenance car au delà d'une certaine limite la machine risque de tomber en panne et par conséquent provoquer un arrêt de la production. Cette constatation nous a incité à fixer une deadline pour la maintenance. Cette deadline est donnée par la formule suivante : à chaque instant t où on crée une nouvelle maintenance m_j on a $\widetilde{d}_j = t + T_{max} + \tau * T^* + p_m$.

L'heuristique HI2 a été alors modifiée pour prendre en compte cette nouvelle hypothèse. Cette heuristique sera appelée HI3. La modification se situe dans le placement du travail $J_i \in TB$ sélectionné avec la règle $PRTT$. En effet, ce travail ne sera pas placé automatiquement mais un test est nécessaire. Ce test consiste à vérifier si ce placement peut entraîner le dépassement de la tâche de maintenance sa deadline. Si tel est le cas

on place plus tôt la maintenance.

3.5.3 Borne Inférieure

Cette borne est une adaptation de la borne inférieure proposée par Jouglet [Jou02] pour la minimisation de la somme des retards sur une machine ne subissant pas de périodes d'indisponibilité.

L'idée générale de cette adaptation est de considérer une tâche de maintenance comme un job et de relaxer le problème à un problème avec préemption. L'algorithme de cette borne inférieure utilise les deux propositions suivantes. Ces deux propositions ne sont valables que dans le cas préemptif.

Proposition 3.1. *[Jou02] Soient J_i et J_k deux jobs tels que $r_i \leq r_k$, $p_i \leq p_k$ et $d_i \leq d_k$, alors il existe un ordonnancement optimal dans lequel J_k commence après la fin de J_i.*

Proposition 3.2. *[Jou02] Soient J_i et J_k deux jobs tels que $r_i \leq r_k$, $p_i \leq p_k$ et $d_i > d_k$, alors échanger les valeurs de d_i et d_k n'augmente pas la valeur optimale du critère somme des retards.*

Nous supposons connaître le nombre de tâches de maintenance N_m à effectuer sur l'horizon de production. On commence par créer la première maintenance m_1 avec $r_1 = T_{min}$ et $d_1 = T_{max} + p_m$ et on l'ajoute à l'ensemble des jobs à ordonnancer. Lorsqu'une tâche de maintenance est placée, on crée dynamiquement la suivante. D'une manière générale, une tâche de maintenance m_k a une date de début au plus tôt $r_k = C_{m_{k-1}} + T_{min}$ et une date de fin au plus tard $d_k = C_{m_{k-1}} + T_{max} + p_m$. Les deux propositions citées ci-dessus nous permettent de calculer une borne inférieure BI selon l'algorithme suivant.

A chaque instant t, on détermine $D = \{J_i / r_i \leq t \wedge p'_i > 0\}$ l'ensemble des jobs disponibles mais non encore finis à l'instant t (p'_i est la durée opératoire restante du job J_i à l'instant t). Soient J_i le job ayant la plus petite durée opératoire restante, et J_k le job ayant la plus petite date de fin au plus tard. Si $d_i = d_k$, c-à-d, J_i a la plus petite date de fin au plus tard, selon la proposition 3.1, il est optimal de placer une unité de ce job. Si ce n'est pas le cas, selon la proposition 3.2, on échange la date de fin au plus tard de J_i avec celle de J_k (J_k a la plus petite date de fin au plus tard). Cette nouvelle instance du problème a une somme de retard optimale inférieure ou égale à celle du problème d'origine. Dans ce nouveau problème, J_i a maintenant la plus petite date de fin au plus tard et la plus petite durée opératoire restante, alors selon la proposition 3.1, il est optimal de placer une unité de ce job. On incrémente t et on réitère le même

procédé jusqu'à avoir placé tous les jobs. Si un job fini est une tâche de maintenance, on crée la tâche de maintenance suivante et on l'insère dans l'ensemble des jobs non encore placés.

L'algorithme de cette borne inférieure ainsi que toutes les heuristiques citées dans les paragraphes précédents ont été programmées en langage C et testées sur différents jeux de données. Elles sont ensuite comparées entre elles ainsi qu'avec la borne inférieure. Tel est le sujet de la section suivante.

3.6 Étude expérimentale

Afin de tester les heuristiques développées dans les sections précédentes, nous avons proposé une méthode de génération des données initiales. Cette méthode est une adaptation de celle proposée par Chu et Portmann [CP92] en tenant compte de la maintenance au moment de la génération des données de production.

Méthode de génération des données initiales :

- Les durées opératoires p_i sont générées uniformément entre 1 et 100 ;
- La période optimale de maintenance préventive est calculée à l'aide de la formule suivante : $T^* = \gamma * \sum_{i=1}^{n} p_i$ avec $\gamma \in \{0.15,\ 0.25,\ 0.5\}$;
- Les tâches de maintenance sont supposées être des tâches de durée moyenne. La durée d'une maintenance est donc donnée par la moyenne des durées opératoires des travaux : $p_m = 1/n * \sum_{i=1}^{n} p_i$;
- Les dates de début au plus tôt r_i sont générées uniformément entre 0 et X, avec $X = \alpha * \left(\sum_{i=1}^{n} p_i + p_m * \frac{\sum_{i=1}^{n} p_i}{T^*} \right)$ et $\alpha \in \{0,\ 0.5,\ 1\}$;
- Les dates de fin au plus tard sont générées selon la loi uniforme de la manière suivante : $d_i - (r_i + p_i) \in U[0,\ Y]$, avec $Y = \beta * \left(\sum_{i=1}^{n} p_i + p_m * \frac{\sum_{i=1}^{n} p_i}{T^*} \right)$ et $\beta \in \{0.15,\ 0.25,\ 0.5\}$;
- Le retard/avance toléré(e) pour la maintenance est donné(e) par la formule suivante : $\Delta T = 0.05 * T^*$.

Étude comparative des heuristiques HS1 et HS2 :

Dans cette partie nous présentons une étude expérimentale des deux heuristiques $HS1$ et $HS2$ proposées dans le cas séquentiel. Nous étudions le pourcentage d'écart entre les résultats des deux heuristiques ainsi que la variation du coût de la maintenance.

Les algorithmes des deux heuristiques ont été programmés en utilisant le langage C. Nous nous sommes intéressés aux problèmes de taille n allant de 20 à 200. Pour chaque

combinaison entre n, α, β et γ, 10 instances sont générées. On prendra pour les premiers tests $\phi_1 = 1$ et $\phi_2 = 1$.

La figure 3.11 représente l'évolution, en fonction du nombre total des travaux (n), du pourcentage d'écart entre le résultat donné par l'heuristique $HS1$ et celui donné par $HS2$ (moyenne sur les 10 instances générées pour chaque problème) pour $\alpha = 0$. Le pourcentage d'erreur est calculé à l'aide de la formule suivante : $\frac{HS1-HS2}{H2} \times 100$.

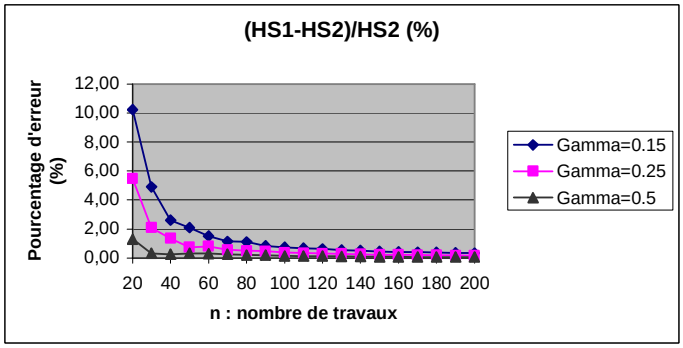


FIG. 3.11 – *Évolution du pourcentage d'erreur entre HS1 et HS2 en fonction du nombre de travaux n*

La figure 3.11 montre que la moyenne d'écart entre $HS2$ et $HS1$ diminue quand le nombre de travaux n augmente. Néanmoins, le pourcentage d'erreur de l'heuristique $HS1$ par rapport à $HS2$ ne dépasse pas les 11%. Le fait que α soit égale à 0 signifie que tous les travaux sont disponibles à $t = 0$ et par conséquent les dates de fin au plus tard sont assez petites. De ce fait, il y aura beaucoup de retard au bout d'un certain temps et par la même les deux heuristiques donnent des résultats assez proches.

Dans le cas où $\alpha > 0$, les résultats expérimentaux ont montrés une nette différence entre les deux heuristiques. La figure 3.12 représente l'évolution du coût total en fonction du nombre de travaux n pour les cas où $\gamma = 0.15$ et $\gamma = 0.5$. On constate bien que le placement de la maintenance (choix de γ) influence sur l'amélioration des résultats des deux heuristiques. Une autre constatation est que dans le cas où le nombre de tâches de maintenance est faible ($\gamma = 0.5$) les deux heuristiques s'éloignent l'une de l'autre plus rapidement que lorsque $\gamma = 0.15$; mais $HS2$ reste toujours meilleure.

L'heuristique $HS2$ a été plus efficace concernant l'optimisation du coût de la production. En effet, le tableau 3.3 représente le coût de production donné par chaque heuristique et pour chaque tâche. Les deux dernières cases comportent les temps d'exécutions en seconde de $HS1$ et $HS2$. On remarque bien que l'heuristique $HS2$ met plus

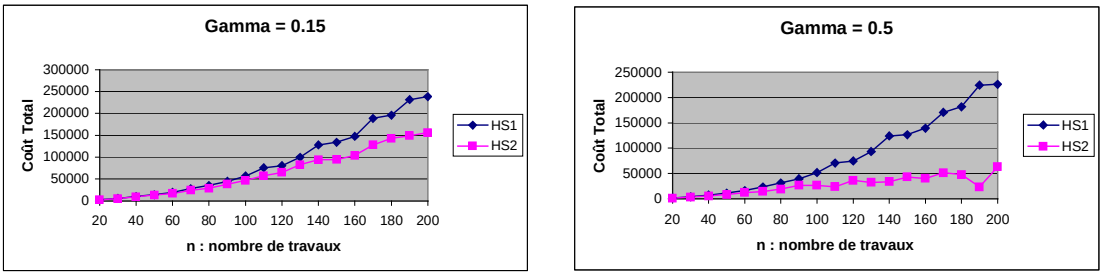


FIG. 3.12 – *Comparaison entre HS1 et HS2 en fonction du nombre de travaux n*

de temps que $HS1$, toutefois les deux heuristiques restent raisonnables du point de vue temps d'exécution.

n	HS1	HS2	CPU-HS1/sec	CPU-HS2/sec
20	5432	4945	0,01	0,32
40	19542	18978	0,02	0,16
60	42141	41427	0,02	0,90
80	74657	73728	0,01	0,18
100	115390	114448	0,01	0,18
120	158076	156969	0,01	0,19
140	233658	232345	0,01	0,20
160	280567	279282	0,02	0,21
180	363695	362156	0,02	0,22
200	433018	431495	0,02	0,23

TAB. 3.3 – *Variation du coût de production et du temps d'exécution de HS1 et HS2*

Bien que le coût total donné par $HS2$ est meilleur que celui donné par $HS1$, cette dernière optimise bien le critère concernant la maintenance. Par exemple, dans le cas où les travaux sont disponibles à $t = 0$ ($\alpha = 0$) et un nombre de tâches de maintenance considérable ($\gamma = 0.15$) la figure 3.13 montre que l'heuristique $HS1$ est meilleure que $HS2$ pour la majorité des nombres de travaux.

Par ailleurs, dans le cas où $\alpha = 1$ (les dates de fin au plus tard sont très larges), l'heuristique $HS2$ optimise mieux le coût de maintenance pour un nombre de travaux inférieur à 80. La figure 3.14 illustre bien ce cas.

En conclusion, si on considère les critères de production et de maintenance séparément on peut dire que chaque heuristique optimise un critère mieux qu'un autre. En effet, $HS1$ optimise mieux le coût de maintenance, tandis que $HS2$ semble plus performante en ce

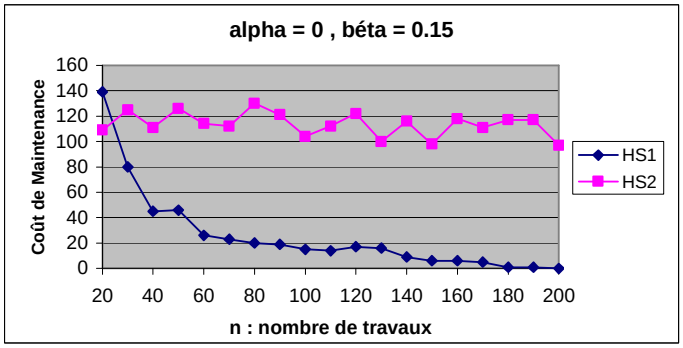


FIG. 3.13 – *Coût de maintenance de HS1 et HS2 en fonction du nombre de travaux n*

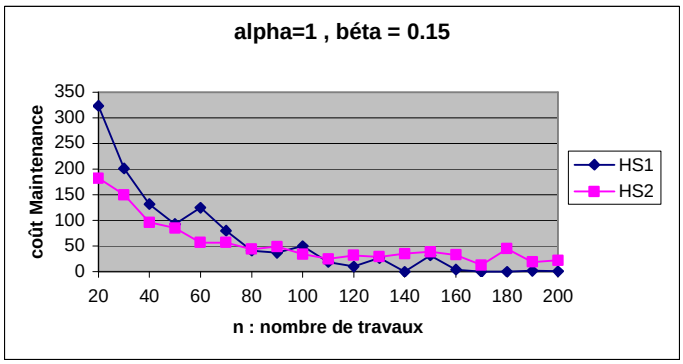


FIG. 3.14 – *Coût de maintenance de HS1 et HS2 en fonction du nombre de travaux n*

qui concerne le coût de production. Toutefois, l'heuristique $HS2$ reste meilleure pour le coût total.

Étude comparative des heuristiques HI1 et HI2 :

Les deux heuristiques HI1 et HI2 ont été programmées à l'aide du langage C et testées sur les mêmes jeux de données utilisés pour le cas séquentiel. L'heuristique HI3 a été ensuite comparée avec HI2 en introduisant la notion de deadline pour les tâches de maintenance.

La figure 3.15 représente les coûts totaux obtenus par les deux heuristiques HI1 et HI2 en fonction du nombre de travaux n. On constate que HI2 améliore HI1 pour la majorité des cas. Cela montre bien l'efficacité de la règle de dominance $PRTTE$ qui tient compte des tâches de maintenance et les place de façon qu'elles ne soient pas trop en avance ou trop en retard. Ce qui évite de faire des interventions inutiles ou bien de retarder la maintenance jusqu'à la panne de l'équipement.

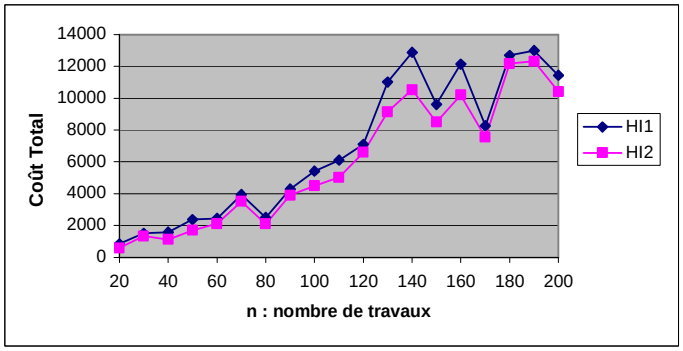


FIG. 3.15 – *Coût Total de HI1 et HI2 en fonction du nombre de travaux n*

Nous introduisons dans le tableau 3.4 les coûts de maintenance obtenus par les trois heuristiques HI1, HI2 et HI3 pour $\alpha = 0$ et $\gamma = 0.15$.

n	HI1	HI2	HI3
20	155	139	137
40	179	179	134
60	193	142	133
80	224	180	159
100	177	143	117
120	221	187	120
140	269	162	164
160	262	176	160
180	291	233	168
200	273	209	173

TAB. 3.4 – *Variation du coût de maintenance*

Nous remarquons bien que HI3 donne le coût de maintenance le moins élevé. En effet, la règle $PRTTE$ permet de distinguer entre les travaux et les tâches de maintenance, ce qui optimise mieux le critère de maintenance et par conséquent le critère total. En ce qui concerne l'ajout de la notion de deadline pour la maintenance, l'heuristique HI3 optimise mieux le critère concernant la maintenance.

La figure 3.16 représente le pourcentage de nombre de fois où HI3 améliore strictement HI2 pour $\alpha = 1$ et $\beta = 0.25$ (les r_i sont diversifiées et d_i sont dispersées). On peut conclure que l'heuristique HI3 améliore le résultat donné par HI2 pour la majorité des problèmes générés. Le pourcentage de nombre de fois où HI3 améliore le résultat donné

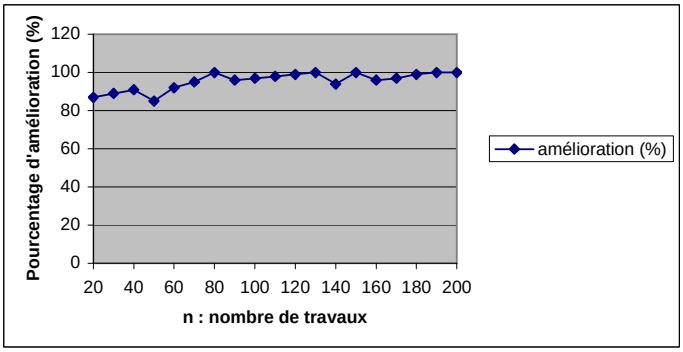


FIG. 3.16 – *Évolution du pourcentage de nombre de fois où HI3 améliore strictement HI2 en fonction du nombre de travaux n*

par l'heuristique HI2 est supérieur à 85% et atteint 100% pour certains problèmes.

Tous ces résultats ont été donnés dans le cas où $\phi_1 = \phi_2 = 1$; c'est à dire qu'on donne les mêmes préférences pour la production et la maintenance. Afin d'analyser l'influence de ces deux variables sur le résultat des heuristiques, on les a testées sur trois ensembles de valeurs de ces variables : $E1 = \{\phi_1 = 0.25,\ \phi_2 = 0.75\}$, $E2 = \{\phi_1 = 0.75,\ \phi_2 = 0.25\}$ et $E3 = \{\phi_1 = 1,\ \phi_2 = 1\}$.

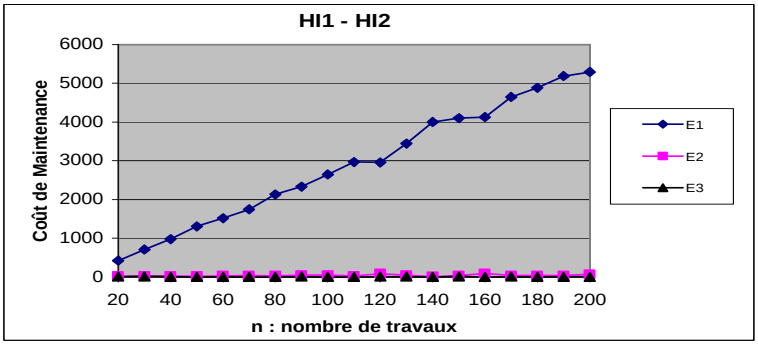


FIG. 3.17 – *Variation de la différence des coût de maintenance en fonction de ϕ_1 et ϕ_2*

La figure 3.17 représente la différence des coûts de maintenance obtenus par HI1 et HI2 en fonction de ϕ_1 et ϕ_2 et pour des travaux disponibles à $t = 0$ ($\alpha = 0$). A partir de cette figure, on peut constater que lorsque $\phi_1 < \phi_2$, HI2 optimise mieux le critère concernant l'aspect maintenance, et devient plus performante lorsque $\phi_1 = 0.25$ et que cette performance augmente quand le nombre de travaux n augmente. Par ailleurs, dans le cas où $\phi_1 \geq \phi_2$ les deux heuristiques donnent des coûts de maintenance assez comparables quoique HI2 reste sensiblement meilleure pour la majorité des nombres de

travaux.

Le but de la troisième heuristique HI3 est de contrôler le retard d'une tâche de maintenance en vue de limiter le risque de panne et par conséquent mieux optimiser le critère concernant la maintenance. Nous reportons dans la figure 3.18 les courbes de variation du coût de maintenance en fonction de ϕ_1 et ϕ_2 pour les heuristiques HI2 et HI3.

Une première constatation est que HI3 optimise mieux le placement des tâches de maintenance. Dans le cas où $\phi_1 < \phi_2$, les deux heuristiques donnent des coûts de maintenance assez comparables. Tandis que lorsque $\phi_1 > \phi_2$, $HI2$ s'éloigne de $HI3$ au fur et à mesure que le nombre de travaux augmente.

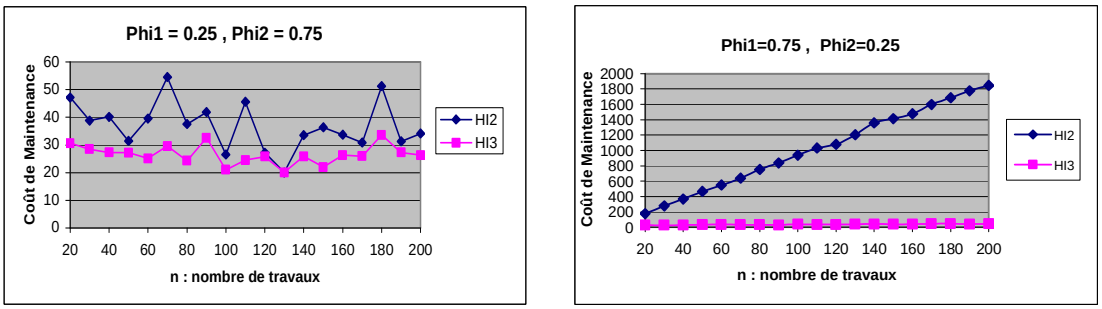


FIG. 3.18 – *variation du coût de maintenance de HI2 et HI3 en fonction de* ϕ_1 *et* ϕ_2

Performance de la borne inférieure LB :

La borne inférieure décrite au paragraphe 3.5.3 a été proposée dans le cas où seul le retard de la maintenance est considéré et où $\phi_1 = \phi_2 = 1$. C'est pour cette raison qu'on ne peut la comparer qu'à l'heuristique HI1. Malheureusement on ne pourra pas l'évaluer par rapport à l'heuristique HI2.

Nous reportons dans le tableau 3.5.3 les résultats de 10 instances d'un problème à 20 travaux avec $\alpha = 0.5$, $\beta = 0.25$ et $\gamma = 0.15$. La première colonne représente le numéro de l'instance testée, la deuxième (resp. troisième) comporte le résultat donné par l'heuristique HI1 (resp. HI2). La valeur de la borne inférieure pour chaque instance est donnée dans la quatrième colonne. Enfin, la cinquième colonne représente la valeur du pourcentage d'erreur de l'heuristique HI2 par rapport à la borne inférieure LB (calculée à l'aide de la formule suivante : $\frac{HI2-LB}{LB} * 100$).

Du tableau 3.5.3, nous constatons que le pourcentage d'erreur de l'heuristique HI2 est assez faible.

Instance	HI1	HI2	LB	$\frac{HI2-LB}{LB} * 100$
1	5189	5189	4696	10.50
2	4658	4621	4187	11.24
3	4903	4866	4268	14.87
4	3917	3886	3553	10.24
5	6653	6586	6007	10.75
6	5995	5936	5209	15.09
7	2961	2951	2795	5.94
8	3990	3970	3620	10.22
9	5584	5584	4665	19.70
10	3930	3920	3413	15.15

TAB. 3.5 – *Pourcentage d'erreur moyen de HI2 par rapport à LB pour* $n = 20$, $\alpha = 0.5$, $\beta = 0.25$ *et* $\gamma = 0.15$

Afin de tester la sensibilité de la borne inférieure à la variation du nombre de travaux, nous avons calculé pour chaque n le pourcentage d'erreur moyen (moyenne sur les 10 instances générées). La figure 3.19 illustre la variation du pourcentage d'erreur moyen en fonction du nombre de travaux n. Nous pouvons bien remarqué que le pourcentage d'erreur moyen n'est pas trop sensible à la variation du nombre de travaux. En effet, ces valeurs sont assez faibles et sont comprises entre 4% et 10 % .

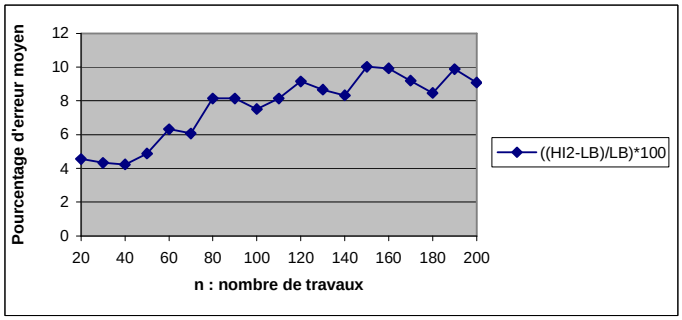


FIG. 3.19 – *Évolution du pourcentage d'erreur moyen de HI2 par rapport à LB en fonction du nombre de travaux n*

3.7 Conclusion

Dans ce chapitre nous nous sommes intéressés au problème d'ordonnancement de la production et de la maintenance sur une machine. Les tâches de maintenance étant supposées flexibles. Le critère d'optimisation prend en compte les deux aspects production et maintenance.

Dans la première partie, nous avons proposé deux heuristiques HS1 et HS2 pour la résolution du cas séquentiel. Le principe de ces dernières est d'ordonnancer les travaux avec une règle de priorité PRTT, puis insérer les tâches de maintenance déjà planifiées. L'étude expérimentale de ces deux heuristiques, montre que HS2 améliore bien le résultat donné par HS1 et ceci grâce à l'énumération de tous les emplacements possibles pour une maintenance et le choix de celui qui donne un coût minimum.

Dans la deuxième partie de ce chapitre, nous nous sommes intéressés au cas de planification intégrée de la production et de la maintenance. Nous avons proposé une règle de dominance locale qui lie un travail et une tâche de maintenance. Cette règle a été utilisée pour construire une heuristique. Dans un souci de comparaison, on proposé une autre heuristique qui assimile une tâche de maintenance à un travail (tâche de production) et ordonnance toutes les opérations avec la règle de priorité PRTT. L'étude expérimentale a mis en valeur la première heuristique essentiellement concernant les performances liées à la maintenance.

Dans un soucis d'amélioration de l'heuristique basée sur la règle de dominance, nous avons introduit la notion de deadline pour une tâche de maintenance. En effet, une intervention préventive ne peut pas être retardée indéfiniment car au delà d'une certaine limite la machine risque de tomber en panne. L'heuristique a été donc modifiée pour prendre en compte les deadlines des tâches de maintenance. Cette modification était efficace pour l'optimisation du coût de la maintenance.

La difficulté du problème nous a incité à proposer des heuristiques. Pour pouvoir tester leur efficacité, nous avons proposé une borne inférieure qui ne tient compte que du retard de la production et de la maintenance. Cette borne s'est avérée assez bonne.

Le cas d'étude d'une machine peut être la base pour résoudre les problèmes à plusieurs machines. Dans les chapitres suivants, nous utilisons la règle de dominance proposée dans l'élaboration de méthodes de résolutions approchées des problèmes à plusieurs machines et plus précisément au problème du flow shop.

Chapitre 4

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

Résumé :*Nous étudions dans ce chapitre le problème d'ordonnancement conjoint de la production et de la maintenance dans un atelier de type flow shop. Le critère d'optimisation considéré est une agrégation de deux critères, l'un lié à la production et l'autre à la maintenance. Nous considérons en premier lieu le cas de flow shop à deux machines et nous proposons un algorithme Branch and Bound qui permet d'obtenir des ordonnancements de permutation. En second lieu, nous nous intéressons au cas du flow shop à plusieurs machines. Pour la résolution, nous développons un algorithme génétique ainsi qu'un codage approprié. Les résultats d'expériences effectuées sur les instances générées selon un schéma particulier, permettent de montrer l'efficacité des approches proposées.*

4.1 Introduction

Dans le chapitre précédent nous avons étudié le problème d'ordonnancement de la production et de la maintenance sur une machine. La difficulté du problème nous a incité à développer des méthodes heuristiques. Une de ces heuristique sera adaptée pour résoudre le problème de flow shop à plusieurs machines ; sujet de ce chapitre.

La première partie de ce chapitre est dédiée au système d'atelier de type Flow Shop à deux machines. Pour la résolution, nous proposons une procédure par séparation et évaluation. Les tâches de maintenance sont supposées flexibles, ce qui signifie que leurs positions et leurs nombres ne sont pas fixés a priori et peuvent être optimisés.

Dans la deuxième partie de ce chapitre, nous considérons le problème d'ordonnancement général de type flow shop dans la cas où une machine subit plusieurs types de maintenance et par conséquent une gamme de maintenance est imposée. Nous proposons pour la résolution un algorithme génétique ainsi qu'un codage approprié. Des simulations sont menées pour attester de l'efficacité des approches proposées.

4.2 Problème d'atelier de type flow shop

Nous étudions dans ce chapitre le problème d'ordonnancement dans un atelier flow shop strictement non-préemptif où les machines sont sujettes à des interventions périodiques de maintenance préventive systématique. Nous supposons que les périodes de maintenance sont connues et flexibles. L'objectif est la minimisation d'une fonction tenant compte des deux aspects production et maintenance. Comme l'indique l'état de l'art dressé dans le deuxième chapitre de la thèse, toutes les études consacrées à la prise en compte de contraintes de disponibilité dans les ateliers de type flow shop sont réduites aux modèles à deux machines. Par ailleurs, la majorité de ces travaux se place dans le contexte où les opérations sont sécables.

4.2.1 Notations et Hypothèses

Pour formuler le problème d'ordonnancement conjoint de la production et de la maintenance dans un atelier de type flow shop, nous allons utiliser les notations données au tableau 4.1. Ces notations nous permettrons de définir et formuler le critère d'optimisation lié à la production et à la maintenance.

Le problème d'ordonnancement de type flow shop strictement non-préemptif avec la prise en compte de périodes de maintenance des machines se définit de la manière

suivante :

- un ensemble de n travaux $J = \{J_1,J_2,\ldots,J_n\}$ doit être réalisé sur m machines $M = \{M_1,M_2,\ldots,M_m\}$;
- chaque travail J_i, composé d'une séquence de m opérations $\{O_{i1},O_{i2},\ldots,O_{im}\}$, visite toutes les machines dans l'ordre $(M_1,M_2,\ldots,M_m)$;
- chaque machine ne peut réaliser qu'une seule opération à la fois et chaque opération nécessite une seule machine à la fois ;
- Les machines sont sujettes à plusieurs types de maintenance et à des périodes différentes. Une gamme de maintenance sera fixée à priori ;
- Les opérations sont strictement non-préemptives, ce qui signifie que l'exécution de toute opération ne peut être interrompue ni par la réalisation d'une intervention de maintenance, ni par celle d'une autre opération.

Afin d'optimiser le placement des tâches de maintenance, nous supposons qu'elles sont flexibles et que leurs dates de début doivent être déterminées durant la procédure d'ordonnancement. Une fenêtre temporelle est donc allouée à l'exécution de chaque tâche. Cette hypothèse se rapproche d'ailleurs de la réalité industrielle, telle que par exemple l'obligation de planifier une tâche de maintenance sur une machine toutes les 100 heures de travail plus ou moins 10 heures. En outre, il est raisonnable de penser que si une machine est libre à un moment donné et si une tâche de maintenance peut s'effectuer à ce moment, il peut être plus intéressant de la réaliser aussitôt que possible, de manière à poursuivre avec une machine dont la probabilité de panne est plus faible. Nous supposons que le nombre de tâches de maintenance n'est pas connu en avance. En effet, ce nombre sera ajusté suivant la longueur de l'horizon de production considéré ; ce qui nous évite de réaliser des interventions de maintenance en plus.

Symbole	Signification
p_{ij}	temps opératoire du travail J_i sur la machine M_j
d_i	date de fin souhaitée du travail J_i
C_{ij}	date d'achèvement du travail J_i sur la machine M_j
N_t	nombre de types de tâches de maintenance
$P^k_{m_j}$ $k = 1,\ldots,N_t$	durée d'une tâche de maintenance de type k sur M_j
$S^k_{m_{lj}}$ $k = 1,\ldots,N_t$	date de début de la $l^{ième}$ maintenance de type k sur M_j
$C^k_{m_{lj}}$ $k = 1,\ldots,N_t$	date d'achèvement de la $l^{ième}$ maintenance de type k sur M_j
T^k_j $k = 1,\ldots,N_t$	période optimale de maintenance de type k sur M_j
ΔT_j	retard/avance toléré pour une tâche de maintenance sur M_j
$T^k_{min_j} = T^k_j - \Delta T_j$	durée minimale entre 2 maintenances consécutives de type k sur M_j
$T^k_{max_j} = T^k_j + \Delta T_j$	durée maximale entre 2 maintenances consécutives de type k sur M_j
Nm_{kj}	nombre de tâches de maintenance de type k sur M_j

TAB. 4.1 – *Notations utilisées*

Étant données les hypothèses et notations présentées ci-dessus, nous donnons dans ce qui suit une formulation du critère d'optimisation considéré dans cette étude. Ce critère tient compte des deux aspects production et maintenance.

4.2.2 Critère d'optimisation

Le but de cette étude est de pouvoir effectuer les interventions de maintenance avec un minimum de retard et dans le cas optimal dans les délais. Ces interventions sont liées entre elles. En effet, l'emplacement d'une tâche de maintenance dépend du placement de celle (de même type) qui la précède. La $l^{ième}$ tâche de maintenance de type k sur M_j effectuée dans les délais vérifie la condition suivante :

$$C^k_{m_{(l-1)j}} + T^k_{min_j} \leq S^k_{m_{lj}} \leq C^k_{m_{(l-1)j}} + T^k_{max_j}$$

Soient

$A^k_{m_{lj}} = \max(0, C^k_{m_{(l-1)j}} + T^k_{min_j} + P^k_{m_j} - C^k_{m_{lj}})$ l'avance de la $l^{ième}$ maintenance de type k sur M_j.

$R^k_{m_{lj}} = \max(0, C^k_{m_{lj}} - T^k_{max_j} - P^k_{m_j} - C^k_{m_{(l-1)j}})$ le retard de la $l^{ième}$ maintenance de type k sur M_j.

Les contraintes imposées par les clients à leurs fournisseurs s'expriment souvent en terme de délai, ce qui nous incite à considérer la somme des retards comme critère de

production. Soit f_p cette fonction objectif définie par :

$$f_p = \sum_{i=1}^{n} \max(0, C_{im} - d_i) \tag{4.1}$$

Du point de vue du fournisseur, le respect des contraintes impose un bon fonctionnement de son système de production. Ceci passe par le respect des périodes de maintenance prévues. Soit f_m la fonction objectif concernant la maintenance. Elle est définie par :

$$f_m = \sum_{j=1}^{m} \sum_{u=1}^{k} \sum_{l=1}^{Nm_{uj}} (A^u_{m_{lj}} + R^u_{m_{lj}}) \tag{4.2}$$

Pour optimiser les deux aspects production et maintenance, on effectue une agrégation des deux critères f_p et f_m. Un ordonnancement conjoint de la production et de la maintenance doit minimiser :

$$f = \phi_1 * f_p + \phi_2 * f_m \tag{4.3}$$

où ϕ_1 et ϕ_2 sont fixées par l'utilisateur suivant le degré d'importance qu'il donne à la production et à la maintenance. Ces coefficients peuvent éventuellement dépendre du nombre de tâches (production et/ou maintenance).

D'après la notation concernant les contraintes de disponibilité des machines introduite par Schmidt [Shm00], le problème d'ordonnancement dans un flow shop à m machines sujettes à des interventions de maintenance peut être noté $Fm, NC_{win-flex} \mid \mid f$.

Le problème d'ordonnancement classique du flow shop, c'est à dire sans indisponibilité des machines, a été démontré $\mathcal{NP}$-*difficile* au sens fort pour le critère somme des retards par Lenstra et *al.* [LRB77] pour un nombre de machines supérieur ou égal à deux. Il en résulte que le problème que nous étudions ici est également $\mathcal{NP}$-*difficile* au sens fort. Ce résultat nous a incité à élaborer essentiellement des méthodes de résolution approchées, mais également un algorithme Branch and Bound limité en taille de problèmes résolus.

La première partie de ce chapitre sera dédiée au cas du flow shop à deux machines. Nous y développerons une procédure par séparation et évaluation pour le cas de flow shop de permutation.

4.3 Flow shop à deux machines

Le problème d'ordonnancement classique du flow shop, c'est à dire sans considération des périodes de maintenance, a été démontré $\mathcal{NP}$-*difficile* au sens fort pour le critère

somme des retards par Lenstra et *al* [LRB77]. L'introduction des périodes de maintenance des machines rendra le problème plus difficile. Cependant, il est impossible de trouver une méthode optimale et qui fournisse la solution en un temps polynomial. Néanmoins, nous avons pu proposer une procédure par séparation et évaluation (PSE) qui se restreint aux ordonnancements de permutation.

Bien que les ordonnancements de permutation ne sont plus dominants pour le critère somme des retards dès qu'il y a une période de maintenance, la procédure PSE proposée a permis de trouver de bonnes solutions pour des cas de figures bien particuliers. Afin de simplifier l'exposé du problème, nous supposons qu'une machine subit un seul type de maintenance. Cependant, la méthode peut être généralisée au cas de plusieurs types de maintenance.

Nous noterons pour le cas d'un seul type de maintenance : $P^1_{m_j} = P_{m_j}$, $S^1_{m_{kj}} = S_{m_{kj}}$, $C^1_{m_{kj}} = C_{m_{kj}}$, $T^1_j = T_j$, $T^1_{min_j} = T_{min_j} = T_j - \Delta T_j$, $T^1_{max_j} = T_{max_j} = T_j + \Delta T_j$ et $Nm_{kj} = Nm_j$.

4.3.1 Méthode de résolution proposée pour le problème de permutation

Nous développons dans cette section une procédure PSE permettant de générer des ordonnancements de permutation pour le problème d'ordonnancement conjoint de la production et de la maintenance dans un flow shop à deux machines. Nous commençons par donner une formulation de la borne inférieure utilisée pour l'évaluation des différents noeuds de l'arborescence.

4.3.1.1 Evaluation des noeuds

Borne inférieure Malgré la difficulté du critère d'optimisation étudié, nous avons proposé une borne inférieure. Cette borne est une adaptation d'une borne inférieure proposée par Pan et *al.*[PCC02] dans le cas du flow shop de permutation classique. Elle est obtenue en considérant les dates de fin des travaux de l'ordonnancement partiel et l'ensemble des travaux non encore ordonnancés. Pour illustrer cette borne, nous introduisons les notations suivantes :

- $P_{i,j}$: somme des i plus petites durées opératoires sur M_j, $j = 1, \ldots, m$;
- $p_{[1],j}$: plus petite durée opératoire sur M_j, $j = 1, \ldots, m$;
- S_1 : ordonnancement obtenu par l'algorithme EDD ;

- S_2 : ordonnancement obtenu par l'algorithme SPT sur M_1 ;
- $d_i(S_1)$: date de fin au plus tard du $i^{ème}$ travail de S_1 ;
- $C_i(Q)$: date de fin du travail J_i dans l'ordonnancement Q ;
- $d_i(Q)$: date de fin au plus tard du travail placé à la position i dans Q ;
- σ : sous-ensemble des travaux déjà ordonnancés ;
- $C_{\sigma j}(Q)$: date de fin du dernier travail placé dans Q (Q formé par les travaux de σ) sur M_j, $j = 1,\ldots,m$;
- $p_{i,2}(Q)$: durée opératoire sur M_2 du travail placé à la $i^{ème}$ position dans Q ;
- x^+ : max(x,0) ;
- λ_k : sous ensemble de travaux qu'on peut insérer entre la $l^{ième}$ et la $(l+1)^{ième}$ maintenance.

Le nombre minimum de tâches de maintenance effectuées avant le $i^{ème}$ travail placé sur M_1 est donné par :

$$K_{i,1} = \Big\lfloor \frac{P_{i,1} - 1}{T_{min_1} + P_{m_1}} \Big\rfloor$$

Le nombre minimum de tâches de maintenance effectuées avant le $i^{ème}$ travail placé sur M_2 est donné par :

$$K_{i,2} = \Big\lfloor \frac{P_{i,2} + p_{[1],1} - 1}{T_{min_2} + P_{m_2}} \Big\rfloor$$

Soient les deux lemmes suivants:

Lemme 1. *[Kim93] Dans un flow shop à deux machines et n travaux, la date de fin du travail placé à la $i^{ème}$ position d'un ordonnancement Q est au moins égale à*

$$\max\{P_{i,1} + p_{[1],2}, P_{i,2} + p_{[1],1}\}$$

Lemme 2. *[PF97] Dans un flow shop à deux machines, la somme des retards de tous les travaux est au moins égale à*

$$\sum_{i=1}^{n} \Big(\max\{P_{i,1} + p_{k,2}(S_2), P_{i,2} + p_{[1],1}\} - d_i(S_1) \Big)^+$$

où k commence de 1 et si $P_{i,1} + p_{k,2}(S_2) - d_i(S_1) > 0$ alors $p_{k,2}(S_2)$ sera remplacé par $p_{k+1,2}(S_2)$.

En adaptant le lemme 1 pour prendre en compte les tâches de maintenance, la date de fin du travail placé à la $i^{ème}$ position d'un ordonnancement Q est au moins égale à :

$$\max\{C_{\sigma 1}(Q) + P_{i,1} + p_{[1],2} + K_{i,1} * P_{m_1}, C_{\sigma 2}(Q) P_{i,2} + K_{i,2} * P_{m_2}\} \tag{4.4}$$

En adaptant le lemme 2 et en utilisant l'equation 4.4, la somme des retards des travaux non encore ordonnancés est au moins égale à :

$$\sum_{i \notin \sigma} \Big(\max \big\{ C_{\sigma 1}(Q) + P_{i1} + p_{k,2}(S_2) + K_{i,1} * P_{m_1}, C_{\sigma 2}(Q) + P_{i2} + K_{i,2} * P_{m_2} \big\} - d_i(S_1) \Big)^+ \tag{4.5}$$

La borne inférieure de la somme des retards des travaux déjà ordonnancés dans Q est égale à :

$$\sum_{i \in \sigma} \Big(C_i(Q) - d_i(Q) \Big)^+ \tag{4.6}$$

La borne inférieure est alors la somme de 4.5 et 4.6. Elle est donnée par :

$$BI = \sum_{i \in \sigma} \Big(C_i(Q) - d_i(Q) \Big)^+ + \sum_{i \notin \sigma} \Big(\max \big\{ C_{\sigma 1}(Q) + P_{i1} + p_{k,2}(S_2) + K_{i,1} * P_{m_1}, \\ C_{\sigma 2}(Q) + P_{i2} + K_{i,2} * P_{m_2} \big\} - d_i(S_1) \Big)^+$$

où $p_{k+1,2}(S_2)$ est utilisé à la place de $p_{k,2}(S_2)$ si $\max \big\{ C_{\sigma 1}(Q) + P_{i1} + p_{k,2}(S_2) + K_{i,1} * P^1_{m_1}, C_{\sigma 2}(Q) + P_{i2} + K_{i,2} * P^1_{m_2} \big\} - d_i(S_1) > 0$.

Borne supérieure Avant que ne commence la construction de l'arborescence, une borne supérieure BS est calculée [KVZ03a]. Elle est donnée par l'heuristique $MEDD$ décrite ci-dessous.

La construction des différents ordonnancements s'effectue sur la première machine puis le même ordre sera conservé sur la deuxième machine (flow shop de permutation). Par exemple, considérons une séquence $s = (J_1, J_2, J_3)$, l'heuristique proposée place d'abord la première opération du travail J_1, puis la première opération du travail J_2 et enfin la première opération du travail J_3. Le même ordre sera conservé sur la deuxième machine.

L'heuristique commence par placer à chaque T_{max_j} les Nm_j tâches de maintenance sur chaque machine. Le nombre de tâche de maintenance à effectuer sur chaque machine est donné par :

$$\begin{cases} Nm_1 = \lfloor \sum_{i=1}^{n} p_{i1}/T_1 \rfloor \\ Nm_2 = \lfloor (\sum_{i=1}^{n} p_{i2} + p')/T_2 \rfloor \end{cases}$$

avec p' est la durée opératoire sur la première machine du travail ayant la plus petite date de fin au plus tard.

Ce placement décompose l'horizon de planification de la machine M_1 (resp. M_2) en Nm_1 (resp. Nm_2) sous-intervalles $I_{1,1}$,$I_{1,2}$,...,I_{1,Nm_1} (resp. $I_{2,1}$,$I_{2,2}$,...,I_{2,Nm_2}).

Pour construire un ordonnancement réalisable, l'heuristique proposée (appelée $MEDD$ pour faire allusion à EDD Modifiée avec insertion de la maintenance) procède selon les étapes suivantes : un ordre de priorité provisoire entre les travaux $\{J_1,J_2,\ldots,J_n\}$ est donné par la règle EDD (placer les travaux selon l'ordre croissant des dates de fin au plus tard). Les travaux sont ensuite pris un par un selon l'ordre donné par EDD. A chaque fois qu'un travail J_i est sélectionné, l'heuristique proposée détermine le premier sous-intervalle sur l'horizon de planification de la machine M_1, dans lequel J_i peut être inséré au plus tôt, sans recouvrir une tâche de maintenance ou une opération déjà ordonnancée. Si J_i ne peut pas être placé dans l'intervalle courant sur la machine M_2 alors on cherchera le premier travail parmi ceux qui ne sont pas encore placés et qui peut être placé dans l'intervalle courant de M_2. Si de tels travaux n'existent pas, alors on place J_i dans l'intervalle suivant.

Si parfois le dernier travail dans un intervalle sur M_1 (resp. M_2) donné se termine après T_{min_1} (resp. T_{min_2}), alors on pourra décaler à gauche une tâche de maintenance tout en restant dans son intervalle de tolérance. L'algorithme 4.1 explique mieux le principe de cette heuristique.

Algorithme 4.1 Algorithme de l'heuristique MEDD

Entrée : $TS = \{J_i/i = 1,\ldots,n\}$ *{l'ensemble des travaux à ordonnancer}*
S_1 l'ordre obtenu par la règle EDD
toutes les tâches de maintenance sont placées

Début
$i = 1$; $l_1 = 1$; $l_2 = 1$
$tf_1 = 0$; /*date de fin du dernier travail placé sur M_1
$tf_2 = 0$; /*date de fin du dernier travail placé sur M_2
$tm_1 = T_{max_1} + P_{m_1}$; /*date de fin de la dernière maintenance placée sur M_1
$tm_2 = T_{max_2} + P_{m_2}$; /*date de fin de la dernière maintenance placée sur M_2
Tant que $TS \neq \emptyset$ **faire**
Si $(p_{i1} \leq tm_1 - tf_1)$ **alors**
Si $(p_{i2} \leq tm_2 - tf_2)$ **alors**
placer J_i à la fin de I_{1,l_1} et I_{2,l_2} ;
$tf_1 = tf_1 + p_{i1}$; $tf_2 = \max(tf_1,tf_2) + p_{i2}$;
$TS = TS \setminus \{J_i\}$;
Sinon
$TB = \{J_k \in S_1/p_{k1} \leq tm_1 - tf_1$ *et* $p_{k2} \leq tm_2 - tf_2\}$;
Si $(TB \neq \emptyset)$ **alors**
placer le premier travail J_k de TB à la fin de I_{1,l_1} et I_{2,l_2} ;
$tf_1 = tf_1 + p_{k1}$; $tf_2 = \max(tf_1,tf_2) + p_{k2}$;
$TS = TS \setminus \{J_k\}$;
Sinon
$l_2 = l_2 + 1$;
placer J_i à la fin de I_{1,l_1} et I_{2,l_2};
$tf_1 = tf_1 + p_{i1}$; $tf_2 = tm_2 + p_{i2}$;
$TS = TS \setminus \{J_i\}$;
Fin si
Fin si
Sinon
passer au travail suivant de TS ;
Fin si
Fin tant que
Fin

4.3.1.2 Sélection

On va raisonner sur la première machine seulement car on est dans le cas d'un flow shop de permutation. L'ordre des travaux obtenu sur la première machine sera reporté

sur la deuxième machines.

Chaque noeud de l'arborescence est un vecteur $\sigma_{k,l} = (\lambda_0,\lambda_1,\ldots,\lambda_{Nm_1})$, où λ_j est un lot comportant tous les travaux pouvant s'exécuter avant une tâche de maintenance.

A chaque niveau $k = 0,\ldots,n+1$ de l'arborescence, on cherche tous les travaux (initialement ordonnés selon la règle EDD) qu'on peut insérer à la fin d'un lot λ_j. Notons $E_k = \{J_i \in J / p_{i,1} + \sum_{l\in\lambda_j} p_{l,1} \leq T_{max_1}\}$ l'ensemble des travaux qui peuvent être insérés dans λ_j. Si E_k est vide, c-à-d aucun travail ne peut être affecté au lot λ_j, alors on place une tâche de maintenance et on crée un nouveau lot λ_{j+1}.

Afin de diminuer le nombre de noeuds générés à chaque niveau, on a appliqué les deux théorèmes suivants sur l'ensemble des travaux de E_k. Le théorème 4.1 permet d'éliminer les travaux qui ne peuvent pas être placés au début. Le théorème 4.2 détermine les relations de précédence entre les travaux.

Théorème 4.1. *[PCC02] Pour un travail J_i, si il existe un travail J_j qui satisfait : a) $p_{i,2} \leq p_{j,2}$ b) $p_{i,1} + p_{i,2} \geq p_{j,1} + p_{j,2}$ et c) $p_{i,2} - d_i \leq p_{j,2} - d_j$ alors il existe un ordonnancement optimal tel que J_i n'est pas le premier.*

Théorème 4.2. *[PCC02] Pour un travail J_i, si il existe un travail J_j qui satisfait : a) $p_{i,1} \geq p_{j,1}$ b) $p_{i,2} = p_{j,2}$ et c) $d_i \geq d_j$ alors il existe un ordonnancement optimal tel que J_i est après J_j.*

4.3.1.3 Branchement

Le branchement se fait en créant un nouveau noeud $\sigma_{k,l} = (\lambda_0,\ldots,\lambda_j \cup \{J_i\},\ldots,\lambda_{Nm_1})$, $l = 1,\ldots,|E_k|$. Ce nouveau noeud est crée en insérant le travail J_i à la fin du lot λ_j.

4.3.1.4 Séparation

Lors de la séparation, un noeud dont la valeur dépasse la borne supérieure BS sera éliminé. A chaque niveau k de l'arbre, le noeud qui a la plus petite borne inférieure sera conservé. Nous détaillons dans l'algorithme 4.2 les différentes étapes de la procédure par séparation et évaluation.

Algorithme 4.2 Algorithme de séparation et évaluation

Entrée : $TS = \{J_i / i = 1, \ldots, n\}$ *{l'ensemble des travaux à ordonnancer}*

Début

$k = 1$;

- calculer la borne supérieure BS

Tant que toutes les solutions ne sont pas implicitement explorées **faire**

- déterminer E_k ;

-appliquer les théorèmes 4.1 et 4.2 sur les travaux de E_k. Soit E'_k l'ensemble des travaux obtenu ;

- créer les noeuds $\sigma_{k,l}$, $l = 1, \ldots, |E'_k|$

Pour tous les noeuds $\sigma_{k,l}$ crées **faire**

Si $\sigma_{k,l}$ est une solution complète **alors**

- calculer sa valeur du critère

Sinon

- calculer BI

Si $BI < BS$ **alors**

- placer $\sigma_{k,l}$ dans la liste des noeuds à explorer

Fin si

Fin si

Fin pour

Fin tant que

Fin

Exemple 4.1. *Considérons un problème de six travaux, les durées opératoires p_{ij} et les dates de fin au plus tard sont données dans le tableau 4.2. Concernant la maintenance, $T_1 = 15$, $T_2 = 12$, $\Delta T_1 = \Delta T_2 = 2$, $P_{m_1} = 1$ et $P_{m_2} = 1$.*

J_i	1	2	3	4	5	6
$p_{i,1}$	4	5	4	7	9	6
$p_{i,2}$	4	8	3	3	8	7
d_i	9	14	16	17	21	23

TAB. 4.2 – *Exemple d'application*

Pour cet exemple $Nm_1 = 2$ et $Nm_2 = 3$. La borne supérieure calculée à l'aide de l'heuristique $MEDD$ a pour valeur 54. Le diagramme de Gantt de la solution obtenue par l'algorithme Branch and Bound est donné dans la figure 4.1. La valeur de la fonction objectif est égale à 49. Notons que pour cet exemple nous avons essayé toutes les permutations possibles et il s'est avéré que l'ordonnancement de permutation obtenu est optimal.

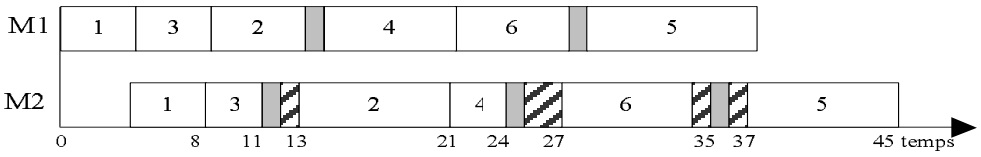


FIG. 4.1 – *Ordonnancement obtenu par l'algorithme de séparation et évaluation*

4.4 Résultats préliminaires

Nous reportons dans le tableau 4.3 les valeurs obtenues par la procédure de séparation et évaluation pour n allant de 10 à 20. La première colonne contient la valeur du critère global, la deuxième donne le temps d'exécution et la troisième contient le nombre de noeuds générés pour chaque problème.

n	PSE	$CPUsecs$	$Noeuds$
10	103	3.81	8718
11	126	4.97	9828
12	265	5.69	15198
13	377	60.84	23549
14	526	199.16	96301
15	613	572.72	116606
16	741	1036.38	2166926
17	801	2176.7	387643
18	1098	2764.9	465987
19	1254	3176.3	501234
20	1357	3425.1	587543

TAB. 4.3 – *Résultats obtenus par la procédure de séparation et évaluation*

Du tableau 4.3, on peut remarquer que le temps d'exécution est raisonnable. Un temps d'exécution est considéré comme raisonnable s'il ne dépasse pas 3600 secondes. Le

nombre de noeuds générés croît sensiblement quand le nombre de travaux augmentent. Ceci explique la limite de cet algorithme en ce qui concerne le nombre maximum de travaux considéré.

Les difficultés rencontrées lors de l'élaboration d'une méthode polynômiale pour le cas d'un flow shop à deux machines nous a orienté vers les méthodes heuristiques pour la résolution du problème à plusieurs machines. Nous passons dans la section suivante à l'étude du problème d'ordonnancement de la production et de la maintenance dans un atelier flow shop à plusieurs machines.

4.5 Flow shop à plusieurs machines

Nous étudions dans cette partie le problème d'ordonnancement de la production et de la maintenance dans un flow shop à plusieurs machines [KVZ04]. Nous supposons qu'une machine subit plusieurs types de maintenance. Cette hypothèse nous parait proche de la réalité industrielle, une gamme de maintenance est souvent fixée pour chaque machine.

4.5.1 Méthode de résolution

Les algorithmes génétiques, décrits en détail dans le premier chapitre de cette thèse, sont connus pour leur capacité à fournir de bonnes solutions aux problèmes d'optimisation combinatoire $\mathcal{NP}$-*difficiles*, en un temps relativement court. En particulier, ils ont prouvé leur efficacité dans la résolution de problèmes d'ordonnancement d'ateliers, notamment celui du Flow Shop (Reeves [Ree95]). Pour la résolution du problème d'ordonnancement de la production et de la maintenance dans un flow shop à plusieurs machines, nous allons proposer un algorithme génétique ainsi qu'un nouveau codage approprié à la maintenance. Nous détaillons par la suite les éléments de l'algorithme génétique mis en oeuvre.

4.5.1.1 Codage des solutions

Le codage est une étape importante dans l'élaboration d'un algorithme génétique. Dans nos premiers tests nous avons utilisé un codage proposé par [Har03] pour le cas de job shop. C'est un codage réel indirect basé sur les travaux. Un chromosome est représenté par une chaîne de $n \times m$ entiers, où n est le nombre de travaux et m est le nombre de machines. Chaque entier indique une opération spécifique d'un travail. Dans ce type de codage, une tâche de maintenance est assimilée à un travail. L'inconvénient d'une telle méthode est qu'on perd les spécificités des tâches de maintenance. En effet, la diversité

des tâches de maintenance exige que leurs codes soient différents. C'est pour cette raison qu'on a proposé un nouveau codage qui tient compte des propriétés intrinsèques à la maintenance.

Nous proposons alors un codage direct basé sur les machines. La procédure de codage d'une solution est composée de deux étapes. Dans la première un chromosome est une suite de vecteurs dont chaque vecteur est une chaîne de nombres entiers représentant les travaux et les tâches de maintenance. Le nombre de vecteurs est égal au nombre de machines. La deuxième étape extrait les différents lots à effectuer avant une tâche de maintenance. Ainsi un vecteur sera divisé en sous-vecteurs chacun représente un lot de travaux.

Exemple 4.2. *Pour illustrer ce codage, on prend à titre d'exemple un problème de flow shop à* 5 *travaux et* 3 *machines dont chaque machine doit subir* 2 *types de maintenance (une maintenance de chaque type). Une contrainte à respecter est la gamme de maintenance ; pour cet exemple une tâche de maintenance de type* 1 *est toujours suivie d'une tâche de maintenance de type* 2*. Les travaux sont numérotés de* 1 *à* 5 *et les maintenances de* 6 *à* 7*. Nous appliquons ci-dessous les deux étapes de codage.*

1. *Les travaux et les tâches de maintenance sont codés de la façon suivante :*

$$\Big(\underbrace{(1\ \ 2\ \ \boldsymbol{6}\ \ 3\ \ 4\ \ \boldsymbol{7}\ \ 5)}_{M_1}\ \underbrace{(1\ \ 3\ \ 2\ \ \boldsymbol{6}\ \ 4\ \ \boldsymbol{7}\ \ 5)}_{M_2}\ \underbrace{(1\ \ 2\ \ \boldsymbol{6}\ \ 4\ \ 3\ \ \boldsymbol{7}\ \ 5)}_{M_3}\Big)$$

2. *Pour pouvoir appliquer les opérateurs génétiques classiques, nous allons transformer le chromosome obtenu lors de la première étape en ne gardant que les travaux sous la forme suivante :*

$$\Big(\underbrace{\big((1\ \ 2)\ \ (3\ \ 4)\ \ (5)\big)}_{M_1}\ \underbrace{\big((1\ \ 3\ \ 2)\ \ (4)\ \ (5)\big)}_{M_2}\ \underbrace{\big((1\ \ 2)\ \ (4)\ \ (3)\ \ (5)\big)}_{M_3}\Big)$$

ou encore

$$\begin{pmatrix} (1\quad\ 2) & (3\ \ 4) & (5) \\ (1\ \ 3\ \ 2) & (\ 4\ ) & (5) \\ (1\quad\ 2) & (4\ \ 3) & (5) \end{pmatrix}$$

Ainsi, dans chaque ligne et entre deux parenthèses successives, il y a une tâche de maintenance qui sera ensuite identifiée selon la gamme constructive de maintenance.

La deuxième étape de l'algorithme génétique étant la génération de la population initiale. Telle est l'objet du paragraphe suivant.

4.5.1.2 Génération de la population initiale

La population initiale peut être générée aléatoirement, par duplication et évolution ou en s'appuyant sur une heuristique. Dans notre étude nous avons essayé trois façons de génération de la population initiale : une génération aléatoire, duplication d'une solution obtenue par une heuristique et enfin génération par combinaison de deux heuristiques. Les tests préliminaires effectués nous ont montré que la troisième méthode donne la meilleure solution et en un temps raisonnable. Dans la suite de notre travail nous utilisons cette dernière pour la génération de la population initiale.

L'algorithme génétique proposé utilise une population générée par une combinaison de deux heuristiques. La première est une adaptation de HI2 proposée pour le cas d'une machine et décrite dans le troisième chapitre. La deuxième utilise la solution donnée par la première heuristique comme solution de départ.

Pour pouvoir adapter HI2 au cas du flow shop, on doit définir pour chaque opération o_{ij} du travail J_i sa date de début au plus tôt r_{ij} et sa date de fin au plus tard d_{ij}. Ces deux variables sont données ci-dessous.

$$\begin{cases} r_{i1} = 0 \\ r_{ij} = \sum_{k=1}^{j-1} p_{ik} \ \forall\ j = 2,\ldots,m \end{cases} \qquad \begin{cases} d_{im} = d_i \\ d_{ij} = d_i - \sum_{k=j+1}^{m} p_{ik} \ \forall\ j = 1,\ldots,m-1 \end{cases}$$

L'adaptation de HI2 au cas du flow shop consiste à appliquer HI2 machine par machine tout en respectant les contraintes de précédence entre les opérations d'un même travail. Nous obtenons à la fin une solution (chromosome) réalisable du problème. A partir de cette solution on va construire la population initiale de l'algorithme génétique. Notons S cette solution.

La duplication du chromosome obtenu peut entraîner une convergence prématurée de l'algorithme génétique vers un optimum local. C'est pour cette raison qu'on a voulu diversifier un peu la population initiale. Cette diversification peut être réaliser par l'application de plusieurs descentes stochastiques sur S et conserver à chaque pas le meilleur trouvé. Un voisin d'une solution est obtenu par insertion et décalage de deux gènes. Nous décrivons mieux ce principe dans la section 4.5.1.4 concernant l'opérateur de mutation. La population finale sera ensuite complétée par duplication des chromosomes obtenus lors des descentes. L'algorithme 4.3illustre la méthode de construction de la population initiale.

Algorithme 4.3 Algorithme de construction de la population initiale

Entrée : S solution obtenue par l'adaptation de HI2

$cpt = 0$, $palier = 0$, $meilleur = f(S)$, $NB = 0$, $ok = 0$;
Tant que ($cpt < seuil$) **faire**
$iter = 0$;
Tant que ($iter < 10$ && $ok = 0$) **faire**
S'= mutation(S) ;
Si ($f(S') \leq meilleur$) **alors**
$meilleur = f(S')$;
ajouter S' à la population ;
$NB = NB + 1$;
$ok = 1$;
Sinon
$iter = iter + 1$;
S = mutation-inverse(S') ;
Fin si
Fin tant que
Fin tant que
Si ($NB <$ taille-population) **alors**
compléter la population par duplication des solutions S' obtenues lors des descentes ;
Fin si
Fin

4.5.1.3 Fonction d'évaluation

Afin de mesurer la force des individus, nous avons utilisé la fonction d'évaluation suivante : $F(x) = CM - f(x) + 1$ où CM représente la plus grande valeur de f 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 f) 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.

4.5.1.4 Opérateurs génétiques

Les opérateurs génétiques nécessaires pour l'évolution des populations et la génération de nouvelles solutions sont la sélection, le croisement et la mutation.

L'opérateur de sélection Pour la mise au point de notre algorithme génétique, nous avons choisi la procédure de *sélection* de Goldberg [Gol89] définie dans le chapitre 1. 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, la sélection choisie maintient la diversité de la population en laissant une chance aux individus faibles pour être sélectionnés. Afin d'éviter que les meilleurs individus soient altérés, nous avons utilisé la reconduite explicite de l'élite dans une proportion fixée de la population. Ainsi, pour notre problème, les 15% meilleurs , de la population sont directement reproduits à l'identique, l'opérateur de selection ne jouant alors que sur les 85% restants.

L'opérateur de croisement Il permet l'exploration de l'espace de recherche en créant de nouveaux individus et est essentiel pour la convergence de l'algorithme génétique. Afin d'éviter la génération de solutions non admissibles, le croisement des solutions de notre problème de flow shop nécessite une adaptation au codage proposé. Nous avons alors adapté le *codage d'ordre maximal* proposé par Falkenauer et Bouffouix [FB91] et qui a pour objectif de garder le maximum possible l'ordre des gènes.

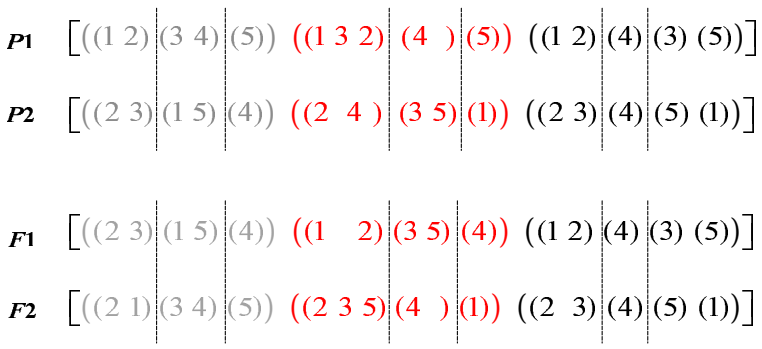


FIG. 4.2 – *Croisement d'ordre maximal de deux individus* $P1$ *et* $P2$

On croise les individus machine par machine. On commence par choisir aléatoirement deux points de coupure (un point de coupure est une position entre deux parenthèses successives), puis les parenthèses médianes sont interchangées. Les gènes manquants

sont par la suite complétés à partir de chaque parent en allant de gauche à droite et en choisissant le premier gène disponible. La cardinalité des parenthèses du parent1 (resp. parent2) est conservée dans le fils2 (resp. fils1). La figure 4.2 représente un exemple d'application de cet opérateur de croisement.

L'opérateur de mutation 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-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 4.3 illustre le fonctionnement de cette mutation.

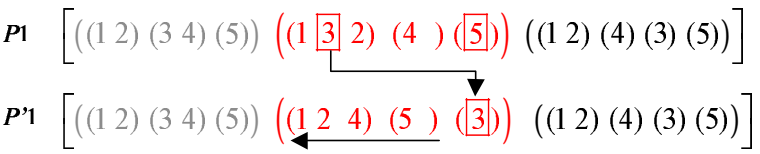


FIG. 4.3 – *Mutation d'un chromosome par insertion-décalage*

Paramètres de l'algorithme génétique 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 de trouver une solution réalisant cette valeur optimale. De même si on connaît une borne inférieure de la solution optimale. L'algorithme s'arrête dès qu'il trouve une solution dont la valeur du critère est la plus proche possible de la borne. Si aucune information n'est connue à priori, le critère d'arrêt peut être un nombre fixe d'itérations.

Pour notre cas, nous avons utilisé un nombre fixe d'itérations comme condition d'arrêt.

Comme nous l'avons présenté dans le chapitre 1, l'efficacité des algorithmes génétiques dépend étroitement du réglage des différents paramètres à fixer. Ces paramètres sont les probabilités de croisement et de mutation, le nombre des générations et la taille de la population.

Les deux derniers paramètres dépendent étroitement de la taille du problème et de la

qualité de la solution souhaitée. Généralement, il faut chercher à établir un compromis entre le temps de calcul et la qualité de la solution. Il est clair que le choix d'une population de taille importante et d'un nombre élevé de générations augmente les chances d'avoir une bonne solution, mais en contre partie, un tel choix engendre un temps de calcul excessif. De plus la taille de la population est limitée par la mémoire du calculateur utilisé. Concernant nos simulations, nous avons fixé la taille des populations entre 100 et 300 individus suivant la taille du problème traité. De plus, des tests préliminaires nous ont permis de fixer le nombre maximal des générations à $NG_{max} = 1000$. Dans la figure 4.4, nous pouvons voir les courbes de convergence pour trois problèmes de tailles différentes et pour une taille de la population initiale égale à 300. Ces trois problèmes sont $Pb1 = \{m = 2, n = 20\}$, $Pb2 = \{m = 4, n = 40\}$ et $Pb3 = \{m = 6, n = 20\}$. Ces courbes représentent la valeur du critère global en fonction du nombre de générations.

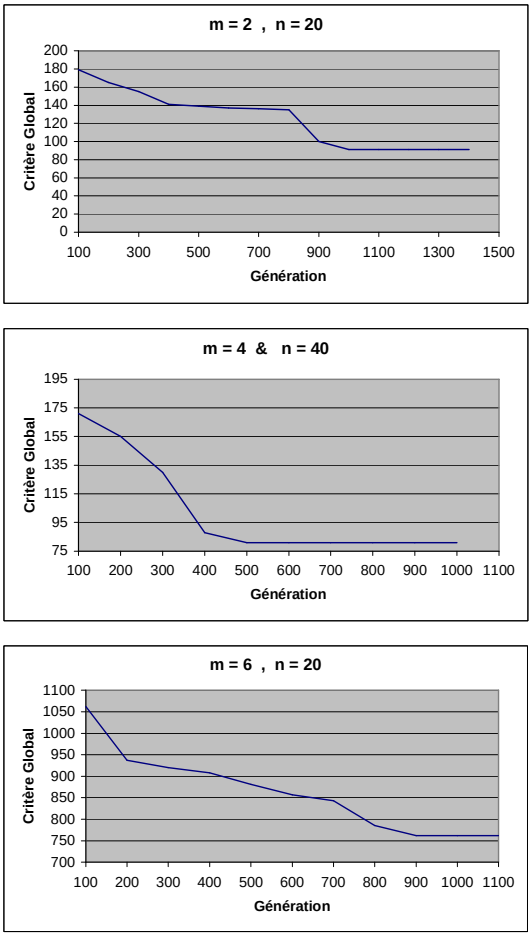


FIG. 4.4 – *Courbes de convergence*

Concernant les probabilités de mutation et de croisement, la littérature montre que le choix de leurs valeurs est d'une importance capitale pour assurer l'efficacité des algorithmes génétiques. Le survol des travaux utilisant ces techniques comme outils d'optimisation met en évidence ce qu'on appelle "l'effet perturbateur" de l'opérateur de mutation sur la convergence. En effet, la mutation joue un rôle perturbateur dans le sens qu'elle dégénère la qualité des individus y opérant. Il s'en suit qu'un taux élevé de mutation ralentit fortement la convergence de la population car on aura un effet d'oscillation aléatoire de la qualité des solutions. D'autre part, un taux insuffisant de mutation présente le risque d'avoir un minimum local de mauvaise qualité comme solution finale. Compte tenu de ces remarques, nous avons opté pour un choix raisonnable pour ces paramètres. Ainsi, nous avons appliqué les taux de mutation et de croisement suivants : $p_m = 0.09$ et $p_c = 0.8$. Ce choix a été motivé par des expérimentations préliminaires sur quelques instances pour le problème abordé.

4.6 Étude expérimentale

Le problème de minimisation de la somme des retards dans un flow shop est souvent limité au cas de deux machines et traité sans contraintes d'indisponibilité ou de maintenance. L'absence de benchmarks concernant les deux aspects production et maintenance nous a incité à développer un schéma de génération des données de production et de maintenance. Ce schéma est une adaptation au cas de la maintenance des schémas proposés par Xu et *al.* [XTS03] et Lee et Chen [LC00].

4.6.1 Schéma de génération des données

Supposons que la gamme constructive de maintenance sur chaque machine est la suivante $\{1\ 2\ \ldots\ N_t\}$. Cela signifie que par exemple une maintenance de type 1 sera toujours suivie d'une maintenance de type 2 et ainsi de suite.

- le nombre de jobs ainsi que le nombre de machines prennent leurs valeurs dans les ensembles suivants : $n \in \{20, \ldots, 100\}$ et $m \in \{2,4,6,8\}$;
- les durées opératoires p_{ij} sont uniformément distribuées entre 1 et 100 ;
- les dates de fin au plus tard d_i sont uniformément distribuées entre $S*(1-\tau-R/2)$ et $S*(1-\tau+R/2)$, où

$$S = \max\left(\max_{1\leq j\leq m} \left\{ \sum_{i=1}^{n} p_{ij} + \min_{i} \sum_{l=1}^{j-1} p_{il} + \min_{i} \sum_{l=j+1}^{m} p_{il} + \sum_{k=1}^{N_t} \left(\frac{E(\sum_{i=1}^{n} p_{ij}/T_j)}{N_t + k} + 1\right) * P_{mj}^{k} \right\},\right.$$

$$\max_i \left\{ \sum_{j=i}^{m} p_{ij} + \sum_{k=1}^{N_t} \left(\frac{E(\sum_{j=i}^{m} p_{ij}/T_j)}{N_t + k} + 1 \right) * P_{mi}^k \right\} \Bigg)$$

avec R et τ sont deux paramètres réglant la dispersion des dates de fin au plus tard (Due Date Range) et le retard moyen (Tardiness Factor). R (resp. τ) prennent leurs valeurs dans $\{0.6,\ 1.2\}$ (resp. $\{0.2,\ 0.4\}$). Dix instances de problèmes ont été générées pour chacune des 4 combinaisons possibles des paramètres R et τ.

- les durées de maintenance sont déterminées comme suit :

  $$P_{mj}^k = k * \sum_{i=1}^{n} p_{ij}/n$$

  où k est tiré aléatoirement de l'ensemble $\{1,2,\ldots,N_t\}$ et $j = 1,\ldots,m$

- on suppose que les périodes de maintenance T_j^k sont multiples de la plus petite période (notée T_j). Elles sont données par la formule suivante :

  $$T_j^k = k * T_j = k * \left(\beta * \sum_{i=1}^{n} (p_{ij} + \sum_{l=1}^{j} \min_i p_{il})/m + \max_k P_{mj}^k \right)$$

  avec $k \in \{1,2,\ldots,N_t\}$,$\beta \in \{0.2,\ 0.4,\ 0.8\}$ et $j = 1,\ldots,m$

- le retard/avance toléré pour une maintenance de type k est :

  $$\Delta T_j = 0.02 * \min_t T_j^k$$

Les différentes méthodes proposées ont été programmées en langage C et testées sur les instances générées selon le schéma décrit ci-dessus. Nous allons dans ce qui suit donner une étude comparative des différentes méthodes proposées.

4.6.2 Étude comparative pour le cas de deux machines

Pour la résolution du cas de flow shop à deux machines, nous avons proposé une procédure par séparation et évaluation. Ce problème est $\mathcal{NP}$-*difficile* au sens fort. Afin de réduire la combinatoire de l'espace de recherche, nous nous restreignons au cas d'un flow shop de permutation.

Dans un premier temps, nous avons essayé de tester notre approche en la comparant à d'autres méthodes. A notre connaissance, aucun des travaux traitant les problèmes de flow shop avec prise en compte de la maintenance ne considère la somme des retards de la production et/ou de la maintenance comme critère d'optimisation. Dans cette section nous allons comparer les résultats obtenus par la procédure par séparation et évaluation avec ceux donnés par l'algorithme génétique proposé pour le cas de plusieurs machines.

Afin d'évaluer l'efficacité de l'algorithme de séparation et évaluation proposé, nous l'avons testé sur quatre problèmes différents en faisant varier les paramètres τ et R. Les deux premiers problèmes ($Pb1$ et $Pb2$) correspondent à $\tau = 0.2$ et les deux derniers ($Pb3$ et $Pb4$) à $\tau = 0.4$. Dans le tableau 4.4 nous reportons les résultats obtenus où pour chaque problème on donne la moyenne sur les dix instances générées pour chaque valeur de τ et R fixées. Les tests numériques effectués ont montré que l'algorithme de séparation et évaluation est limité en nombre de travaux. En effet, ce dernier peut résoudre jusqu'à 20 travaux en un temps raisonnable.

n	Problème	PSE	Algorithme Génétique	Ecart($AG - PSE$)
10	Pb1	127.4	131.6	4.2
	Pb2	108.7	121.19	12.48
	Pb3	504.8	460.5	-44.3
	Pb4	634.5	557.1	-77.4
12	Pb1	270.9	279.7	8.8
	Pb2	45.3	50.7	5.4
	Pb3	658.7	625.7	-33
	Pb4	508.6	487.5	-21.1
14	Pb1	91.5	102.3	10.8
	Pb2	97.9	110.6	12.7
	Pb3	546.2	524.5	-21.7
	Pb4	616	619.4	3.4
16	Pb1	99.2	105.4	6.2
	Pb2	68.4	77.22	8.82
	Pb3	866.1	840.6	-25.5
	Pb4	681.56	635.6	-45.96
18	Pb1	103.4	116	12.6
	Pb2	79.1	87.3	8.2
	Pb3	790	777.7	-12.3
	Pb4	1098.1	1126.9	28.8
20	Pb1	135.7	140.2	4.5
	Pb2	69.7	75.1	5.4
	Pb3	1100	1060.5	-39.5
	Pb4	768	722.1	-45.9

TAB. 4.4 – *Comparaison entre PSE (pour des flow shops de permutation) et Algorithme Génétique (pour des flow shops généraux)*

Du tableau 4.4 nous pouvons remarquer que l'algorithme de séparation et évaluation

donne les meilleurs résultats lorsque le paramètre d'ajustement du retard moyen τ est faible. Par conséquent, les ordonnancements de permutations donnent d'assez bon résultats lorsqu'il y a peu de retard (τ est faible). Pour des valeurs de τ élevées l'algorithme de séparation et évaluation est dominé par l'algorithme génétique. Ceci est du au fait que les algorithmes génétiques fournissent des ordonnancements qui ne sont pas forcement de permutation. C'est une preuve numérique que les ordonnancements de permutation ne sont pas dominants en présence de la maintenance.

Les résultats figurant dans le tableau 4.4 permettent également de montrer l'efficacité de l'algorithme génétique développé pour le cas de plusieurs machines. En effet, pour les instances où la PSE donne la meilleure solution, les écarts relatifs entre celle ci et la solution donnée par l'algorithme génétique n'excèdent pas 13%.

4.6.3 Étude comparative pour le cas de plusieurs machines

Pour la résolution du problème d'ordonnancement conjoint de la production et de la maintenance dans un flow shop à plusieurs machines, nous avons développé un algorithme génétique ainsi qu'un codage approprié à la production et à la maintenance.

Afin de justifier notre choix en ce qui concerne la méthode de génération de la population initiale, nous avons comparé les résultats donnés par deux méthodes différentes. La première est une génération aléatoire et la deuxième est la méthode de descente proposée décrite dans le paragraphe 4.5.1.2. Dans le tableau 4.5 nous reportons les résultats obtenus par l'algorithme génétique en utilisant la génération aléatoire et la descente pour construire la population initiale et pour $\phi_1 = 0.5$ et $\phi_2 = 0.75$. Les deux dernières colonnes représentent les temps d'exécution de l'algorithme génétique en utilisant respectivement la génération aléatoire et la génération par la descente.

Du tableau 4.5 nous pouvons remarquer que la génération par la méthode de descente donne dans la majorité des cas la meilleure solution. En effet, plus le nombre de travaux augmente plus les résultats donnés par la génération aléatoire sont médiocres. Une autre constatation concerne les temps d'exécution. Ces derniers sont équivalents pour les deux méthodes de génération et n'excèdent pas 8 minutes pour le plus grand problème testé. Dans la suite de nos tests, nous utiliserons la méthode de descente pour générer la population initiale de l'algorithme génétique.

En raison de l'absence de méthodes traitant le même problème étudié dans cette section, nous avons testé l'impact de la variation de quelques paramètres décrivant le problème sur la performance de l'algorithme génétique. Ces paramètres sont les coefficients d'agrégation ϕ_1 et ϕ_2 et les périodes de maintenance.

n	m	Gen-aleatoire	Gen-descente	CPU-alea/secs	CPU-desc/secs
20	2	99.9	36.25	39.9	40.82
20	4	334.53	87.38	64.1	69.11
20	6	1106.36	272.51	88.63	89.7
20	8	2242.79	479.16	118.49	120.97
40	2	143.72	43.39	57.97	58.91
40	4	1071.51	150.62	102.24	103.17
40	6	2829.68	354.16	152.43	150.07
40	8	5479.15	591.69	206.7	205.76
60	2	233.53	55.9	91.07	91.49
60	4	1683.09	152.02	174.49	169.01
60	6	4478.74	375.25	249.3	232.41
60	8	7353.77	431.92	315.26	317.08
80	2	306.87	25.44	108.436	113.27
80	4	2039.25	111.58	204.84	215.3
80	6	5672.95	314.28	327.61	323.12
80	8	8154.27	457.89	452.04	448.61
100	2	606.74	67.34	157.93	159.32
100	4	3787.5	191.26	269.75	271.83
100	6	6899.25	305.88	433.8	426.08
100	8	11240.31	397.23	519.49	509.64

TAB. 4.5 – *Résultats de 10 exécutions indépendantes par l'AG pour* $\phi_1 = 0.5$ *et* $\phi_2 = 0.75$

Avant de commencer les tests numériques nous avons normalisé le critère d'optimisation étudié. Dans ce cas $f_p = \frac{1}{n}\sum_{i=1}^{n} \max(0, C_{im} - d_i)$ et
$f_m = \frac{1}{N_T}\sum_{j=1}^{m}\sum_{u=1}^{k}\sum_{l=1}^{Nm_{uj}}(A^u_{m_{lj}} + R^u_{m_{lj}})$, avec $N_T = \sum_{j=1}^{m}\sum_{u=1}^{k} Nm_{uj}$ le nombre total de tâches de maintenance sur toutes les machines. Par cette normalisation les valeurs obtenues sont des moyennes par tâches (production ou maintenance).

Le tableau 4.6 présente la valeur initiale , valeur moyenne et meilleure valeur du critère trouvées par l'algorithme génétique en dix exécutions indépendantes de chaque problème généré.

Nous pouvons en déduire que l'algorithme génétique est relativement stable exceptés quelques cas, en ce sens que la différence entre la mauvaise et la meilleure valeur du critère pour dix exécutions est assez faible.

n	m	initiale	mauvaise	moyenne	meilleure
20	2	101.34	66.63	67.32	63.13
40	2	73.76	61.31	56.59	45.8
60	2	112.23	95.57	77.67	66.24
30	4	215.74	131.62	124.25	118.76
50	4	145.69	109.62	106.52	102.73
70	4	244.91	192.69	187.69	177.75
20	6	376.54	334.26	330.88	327.16
40	6	588.71	513.9	512.08	510.83
60	6	493.84	480.58	467.83	449.72
80	8	605.53	556.59	554.80	550.34
90	8	667.95	631.05	622.90	615.18
100	8	588.25	556.24	550.91	546.38

TAB. 4.6 – *Résultats de 10 exécutions indépendantes par l'AG*

Nous allons maintenant étudier l'influence de la longueur des périodes de maintenance préventives sur les coûts de production et de maintenance. Trois cas d'études seront présentés dans les sections suivantes, à savoir les cas de périodes courtes, moyennes et longues.

4.6.3.1 Cas de Périodes de maintenance courtes

Dans ce cas les interventions de maintenance préventive sont fréquentes ($\beta = 0.2$). Nos tests porteront sur l'impact des coefficients d'agrégation, ϕ_1 et ϕ_2, sur les coûts de production et de maintenance. Nous avons donc fixé le coefficient d'agrégation lié à la production et nous avons varié celui lié à la maintenance.

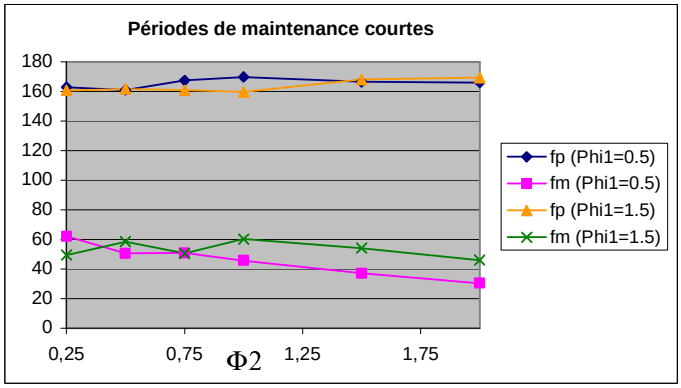


FIG. 4.5 – *Variation des coûts de production et maintenance en fonction de* ϕ_2

D'après la figure 4.5 nous pouvons constater que lorsqu'on donne moins d'importance à la production ($\phi_1 = 0.5$), la courbe de f_p croît légèrement tandis que celle de f_m décroît sensiblement. Le cas opposé, c-à-d lorsqu'on privilégie la production ($\phi_1 = 1.5$), la courbe de f_p croît plus rapidement que dans le premier cas. Quand à la maintenance, son coût est assez stable.

4.6.3.2 Cas de Périodes de maintenance moyennes

Dans ce cas les interventions de maintenance sont peu fréquentes. De la figure 4.6 nous remarquons que le fait de donner plus d'importance à la maintenance accroît f_p et réduit f_m d'une manière considérable. Tandis que dans le cas où le coefficient d'agrégation lié à la production est élevé, les courbes de f_p et f_m retrouvent une certaine stabilité.

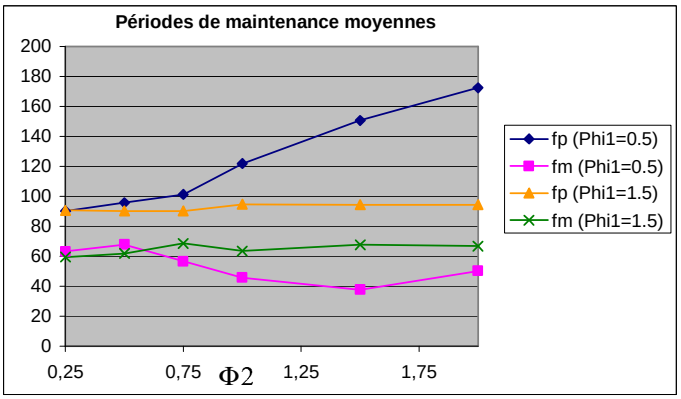


FIG. 4.6 – *Variation des coûts de production et maintenance en fonction de* ϕ_2

4.6.3.3 Cas de Périodes de maintenance longues

Le troisième cas étudié concerne des périodes de maintenance longues. En effet, dans ce cas le nombre d'interventions de maintenance prévues sur l'horizon de production est assez faible. Les mêmes remarques mentionnées dans le cas de périodes de maintenance de longueurs moyennes et pour $\phi_1 = 0.5$ restent vraies. Tandis que lorsque $\phi_1 = 1.5$ la courbe de f_p croît ainsi que celle de f_m décroît sensiblement. Cette différence par rapport au cas précédent est due au nombre d'interventions de maintenance. En effet, lorsqu'on donne plus d'importance à la production, plus on a des tâches de maintenance plus l'équilibre entre les coûts de production et de maintenance est établi.

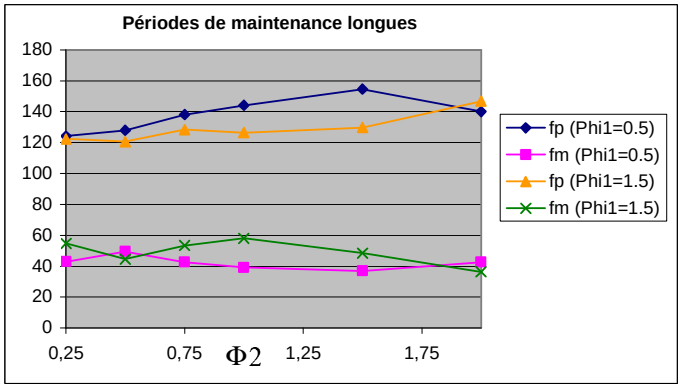


FIG. 4.7 – *Variation des coûts de production et maintenance en fonction de* ϕ_2

En conclusion, selon le nombre d'interventions de maintenance, le réglage des paramètres d'agrégation dépend des priorités qu'on donne à la production et à la maintenance. En effet, dans le cas où le nombre de maintenances est élevé il est préférable de donner plus d'importance à la maintenance. Tandis que dans les cas de périodes moyennes et longues, il est préférable de donner plus d'importance à la production.

4.7 Conclusion

Peu d'entreprises pensent encore aujourd'hui que " la maintenance est un mal nécessaire ". Cependant, peu d'entre elles réalisent que le moindre accroc dans l'efficacité ou la pertinence de la maintenance peut avoir des conséquences indirectes extrêmement préjudiciables pour d'autres fonctions de l'entreprise. Un manque de fiabilité d'une pompe peut générer : des retards de livraison, des pertes de clients, des stocks de produits finis plus importants, des difficultés de trésorerie, des heures supplémentaires, de la fatigue inutile voire même des problèmes de sécurité. L'optimisation des délais et des coûts sont donc 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 présenté dans ce chapitre des méthodes de résolution pour les problèmes d'ordonnancement conjoint de la production et de la maintenance dans un flow shop à deux et à plusieurs machines.

Pour le problème de flow shop à deux machines, nous avons proposé un algorithme de séparation et évaluation qui se restreint aux ordonnancements de permutation. Tandis que pour le cas à plusieurs machines nous avons développé un algorithme génétique ainsi

qu'un codage approprié.

Une étude expérimentale a été menée sur des instances générées selon un schéma proposé. Les résultats obtenus ont montré que l'algorithme de séparation et évaluation n'est efficace que pour des cas bien particuliers et que l'algorithme génétique donne d'assez bonnes solutions. Afin d'aider l'expert à choisir les coefficients d'agrégation, nous avons étudié l'influence de ces coefficients sur les coûts de production et de maintenance et ce selon la fréquence des interventions de maintenance. A partir de ces tests, nous avons constaté que dans le cas où les interventions de maintenance sont très fréquentes il est préférable de donner plus d'importance à la maintenance. Tandis que dans le cas d'interventions moins ou peu fréquentes, le coefficient lié à la production doit être supérieur à celui lié à la maintenance.

Conclusion générale et perspectives

Le contexte de notre travail concerne l'ordonnancement des activités de maintenance préventive systématique dans les systèmes de production. Nous avons considéré deux systèmes de production différents : le cas d'une seule machine et l'atelier de type flow shop. Le critère d'optimisation étudié prend en considération les deux aspects production et maintenance. L'aspect production est qualifié par la somme des retards des travaux. Quand à la maintenance, elle est évaluée par la somme des avances/retards des interventions préventives.

Nous avons pour cela élaboré des méthodes de résolution aussi bien dans le cas d'une seule machine que dans le cas de flow shop à deux et à plusieurs machines. Les règles de priorité et de dominance ainsi que les algorithmes génétiques qui ont fait leur preuve dans le domaine de l'ordonnancement ont été à la base de notre étude.

Notre contribution comporte 3 volets :

- le premier volet prend appui sur une règle de priorité proposée par Chu dans le cas classique d'ordonnancement sur une machine. Nous avons utilisé cette règle conjointement avec une règle de dominance afin de proposer des heuristiques permettant la résolution approchée du problème aussi bien dans le cas d'ordonnancement séquentiel qu'intégré. Nous avons ensuite développé une borne inférieure du critère étudié dans le cas où seul le retard de la maintenance est considéré. Dans un souci de validation, les heuristiques proposées ont été élaborées sur des problèmes générés aléatoirement et comparées entre elles et à la borne inférieure ;
- le deuxième volet propose un algorithme par séparation et évaluation permettant de générer des ordonnancements de permutation du problème conjoint de la production et de la maintenance au sein du flow shop à deux machines. Afin d'accélérer la convergence de cet algorithme, nous avons élaboré une borne inférieure. La complexité du problème initialement étudié rend la recherche d'une solution optimale une tâche très difficile voire impossible. Nous nous sommes donc contentés de chercher des ordonnancements de permutation qui se sont avérés de bonne qualité dans

des conditions particulières ;

– le troisième volet étend l'étude au cas du flow shop à plusieurs machines. Nous avons proposé dans ce cas un algorithme génétique avec un codage approprié des solutions génétiques. Nous avons couplé une des heuristiques proposées dans le cas d'une seule machine avec une descente stochastique pour générer une population initiale hétérogène. Notre algorithme génétique a l'avantage d'explorer largement l'espace de recherche et par conséquent de générer des ordonnancements de très bonne qualité. Dans le cas de flow shop à deux machines, nous avons élaborée une comparaison entre les résultats donnés par l'algorithme génétique et ceux obtenus par la méthode de séparation et évaluation. Pour le flow shop à plusieurs machines, nous avons testé l'efficacité de l'algorithme génétique par une variation de quelques paramètres du problème tels que les degrés d'importance de la production et de la maintenance (figurant dans la fonction objectif). Les résultats expérimentaux ont confirmé les bonnes performances de l'algorithme génétique.

Après la mise au point de trois heuristiques pour résoudre le problème d'ordonnancement conjoint de la production et de la maintenance sur une seule machine, nous nous focaliserons sur la recherche d'une méthode exacte. Ceci en "fouillant" les caractéristiques du problème étudié afin d'extraire des propriétés liées à la maintenance et à la production.

Nous proposons aussi l'élaboration des bornes inférieures du critère étudié dans le cas du flow shop sans relaxation du problème initial. Ces bornes inférieures permettant d'améliorer la borne proposée dans la thèse et par conséquent accélérer la convergence de l'algorithme de séparation et évaluation.

Le cas des problèmes d'ordonnancements statiques reste loin de la réalité industrielle. En effet, l'ordonnancement précis des tâches de maintenance et de production déterminé dans ce cadre est inévitablement perturbé par des aléas de production. En particulier, une politique de maintenance préventive, même très efficace, limite le risque de défaillances mais ne les élimine pas complètement. Il est donc parfois nécessaire de réagir à une panne en exécutant une opération de maintenance corrective le plus tôt possible. Nous proposons dans ce cas d'étendre nos heuristiques ainsi que l'algorithme génétique 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.

Une des contraintes cruciales à ne pas négliger, sera la gestion des ressources humaines de maintenance. En effet, de plus en plus le service maintenance est une composante externe des entreprises. Les équipes de maintenance doivent donc être constituées en fonction des besoins spécifiques à chaque atelier de l'entreprise. Il apparaît alors un problème de gestion des compétences des personnels, couplé au problème de planification des tâches de maintenance.

Pour faire face à une demande de la part des utilisateurs, les constructeurs se dirigent de plus en plus vers des services de maintenance et de contrôle à distance. Ces services sont appelés de télémaintenance coopérative. Dans ce cas le personnel de maintenance peut effectuer son travail à distance (télémaintenance) mais il peut le faire en collaboration avec d'autres experts (travail coopératif). L'accent est alors mis sur l'aspect mobilité des membres coopérants. Cette nouvelle solution permettra d'augmenter, entre autres, la disponibilité, la mobilité, le rendement, la qualité, et de diminuer les coûts occasionnés par les défaillances des équipements.

Bibliographie

[AARE03] H. Allaoui, A. Artiba, F. Riane, and S.E. Elmaghraby. On the two machines flow shop with availability constraints. In *Proceedings for the International Conference on Industrial Engineering and production Management*, Porto, Portugal, mai 26-28 2003.

[ABFK89] I. Adiri, J. Bruno, E. Frostig, and A.H.G. Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26:679–696, 1989.

[AG99] M.S. Akturk and E. Gorgulu. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112:81–97, 1999.

[Agg01] R. Aggoune. Minimizing the makespan for the flow shop scheduling problem with availability constraints. In *Operational Research Peripatetic Post-Graduate Programme, ORP3*, Paris, 2001.

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

[AM98] A. Allahverdi and J. Mittenthal. Dual criteria scheduling on a two-machine flow shop subject to random breakdowns. *Int. Trans. Opl. Res.*, 5(4):317–324, 1998.

[AO00] M.S. Akturk and D. Ozdemir. An exact approach to minimizing total weighted tardiness with release dates. *IIE Transactions*, 32:1091–1101, 2000.

[AO01] M.S. Akturk and D. Ozdemir. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational research*, 135:394–412, 2001.

[ARPW90] T.S. Abdul-Razacq, C.N. Potts, and L.N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26:235–253, 1990.

[AS01] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110:85–99, 2001.

[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.

[Bak74] K.R. Baker. *Introduction to sequencing and scheduling.* John Wiley and Sons, 1974.

[BB82] K.R. Baker and J.W.M. Bertrand. A dynamic priority rule for scheduling against due dates. *Journal of Operations Management*, 3:37–42, 1982.

[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.

[BBF+01] J. Blazewicz, J. Breit, P. Formanowicz, W. Kubiak, and G. Schmidt. Heuristic algorithms for the two-machine flowshop with limited machine availability. *The International Journal of Management Science*, 29:599–608, 2001.

[BDF+00] J. Blazewicz, M. Drozdowski, P. Formanowicz, W. Kubiak, and G. Schmidt. Scheduling preemptable tasks on parallel processors with limited availability. *Parallel Computing*, 26:1195–1211, 2000.

[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.

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

[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.

[BO87] F. Boucly and A. Ogus. *Le management de la maintenance.* AFNOR, Paris, 1987.

[Bou88] A. Boulenger. *Vers le zéro panne avec la maintenance conditionnelle, Guides de l'utilisateur.* AFNOR, Paris, 1988.

[BPH82] J.H. Blackstone, D. Phillips, and G.L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job-shop operations. *International Journal of Production Research*, 20(1):27–45, 1982.

[BPS86] L. Boyer, M. Poirée, and E. Salin. *Précis d'organisation et de gestion de la production*, volume 608 p. Eds. Les Editions d'Organisation, 1986.

[Bre00] J. Breit. *Heuristische ablaufplanungsverfahren fur Flowshops und Openshops mit beschrankt verfugbaren prozessoren.* Ph.d thesis, University of Saarland, Saarbrucken, 2000.

[BSS01] J. Breit, G. Schmidt, and V.A. Strusevich. Two-machine open shop scheduling with an availability constraint. *Operations Research Letters*, 29:65–77, 2001.

[BSS02] J. Breit, G. Schmidt, and V.A. Strusevich. Non-preemptive two-machine open shop with non-availability constraints. In *Project Management and Scheduling*, Valence, Spain, 2002.

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

[CC88] J. Carlier and P. Chrétienne. *Problèmes d'ordonnancement.* Masson, Paris, 1988.

[CC00] Y.L. Chung and Z.L. Chen. Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics*, 47:145–165, 2000.

[CDM91] A. Colorni, M. Dorigo, and V. Maniezzo. An investigation of some properties of an ant algorithm in parallel problem solving from nature 2. Technical report, Manner and B. Manderick eds, North-Holland : Amsterdam, 509-520, 1991.

[Cha87] P. Chapouille. « *Maintenabilité - maintenance* », *Génie Industriel*, volume T4305. Techniques de l'ingénieur, Paris, 1987.

[Chu92] C. Chu. A branch and bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics*, 39:265–283, 1992.

[Chu95] C. Chu. *Ordonnancement de la production: approches théoriques nouvelles.* PhD thesis, Université de Metz, France, 1995.

[Coo71] S.A. Cook. The copmplexity of theorem proving procedures. In Association of Computing Machinery, editor, *Proceedings of the third annual ACM symposium on the theory of computing*, pages 151–158, New York, 1971.

[Cou00] T. Coudert. *Apport des systèmes muti-agents pour la négociation en ordonnancement : application aux fonctions production et maintenance.* PhD thesis, Laboratoire Génie de Production, Ecole Nationale d'Ingénieurs de Trabes, 2000.

[CP89a] J. Carlier and E. Pinson. An algorithm for solving the job shop problem. *Management Science*, 35:164–176, 1989.

[CP89b] C. Chu and M.C. Portmann. Minimizing total tardiness for the one-machine scheduling problem. rapport de recherche 1023, INRIA, 1989.

[CP92] C. Chu and M.C. Portmann. Some new efficient methods to solve the $n|1|r_i|\sum t_i$ scheduling problem. *European Journal of Operational Research*, 58:404–413, 1992.

[CPP95] 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.

[CTV92] F. Della Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. In *3rd International Workshop on PMS*, Como, Italy, 1992.

[CW00] T.C.E. Cheng and G. Wang. An improved heuristic for the two-machine flowshop scheduling with an availability constraint. *Operations Research Letters*, 26:223–229, 2000.

[Dav85] L. Davis. Job-shop scheduling with genetic algorithms. In *1st Int. conf. on Genetic Algorithms and their applications*, pages 136–140, Lawrence Erlbaum (Ed.), Hillsdale, New Jersey, 1985.

[Dav91] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[Den99] S. Deniaud. *Contribution à l'élaboration d'un modèle technico-économique de maintenance préventive. Cas de lignes de fabrication industrielles.* PhD thesis, UFR sciences et techniques de l'univesité de Franche-Comté, ENI de Belfort, 1999.

[DL90] J. Du and J. Y-T. Leung. Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 15(3):483–495, August 1990.

[Emm69] H. Emmons. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17:701–715, 1969.

[FB91] E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 824–829, 1991.

[FG92] S. Fougerousse and J. Germain. *Pratique de la maintenance industrielle par le coût global.* Afnor gestion, 1992.

[Fin99] G. Finke. Ordonnancement. Rapport de cours, INPG, 1999.

[Fog00] D. Fogel. *Evolutionary Computation : Toward a new philosophy of machine intelligence (second edition).* IEEE Press, 2000.

[Gen95] L. Geneste. *Outils d'aide à la décision pour le pilotage d'atelier.* PhD thesis, Université Paul Sabatier en Informatique Industrielle, Laboratoire Génie de Production, 1995.

[Git94] C.W. Gits. Structuring maintenance control systems. *International Journal of Operations and Production Management*, 14, No. 17:5–17, 1994.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability : A guide of the theory of NP-completeness.* W.H. freeman and Company, San Fransisco, 1979.

[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. Rinooy Kan. optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Descrete Mathematics*, 5:287–326, 1979.

[Glo77] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[Glo86] F. Glover. Future paths for integer programming and links to atificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[Gol89] D. Golberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing, 1989.

[GOT93] GOTHA. Les problèmes d'ordonnancement. *RAIRO-Recherche opérationnelle/ Operation research*, 27(1):77–150, 1993.

[GP85] M. Gabriel and Y. Pimor. *Maintenance assistée par ordinateur.* Masson, Paris, 1985.

[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 *In Proceedings of the 3td International Conference on Modelisation and simuling MOSIM*, pages 199–203, Troyes, France, Avril 2001.

[Har03] Y. Harrath. *Contribution à l'ordonnancement conjoint de la production et de la maintenance : application au cas d'un Job Shop.* Thèse de doctorat, Université de Franche Comté, France, 2003.

[Hol75] J. Holland. Adaptation in natural and artifical systems. *(Cambrige, Mass : MIT press*, 1975.

[Jac55] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report 43, Management research project, University of California, Los Angeles, 1955.

[Jou02] A. Jouglet. *Ordonnancer une machine pour minimiser la somme des coûts.* PhD thesis, Laboratoire heudiasyc, Université de Thechnologie de Compiègne, 2002.

[KBF+02] W. Kubiak, J. Blazewicz, P. Formanowicz, J. Breit, and G. Schmidt. Two-machine flow shops with limited machine availability. *European Journal of Operational Research*, 136:528–540, 2002.

[KGV83] S. Kirkpatrick, C.D. Gellatt, and M.P. Vecchi. Optimazation by simulated annealing. *Science*, 220:671–680, 1983.

[Kim93] Y. Kim. A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers & Operations Research*, 20:391–401, 1993.

[KLL75] A.H.G. Rinnooy Kan, B.J. Lageweg, and J.K. Lenstra. Minimizing total cost in one-machine scheduling. *Operations Research*, 23:908–927, 1975.

[KM88] M. Kaspi and B. Montreuil. On the scheduling of identical parallel processes with arbitrary initial processor available time. Rapport de recherche, School of Industrial Engineering, Purdue University, december 1988.

[KVZ02] J. Kaabi, C. Varnier, and N. Zerhouni. Heuristics for scheduling maintenance and production on a single machine. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics, SMC'2002, sur CD ROM, 5 pages*, Hammamet, Tunisie, october 6-9 2002.

[KVZ03a] J. Kaabi, C. Varnier, and N. Zerhouni. ordonnancement de la production et de la maintenance : cas d'un atelier de type flow shop à deux machines. *APII-JESA*, 37:641–660, décembre 2003.

[KVZ03b] J. Kaabi, C. Varnier, and N. Zerhouni. Scheduling with machine availability : An integrated approach. In *Proc.of International conference on industrial Engineering and Production Management, IEPM'03, sur CD ROM, 7 pages*, Porto, Portugal, mai 26-28 2003.

[KVZ04] J. Kaabi, C. Varnier, and N. Zerhouni. Ordonnancement conjoint de la production et de la maintenance dans un flow shop. In *5ème Conférence Francophone de Modélisation et Simulation*, Nantes, France, 1-3 septembre 2004.

[Las93] S. Lash. Genetic algorithms for weighted tardiness scheduling on parallel machines. Technical report 93-01, Department of industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, USA, 1993.

[Law77] E.L. Lawler. A pseudo-polynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[LBB02] T. Lorigeon, J.C. Billaut, and J.L. Bouquard. Availability constraint for a single machine problem with heads and tails. In *Project Management and Scheduling*. Valence, Spain, 2002.

[LC95] W. Li and J. Cao. Stochastic scheduling on a single machine subject to multiple breakdowns according to different probabilities. *Operations Research Letters*, 18:81–91, 1995.

[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.

[Lee91] C.Y. Lee. Parallel machine scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30:53–61, 1991.

[Lee96] C.Y. Lee. Machine scheduling with an availability contraint. *Journal of Global Optimization*, 20:395–416, 1996.

[Lee97] C.Y. Lee. Minimizing the makespan in two-machine flowshop scheduling problem with an availability constraint. *Operations Research Letters*, 20:129–139, 1997.

[Lee99] C.Y. Lee. Two-machine flowshop scheduling problem with availability constraints. *European Journal of Operational research*, 114:420–429, 1999.

[Lim91] S.D. Liman. *Scheduling with capacities and due-dates*. These de doctorat, Industrial and Systems Engineering Department, University of Florida, USA, 1991.

[LK96] K.M. Lee and T. Yama Kawa. A genetic algorithm for general machine scheduling problems. *Int. Journal of Knowlege-Based Electronics Systems*, 2:60–66, 1996.

[LL92] C.Y. Lee and S.D. Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta informatica*, 29:375–382, 1992.

[LL93] C.Y. Lee and S.D. Liman. Capacitated two-parallel machines scheduling to minimize sum of job completion times. *Discrete Applied Mathematics*, 41:211–222, 1993.

[LP93] L. Lu and M.E. Posner. An np-hard open shop scheduling problem with polynomial average time complexity. *Mathematics of Operations Research*, 18:12–38, 1993.

[LRB77] J.K. Lenstra, K.A. Rinnooy, and P. Brucker. Complexity of machine scheduling problems. *Journal of Operational Research Society*, 50:343–362, 1977.

[LS95] Z. Liu and E. Sanlaville. Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Applied Mathematics*, 58:253–280, 1995.

[LYH98] G.H. Lin, E.Y. Yao, and Y. He. Parallel machine scheduling to maximize the minimum load with nonsimultaneous machine available times. *Operations Research Letters*, 22:75–81, 1998.

[Lyo92] P. Lyonnet. *La maintenance : mathématiques & méthodes*. Lavoisier - Paris, 1992.

[Mat96] D.C. Mattfeld. Evolutionary search and the job shop. Investigations on genetic algorithms for production scheduling, Physica Verlag, 1996.

[Mes99] K. Mesghouni. *Application des algorithmes évolutionnistes dans les problèmes d'optimisation en ordonnancement de production*. PhD thesis, Université de Lille1, France, 1999.

[Mül93] H. Mülhenbein. *Evolutionary Algorithms : Theory and Applications*. Wiley, 1993.

[Mon87] F. Monchy. *La fonction Maintenance : formation à la gestion de la maintenance industrielle*. Collection technologies, Masson, Paris, 1987.

[Mon03] F. Monchy. *Maintenance : Méthodes et organisations*. Dunod, Paris, 2003.

[Moo68] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

[Mos94] G. Mosheiov. Minimizing the sum of job completion times on capacitated parallel machines. *Mathl. Comput. Modelling*, 20:91–99, 1994.

[MR95] T.E. Morton and P. Ramnath. Guided forward search in tardiness scheduling of large one machine problems. Technical report, Intelligent Scheduling Systems, Kluwer Academic Publishers, Hingham, MA, 1995.

[Nak89] S. Nakajima. *La Maintenance Productive Totale (TPM)*. afnor gestion, 1989.

[NEH83] M. Nawaz, JR. E.E. Enscore, and I. Ham. *A heuristic algorithm for the m machine, n job flow shop sequencing problem*, volume 11. Omega, 1983.

[Ogu88] A. Ogus. «*Maintenance*», volume A8450. Techniques de l'Ingénieur, Paris, 1988.

[PCC02] J.C-H. Pan, J-S. Chen, and C-M. Chao. Minimizing tardiness in a two-machine flow-shop. *Computers & Operations Research*, 29:869–885, 2002.

[PF97] JCH. Pan and ET. Fan. Two-machine flow-shop scheduling to minimize total tardiness. *International Journal of Systems Science*, 28:405–414, 1997.

[Pin95] M. Pinedo. *Scheduling Theory, Algorithms, and Systems.* Prentice-Hall, 1995.

[PL94] N.M. Paz and W. Leigh. Maintenance scheduling : issues, results and research needs. *International Journal of Operations & Production Management*, 14, No. 8:47–69, 1994.

[Por96] M.C. Portmann. Genetic algorithms and scheduling : a state of the art and some propositions. In *Proceedings of the workshop on production planning and control*, Mons, Belguim, september 9-11 1996.

[QCT99] X. Qi, T. Chen, and F. Tu. Sceduling the maintenance on a single machine. *Journal of the Operational Research Society*, 50:1071–1078, 1999.

[QYB02] X.D. Qi, G. Yin, and J.R. Birge. Single-machine scheduling with random machine breakdowns and randomly compressible processing times. *Stochastic Analysis and Applications*, 18(4):635–653, 2002.

[Rac87] R.M.V. Rachamadugu. A note on weighted tardiness problem. *Operations Research*, 35:450–452, 1987.

[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.

[RDET86] R.S. Russel, E.M. Dar-El, and B.W. Taylor. A comparative analysis of the covert job sequencing rule using various shop. *International Journal of Production research*, 27:1523–1540, 1986.

[Ree95] C.R. Reeves. A genetic algorithm for flow shop sequencing. *Computers and Research*, 22:5–14, 1995.

[Roj97] M.-C. Riff Rojas. *Résolution de problèmes de satisfaction de contraintes avec des algorithmes évolutionnistes.* PhD thesis, Ecole Nationale des Ponts et Chaussées, 1997.

[RS64] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS-9 bis, SEMA, Paris, France, 1964.

[RW88] F. Rodammer and K.P. White. A recent survey of production scheduling. *IEEE transaction on systems, man and cybernetics*, 6(18), 1988.

[Sad02] C. Sadfi. *Problèmes d'ordonnancement avec minimisation des encours.* Thèse de doctorat, Institut National Polytechnique de Grenoble, 2002.

[Sch81] H-P. Schwefel. *Numerical optimization of computer models.* Wiley, Chichester, 1981.

[Shm84] G. Shmidt. Scheduling on semi-identical processors. *Z. Oper. Res.*, 28:153–162, 1984.

[Shm88] G. Shmidt. Scheduling independent tasks with deadlines on semi-identical processors. *Journal of Operational Research Society*, 39:271–277, 1988.

[Shm00] G. Shmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

[Smi56] W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[SS97] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. Rapport de recherche, Université de Saarland, Saarbruken, Allemagne, 1997.

[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.

[Sys89] G. Syswarda. Uniform cross-over in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[TGGP01] E.D. Taillard, L.M. Gambardella, M. Gendreau, and J.Y. Potvin. Adaptive memory programming : a unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16, 2001.

[VM87] A.P.J. Vepsalainen and T.E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33:1035–1047, 1987.

[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.

[Woo97] G.W. Woods. *A hybrid Genetic Algorithm that adapts to binary and real coded operators*. PhD thesis, Department of Computer Science RMIT, 1997.

[WX02] X. Wang and J. Xie. Two-stage flexible flowshop scheduling with limited machine availability. In *The First International Conference on Information and Management Sciences*, Xi'an, China, May 2002.

[WX03] X. Wang and J. Xie. Branch and bound algorithm for flexible flowshop with limited machine availability. *Asian Information-Science-Life*, 1(3), 2003.

[XTS03] Y. Xu, Y. Tian, and N. Sannomiya. Three-stage tabu search for solving large-scale flow shop scheduling problems. *Transactions of Institute of Electrical Engineers of Japan*, 123(3):601–608, 2003.

[Zwi96] G. Zwingelstein. *La maintenance basée sur la fiabilité, Guide pratique d'application de la RCM, série Diagnostic et Maintenance*. Hermès, Paris, 1996.