Ordonnancement d'ateliers sous contraintes de disponibilité des machines

Riad Aggoune

N° d'ordre :

Année 2002

THESE

présentée à

L'UNIVERSITE DE METZ
Mathématiques, Informatique, Mécanique

pour obtenir le grade de

DOCTEUR EN SCIENCES

Spécialité : Automatique (Recherche Opérationnelle)

présentée et soutenue publiquement par

Riad AGGOUNE

Ordonnancement d'Ateliers sous Contraintes de Disponibilité des Machines

le 6 décembre 2002 devant le jury :

Rapporteurs

Abdelhakim Artiba	Professeur aux Facultés Universitaires Catholiques de Mons
Pierre Ladet	Professeur à l'Institut National Polytechnique de Grenoble

Examinateurs

Jean-Charles Billaut	Professeur à l'Université de Tours
Stéphane Dauzère-Pérès	Professeur à l'Ecole des Mines de Nantes
Gauthier Sallet	Professeur à l'Université de Metz
Jos Schaefers	Directeur du LTI au CRP Henri Tudor du Luxembourg

Directeur de Thèse

Marie-Claude Portmann	Professeur à l'Institut National Polytechnique de Lorraine

N° d'ordre : Année 2002

THESE

présentée à

L'UNIVERSITE DE METZ
Mathématiques, Informatique, Mécanique

pour obtenir le grade de

DOCTEUR EN SCIENCES

Spécialité : Automatique (Recherche Opérationnelle)

présentée et soutenue publiquement par

Riad AGGOUNE

Ordonnancement d'Ateliers sous Contraintes de Disponibilité des Machines

le 6 décembre 2002 devant le jury :

Rapporteurs

Abdelhakim Artiba	Professeur aux Facultés Universitaires Catholiques de Mons
Pierre Ladet	Professeur à l'Institut National Polytechnique de Grenoble

Examinateurs

Jean-Charles Billaut	Professeur à l'Université de Tours
Stéphane Dauzère-Pérès	Professeur à l'Ecole des Mines de Nantes
Gauthier Sallet	Professeur à l'Université de Metz
Jos Schaefers	Directeur du LTI au CRP Henri Tudor du Luxembourg

Directeur de Thèse

Marie-Claude Portmann	Professeur à l'Institut National Polytechnique de Lorraine

A Maman, Papa, Woihida, Wassila et Farid

A Marielle et Bilel

Remerciements

Ce travail doctoral a été réalisé au sein de l'équipe Macsi de l'INRIA Lorraine et du Loria, ainsi qu'au Laboratoire des Technologies Industrielles du Centre de Recherche Public Henri Tudor (Luxembourg).

Je tiens tout d'abord à remercier mon directeur de thèse, le Professeur Marie-Claude Portmann, pour toute sa disponibilité, son enthousiasme et ses précieux conseils qui m'ont permis de mener à bien cette recherche. Qu'elle trouve ici toute ma reconnaissance et ma profonde sympathie.

Je tiens également à remercier Monsieur Jos Schaefers, directeur du Laboratoire des Technologies Industrielles, pour toute la confiance et le soutien dont il m'a gratifié durant ces trois années.

Je suis très honoré par la présence dans mon jury de Monsieur Abdelhakim Artiba, Professeur aux Facultés Universitaires Catholiques de Mons (Belgique), qui a accepté de rapporter cette thèse.

Je tiens également à exprimer toute ma gratitude à Monsieur Pierre Ladet, Professeur à l'Institut National Polytechnique de Grenoble, qui m'a fait l'honneur d'évaluer ce travail en tant que rapporteur.

Toute ma reconnaissance s'adresse en outre à Messieurs Jean-Charles Billaut, Professeur à l'Université de Tours, Gauthier Sallet, Professeur à l'Université de Metz, et Stéphane Dauzère-Pérès, Professeur à l'Ecole des Mines de Nantes, qui ont accepté de faire partie de mon jury.

Merci aux membres des équipes Macsi et du LTI, au sein desquelles j'ai pu évoluer durant ces dernières années, dans des ambiances tant agréables que motivantes.

Mes plus vifs remerciements vont à mon ami Yazid auprès de qui j'ai énormément appris. Je le remercie pour tous ses conseils, ses encouragements et toute son amitié.

Enfin, que tous ceux qui ont participé de près ou de loin à l'aboutissement de ce travail et que je n'ai pas cité, trouvent ici l'expression de mes remerciements les plus sincères.

Table des matières

Liste des figures

Chapitre 1 :

Chapitre 2 :

Chapitre 3 :

Chapitre 4 :

Liste des tableaux

Liste des algorithmes

Introduction Générale

De nos jours, toute entreprise est soumise à la concurrence, et ce, quel que soit le secteur économique dont elle dépend. Dans celui de l'industrie, une plus grande compétitivité passe par l'amélioration des rendements en terme de coûts de production et de délais de livraison. Aussi, est-on amené à résoudre des problèmes d'ordonnancement, qui peuvent être définis de la manière suivante : étant donnés un ensemble de tâches à accomplir et un ensemble de ressources permettant leur réalisation, le problème d'ordonnancement consiste à affecter les différentes ressources aux tâches et à déterminer les dates d'exécution de ces tâches, de manière à optimiser une fonction objectif.

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 dates et durées des indisponibilités sont connues dans certains cas : congés de personnel, opérations de maintenance sur les machines d'un atelier, etc. D'autres indisponibilités telles que les pannes de machines ou les absences de personnel, par exemple pour raison médicale, ne sont pas prévisibles.

La présence de ces 'trous' dans un planning de production influe de manière significative sur le processus de fabrication et tout ordonnancement réaliste se doit d'en tenir compte. Une manière de pallier l'indisponibilité d'une ressource est d'affecter sa charge de travail à une ressource de remplacement. Mais une ressource capable d'assurer cette prise en charge, aussi bien en terme de capacité qu'en terme de compétence, n'existe pas forcément.

1. Objectif de la thèse

L'objectif de cette thèse est d'apporter une contribution à l'ordonnancement des ateliers de production avec la prise en compte de contraintes de disponibilité des machines. Les problèmes abordés sont statiques, ce qui signifie que toutes les données sont fixes et connues à l'avance. Par ailleurs, nous considérons uniquement le contexte déterministe de périodes d'indisponibilités dues à des travaux de maintenance sur les machines telles que des opérations de nettoyage, de réglage ou de contrôle. Les modèles d'ateliers que nous étudions sont caractérisés par la présence de machines en un seul exemplaire. En cas d'indisponibilité d'une machine les opérations qui doivent être exécutées sur cette dernière ne peuvent être dirigées vers une machine de remplacement.

Concernant les opérations à ordonnancer, nous supposons tout au long de cette thèse qu'elles sont *strictement non-préemptives*, ce qui signifie que l'exécution d'une opération ne peut être interrompue ni par la réalisation d'une tâche de maintenance ni par celle d'une autre opération.

Plusieurs problèmes d'ordonnancement qui n'ont pas encore été abordés dans la littérature, du moins à notre connaissance, sont traités dans cette thèse. En particulier, les modèles du flow shop à plus de deux machines et celui du job shop, avec des indisponibilités correspondant à des opérations de maintenance préventive sont étudiés. Par ailleurs, deux hypothèses caractérisant l'activité de maintenance sont considérées: soit les tâches de maintenance sont totalement fixes, ce qui est communément étudiés dans la littérature, soit elles sont flexibles et leurs positions peuvent être optimisées au sein de fenêtres temporelles. Nous proposons pour des problèmes d'ordonnancement à deux travaux des résultats de complexité originaux, et développons pour les problèmes généraux des méthodes exactes et approchées de résolution.

2. Organisation du manuscrit

Le manuscrit est organisé de la manière suivante. Le premier chapitre présente, dans un premier temps, des généralités sur les problèmes d'ordonnancement d'ateliers, notamment les modèles classiques de la littérature et les principales méthodes de résolution. Dans un second temps, nous dressons un état de l'art sur les problèmes avec indisponibilités des machines.

Le second chapitre est consacré à l'étude du problème d'ordonnancement du job shop avec seulement deux jobs (ou travaux) à ordonnancer et des contraintes de disponibilité sur les machines. Nous y montrons que le problème de la minimisation du makespan, dans le cas d'opérations strictement non-préemptives, peut être résolu par un algorithme polynomial. Cet algorithme est ensuite étendu à la considération de tout critère régulier, ainsi qu'à la prise en compte de contraintes additionnelles.

Dans le troisième chapitre, nous proposons des méthodes de résolution approchées pour les problèmes généraux, c'est-à-dire à plusieurs jobs, du flow shop et du job shop. Ces méthodes sont pour la plupart basées sur les algorithmes polynomiaux développés au chapitre précédent, ainsi que sur des méthodes de recherche locales. Des expériences numériques permettant de tester et comparer les différentes approches proposées sont ensuite décrites.

Le dernier chapitre de la thèse est dédié à la résolution de manière exacte des problèmes d'ordonnancement du chapitre précédent. En particulier, nous y présentons des méthodes par séparation et évaluation. Les calculs des bornes inférieures sont basés sur des extensions des algorithmes polynomiaux développés pour les problèmes à deux jobs. Des résultats d'expériences sont rapportés en fin de chapitre pour montrer l'efficacité des méthodes exactes, d'une part, et valider les résultats des méthodes heuristiques, d'autre part.

Le manuscrit se termine par nos conclusions et la présentation des différentes perspectives de recherche envisagées.

Chapitre 1 Etat de l'art

Dans ce premier chapitre, nous dressons un état de l'art portant sur les problèmes d'ordonnancement dans les ateliers de production avec la prise en compte de contraintes de disponibilité des machines. Nous proposons, en premier lieu, quelques généralités sur les problèmes d'ordonnancement. Les notions de base ainsi que les modèles classiques de la littérature sont définis. En particulier, nous décrivons les systèmes d'ateliers étudiés dans cette thèse, à savoir ceux du flow shop et du job shop. En second lieu, nous présentons les principales méthodes de résolution des problèmes d'ordonnancement d'atelier, notamment les algorithmes génétiques et la recherche tabou, ainsi que la procédure par séparation et évaluation. Enfin, un état de l'art portant sur les problèmes d'atelier classiques, puis sur l'ordonnancement sous contraintes de disponibilité est proposé.

1. Introduction

Qu'il s'agisse de services ou de production industrielle, toute entreprise subit la pression de la concurrence. La recherche d'une plus grande compétitivité est donc un besoin perpétuel, auquel permet de répondre, du moins en partie, la résolution de problèmes d'ordonnancement. L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de production, qui vise à améliorer l'efficacité d'une entreprise en termes de coûts de production et de délais de livraison. Formellement, un problème d'ordonnancement peut se définir de la manière suivante : étant donnés un ensemble de travaux à accomplir et un ensemble de ressources permettant leur réalisation, le problème d'ordonnancement consiste à déterminer, d'une part l'affectation des ressources aux différentes tâches composant les travaux et, d'autre part, les dates d'exécution de ces tâches, de telle sorte qu'une fonction objectif donnée soit optimisée.

En raison de la diversité de leurs champs d'application, notamment le secteur industriel (Pinedo [1995]) et celui de l'informatique (Blazewicz et al. [1996a]), les problèmes d'ordonnancement ont été largement étudiés depuis cinquante ans et plus particulièrement durant les trois dernières décennies. Aussi, pouvons-nous citer parmi les nombreux ouvrages de référence qui ont été publiés Conway et al. [1967], Baker [1974], Rinnooy Kan [1976], French [1982], Carlier et Chrétienne [1988], Tanaev et al. [1994a] et [1994b], Brucker [1998], Esquirol et Lopez [1999], Lopez et Roubellat [2001].

Dans ce chapitre, nous présentons les problèmes d'ordonnancement rencontrés dans les ateliers de production avec la prise en compte de contraintes de disponibilité sur les machines. Les principaux résultats portant sur la prise en compte de trous stochastiques sont cités, et les études développées dans le contexte d'indisponibilités fixes sont décrites. Cependant, les travaux de recherche proposés dans cette thèse sont focalisés sur les problèmes déterministes et statiques, pour lesquels toutes les données (durées des opérations à réaliser, contraintes de précédence entre ces opérations) sont fixes et connues à l'avance. En particulier, nous supposerons dans les chapitres suivants que les indisponibilités des machines sont dues à des activités de maintenance préventive, telles que des opérations de nettoyage, de réglage ou de contrôle. Les positions des 'trous' (périodes d'indisponibilité) sur chaque machine, ainsi que leurs longueurs (temps d'indisponibilité) seront donc fixées a priori.

Dans la première section de ce chapitre, nous présentons quelques généralités sur les problèmes d'ordonnancement dans les ateliers de production. Nous commençons par décrire les modèles classiques d'atelier, dont ceux du flow shop et du job shop qui servent de modèles de base à notre recherche, puis sont introduites les notations et les modes de représentation des ordonnancements. Enfin, nous définissons la notion de complexité des problèmes d'optimisation combinatoire.

La deuxième section est consacrée aux principales méthodes de résolution des problèmes d'ordonnancement. En particulier, les méthodes approchées et exactes que nous employons par la suite, respectivement les méta-heuristiques et la procédure par séparation et évaluation, sont décrites.

Dans la dernière section de ce chapitre, nous présentons, en premier lieu, les principaux résultats de la littérature concernant les problèmes classiques d'ordonnancement d'ateliers. En second lieu, nous dressons un état de l'art sur les problèmes d'ordonnancement avec la prise en compte de disponibilité des machines.

2. Ordonnancement dans les ateliers de production

2.1. Généralités

Comme l'indique la définition donnée en introduction, les problèmes d'ordonnancement sont caractérisés par plusieurs entités que sont les ressources, les tâches à réaliser et les fonctions objectif.

Les ressources nécessaires à l'exécution de tâches peuvent être de différents types. Une ressource dont la quantité diminue au fur et à mesure de son utilisation est dite consommable. Il convient alors d'en assurer le ré-approvisionnement éventuel. C'est le cas par exemple des matières premières ou des composants dans une entreprise de fabrication ou encore du financement d'un projet. Lorsqu'une ressource utilisée pour la réalisation d'une tâche reste disponible en même quantité (équipe, machine d'un atelier), on parle de ressource renouvelable. Les ressources renouvelables qui ne peuvent exécuter qu'une opération à la fois sont qualifiées de disjonctives (une machine-outils disponible en un exemplaire), celles qui peuvent prendre en charge plusieurs opérations à la fois, telles que les équipes d'ouvriers, sont appelées ressources cumulatives.

Par ailleurs, l'ensemble des ressources est régi par des contraintes portant sur l'utilisation et la disponibilité de chacune d'entre elles. Par exemple, il se peut que la réalisation des différentes tâches nécessite l'utilisation simultanée de plusieurs ressources. On parle dans ce cas de problèmes multi-ressources [Drozdowski 1996].

Les tâches sont définies comme étant des entités élémentaires des différents travaux à réaliser. Elles sont caractérisées par une durée d'exécution et peuvent, selon les problèmes, être réalisées sans interruption (on parle dans ce cas de tâches non-préemptives) ou par morceaux (tâches préemptives).

Deux types de contraintes temporelles caractérisent l'ensemble des tâches dans la plupart des problèmes. D'une part, les contraintes de précédence définissent l'ordre partiel ou total d'exécution des tâches. D'autre part, les contraintes de temps alloué, propres à chacune des tâches, expriment les dates limites de leur exécution. En particulier, une tâche peut avoir une date de disponibilité et une date échue ou date de fin souhaitée.

Les fonctions objectif sont les critères à optimiser qui permettent d'apprécier la qualité de toute solution à un problème d'ordonnancement. Il existe des critères liés à l'utilisation des ressources tels que la charge des machines d'un atelier. D'autres sont liés au temps comme la date d'achèvement de la réalisation des travaux, connue sous le nom de *makespan*. Par ailleurs, on parle de *critère régulier* lorsque l'avancement de l'exécution d'une tâche, sans en retarder d'autres, ne dégrade pas la valeur du critère.

2.2. Problèmes d'ordonnancement d'atelier

Les problèmes d'ordonnancement d'atelier les plus communément étudiés se restreignent au contexte de ressources renouvelables disjonctives. En effet, les ressources sont supposées être des machines ne pouvant exécuter qu'une opération à la fois (le terme de tâche étant plutôt réservé à l'ordonnancement de projet, Demeulemeester et Herroelen. [2001]). Par ailleurs, toute opération nécessite une seule machine déjà affectée pour sa réalisation. Les contraintes de précédence entre les opérations des différents travaux sont soit données par une gamme linéaire représentant l'ordre de fabrication (*flow shop* et *job shop*), soit non fixées (*open shop*). Lorsque l'affectation des machines nécessaires à l'exécution des opérations est fixée a priori, le problème d'ordonnancement d'atelier se réduit à la détermination des dates de début d'exécution des opérations (ou de chaque partie d'opération lorsqu'elles sont morcelables).

Dans ce qui suit, nous décrivons les problèmes classiques d'atelier, notamment ceux du flow shop et du job shop sur lesquels se focalisent les études proposées dans les chapitres suivants.

a) Problèmes à une machine

Les problèmes d'atelier à une machine consistent à ordonnancer des travaux constitués d'une seule opération. Il est évident que si tous les travaux sont disponibles à l'instant zéro et si le critère à minimiser est le makespan, toute séquence de traitement des travaux aboutit à une solution optimale. Cependant, l'ajout de contraintes et la considération d'autres critères rendent le problème plus difficile à résoudre (Carlier [1982]). L'étude des problèmes à une machine présente un intérêt pratique, car il est fréquent dans les ateliers qu'une machine, appelée goulet d'étranglement, constitue un lieu où se concentrent les conflits, les attentes, les synchronisations etc. Par ailleurs, la résolution de problèmes à une machine permet de développer des méthodes pour la résolution de problèmes plus complexes à plusieurs machines (Carlier et Pinson [1989], Brinkkötter et Brucker [2001]).

b) Problèmes à machines parallèles

Les problèmes d'atelier à machines parallèles sont une généralisation des problèmes à une machine qui sont caractérisés par le fait que chaque opération peut être réalisée par n'importe quelle machine mais n'en nécessite qu'une seule. Le problème d'ordonnancement consiste donc à déterminer l'affectation des opérations aux machines puis leurs dates d'exécutions. Selon les caractéristiques des machines, trois types de problèmes à machines parallèles sont considérés dans la littérature :

- Les problèmes à machines identiques pour lesquels les durées opératoires ne dépendent pas des machines.
- Les problèmes à machines uniformes, pour lesquels la durée d'une opération varie uniformément en fonction de la performance de la machine choisie.
- Les problèmes à machines indépendantes, pour lesquels les durées opératoires dépendent complètement des machines employées.

Deux approches sont généralement utilisées pour la résolution des problèmes à machines parallèles. D'une part, les *approches hiérarchiques* qui résolvent le problème d'affectation des opérations aux machines et ensuite les problèmes d'ordonnancement sur chaque machine. D'autre part, les *approches intégrées* qui résolvent simultanément les problèmes d'affectation et de séquencement. Notons que ces dernières sont largement plus répandues dans la littérature que ne le sont les approches hiérarchiques.

c) Problèmes d'atelier à cheminement unique (Flow shop)

Les problèmes d'atelier à cheminement unique sont des problèmes où toutes les machines sont généralement disposées en ligne (en 'I', 'S' ou 'U' par exemple). Les travaux à réaliser sont composés de plusieurs opérations et visitent toutes les machines dans le même ordre. Un cas particulier souvent étudié dans la littérature est le flow shop de permutation. Il se caractérise par le fait que la séquence de passage des travaux est la même sur toutes les machines. Cette contrainte est obligatoire lorsqu'il n'y a pas de place pour stocker les produits entre deux machines consécutives. Un état de l'art portant sur les problèmes d'ordonnancement de type flow shop figure en dernière partie de chapitre.

d) Problèmes d'atelier à cheminements multiples (Job shop)

Le problème d'atelier de type job shop est une généralisation directe de celui du flow shop. En effet, les travaux sont également composés de plusieurs opérations et les gammes de fabrication sont fixées. La différence avec le modèle du flow shop réside dans le fait que l'ordre de passage des travaux sur les machines peut être différent d'un travail à l'autre. Les principaux résultats de la littérature concernant les problèmes d'ordonnancement de type job shop sont présentés dans la dernière partie de ce chapitre.

e) Problèmes d'atelier à cheminements libres (Open shop)

Les problèmes à cheminements libres constituent le troisième type de problèmes d'atelier multi-machines. Ils sont caractérisés par le fait que les gammes de fabrication des différents travaux ne sont pas fixées a 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 littérature.

2.3. Notations

Avant de présenter la manière usuelle de décrire les problèmes d'ordonnancement d'atelier, nous commençons par introduire quelques-unes des notations que nous adoptons dans la suite de ce manuscrit (tableau 1.1).

Tableau 1.1 : Notations

$\mathfrak{I} = \{J_1, J_2, \ldots, J_n\}$	: ensemble des travaux ou jobs à réaliser
n	: nombre de travaux
$M = \{M_1, M_2, \ldots, M_m\}$	: ensemble des machines de l'atelier
m	: nombre de machines
O_{ij}	: $j^{ème}$ opération du job J_i
p_{ij}	: temps opératoire de O_{ij}
r_i	: date de disponibilité du job J_i
d_i	: date de fin souhaitée de J_i
w_i	: coefficient de pondération associé à J_i
S_{ij}	: date de début de O_{ij}
C_{ij}	: date de fin de O_{ij}
C_i	: date de fin de J_i
$C_{max} = \max \{C_i, i = 1,\ldots, n\}$	: date de fin de tous les jobs ou *makespan*

Comme nous l'avons souligné précédemment, un problème d'ordonnancement d'atelier est complètement défini par la donnée des caractéristiques des machines et des travaux, ainsi que par celle des différents critères considérés.

Plusieurs notations des problèmes d'ordonnancement existent dans la littérature, notamment celle introduite par Conway et al. [1967] qui comporte quatre champs A | B | C | D, contenant respectivement les caractéristiques des machines, celles des opérations, les contraintes de précédence et les critères objectifs. Mais cette classification a été conçue pour des problèmes d'ordonnancement de base. Aussi a-t-elle été améliorée par Mac Carthy et Liu [1993], pour la représentation de problèmes plus généraux.

Nous adoptons dans ce qui suit la notation la plus utilisée en ordonnancement, à savoir celle introduite par Graham et al. [1979] qui décrit les problèmes d'ordonnancement en trois champs $\alpha \mid \beta \mid \gamma$.

- Le premier champ $\alpha = \alpha_1\ \alpha_2$ décrit les caractéristiques des machines, où α_1 représente le type d'atelier considéré et α_2 le nombre de machines.
- Le champ β représente l'ensemble des contraintes sur les travaux.
- Le champ γ spécifie la fonction objectif à optimiser.

Par exemple, F2 | r_i | C_{max} désigne le problème de la minimisation du makespan dans un environnement de type flow shop à deux machines, où chaque job J_i ne peut s'exécuter avant sa date de disponibilité r_i.

2.4. Représentations des ordonnancements

Nous présentons dans ce qui suit deux manières de représenter les solutions d'un problème d'ordonnancement d'atelier : le diagramme de Gantt et le graphe disjonctif.

a) Diagramme de Gantt

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 leur temps opératoires. L'axe des abscisses représentant 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.

Considérons le problème J3 | n = 3 | C_{max} avec les gammes opératoires suivantes :

$\mathbf{J_1}$: M_1 (1) M_2 (2) M_3 (2)

$\mathbf{J_2}$: M_2 (2) M_3 (2) M_1 (3)

$\mathbf{J_3}$: M_3 (2) M_1 (3) M_2 (1)

où les nombres entre parenthèses correspondent aux différentes durées opératoires. La figure 1.1 représente une solution réalisable du problème considéré dont le makespan est égal à 8 unités de temps.

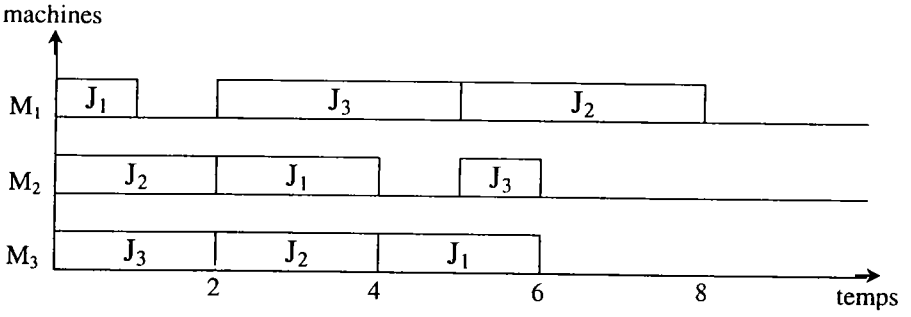


Figure 1.1 : Diagramme de Gantt

b) Graphe disjonctif

Contrairement au diagramme de Gantt qui permet uniquement de visualiser les solutions, le graphe disjonctif est en plus un outil de modélisation abondamment utilisé pour résoudre les problèmes d'atelier multi-machines (Jain et Meeran [1999]).

Le graphe disjonctif a été proposé par Roy et Susmann [1964] pour l'ordonnancement des job shop. Les sommets du graphe représentent les différentes opérations composant les travaux, auxquels s'ajoutent deux sommets fictifs appelés source et puits correspondant respectivement au début et à la fin des jobs.

On distingue deux sortes d'arcs reliant les sommets : les arcs conjonctifs et les arcs disjonctifs. Un arc conjonctif connecte deux opérations consécutives d'un même job, décrivant ainsi la contrainte de précédence liant ces opérations. Un arc disjonctif connecte deux opérations appartenant à des travaux distincts qui utilisent une 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'extrémité droite, excepté les arcs partant de la source qui sont pondérés par 0.

Une solution réalisable du problème d'ordonnancement est obtenue en choisissant, pour chaque arc disjonctif, une orientation telle que le graphe résultant soit acyclique, c'est-à-dire sans circuits. On dit alors que le graphe disjonctif est arbitré (Carlier et Chrétienne [1988]).

La figure 1.2 est la représentation par le graphe disjonctif arbitré de la solution illustrée par le diagramme de Gantt précédent (figure 1.1). Par définition, la valeur du makespan est donnée par la longueur du plus long chemin allant de la source au puits.

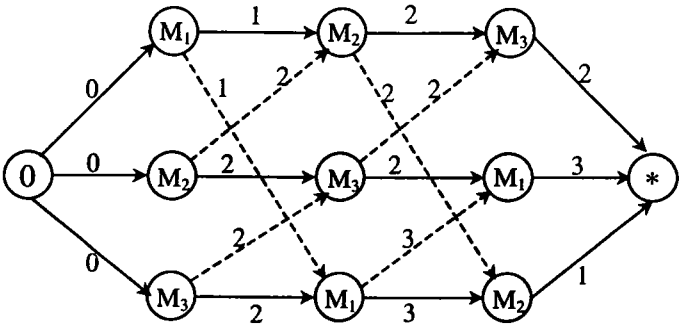


Figure 1.2 : Graphe disjonctif arbitré

2.5. Complexité des problèmes d'ordonnancement

Pour un problème d'ordonnancement de job shop à n travaux et m machines Jm || C_{max}, le nombre de séquences d'opérations qu'il est possible de former sur chacune des machines est égale à n !.

Si ces n ! séquences sont choisies indépendamment pour chaque machine, cela donne $(n !)^m$ solutions potentielles, certaines séquences ne conduisant pas à des solutions réalisables. Pour une instance de taille 10 × 10, le nombre de solutions potentielles est ainsi de l'ordre de 3.95×10^{65}. L'énumération complète des solutions réalisables pour trouver la solution optimale n'est donc pas envisageable, puisque le nombre de séquences possibles croît exponentiellement avec n et m, et Jm || C_{max} est un problème d'optimisation combinatoire.

La théorie de la complexité [Cook 71, Karp 72, Garey and Johnson 79] 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 classes de difficulté. En particulier, elle fixe la limite entre les problèmes dits polynomiaux et les problèmes dits NP-difficiles.

En théorie de la complexité, la distinction est faite entre problèmes d'optimisation et problèmes de décision : Un problème de décision associé à un problème d'ordonnancement consiste à répondre par vrai ou faux à la question :

Pour une instance donnée, existe-il une solution réalisable du problème d'ordonnancement telle que la valeur de la fonction objectif ne dépasse pas une constante fixée ?

Si la réponse est négative, le problème de décision est dit NP-complet (NP pour *Non deterministic Polynomial*). Dans ce cas, le problème d'ordonnancement correspondant est dit NP-difficile. En particulier, cela signifie qu'il n'existe aucun algorithme polynomial permettant de le résoudre.

Un problème d'ordonnancement est dit *polynomial* s'il existe un algorithme qui le résout en un temps polynomial de la taille du problème. Certains problèmes peuvent être résolus par des algorithme polynomiaux des données (durées opératoires, par exemple). De tels algorithmes sont appelés *pseudo-polynomiaux.*

Un problème d'ordonnancement P_b est *NP-difficile au sens faible*, s'il est possible de trouver un algorithme de résolution pseudo-polynomial.

S'il n'existe pas d'algorithme pseudo-polynomial permettant de résoudre P_b, le problème est *NP-difficile au sens fort.*

3. Méthodes de résolution

Nous présentons dans cette section quelques-unes des méthodes d'optimisation combinatoire utilisées pour la résolution des problèmes d'ordonnancement d'atelier. En premier lieu sont décrites des méthodes exactes, notamment la procédure par séparation et évaluation, la programmation dynamique, ainsi que la programmation linéaire. Parmi les méthodes approchées, nous choisissons de détailler la description de deux méta-heuristiques, à savoir les algorithmes génétiques et la recherche tabou.

3.1. Méthodes exactes

Les méthodes de résolution exactes permettent de trouver les solutions optimales pour des problèmes d'optimisation combinatoire, grâce à une exploration intelligente de l'espace des solutions. Pour les problèmes d'ordonnancement NP-difficiles, les méthodes les plus utilisées sont les procédures par séparation et évaluation, la programmation dynamique et la programmation linéaire en nombres entiers.

a) Procédure par séparation et évaluation

La procédure par séparation et évaluation ou PSE est une méthode de recherche par énumération implicite des solutions. Elle a été introduite pour la première fois par Dantzig et al. [1954] pour la résolution du problème du voyageur de commerce.

La PSE consiste à construire une arborescence dont la racine correspond à l'espace des solutions du problème initial. Dans un problème de minimisation, un majorant (BS) de la valeur de la fonction objectif pour une solution réalisable est avant tout déterminé. Comme son nom l'indique, deux actions principales interviennent dans une procédure par séparation et évaluation.

La séparation consiste à décomposer un sommet représentant l'espace des solutions d'un problème en une partition (ou plus rarement un recouvrement) de sous-ensembles disjoints de tailles inférieures. L'utilisation seule de la séparation 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), le sommet est éliminé. Par ailleurs, l'exploration d'un sommet est arrêtée s'il s'agit d'une feuille, c'est-à-dire si on ne peut plus effectuer de séparation. 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 procédure par séparation et évaluation.

Algorithme 1.1 : Procédure par séparation et évaluation

Etape 1 : *Initialisation*
Calculer une borne supérieure BS

Etape 2 : *Séparation*
Sélectionner le sommet à séparer et créer ses fils

Etape 3 : *Evaluation*
Pour chaque nouveau sommet créé **Faire**
Calculer la borne inférieure BI
Si un des sommets représente une solution complète **Alors**
Calculer sa valeur de critère objectif et mettre à jour BS
FinSi
FinPour

Etape 4 : *Elimination*
Eliminer tout sommet tel que BI > BS

Etape 5 : *Critère d'arrêt*
Si tous les sommets ont été éliminés **Alors** arrêter
Sinon aller à l'étape 2

Les performances de la méthode dépendent évidemment de la qualité des bornes inférieures et supérieures (elles doivent être serrées) 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, où sont explorés en premier les sommets les plus récemment créés, c'est-à-dire ceux du niveau de l'arborescence le plus grand.
- La largeur d'abord, qui consiste à explorer tous les sommets d'un niveau avant de passer au suivant.
- La manière progressive, où la priorité est donnée au sommet le plus prometteur, c'est-à-dire ayant la meilleure évaluation.

De nombreuses PSE figurent dans la littérature de l'ordonnancement. Une référence dans le domaine est celle développée par Carlier et Pinson [1989] pour le problème de la minimisation du makespan dans un job shop. La PSE leur a permis de résoudre une instance à 10 jobs et 10 machines, proposée par Fisher et Thomson [1963] et restée ouverte jusque-là.

b) Programmation dynamique

La programmation dynamique est une méthode qui permet de résoudre de nombreux problèmes d'optimisation avec contraintes linéaires ou non, dont la fonction objectif présente la propriété dite de décomposabilité (Gondran et Minoux [1983]). Elle a été proposée par Bellman [1957] pour une recherche de plus court chemin dans un graphe.

Le principe de la programmation dynamique est de ramener la résolution d'un problème initial P à celle d'une série de sous-problèmes (P_0, P_1,..., P_n). Ces sous-problèmes sont liés par une relation de récurrence portant sur la valeur de la fonction objectif. Ainsi, un sous-problème P_k est résolu optimalement en tenant compte des informations obtenues lors de la résolution des problèmes P_0 à P_{k-1}. Pour obtenir la solution optimale au problème P, il suffit de parcourir à rebours (de P_n à P_0) l'ensemble des décisions prises et conservées à chaque résolution de sous-problème.

Cette méthode peut s'avérer coûteuse en temps et en mémoire du fait qu'il est nécessaire de conserver des informations obtenues au cours de la recherche. Cependant, elle permet de fournir des algorithmes polynomiaux pour des cas particuliers. Un exemple typique est la recherche du plus court chemin dans un graphe valué.

Dans le domaine de l'ordonnancement, la programmation dynamique a été utilisée pour développer des algorithmes polynomiaux et pseudo-polynomiaux pour des problèmes à une machine et à machines parallèles (Lawler [1973], Schrage et Baker [1978], Potts et Van Wassenhove [1982], Carlier et Chrétienne [1988]).

c) Programmation linéaire

Un programme linéaire est un outil de modélisation des problèmes d'optimisation combinatoire, où les contraintes sont des inéquations et la fonction objectif est une équation, linéaires des variables de décision.

Lorsque les variables de décision sont réelles, on parle de programmes linéaires continus. Ces programmes sont connus comme étant polynomiaux (algorithmes de Kachian [1979] et de Karmarkar [1984]). En pratique, c'est l'algorithme du Simplexe (Dantzig [1951]) qui est le plus utilisé, bien que sa complexité théorique soit exponentielle.

L'idéal pour la résolution d'un problème est de pouvoir le modéliser en un programme linéaire continu. Mais la majorité des problèmes d'ordonnancement ne peut être modélisée que par des programmes en variables entières ou mixtes (réelles et entières), qui sont NP-difficiles. Aussi, des PSE sont utilisées pour leur résolution (Gomory [1958], ainsi que tout progiciel commercialisé de programmation linéaire en nombres entiers). Notons que les bornes inférieures sont obtenues en résolvant les programmes linéaires relâchés, c'est-à-dire où les variables sont supposées réelles.

3.2. Méthodes approchées

La majorité des problèmes d'optimisation combinatoire est NP-difficile. L'application de méthodes exactes de résolution n'est alors raisonnable que si la taille du problème n'est pas trop importante. En effet, dans le cas contraire, la détermination des solutions optimales présente un coût élevé en termes de temps de calcul et d'espace mémoire. Aussi est-il préférable d'appliquer des méthodes approchées de résolution ou heuristiques.

Ces dernières visent à obtenir à moindre coût des bonnes solutions, c'est-à-dire aussi proches que possible de l'optimum. Dans certains cas, il est en outre possible de quantifier l'écart entre la meilleure solution fournie par une méthode approchée et la solution optimale du problème. On parle alors d'heuristique à performance garantie.

Nous présentons dans ce qui suit une classe prépondérante d'heuristiques utilisées pour la résolution de problèmes d'optimisation combinatoire. Il s'agit des méta-heuristiques qui peuvent se définir comme étant des assemblages de plusieurs heuristiques. Le succès des méta-heuristiques tient du fait qu'elles permettent d'intégrer les différentes contraintes pratiques des problèmes, sont faciles à implémenter, et proposent des solutions de bonne qualité.

Parmi les techniques qui ont prouvé leur efficacité dans la résolution de problème d'optimisation combinatoire, nous pouvons citer les *algorithmes génétiques* (Holland [1975]), le *recuit simulé* (Kirkpatrick [1983]), la *recherche tabou* (Glover [1986]), la *recherche dispersée* (Glover [1977]) et la *colonie des fourmis* (Colorni et al. [1991]). Le lecteur intéressé trouvera une comparaison de ces différentes méthodes dans le récent état de l'art de Taillard et al. [2001].

Dans ce qui suit, sont décrites deux méta-heuristiques que nous appliquons pour la résolution des problèmes traités dans cette thèse : les algorithmes génétiques et la recherche tabou.

a) Algorithmes génétiques

Les algorithmes génétiques sont des méthodes d'optimisation qui peuvent être considérées comme des algorithmes évolutionnistes ou de population construisant des solutions en en combinant d'autres.

Le principe des algorithmes génétiques est de simuler le processus d'évolution naturelle des espèces sexuées. Deux lois importantes caractérisent l'évolution d'une population d'êtres sexués : la transmission des caractères héréditaires lors de la reproduction et la loi de survie au sein de la population. Plus précisément, la reproduction de deux individus permet de créer des enfants plus ou moins différents de leurs parents. Un enfant hérite d'une partie du patrimoine génétique de chacun de ses parents mais possède aussi des caractéristiques qui lui sont propres. Ainsi, un enfant qui aura hérité de 'bonnes caractéristiques' sera bien adapté au milieu. Son espérance de vie et ses chances de se reproduire seront plus grande que celle d'un enfant qui aura hérité de 'mauvaises caractéristiques' (ou dont le patrimoine génétique aura subi de mauvaises mutations).

Holland [1975] a proposé pour la première fois cette analogie entre théorie de l'évolution et méthode d'optimisation combinatoire basée sur l'assimilation de l'ensemble des solutions d'un problème à une population d'individus. Cette méthodologie a par la suite été abondamment développée (Goldberg [89], Davis [91]).

Contrairement à d'autres méta-heuristiques telles que la recherche tabou ou le recuit simulé qui opèrent sur une seule solution, les algorithmes génétiques démarrent d'une population initiale. Cette population composée de chromosomes est générée aléatoirement ou au moyen d'une heuristique. Chaque chromosome est un ensemble de gènes caractérisant une solution du problème d'optimisation. Le but d'un algorithme génétique est alors de faire évoluer la population initiale vers une population d'individus plus forts, c'est-à-dire ayant les meilleures caractéristiques possibles. En pratique, cela correspond à visiter l'espace des solutions pour trouver les meilleures d'entre elles. La progression d'une population à la suivante est obtenue par l'application de deux opérateurs de base appelés croisement et mutation. L'algorithme 1.2 suivant décrit le schéma général des algorithmes génétiques.

Algorithme 1.2 : Algorithmes génétiques

Etape 1 : *Initialisation*

- Choisir un codage et générer une population initiale
- Evaluer chaque chromosome de la population

Etape 2 : *Evolution de la population*

Tant que une condition d'arrêt n'est pas satisfaite **Faire**

- Créer des nouveaux individus (croisement, mutation)
- Evaluer la force de ces individus
- Remplacer tout ou une partie de la population par ces nouveaux individus

FinTantque

La première étape d'un algorithme génétique est le choix d'un codage permettant d'associer un individu de la population à une solution du problème à résoudre. A l'origine, les solutions étaient codées sous forme de vecteurs binaires (Goldberg [1989]). Les applications plus récentes des algorithmes génétiques font appel à d'autres codages plus naturels tels que des permutations (Reeves [1995]). Deux sortes de codages couramment utilisés pour la représentation des solutions sont le codage direct et indirect. 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.

Une fois la population initiale générée (aléatoirement ou grâce à des heuristiques), chaque individu se voit attribuer une force qui permet de quantifier son importance au sein de la population. Jusqu'à ce qu'une condition d'arrêt soit satisfaite (souvent un nombre d'itérations fixé), la population initiale est améliorée. Cette amélioration consiste à remplacer une partie de la population (schéma de Davis [91]) ou la population complète (schéma de Goldberg [89]) par de nouveaux individus. Les nouveaux individus sont obtenus par application de deux opérateurs génétiques à des individus sélectionnés dans la population courante. Plusieurs modes de sélection existent dans la littérature, notamment la roulette (Goldberg [89]) qui attribue à chaque individu une probabilité proportionnelle à sa force.

Le premier opérateur génétique permettant de créer de nouveaux individus est l'opérateur de croisement qui consiste à combiner les gènes de deux chromosomes, dans l'espoir d'en conserver les qualités. L'un des croisement les plus simples est le croisement en un point noté 1X (Goldberg [89]). Ce dernier consiste à choisir de manière aléatoire un point de croisement qui décompose chacun des chromosomes-parents C_1 et C_2 en deux sous-chaînes C_{11}, C_{12} et C_{21}, C_{22}. Le premier enfant est obtenu par concaténation de C_{11} et C_{22}, le deuxième enfant à partir de C_{21} et C_{12}. D'autres croisements plus élaborés figurent dans la littérature. Pour les croisements de permutation, nous pouvons citer en exemple les croisements OX pour *Order Crossover* (Oliver et al. [1987]), LOX pour *Linear Order Crossover* (Falkenauer et Bouffouix [1991]), PMX pour *Partially Mapped Crossover* (Goldberg et Lingle [1985]) et ERX pour *Edge Recombination Crossover* (Whitley [1989]).

Le second opérateur intervenant dans le processus de génération de nouveaux individus est l'opérateur de mutation. Il consiste à perturber des chromosomes choisis aléatoirement en modifiant certaines de leurs caractéristiques. Dans le cas d'un codage par des permutations, la mutation peut être l'échange des positions de certains gènes. L'utilisation de cet opérateur permet d'éviter une convergence trop rapide vers une sous-population limitée dans ces gènes, au détriment de la qualité de cette solution. La mutation assure donc la diversification de la population ce qui revient à donner une chance d'être visitées à des parties supplémentaires de l'espace des solutions.

Le choix des procédures intervenant dans les algorithmes génétiques (codage, croisement, etc.) doit évidemment être effectué en fonction du problème considéré. Par ailleurs, la performance de tout algorithme génétique dépend de caractéristiques qui lui sont propres, notamment la taille de la population, les probabilités d'application des opérateurs de croisement et de mutation. Nous renvoyons le lecteur intéressé à Whitley [1989] pour plus de détails sur les algorithmes génétiques et à Portmann et Vignier [2001] pour des applications aux problèmes d'ordonnancement d'atelier.

b) Recherche tabou

La recherche tabou est une méta-heuristique qui permet de résoudre des problèmes d'optimisation combinatoire par améliorations successives d'une solution.

Le principe de la recherche tabou est d'explorer intelligemment l'espace des solutions d'un problème en évitant d'être piégé par un optimal local, et au moyen de deux stratégies spécifiques. La première de ces stratégies est l'intensification qui consiste à encourager la recherche dans des régions de l'espace jugées prometteuses. La seconde stratégie est la diversification qui est désignée pour guider la recherche vers de nouvelles régions de l'espace des solutions.

L'idée de la recherche tabou a été proposée pour la première fois dans les années soixante dix par Glover [1977]. Mais le principe de la recherche tabou, tel qu'il est appliqué actuellement n'a été développé qu'à partir du milieu des années 80 par Glover [1986], et de manière indépendante par Hansen [1986].

Parmi les méta-heuristiques utilisées pour la résolution de problèmes d'optimisation combinatoire, la recherche tabou est la seule qui soit considérée comme fonctionnant avec une mémoire, ou plutôt un ensemble de mémoires (Glover [1997]). En effet, deux structures particulières caractérisent la recherche tabou : la mémoire explicite et la mémoire attributive.

La mémoire explicite, qui permet de consigner les meilleures solutions trouvées au fil de la recherche, sert de base à la stratégie de diversification. La mémoire attributive quant à elle enregistre les attributs qui permettent de passer d'une solution à une autre. Pour un problème d'ordonnancement d'atelier, par exemple, les attributs peuvent être des permutations d'opérations.

L'algorithme 1.3 suivant décrit le schéma général de la recherche tabou pour une fonction objectif f à minimiser.

Algorithme 1.3 : Recherche tabou

Etape 1 : *Initialisation*

- Générer une solution initiale s
- Evaluer cette solution : calculer f(s)

Etape 2 : *Amélioration de la solution*

Tant que une condition d'arrêt n'est pas satisfaite **Faire**

- Trouver le meilleur mouvement autorisé dans le voisinage de s
- Mettre à jour les structures de mémoires
- Construire la solution s' associée au mouvement retenu

FinTantque

La première étape de la recherche tabou, comme dans toute méthode de recherche locale, consiste à générer une solution initiale de qualité plus ou moins bonne. Cette solution peut être construite par une heuristique ou générée aléatoirement.

Tant qu'une condition d'arrêt n'est pas satisfaite, la solution de départ est alors progressivement améliorée. Plusieurs conditions sont utilisées dans la littérature, notamment celles qui consistent à stopper la recherche après un nombre fixé d'itérations ou encore après un nombre d'itérations sans amélioration de la solution.

La procédure d'amélioration appliquée à chaque itération se décompose de la manière suivante. En premier lieu, un voisinage correspondant à une région de l'espace des solutions est associé à la solution courante. Ce voisinage est constitué de toutes les solutions obtenues par application d'une opération appelée *mouvement*.

La taille du voisinage pouvant être considérable, certains chercheurs préfèrent se limiter à une partie stratégique de ce voisinage (Glover [1997]). Le meilleur mouvement parmi ceux qui sont autorisés, c'est-à-dire celui qui permet d'obtenir la meilleure solution du voisinage, est alors sélectionné pour construire la nouvelle solution. Les attributs de ce mouvement sont consignés dans la *liste tabou* T et conservés durant les | T | itérations suivantes, | T | étant la taille de la liste tabou, ou *tenure* (Glover [1995]).

Les informations données par la liste tabou sont utilisées pour établir une *restriction* (appelée restriction tabou), qui permet de classer certains mouvements comme étant interdits, et permet ainsi d'éviter de retourner à des solutions déjà visitées dans un passé récent (phénomène de cyclage).

Enfin, le statut tabou d'une solution n'est pas définitif et peut être éliminé si certaines conditions exprimées par le *critère d'aspiration* sont satisfaites. Le critère d'aspiration confère donc plus de souplesse à la recherche tabou en lui permettant d'accepter un mouvement lors de la visite du voisinage même si ce mouvement est tabou. L'un des critères d'aspiration les plus utilisés est le critère d'aspiration global qui consiste à sélectionner un mouvement tabou s'il permet d'améliorer la meilleure valeur trouvée de la fonction objectif.

Les listes tabous peuvent être explicites ou attributives selon les restrictions utilisées. Le choix du type de liste dépend du problème considéré. De manière empirique, il a été démontré dans la littérature qu'il était préférable que la taille de la liste tabou augmente avec la taille des données.

Notons que fixer une petite taille de liste augmente le risque de cyclage, alors qu'une grande taille peut faire diminuer la qualité de la solution. Pour plus de détails sur les différents paramètres de la recherche tabou, nous renvoyons le lecteur intéressé aux articles de Glover [1989, 1990], Glover et al. [1993], Glover et Laguna [1997].

4. Etat de l'art

Cette section est dédiée à la présentation des principaux résultats obtenus dans la littérature pour les problèmes d'ordonnancement d'ateliers. La première sous-section est dédiée aux problèmes classiques sans contraintes de disponibilité des machines. La deuxième sous-section est consacrée aux à ordonnancements avec prise en compte des indisponibilités des machines.

4.1. Problèmes sans contraintes de disponibilité

Nous rappelons, en premier lieu, quelques-unes des règles de priorité (définies il y a plus de trente ans) permettant de résoudre de manière optimale certains problèmes à une machine. Nous verrons ultérieurement si elles restent applicables en présence de contraintes de disponibilité. En second lieu, sont dressés des états de l'art portant respectivement sur les problèmes d'ordonnancement de type flow shop et job shop.

a) Problème à une machine

Pour la minimisation du makespan dans le problème à une machine de base (sans contraintes temporelles, ni dates de disponibilité des opérations), tout ordonnancement sans délai, c'est-à-dire dont les opérations sont calées au plus tôt, est optimal.

La règle WSPT (pour *Weighted Shortest Processing Time first*) consiste à ordonnancer les opérations dans l'ordre croissant des rapports p_i / w_i où w_i et p_i désignent respectivement le poids et la durée de l'opération i. Proposée par Smith [1956], cette règle de priorité permet de minimiser le temps moyen pondéré de séjour et le volume du stock d'encours.

En l'absence de coefficients de pondération sur les opérations, la règle WSPT se réduit à ordonnancer les opérations selon les durées opératoires croissantes. C'est la règle SPT (*Shortest Processing Time first*) qui minimise le temps moyen de séjour et la durée moyenne de réalisation. Si des dates de fin au plus tard sont souhaitées, l'ordonnancement des opérations selon la règle SPT permet de minimiser l'écart moyen par rapport aux délais.

La règle EDD (*Earliest Due Date first*) connue également sous le nom de règle de Jackson [1955] permet, quant à elle, de minimiser le plus grand retard. Elle consiste à ordonnancer les opérations dans l'ordre croissant de leur date échue.

L'algorithme de Moore et Hodgson [1968] permet de minimiser le nombre d'opérations en retard. Il consiste à construire progressivement un ordonnancement selon la règle EDD avec la modification suivante : Dès qu'un retard apparaît dans l'ordonnancement partiel initial, l'opération de plus longue durée déjà placée est reportée en fin d'ordonnancement.

b) Flow shop

Le problème d'ordonnancement de type flow shop est un problème classique d'atelier qui consiste à ordonnancer un ensemble de n jobs (ou travaux) sur un ensemble de m machines $M = \{M_1, M_2, \ldots, M_m\}$ disposées en série. La particularité du flow shop est que les gammes de fabrication des jobs sont fixées et identiques. Plus précisément, tous les jobs doivent visiter l'ensemble des machines selon l'ordre $(M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_m)$. Les machines sont supposées disponibles tout au long de l'horizon de planification et ne peuvent exécuter qu'une seule opération à la fois. Par ailleurs, une opération ne peut s'exécuter que sur une machine à la fois. Un cas particulier très souvent étudié dans la littérature est le flow shop de permutation. Ce modèle est caractérisé par le fait que la séquence de traitement des jobs est la même pour toutes les machines.

A quelques exceptions près, les problèmes de type flow shop sont NP-difficiles. Garey et al. [1976] ont en effet montré que pour un nombre de machines supérieur ou égal à trois, le problème était NP-difficile au sens fort, et ce, quel que soit le critère considéré. Les auteurs ont en outre prouvé que pour tous les critères, excepté le makespan, l'ordonnancement d'un flow shop à deux machines était NP-difficile.

Dans le cas d'un flow shop à deux machines, Johnson [1954] a démontré que la minimisation du makespan était polynomiale, en proposant d'ordonnancer les travaux selon la règle suivante :

Un travail i précède un travail j dans une séquence optimale si leurs temps opératoires vérifient la relation $\min(p_{i1}, p_{j2}) < \min(p_{j1}, p_{i2})$.

Gilmore et Gomory [1964] ont montré qu'en ajoutant la contrainte *sans attente* (nwt), le problème de la minimisation du makespan sur deux machines (noté F2 | nwt | C_{max}) restait polynomial. Cette contrainte signifie que les travaux doivent passer sur la deuxième machine dès la fin de leur exécution sur la première. Ce cas se présente par exemple dans le domaine de la métallurgie quand une pièce quittant une machine doit être directement prise en charge par la suivante. L'algorithme proposé par Gilmore et Gomory est basé sur la transformation du problème F2 | nwt | C_{max} en un problème de voyageur de commerce avec des distances particulières.

En marge de ces résultats de complexité, des résultats importants, dits de *dominance* ont été établis. Prouver qu'un ensemble d'ordonnancements est dominant pour un problème signifie qu'au moins une solution optimale appartient à cet ensemble. En particulier, cela permet de restreindre la recherche (et de manière considérable) aux ordonnancements dominants.

Pour les problèmes à deux machines, les ordonnancements de permutation, c'est-à-dire tels que la séquence d'entrée des travaux est la même sur les deux machines, sont dominants pour tous les critères réguliers. Ce résultat reste bien sûr valable pour le problème à deux machines et sans attente.

Dans le cas de trois machines, les ordonnancements de permutation sont dominants seulement si le critère considéré est le makespan.

Campbell, Dudeck et Smith [1970] ont prouvé que pour la minimisation du makespan dans un flow shop à m machines, les ordonnancements dominants étaient tels que les séquences des travaux sont identiques sur les deux premières machines ainsi que sur les deux dernières.

En ce qui concerne le cas général avec plusieurs machines, les recherches se sont essentiellement portées sur le flow shop simplifié de permutation. Notons que si cette restriction permet d'en diminuer la difficulté (au sens propre), elle ne garantit en aucun cas l'optimalité des solutions pour le problème non restreint.

Plusieurs méthodes exactes ont été proposées pour la résolution des problèmes de flow shop à *m* machines, mais c'est surtout sur le développement d'heuristiques que se sont concentrés les efforts des chercheurs. Par ailleurs, le critère le plus étudié dans la littérature consacrée au flow shop est sans conteste le makespan.

Ignall et Schrage [1965] ont été parmi les premiers à développer une procédure par séparation et évaluation pour le problème $Fm \mid prmu \mid C_{max}$. Des méthodes exactes ont été proposées par Carlier et Rebaï [1996], Cheng et al. [1997] pour la résolution du problème $Fm \parallel C_{max}$. Péridy et al. [1998] ont élaboré des règles d'élimination permettant d'accélérer la recherche des solutions optimales aux problèmes $Fm \mid prmu \mid C_{max}$ et $Fm \parallel C_{max}$. Récemment, Della Croce et al. [2002a] ont développé une PSE pour la minimisation de la somme des dates de fin de travaux dans un flow shop à deux machines.

Pour des problèmes de plus grandes tailles, plusieurs heuristiques permettant d'approcher la solution optimale ont été développées par Park et al. [1984], Taillard [1990], Nowicki et Smutnicki [1996a, 1996b]. Une comparaison de différentes méthodes de recherche locale a été réalisée par Glass et Potts [1996]. Nowicki [1999] a développé une recherche tabou performante pour le cas où la capacité du stock entre les machines est limitée.

Les récentes études des problèmes d'ordonnancement dans les ateliers de type flow shop portent essentiellement sur des modèles à deux ou trois machines. Pour ces problèmes de tailles réduites, les auteurs démontrent des propriétés de dominance des ordonnancements optimaux (Cheng et al. [2001]) ou étudient l'impact, notamment sur la complexité du problème, de la prise en compte de contraintes additionnelles (Kyparisis et Koulamas [2000], Kashyrskikh et al. [2001], Epek et al. [2002]).

D'autres axes de recherche focalisant l'attention des chercheurs sont la considération de fonctions objectif particulières telle que le minimisation du nombre de travaux en retard (Della Croce et al. [2002b]) ou encore l'optimisation simultanée de plusieurs critères (Framinan et al. [2002]). Par ailleurs, le développement d'heuristiques de plus en plus élaborées se poursuit (Ladhari et Haouri [2002], Steinhöfel [2002]).

c) Job shop

Le problème d'ordonnancement de type job shop est sans aucun doute l'un des problèmes les plus étudiés dans la littérature (Jain et Meeran [1999]). Ceci s'explique, d'une part, par l'importance théorique du modèle, et d'autre part, par les nombreuses applications pratiques qui peuvent être modélisées de la sorte.

Le job shop est une généralisation directe du système d'atelier de type flow shop, pour lequel l'ordre de passage des jobs sur les machines peut être différent. Le problème étant fortement combinatoire, les premiers efforts des chercheurs se sont portés sur des cas particuliers avec un nombre fixe de machines et de jobs.

Jackson [1956] a proposé une généralisation des règles de Johnson pour la résolution du cas particulier avec deux machines $J2 \parallel C_{max}$ par un algorithme polynomial.

Le problème à deux jobs $J \mid n = 2 \mid f$ (où f est un critère régulier) peut lui aussi être résolu optimalement en un temps polynomial. Les résultats de complexité sur les problèmes à deux jobs sont dus à Sotskov [1985] et Brucker [1988]. Ces derniers ont utilisé la représentation géométrique développée par Akers et Friedman [1955], Szwarc [1960], Hardgrave et Nemhauser [1963], pour la construction de l'algorithme polynomial appelé *approche géométrique*. Cette approche permet de trouver la solution optimale pour le problème du job shop à deux jobs, avec n'importe quelle fonction objectif régulière.

Pour résoudre les problèmes d'ordonnancements de job shop généraux (c'est-à-dire à plus de deux travaux), plusieurs procédures par séparation et évaluation ont été proposées dans la littérature.

La première méthode efficace a été développée par McMaron et Florian [1987]. Carlier et Pinson [1989] ont proposé une procédure qui a permis de résoudre une instance à 10 jobs et 10 machines, proposée par Fisher et Thomson [1963] et qui était restée ouverte. Depuis, des procédures plus efficaces ont été proposées pour résoudre les instances les plus difficiles en un temps modeste (Carlier et Pinson [1990], Applegate et Cook [1991], Brucker et al. [1994a]). Toutes ces méthodes sont basées sur le modèle du graphe disjonctif.

Les méthodes exactes ayant révélées leurs limites pour la résolution optimale des problèmes de grandes tailles, des méthodes heuristiques permettant d'approcher la solution optimale ont été proposées.

Parmi ces méthodes approchées on trouve aussi bien des méthodes basées sur des règles de priorité (Panwalkar et Iskander [1977]), des méthodes basées sur l'étude de la machine goulot (Adams et al. [1988], Dauzère-Pérès et Lasserre [1993], Balas et al. [1995, 1998]) ou encore des méthodes plus sophistiquées basées sur des recherches locales (Glover et Greenberg [1989], Nowicki et Smutnicki [1996]). Une comparaison entre plusieurs d'entre elles a été réalisée par Van Larhoven et al. [1992].

Pour plus de détails sur toutes les méthodes de résolution appliquées au modèle du job shop, nous renvoyons le lecteur intéressé aux états de l'art figurant dans les articles de Aarts et al. [1994], Blazewicz et al. [1996], Jain et Meeran [1999].

Comme pour le flow shop, les études récentes consacrées aux problèmes d'ordonnancement dans les ateliers de type job shop portent sur l'établissement de règles de dominance et de résultats de complexité (Guinet [2000]) pour des problèmes de tailles réduites, sur la considération de critères particuliers (Kravchenko [2002]), sur la prise en compte de contraintes additionnelles (Mosheiov [2002]) et sur l'amélioration de méthodes de résolution (Pezzella et Merelli [2000], Blazewicz et al. [2000b], Dorndorf et al. [2002]).

4.2. Problèmes avec contraintes de disponibilité

Comparé à la littérature dédiée aux problèmes classiques d'ordonnancement, les études où sont prises en compte les indisponibilités des machines ne sont pas très nombreuses. Ce domaine de recherche est en effet relativement récent. Cependant, l'ordonnancement sous contraintes de disponibilité attire de plus en plus l'attention des chercheurs (Lorigeon et al. [2002], Sadfi [2002], Kubiak et al. [2002]).

Nous présentons dans ce qui suit les principaux résultats de complexité concernant l'ordonnancement sous contraintes de disponibilité des machines. Nous focalisons notre état de l'art sur le contexte déterministe où les périodes d'indisponibilité sont connues à l'avance.

En effet, dans le cas où les indisponibilités sont inconnues (en raison de pannes par exemple), le problème d'ordonnancement peut être considéré comme dynamique et des méthodes de résolution telles que les algorithmes en ligne sont nécessaires. Nous renvoyons le lecteur intéressé aux articles de Glazebrook [1987], Birge et Glazebrook [1988], Adiri et al. [1989], Birge et al. [1990], Pinedo [1995], Li et Cao [1995], Qi et al. [1997], Lee et Leon [1998], Graves et Lee [1999], qui portent sur des problèmes à une machine, ainsi qu'à l'article de Lee et Chen [2000], pour des problèmes à machines parallèles.

Comme le soulignent Sanlaville et Schmidt [1997], la disponibilité limitée des machines a d'abord et surtout été prise en compte dans le cadre de problèmes à machines parallèles et de problèmes à une machine. Par la suite, plusieurs études ont été consacrées aux systèmes de type flow shop et open shop, essentiellement à deux machines.

Plusieurs notations sont employées dans la littérature pour caractériser les contraintes de disponibilité des machines. Les plus utilisées figurent dans Pinedo [1995], Lee [1996], Schmidt [2000] et Kubiak et al. [2002]. Par souci de clarté et étant données les différences de ces notations, nous choisissons de ne pas les introduire dans ce qui suit.

a) Interruption des opérations par des indisponibilités de machines : définitions

Dans la littérature dédiée à l'ordonnancement sous contraintes de disponibilité des machines, trois cas fondamentaux caractérisant les opérations à réaliser sont considérés : les cas *sécables*, *non-sécables* et *semi-sécables*.

- Une opération interrompue par une période d'indisponibilité est dite *sécable* si son exécution peut reprendre dès que la machine concernée est à nouveau disponible.
- Dans le cas où l'on doit reprendre du début l'exécution d'une opération interrompue, on parle d'opération *non-sécable*.
- Une opération interrompue est dite *semi-sécable* si une partie de la portion de l'opération exécutée avant la période d'indisponibilité est à ré-exécuter dès que la machine concernée est à nouveau disponible (en plus de la portion restante à réaliser).

Notons que le cas semi-sécable généralise les cas sécables et non-sécables. En effet, si la partie à reprendre d'une opération correspond à toute la portion déjà exécutée, on retrouve le cas non-sécable. Si par contre on n'a rien à reprendre de ce qui a déjà été exécuté, il s'agit uniquement de réaliser la portion restante de l'opération, dès la disponibilité de la machine, ce qui correspond au cas sécable.

b) Problèmes à une machine

b_1) Cas sécable

Lee [1996] a étudié le problème à une machine sous différentes mesures de performance et a identifié quelques problèmes polynomiaux. En particulier, l'auteur a prouvé que, pour la minimisation du makespan dans le cas sécable, toute séquence était optimale. L'auteur a en outre montré qu'ordonnancer les opérations selon la règle SPT permettait de résoudre de manière optimale la minimisation de la somme des dates de fin des travaux. De même, les minimisations du plus grand retard et du nombre de travaux en retard sont solubles de manière optimale par les règles EDD et une modification de la règle de Moore-Hodgson, respectivement.

La minimisation de la somme des retards pondérés dans le cas classique (c'est-à-dire sans indisponibilités des machines) peut être résolue de manière optimale en ordonnançant les travaux selon la règle SWPT. Mais lorsque des contraintes de disponibilité sont prises en compte, Lee [1996] a prouvé que le problème devenait NP-difficile au sens faible même si tous les coefficients de pondération étaient supposés égaux aux durées des opérations associées. Par ailleurs, un algorithme de programmation dynamique, ainsi que plusieurs heuristiques ont été proposé pour résoudre le problème dans le cas d'une période d'indisponibilité.

Lorigeon et al. [2002] se sont intéressés au problème à une machine, avec une période d'indisponibilité, en considérant des dates de disponibilité et des durées de latence sur les opérations. Les auteurs ont proposé une borne inférieure, deux bornes supérieures ainsi qu'une procédure par séparation et évaluation. Notons que dans le cas où la machine est disponible de manière continue, le problème est NP-difficile au sens fort.

b_2) Cas non-sécable

Adiri et al. [1989] ont étudié le problème à une machine, supposée indisponible durant certaines périodes, avec pour objectif la minimisation de la somme des dates de fin des travaux. Le contexte déterministe où les indisponibilités sont connues à l'avance ainsi que le contexte stochastique où les positions et durées des indisponibilités sont aléatoires ont tous deux été considérés. En particulier, dans le cas déterministe, il a été démontré que le problème était NP-difficile au sens faible, même si l'on ne considérait qu'une seule indisponibilité.

Lee et Liman [1992] ont développé une preuve plus simple pour montrer la NP-difficulté du problème déterministe considéré par Adiri et al. [1989]. Par ailleurs, les auteurs ont prouvé qu'un algorithme ordonnançant les opérations selon la règle SPT avait une garantie de performance égale à $^9/_7$.

Lee [1996] a montré que la minimisation du makespan dans le cas non-sécable était NP-difficile au sens faible, dès qu'une période d'indisponibilité était considérée. Par ailleurs, l'auteur a prouvé qu'un algorithme ordonnançant les opérations selon la règle LPT avait une erreur relative égale à $^1/_3$.

Dans le même article, il est démontré que les minimisations du plus grand retard, du nombre de travaux en retard et de la somme pondérée des dates de fin des opérations sont NP-difficiles au sens faible. En outre, l'algorithme EDD permet de résoudre le premier problème avec une erreur relative égale à p_{max} (plus grande durée opératoire) et le second problème peut être résolu par la règle de Moore-Hodgson avec une erreur relative égale à 1. Pour le troisième problème, Lee [1996] a prouvé que le ratio de performance de l'algorithme SWPT pouvait être arbitrairement grand, même si les tous les coefficients de pondération étaient choisis égaux aux durées opératoires.

Récemment, Sadfi [2002] s'est intéressé au problème étudié par Lee et Liman [1992]. L'auteur a développé un algorithme de programmation dynamique et une heuristique ayant une garantie de performance égale à $^{19}/_{17}$.

c) Problèmes à machines parallèles

c_1) Cas sécable

Schmidt [1984] a étudié le problème d'ordonnancement de n travaux sur m machines parallèles, avec pour objectif la minimisation du makespan. L'auteur a proposé un algorithme de complexité $O(n + m \log n)$ permettant de construire des ordonnancements préemptifs réalisables dans le cas où toutes les machines sont disponibles durant un nombre arbitraire de périodes.

Dans Schmidt [1988], des dates de disponibilité et de fin souhaitée additionnelles sont prises en compte et il est prouvé que le problème est soluble en $O(n\ m \log n)$ étapes. Si aucune date de disponibilité n'est imposée, la minimisation du plus grand retard peut être obtenue en un temps proportionnel à $O(n\ m \log n)$.

Kaspi et Montreuil [1988] et Liman [1991] ont prouvé qu'ordonnancer les travaux selon l'ordre croissant des durées opératoires (règle SPT) était un ordre optimal pour la minimisation de la somme des dates de fin des travaux, et ceci, dans le contexte de machines parallèles non disponibles à l'instant zéro.

Lee [1991] a considéré le problème à machines parallèles identiques en supposant que ces machines n'étaient pas disponibles à l'instant zéro, avec pour objectif la minimisation du makespan. L'auteur a proposé une heuristique avec une erreur relative de $^1/_2$ basée sur la règle LPT, puis une amélioration de cet algorithme ayant pour erreur $^1/_3$.

Liu et Sanlaville [1995, 1997] ont étudié différents schémas de disponibilité sur des machines parallèles avec contraintes de précédence sur les travaux. Les auteurs ont prouvé que la minimisation du makespan était polynomiale pour des schémas arbitraires d'indisponibilité et des contraintes de précédence de type chaînes. Dans le cas de contraintes de précédence arbitraires, le résultat est vrai seulement pour deux machines.

Lee [1996] a développé un algorithme de programmation dynamique permettant de résoudre de manière optimale le problème considéré par Kaspi et Montreuil [1988].

Lin et al. [1998] se sont intéressé à la maximisation de la plus petite date de fin de travail dans un environnement à m machines parallèles indisponibles à l'instant zéro et ont montré que l'ordre LPT avait une erreur au pire cas égale à $(2m\text{-}1) / (3m\text{-}2)$. Notons que le critère considéré dans cet article permet d'équilibrer les niveaux d'utilisation des machines.

c_2) Cas non-sécable

Ullman [1975] a été le premier à aborder la minimisation du makespan sur m machines parallèles avec contraintes de disponibilité. Dans le cas d'opérations non-sécables, l'auteur a démontré que le problème était NP-difficile au sens faible.

Lee et Liman [1993] ont étudié le problème à deux machines parallèles en supposant que l'une de ces machines n'était plus disponible à partir d'une certaine date. Les auteurs ont prouvé que le problème de la minimisation de la somme des dates de fin des travaux était NP-difficile au sens faible et ont proposé un algorithme de programmation dynamique, ainsi qu'une heuristique basée sur la règle SPT, pour sa résolution. La garantie de performance de leur heuristique est de $^3/_2$.

Mosheiov [1994] a considéré le même problème avec l'hypothèse que chaque machine n'était disponible que durant une certaine période. L'auteur a montré que pour le problème à m machines, l'ordonnancement des travaux selon la règle SPT était asymptotiquement optimal, lorsque le nombre de jobs tendait vers l'infini.

Un problème qui s'apparente à celui de Mosheiov [1994] est le problème d'ordonnancement avec fenêtres temporelles. Dans ce type de problèmes, chaque travail doit être réalisé durant une certaine période, ce qui revient à considérer des contraintes de disponibilité sur les travaux plutôt que sur les machines. Pour plus de détails sur l'ordonnancement avec fenêtres temporelles, nous renvoyons le lecteur intéressé à Lei et Wong [1990], Kraemer et Lee [1993].

Lee [1996] a prouvé que la minimisation du makespan dans un environnement à m machines parallèles était NP-difficile. Les performances de deux heuristiques ont en outre été analysées : la règle SPT et l'ordonnancement de liste LS, qui consiste à affecter toute opération (étant donné un ordre quelconque de traitement des travaux) à la machine qui lui confère la plus petite date de fin. Les algorithmes SPT et LS ont une erreur relative égale à $(m+1) / 2$ et m respectivement.

Dans le même article, l'auteur a montré que la minimisation de la somme pondérée des dates de fins de travaux pour un atelier à deux machines était NP-difficile. Un algorithme de programmation dynamique a par ailleurs été développé pour résoudre le problème de manière optimale dans le cas où tous les coefficients de pondération sont égaux à 1 et où la première machine est continuellement disponible.

d) Problèmes de type flow shop

Les études portant sur les problèmes de type flow shop avec contraintes de disponibilité sont relativement récentes. Par ailleurs, elles ne concernent à notre connaissance que des flow shop à deux machines.

Lee [1997] a étudié la minimisation du makespan dans un flow shop à deux machines, en considérant une seule période d'indisponibilité et des opérations sécables. L'auteur a montré que le problème était NP-difficile au sens faible quelle que soit la machine concernée par l'indisponibilité et proposé des algorithmes pseudo-polynomiaux basés sur la programmation dynamique. L'auteur a également développé une heuristique avec une performance garantie de $^3/_2$ (resp. $^4/_3$) dans le cas où la période d'indisponibilité se produit sur la première machine (resp. sur la deuxième), l'erreur relative obtenue par application de l'algorithme de Johnson étant égale à 1 (resp. $^3/_2$). Dans le même article, il a été démontré que pour une ou les deux machines indisponibles à l'instant zéro, le problème était soluble de manière optimale par l'algorithme de Johnson.

Dans Lee [1999], la minimisation du makespan pour un flow shop à deux machines est étudiée dans les cas sécables, non-sécables et semi-sécables. Dans le cas sécable, il est démontré que si une indisponibilité par machine est considérée, l'algorithme de Johnson est optimal si les indisponibilités se produisent à des dates identiques (et pas forcément à partir de l'instant zéro) sur les deux machines. Par ailleurs, si les deux machines sont indisponibles à des dates différentes et même si les durées des périodes d'indisponibilité sont égales, le problème devient NP-difficile au sens faible.

Dans le cas non-sécable, Lee ([1999]) a proposé une heuristique ayant une erreur relative égale à 1, lorsque la période d'indisponibilité est imposée sur la première machine. Si la période d'indisponibilité est imposée sur la deuxième machine, l'algorithme de Johnson a une erreur relative égale à 1.

Lee [1999] a généralisé les différents résultats de complexité de Lee [1997] au cas où les opérations sont semi-sécables. Plus précisément, l'auteur a montré que la minimisation du makespan était NP-difficile au sens faible si une période d'indisponibilité était considérée sur l'une ou l'autre des machines. Dans le cas où l'indisponibilité est imposée sur la première machine, l'auteur a développé un algorithme de programmation dynamique, et prouvé que l'algorithme de Johnson avait une erreur relative égale à 1. Si l'indisponibilité se produit sur la deuxième machine, l'algorithme de Johnson a une erreur relative égale à $\max\{^1/_2, \mu\}$, où μ est la portion à ré-exécuter de l'opération semi-sécable interrompue par la période d'indisponibilité.

Par ailleurs, il est montré dans le même article que si les deux machines ont toutes deux une période d'indisponibilité (qui ne commence pas à l'instant zéro), le problème est NP-difficile au sens faible, même si les dates de début et de fin d'indisponibilité sont identiques sur les deux machines. Dans ce cas, l'algorithme de Johnson a une erreur relative égale à μ. Enfin, si les périodes d'indisponibilité des deux machines débutent à l'instant zéro, l'algorithme de Johnson permet encore de résoudre la minimisation du makespan de manière optimale.

Espinouse et al. [1999] se sont intéressés au problème de minimisation du makespan dans un flow shop sans attente à deux machines, avec contraintes de disponibilité. Les auteurs ont tout d'abord montré que le problème était NP-difficile au sens faible si une seule période d'indisponibilité était considérée et ont développé une heuristique à performance garantie égale à 2 dans les cas sécables et non-sécables. D'autre part, il a été démontré que si plusieurs périodes d'indisponibilité étaient prises en compte, le problème devenait NP-difficile au sens fort et qu'il n'existait pas d'heuristique à performance garantie (qu'il s'agisse d'opérations sécables ou non).

Dans Cheng et Wang [1999], les auteurs ont considéré un flow shop à deux machines, chacune d'elles étant indisponible une seule fois, en supposant les périodes d'indisponibilité consécutives et les opérations semi-sécables. Une heuristique, dont la borne au pire cas est égale à $^2/_3$ dans le cas non-sécable, y est développée pour la minimisation du makespan.

Cheng et Wang [2000] ont traité le problème de la minimisation du makespan pour un flow shop à deux machines en considérant des opérations sécables et une période d'indisponibilité sur la première machine. Les auteurs ont démontré que la borne au pire cas égale à $^1/_2$ trouvée par Lee [1997] était serrée, puis ont développé une heuristique ayant une performance garantie égale à $^4/_3$.

Blazewicz et al. [2000a] ont proposé une implémentation parallèle d'une procédure par séparation et évaluation pour la minimisation du makespan dans un flow shop à deux machines où plusieurs indisponibilité par machine sont considérées et où les opérations sont sécables.

Deux heuristiques, ayant une borne au pire cas égale à $^5/_3$ et permettant donc d'améliorer les résultats de Espinouse et al [1999] (flow shop sans attente à deux machines, avec contraintes de disponibilité) ont été développées par Wang et Cheng [2001].

Dans Blazewicz et al. [2001] deux heuristiques constructives ainsi qu'un recuit simulé ont été développées pour résoudre le problème traité dans Blazewicz et al. [2000]. La première heuristique 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.

Braun et al. [2001] ont étudié la stabilité des ordonnancements pour un flow shop à deux machines en présence d'une période d'indisponibilité sur chaque machine. En particulier, les auteurs ont montré que pour la minimisation du makespan avec des opérations sécables, l'ordre de Johnson était encore dominant si les indisponibilités étaient suffisamment petites.

Kubiak et al. [2002] ont également considéré plusieurs périodes d'indisponibilité et des opérations sécables dans un flow shop à deux machines. Les auteurs ont montré que la minimisation du makespan était NP-difficile au sens fort même si toutes les indisponibilités ne concernaient qu'une machine. Dans le cas où les indisponibilités se produisent sur la première machine, une heuristique à performance garantie égale à 2 de complexité $O(n \log n)$ est proposée et il est montré qu'il n'existe pas d'heuristique à performance garantie à partir de deux indisponibilités, si au moins une d'entre elles est sur la deuxième machine. Par ailleurs, après avoir montré qu'ordonnancer les jobs selon la règle de Johnson entre deux périodes d'indisponibilité consécutives était optimal, les auteurs ont développé une procédure par

séparation et évaluation pour résoudre le problème d'affectation des jobs aux différents intervalles de disponibilités des machines.

e) Problèmes de type open shop

Lu et Posner [1993] ont proposé un algorithme polynomial pour la minimisation du makespan dans un open shop à deux machines, l'une d'entre elles n'étant pas disponible à l'instant zéro, et ce, dans le cas d'opérations non-préemptives.

Vairaktarakis et Sahni [1995] ont montré que pour des nombres quelconques de machines et de périodes d'indisponibilité, la minimisation du makespan dans le cas d'opérations préemptives était un problème polynomial.

Dans sa thèse, Breit [2000] s'est intéressé, entre autres, au problème d'ordonnancement de type open shop à deux machines en présence de contraintes de disponibilité des machines. En particulier, l'auteur a prouvé qu'il n'existait pas d'heuristique à performance garantie pour la minimisation du makespan dans le cas d'opérations sécables, si une machine avait une période d'indisponibilité et l'autre deux périodes.

Dans Breit et al. [2001], la minimisation du makespan est considérée dans un environnement de type open shop à deux machines, dont l'une est indisponible durant une seule période. Les auteurs ont prouvé que le problème sécable était NP-difficile au sens faible et développé une heuristique ayant un ratio au pire cas égal à $^4/_3$. Dans le cas où plusieurs indisponibilités de produisent sur une seule machine, une heuristique ayant une erreur de 2 a également été développée.

La minimisation du makespan pour un open shop à deux machines et dans le cas non-sécable a été étudiée dans Breit et al. [2002]. Les auteurs y ont d'abord supposé qu'une des machines avait plusieurs périodes d'indisponibilité et montré qu'il n'existait pas d'heuristiques à performance garantie. Puis, une heuristique ayant un ratio au pire cas égal à 2 (resp. $^4/_3$) a été développée pour le cas d'une période d'indisponibilité sur chaque machine (resp. sur une seule machine).

5. Conclusion

Nous avons présenté dans ce chapitre quelques notions de base concernant les problèmes d'ordonnancement dans les ateliers de production. En particulier, ont été décrites les méthodes de résolution approchées et exactes que nous utilisons par la suite, à savoir les algorithmes génétiques, la recherche tabou et la procédure par séparation et évaluation. En outre, nous avons dressé un bref état de l'art portant sur les modèles classiques qui servent de base à notre étude.

Concernant les problèmes d'ordonnancement avec la prise en compte de contraintes de disponibilité des machines, nous avons présenté, après avoir cité quelques résultats pour le contexte stochastique, les ouvrages de la littérature consacrés au contexte déterministe.

Il est à retenir de cette étude bibliographique que tous les travaux antérieurs traitant de l'ordonnancement de flow shop avec la prise en compte de périodes d'indisponibilité sont focalisés sur le cas de deux machines seulement. Par ailleurs, il n'existe pas, du moins à notre connaissance, d'ouvrage consacré au problème d'ordonnancement de type job shop avec contraintes de disponibilité des machines.

Aussi, notre étude est-elle motivée par la volonté de répondre en partie à ce manque, en se proposant d'aborder, dans les chapitres suivants les problèmes du flow shop général, c'est-à-dire à plusieurs machines, ainsi que celui du job shop.

Chapitre 2

Job Shop à Deux Jobs avec Contraintes de Disponibilité des Machines

Nous étudions dans ce chapitre la complexité du problème d'ordonnancement de type job shop en considérant seulement deux travaux à ordonnancer, ainsi que des contraintes de disponibilité sur les machines. La deuxième section du chapitre est consacrée au rappel de l'approche géométrique qui permet de résoudre de manière polynomiale le problème du job shop classique à deux jobs. En se basant sur cette approche, nous proposons dans la troisième section un nouvel algorithme polynomial pour le problème de la minimisation du makespan, dans le cas d'opérations strictement non-préemptives. Puis, nous étendons notre approche à la minimisation de tout critère régulier, ainsi qu'à la prise en compte de contraintes additionnelles.

Les résultats développés dans ce chapitre ont été présentés aux conférences internationales CO'02 (International Symposium on Combinatorial Optimization, Paris, 2002) et ECCO XV (XV Conference of the European Chapter on Combinatorial Optimization, Lugano, Switzerland, 2002).

1. Introduction

Nous étudions dans ce chapitre la complexité du problème d'ordonnancement de type job shop à deux jobs avec la prise en compte de contraintes de disponibilité des machines. Comme l'indique l'état de l'art figurant dans le premier chapitre de cette thèse, il n'existe pas à notre connaissance d'étude théorique consacrée à l'ordonnancement de job shops où sont prises en compte des périodes d'indisponibilités des machines. Aussi, proposons-nous dans un premier temps un algorithme de résolution polynomial pour la minimisation du makespan. Cet algorithme est basé sur une extension de l'approche géométrique pour le cas du job shop classique à deux jobs.

Par souci de clarté, nous commençons par décrire en détail, dans la section suivante, l'approche géométrique classique. Cette dernière est un algorithme de complexité polynomiale développé pour la résolution des problèmes d'ordonnancement de type job shop sans la prise en compte de contraintes de disponibilité des machines.

La section 3 de ce chapitre est consacrée à l'étude du problème d'ordonnancement de type job shop à deux jobs avec contraintes de disponibilité, dans le cas d'opérations strictement non-préemptives. Un algorithme polynomial original est proposé pour la minimisation du makespan, puis étendu à la considération de tout critère régulier.

Enfin, nous proposons des extensions de l'algorithme polynomial développé en section 3, de manière à prendre en compte des contraintes additionnelles, à savoir des contraintes de précédence et de disponibilité sur les opérations à ordonnancer.

2. Approche géométrique classique

L'approche géométrique a été proposé pour la première fois par Akers et Friedman [1955]. Elle consiste à réduire la résolution du problème d'ordonnancement du job shop à deux jobs en un problème de recherche de plus court chemin. Sotskov [1985] et Brucker [1988] ont déterminé la complexité exacte de l'approche géométrique pour la minimisation de n'importe quel critère objectif. Nous expliquons dans ce qui suit le principe de l'approche pour la minimisation du makespan.

a) Le problème $J \mid n=2 \mid C_{max}$

Le problème d'ordonnancement de type job shop à deux jobs $J \mid n=2 \mid C_{max}$ se définit de la manière suivante. Un ensemble de m machines doit réaliser deux travaux J_1, J_2. Chaque travail J_i est composé d'une séquence linéaire de n_i opérations $\{O_{i1}, O_{i2}, \ldots, O_{ini}\}$. Les machines ne peuvent réaliser qu'une opération à la fois. Une opération O_{ij} nécessite une seule machine à la fois et dure p_{ij} unités de temps. Par ailleurs, la préemption des opérations n'est pas permise. Résoudre le problème d'ordonnancement revient donc à déterminer les séquences d'entrée des opérations sur les machines de telle sorte que le makespan soit minimal.

Le problème d'ordonnancement peut être représenté dans le plan à deux dimensions avec obstacles (Akers et Friedman [1955]), défini de la manière suivante :

- Chaque axe est décomposé en n_i sous-intervalles rangés selon la gamme opératoire du job qu'il représente. La longueur de l'intervalle I_{ij} est proportionnelle à la durée de l'opération O_{ij} correspondante.

- Le rectangle défini par les intervalles I_{1j} et I_{2k} constitue un obstacle si les deux opérations O_{1j} et O_{2k} partagent la même machine.

- Les bordures supérieure et à droite du rectangle défini par le point d'origine O et le point final F définissant la fin de réalisation des deux jobs constituent l'obstacle final.

Une solution réalisable au problème d'ordonnancement correspond à un chemin allant de l'origine O au point final F. Un tel chemin est composé de segments horizontaux, verticaux et diagonaux. Un segment horizontal (resp. vertical) exprime la progression exclusive du job J_1 (resp. J_2), tandis qu'un segment diagonal modélise l'exécution simultanée des deux jobs (sur des machines différentes). D'autre part, tout chemin doit éviter l'intérieur des obstacles. Ceci s'explique par le fait que deux opérations ne peuvent être exécutées simultanément sur la même machine et ne peuvent être interrompues.

La longueur L d'un chemin, qui correspond à la durée de l'ordonnancement associé, est donnée par :

$$L = L_{(Hor)} + L_{(Vert)} + L_{(Diag)} / \sqrt{2},$$

où $L_{(Hor)}$ est la somme des longueurs des segments horizontaux, $L_{(Vert)}$ celle des segments verticaux et $L_{(Diag)}$ celle des segments diagonaux. Par conséquent, la recherche de l'ordonnancement qui minimise le makespan est équivalent à la recherche du plus court chemin dans le plan.

La figure 2.1 suivante représente, en gras, le plus court chemin dans le plan pour un problème d'ordonnancement à deux jobs définis par les gammes linéaires suivantes :

J_1 : $M_1(1)\ M_2(3)\ M_3(3)$

J_2 : $M_2(2)\ M_3(2)\ M_1(1)$

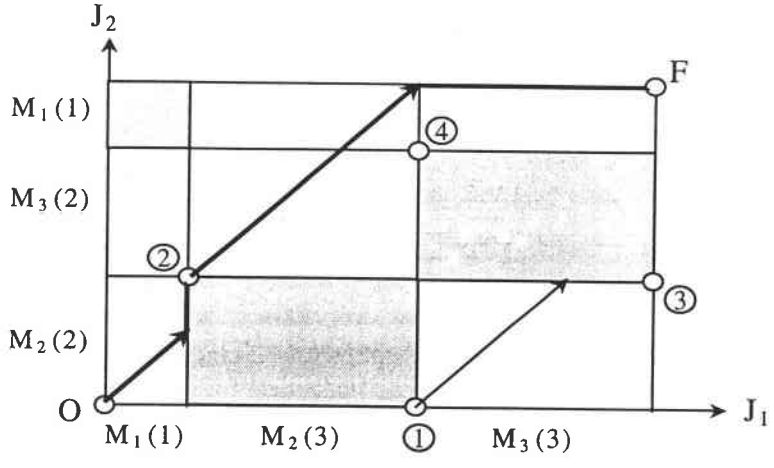


Figure 2.1 : Plus court chemin dans le plan

La recherche du plus court chemin dans le plan défini précédemment peut être transformée en une recherche de plus court chemin dans un réseau acyclique N approprié, ce dernier problème pouvant être résolu en un temps linéaire du nombre d'obstacles.

Le réseau N = (V, E, d) est construit de la manière suivante :

- L'ensemble V des sommets du réseau est composé de l'origine O, du point final F ainsi que des coins sud-est (SE) et nord-ouest (NW) des différents obstacles.

- Chaque sommet $i \in V \setminus \{F\}$ est adjacent à au plus deux arcs partant de i. Ces arcs sont obtenus en progressant diagonalement (dans la direction nord-est) à partir de i jusqu'à ce que la bordure du rectangle défini par les points O et F, ou bien celle d'un obstacle D, soit heurtée :
    - Dans le premier cas, F est l'unique successeur direct de i et l'arc (i, F) est constitué du chemin qui progresse diagonalement depuis i jusqu'à la bordure finale puis qui la longe pour arriver à F (figure 2.2.a).
    - Si un obstacle D est rencontré, alors deux arcs (i, j) et (i, k) sont créés, où j et k désignent respectivement les coins NW et SE de l'obstacle D, ces deux sommets étant les deux successeurs directs de i (figure 2.2.b).

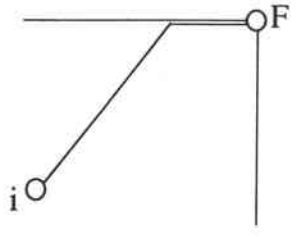


Figure 2.2.a : 1er cas

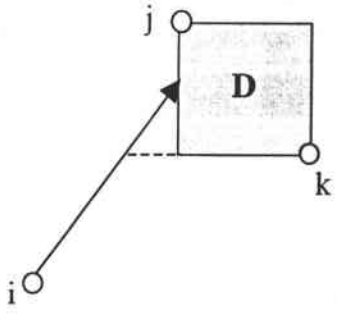


Figure 2.2.b : 2ème cas

Figure 2.2 : Successeurs directs d'un sommet v_i

- La longueur d(i, j) d'un arc (i, j) est égale à longueur de la partie horizontale ou verticale, à laquelle s'ajoute celle de la projection sur un des axes de la partie diagonale.

Un chemin allant de O à F dans le réseau N défini de la sorte correspond à un ordonnancement réalisable pour le problème d'ordonnancement à deux jobs, et la longueur du chemin est égale à la valeur du makespan associé. Ainsi la détermination d'un ordonnancement optimal est équivalente à la recherche d'un chemin O-F de longueur minimale dans le réseau N construit.

La figure 2.3 suivante représente le réseau associé à la recherche du plus court chemin dans le plan qui est donnée par la figure 2.1.

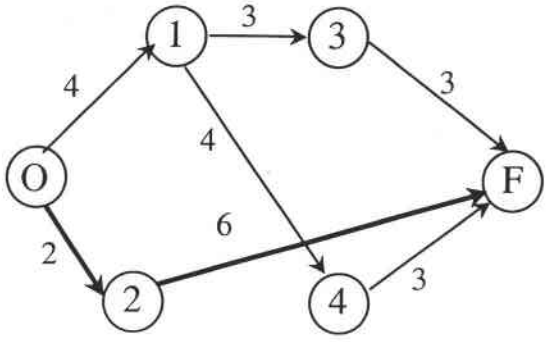


Figure 2.3 : Réseau acyclique

Brucker [1988] a démontré que le réseau N pouvait être construit en O(r log r) étapes, où r est le nombre d'obstacles, égal au plus à $O(n_1\ n_2)$. Par ailleurs, le plus court chemin dans un réseau acyclique peut être obtenu en un temps proportionnel à O(r). Ainsi, la complexité du problème J | n=2 | C_{max} est au plus égale à O(r log r).

b) Le problème J | n=2 | f

Le problème J | n=2 | *f* peut également être résolu de manière polynomiale, pour toute fonction objectif $f(C_1, C_2)$ régulière, où C_i représente la date de fin d'exécution du job J_i. Pour cela, il suffit d'évaluer la fonction objectif chaque fois qu'un chemin heurte la bordure de l'obstacle final défini par l'origine O et le point final F. En effet, ces points de croisement correspondent à la fin de l'exécution de l'un des deux jobs (les réalisations des deux jobs étant achevées en F).

Considérons un chemin P heurtant la bordure droite de l'obstacle final en un point A (figure 2.4). La date de fin d'exécution du job J_1 est égale à la longueur de la partie du chemin C comprise entre les points O et A. Pour calculer C_2, il suffit d'ajouter à C_1 la longueur du segment compris entre A et F.

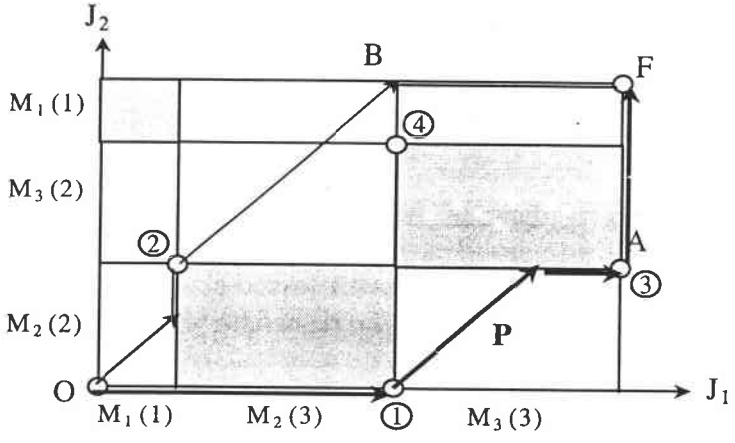


Figure 2.4 : Points de croisement

Notons que les résultats de complexité restent vrais pour le problème de job shop multiprocesseur noté JMPT | n=2 | f, pour lequel la réalisation d'une opération peut nécessiter en même temps plusieurs machines (processeurs) (Brucker [1998]).

c) Cas d'opérations préemptives

Depuis les travaux de Sotskov [1985] et Brucker [1988], nombreuses ont été les généralisations de l'approche géométrique. Sotskov [1991] a proposé une extension de l'approche pour le cas où la préemption des opérations est permise.

Dans cette approche, le réseau N est généralisé pour prendre en compte cette dernière hypothèse, à travers l'ajout de nouveaux sommets et la modification de la définition des successeurs directs d'un sommet.

Plus précisément :

- Si la bordure inférieure d'un obstacle D est heurtée (figure 1.3(a)), le coin sud-est et, cette fois, le point A obtenu en progressant verticalement à l'intérieur de l'obstacle D jusqu'à atteindre la bordure supérieure sont les deux successeurs directs du sommet de départ v_i.
- Si par contre, la bordure gauche de D est heurtée (figure 1.3(b)), le coin nord-ouest ainsi qu'un nouveau point B obtenu en progressant horizontalement à l'intérieur de l'obstacle jusqu'à atteindre la bordure droite sont les deux successeurs de v_i.

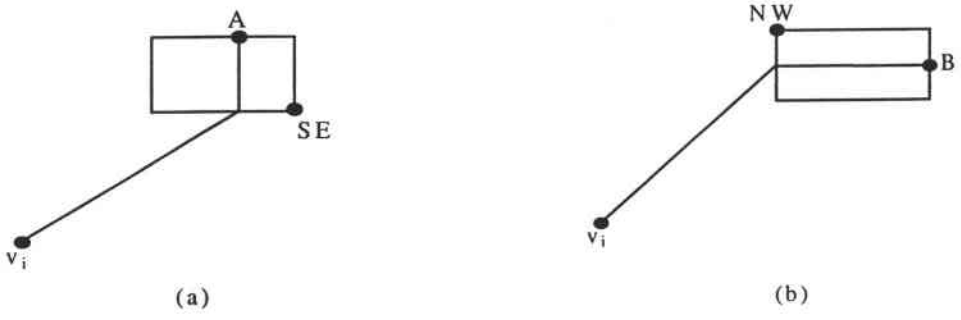


Figure 2.5 : Nouveaux successeurs d'un sommet v_i

Cette nouvelle approche permet de trouver les solutions optimales pour la minimisation de tout critère objectif régulier. Sa complexité est au plus égale à $O(n^3_{max})$, où $n_{max} = \max\{n_1, n_2\}$.

d) Contraintes de précédence et de disponibilité

Brucker et Jurisch [1993] ont proposé une autre extension de l'approche géométrique pour la résolution du problème J | n = 2 ; prec ; r_i | max($c_{ij} + q_{ij}$), où c_{ij} est la date de fin de l'opération O_{ij}, et q_{ij} est la durée de latence associée à l'opération O_{ij}. La complexité de l'approche proposée est au plus $O(N_{max}^3)$, où N_{max} est le nombre d'obstacles, au plus égal à $n_1 n_2$.

Les auteurs ont utilisé cette extension, permettant de prendre en compte des contraintes de précédence et des dates de disponibilités des opérations, pour le calcul des bornes inférieures dans une procédure par séparation et évaluation (Brucker et al. [1994]). Cette dernière a été développée pour résoudre le problème général à plus de deux jobs.

Pour la prise en compte de contraintes de précédence entre les opérations partageant la même machine, la modification réside dans l'interdiction de certains passages lors de la détermination du plus court chemin dans le plan. Pour cela, des barrières sont ajoutées au plan selon les contraintes de précédence à respecter. Ainsi, si l'opération O_{1j} doit être réalisée avant l'opération O_{2k} (ce qui est noté $O_{1j} \prec O_{2k}$), la disposition d'une barrière impose à tout chemin de progresser à droite de l'obstacle défini par les deux opérations (figure 2.6.a). Le cas où $O_{2k} \prec O_{1j}$ est analogue : la progression doit s'effectuer à gauche de l'obstacle (figure 2.6.b).

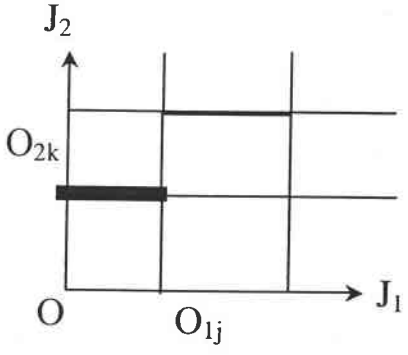


Figure 2.6.a : $O_{1j} \prec O_{2k}$

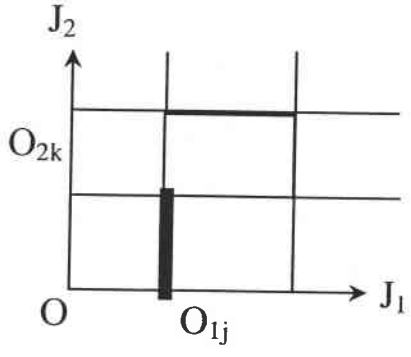


Figure 2.6.b : $O_{2k} \prec O_{1j}$

Figure 2.6 : Contraintes de précédence

Les contraintes de disponibilités des opérations sont quant à elles intégrées à travers l'ajout d'un troisième axe représentant le temps (figure 2.7).

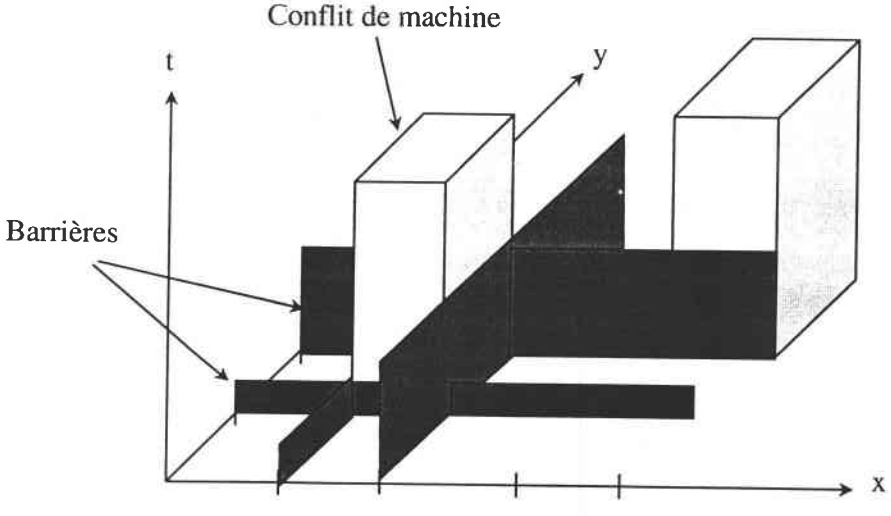


Figure 2.7 : Contraintes de disponibilité des opérations.

Dans ce cas, les obstacles deviennent des pavés que tout chemin doit contourner. Des barrières en deux dimensions sont ajoutées pour modéliser les contraintes de disponibilités des opérations. Un chemin qui heurte une telle barrière doit progresser dans la direction de l'axe des temps jusqu'à atteindre la bordure supérieure de la barrière. En pratique, cela correspond à attendre la disponibilité de l'opération concernée.

e) Modèles plus généraux

Nous terminons cette section en citant les autres extensions de l'approche géométrique utilisées pour la résolution des problèmes d'ordonnancement à deux jobs, dans le cadre de modèles plus généraux que ceux étudiés dans cette thèse :

- Brucker et Schlie [1990] : job shop flexible, pour lequel l'affectation des opérations aux machines n'est pas fixée a priori, dans le cas où les machines capables de réaliser une opération ont la même vitesse. La complexité de l'approche est au plus $O(r^3)$, où r est le nombre d'obstacles. L'approche est applicable à toute fonction objectif régulière.

- Jurisch [1995] : extension du modèle précédent par l'introduction de contraintes de précédence et de disponibilité des opérations, et ceci, pour la minimisation du plus grand retard. Le modèle considéré est noté JMPM | n=2, prec, r_i | L_{max}. La complexité de l'approche est au plus $O(n_{max}^5)$, où $n_{max} = \max(n_1, n_2)$.

- Mati [2001] : job shop multi-ressources avec blocage, pour lequel une opération peut nécessiter plusieurs ressources en même temps, chacune d'elles étant disponible en plusieurs exemplaires (multi-ressources), et où la libération des ressources à la fin d'une opération n'est réalisée que si les ressources nécessaires à l'opération suivante sont toutes disponibles (blocage). La complexité de l'approche est au plus $O(n_1 n_2 (n_1+n_2))$. L'approche est applicable à tout critère régulier.

- Mati et al. [2001] : cas particulier du modèle précédent dans le cas où les ressources sont disponibles en une seule unité, avec la prise en compte de contraintes de précédence.

- Mati et al. [2002] : généralisation du problème étudié par Brucker et Schlie [1990] au cas où les durées opératoires dépendent des machines et ceci, en considérant que l'affectation des opérations aux machines est fixée pour un seul des deux jobs. La complexité de l'approche pour la minimisation de tout critère régulier est au plus $O(k\, n_1^2\, n_2)$, où k est le nombre maximal de machines candidates à l'exécution d'une opération.

- Mati [2002] : job shop flexible avec blocage, dans le cas où les machines ont la même vitesse. La complexité de l'approche pour la minimisation de toute fonction objectif régulière est au plus $O(m_1\, m_2\, (m_1 + m_2))$, où $m_i = K_i \times n_i$, K_i étant le nombre maximal de machines candidates à l'exécution d'une opération du job J_i.

- Mati [2002] : job shop flexible avec blocage dans le cas où les machines n'ont pas la même vitesse et où la flexibilité de l'un des deux jobs est fixée, pour la minimisation de toute fonction objectif régulière. La complexité de l'approche est au plus $O(m_1\, n_2\, (m_1 + n_2))$.

Toutes les études citées ci-dessus sont consacrées à l'optimisation de fonctions objectif régulières. Agnetis et al. [2001] ont étudié le problème du job shop à deux jobs en considérant des critères non-réguliers particuliers : la classe des fonctions quasi-convexes. Les auteurs ont développé un algorithme pseudo-polynomial basé sur l'approche géométrique, de complexité O(rlogr + log H), où r est le nombre maximal d'obstacles et H, la durée opératoire maximale.

3. Job shop à deux jobs avec contraintes de disponibilité des machines

Nous étudions dans cette section la complexité des problèmes d'ordonnancement de type job shop en considérant uniquement deux jobs à ordonnancer ainsi que des contraintes de disponibilité sur les machines. Nous focalisons en premier lieu notre recherche sur le cas d'opérations strictement non-préemptives avec pour objectif la minimisation du makespan. Nous proposons une extension de l'approche géométrique permettant une résolution optimale en un temps polynomial du problème. Les résultats sont ensuite étendus à la considération de toute fonction objectif régulière, et à celle de contraintes supplémentaires.

3.1 Description du problème

Le problème d'ordonnancement de type job shop à deux jobs, avec la prise en compte de contraintes de disponibilité des machines se définit de la manière suivante :

- Un ensemble de m machines doit réaliser deux travaux J_1, J_2.
- Chaque travail J_i est composé d'une séquence linéaire de n_i opérations $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$.
- Chaque machine ne peut réaliser qu'une opération à la fois et chaque opération nécessite une seule machine à la fois, durant p_{ij} unités de temps.
- Les machines peuvent être indisponibles durant certaines périodes, dont les dates et durées sont connues a priori. Nous supposons qu'elles sont soumises à des activités de maintenance préventive.
- Chaque machine peut subir plusieurs tâches de maintenance. Notons K le nombre maximal de tâches de maintenance effectuées sur une même machine.
- 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 tâche de maintenance, ni par celle d'une autre opération.
- L'objectif est de déterminer les séquences d'entrée des opérations sur les machines de telle sorte que le makespan soit minimal.

D'après la notation concernant les contraintes de disponibilités des machines introduite par Schmidt [2000], le problème d'ordonnancement peut être noté J, $N_{Cwin} \mid n=2 \mid C_{max}$, où N_{Cwin} signifie que les périodes d'indisponibilités, c'est-à-dire les tâches de maintenance, sont arbitrairement distribuées sur les machines.

3.2 Approche géométrique temporisée

Nous développons dans ce qui suit une extension de l'approche géométrique décrite dans la section précédente, qui permet de résoudre le problème J, $N_{Cwin} \mid n=2 \mid C_{max}$ de manière exacte et en un temps polynomial.

Cette extension est basée sur la définition et l'introduction de nouveaux sommets, ainsi que sur une nouvelle manière dynamique de progresser d'un sommet aux suivants, permettant la prise en compte de l'évolution du temps et des périodes d'indisponibilité des machines. Nous désignons cette extension par *approche géométrique temporisée*.

3.2.1 Notions de base

Nous commençons par décrire dans cette sous-section, les caractéristiques de l'approche géométrique temporisée, notamment la représentation géométrique utilisée, la caractérisation des sommets et la manière dont sont prises en compte les contraintes de disponibilité des machines.

a) Représentation géométrique

L'approche géométrique temporisée est basée sur le principe de l'approche classique en ce sens qu'elle consiste également à construire un réseau N = (V, E, d) permettant de trouver l'ordonnancement optimal de deux jobs. Les sommets du réseau sont obtenus par une progression non plus dans un plan avec obstacles, mais dans un espace E_T (pour espace-temps) à trois dimensions. Plus précisément, E_T est la réunion disjointe (infinie, dénombrable) de plans parallèles E_t ($t \in N$). Ces plans sont tous identiques et correspondent à la représentation à deux axes et avec obstacles, des jobs à ordonnancer (figure 2.8).

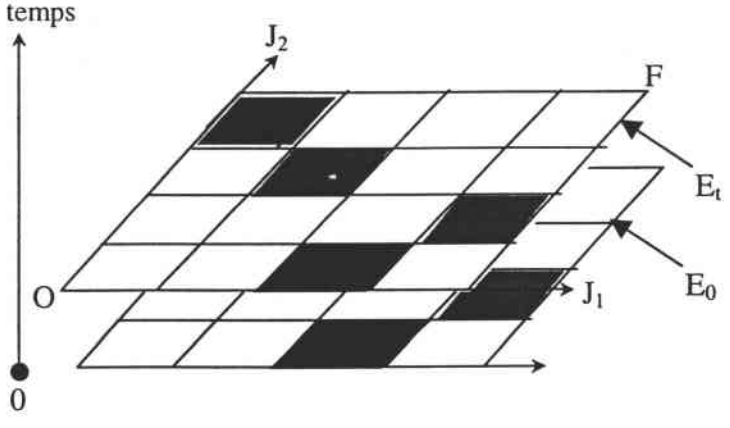


Figure 2.8 : Espace-temps E_T

Dans l'approche géométrique classique les différents sommets générés résultent de la progression de l'un ou des deux jobs. Dans l'approche géométrique temporisée, il se peut qu'un sommet soit obtenu par une progression exclusive du temps. Les opérations des deux jobs à réaliser sont alors en attente, c'est-à-dire que la progression dans un plan avec obstacle E_t est arrêtée, et seul le temps s'écoule, ce qui revient à passer sur un plan E_{t+h} ($h > 0$) de coordonnée temporelle supérieure.

Comme nous allons le montrer dans ce qui suit, le nombre de sauts d'un plan E_t à un plan supérieur E_{t+h} est limité. Aussi pouvons-nous simplifier la représentation en trois dimensions précédente en considérant uniquement la projection P de l'espace-temps E_T sur le plan de coordonnée temporelle nulle. Nous revenons ainsi à la représentation classique à deux axes avec obstacles (figure 2.9). Par abus de langage, nous continuerons à parler de chemins dans le plan avec obstacles P, sous-entendu projections sur le plan P de chemins de l'espace temps E_T.

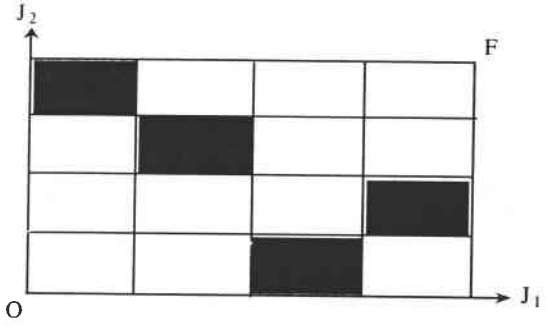
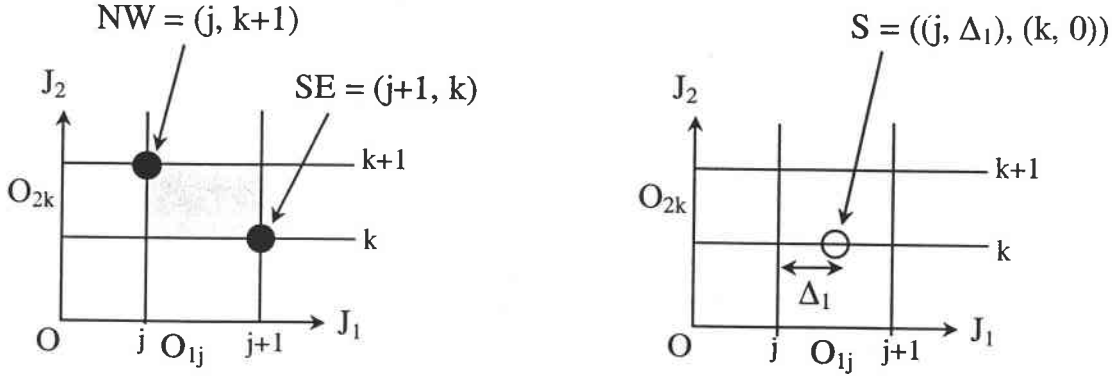


Figure 2.9 : Plan avec obstacles associé à E_T

b) Caractérisation des sommets et définitions

Dans l'approche géométrique classique, les sommets du réseau sont les coins nord-ouest (NW) et sud-est (SE) des obstacles du plan. Ces coins sont par définition situés aux extrémités des intervalles correspondants aux opérations en conflit. Chaque sommet peut donc être défini de manière unique grâce à ses coordonnées dans le plan. L'abscisse x du sommet correspond à l'opération du job J_1 à exécuter, l'ordonnée y du sommet correspond à celle du job J_2 (figure 2.10.a).

Dans le cas de contraintes de disponibilité, certains sommets (projetés sur P) peuvent être situés à l'intérieur des différents intervalles ou plus précisément entre deux lignes délimitant une opération. Nous ajoutons donc à la définition classique une information (pour chaque coordonnée) qui consiste à repérer la durée de la portion de l'opération associée déjà exécutée (figure 2.10.b).



(a) caractérisation classique

(b) nouvelle caractérisation

Figure 2.10 : Caractérisation des sommets

Il est évident que cette nouvelle caractérisation des sommets permet de repérer les sommets classiques que sont les coins nord-ouest et sud-est des obstacles de partage des machines. En effet, un sommet v aura pour coordonnées :

- $v = ((j, 0), (k, 0))$ si v est situé à l'intersection de l'horizontale k et de la verticale j
- $v = ((j, \Delta_1), (k, 0))$ si v est situé sur la ligne horizontale k (entre les verticales j et j+1)
- $v = ((j, 0), (k, \Delta_2))$ si v est situé sur la ligne verticale j (entre les horizontales k et k+1)

Définitions :

Un sommet $v = ((j, \Delta_1), (k, \Delta_2))$ est un *sommet régulier* si v est situé sur la grille définie par les lignes horizontales et verticales, c'est-à-dire si $\Delta_1 = \Delta_2 = 0$.

Un sommet $v = ((j, \Delta_1), (k, \Delta_2))$ est un *sommet singulier* si v est situé hors de la grille, c'est-à-dire si $\Delta_1 = 0$ ou $\Delta_2 = 0$ exclusivement.

Notons que, dans tous les cas, $\Delta_1 \times \Delta_2 = 0$, ce qui signifie que tous les points définis ci-dessus sont situés sur au moins une ligne délimitant une opération.

Par ailleurs, à tout sommet v obtenu par une progression (géométrique ou temporelle) dans le plan avec obstacles P, nous associons une *date au plus tôt*, noté h(v). Cette date au plus tôt correspond à la plus petite durée nécessaire pour atteindre ce sommet depuis l'origine O (par convention, h(O) = 0). La date au plus tôt h(v) ainsi définie est égale à la longueur du plus court chemin allant de O à v, v étant la projection sur le plan P du point (v, h(v)) de l'espace temps E_T.

c) Tests de disponibilités

Afin de prendre en compte les contraintes de disponibilité des machines, des tests sont effectués à chaque exploration de sommet afin de déterminer si les machines nécessaires à l'exécution des opérations suivantes sont disponibles.

Connaissant la date au plus tôt h(v) du sommet v, nous essayons d'exécuter au plus tôt, et si possible en parallèle, les opérations en cours ou à démarrer des deux travaux. Un test de disponibilité dans la direction J_1 consiste alors à déterminer l'instant de disponibilité $T_{1, h(v)}$ de la machine nécessaire à la réalisation de l'opération suivante du job J_1 (figure 2.11). Il en est de même pour le job J_2.

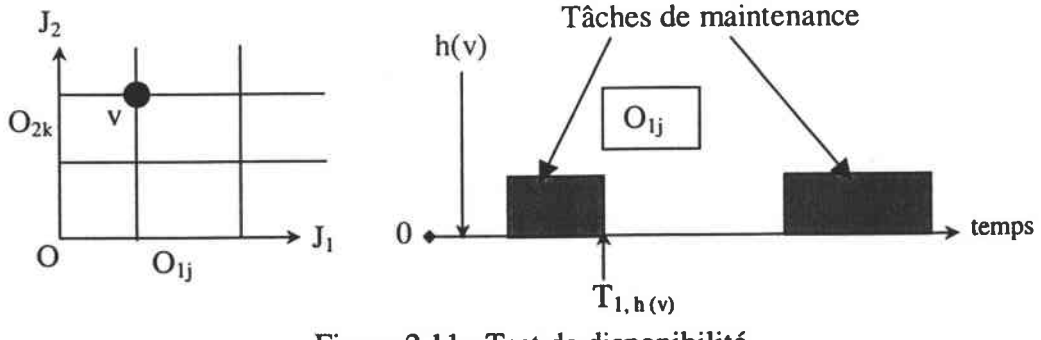


Figure 2.11 : Test de disponibilité

Sur l'exemple illustré par la figure 2.11, l'opération O_{1j} ne peut être exécutée dès l'instant h(v) car elle ne pourrait s'achever avant la première tâche de maintenance prévue sur la machine. Les opérations étant supposées strictement non-préemptives, O_{1j} ne peut démarrer (au plus tôt) qu'à la date $T_{1,\,h\,(v)}$ correspondant à la fin de la tâche de maintenance qui pose problème.

Dans ce qui suit nous allons décrire le principe de fonctionnement de l'approche géométrique temporisée. Plusieurs entités sont donc à définir, à savoir :

- l'ensemble des sommets du réseau,
- la méthode de progression d'un sommet à ses successeurs,
- le calcul de la distance entre deux sommets.

Nous verrons ensuite comment se construit le réseau associé à la recherche de plus court chemin dans le plan avec obstacles P, c'est-à-dire dans quel ordre sont explorés les sommets et à quel moment s'effectue la recherche du plus court chemin.

3.2.2 Principe de fonctionnement

a) Ensemble des sommets

L'ensemble des sommets du réseau construit par notre approche est constitué des trois types de sommets suivants :

- Les sommets *réguliers*, situés à l'intersection des lignes horizontales et verticales, et dont font partie les coins nord-ouest et sud-est des obstacles de partage des machines. Ils correspondent à la fin d'au moins une opération de l'un des travaux.
- Les sommets *singuliers*, situés sur une ligne horizontale (resp. verticale), illustrant le fait que l'exécution de l'opération du job J_1 (rep. J_2) a déjà commencé (cf. figure 2.8.b). Ils correspondent au fait que l'exécution de l'un des jobs se poursuit, tandis que celle de l'autre démarre.
- Les sommets réguliers *d'attentes*, également situés à l'intersection de deux lignes, pour lesquels l'exécution des opérations du job J_1 et J_2 n'a pas encore commencé. Ils correspondent à la fin d'indisponibilité d'une machine devant exécuter un des jobs.

Un sommet singulier est créé si à l'instant courant t, la progression de seulement l'un des deux jobs est possible. Un sommet régulier d'attente (ou sommet d'attente) est quant à lui créé si la progression n'est possible dans aucune direction, c'est-à-dire pour aucun des deux jobs. Dans une représentation à trois dimensions (c'est-à-dire dans un espace-temps E_T), la création d'un sommet d'attente correspond à un saut d'un plan Et à un plan de coordonnée temporelle supérieure. Mais comme nous choisissons de nous restreindre à la représentation en deux dimensions sur le plan P (P étant la projection de Et sur le plan de coordonnée temporelle nulle), un sommet d'attente est une duplication du sommet régulier ayant les mêmes coordonnées géométriques.

Notons que les créations de sommets d'attente sont les seules situations pour lesquelles la progression réelle, c'est-à-dire dans l'espace-temps E_T, est une progression exclusivement temporelle. En outre, la duplication de sommets du plan P permet de modéliser ces situations particulières. La progression dans le plan avec obstacles, que nous détaillons dans ce qui suit, est donc fidèle à la progression qui s'effectue en réalité dans l'espace-temps E_T.

b) Méthode de progression

Considérons un sommet $v = ((j, \Delta_1), (k, \Delta_2))$, et sa date au plus tôt $h(v)$. Nous commençons par développer le cas où v est un sommet régulier ($\Delta_1 = \Delta_2 = 0$), c'est-à-dire que l'exécution des opérations O_{1j} du job J_1 et O_{2k} du job J_2 n'a pas encore commencé, puis nous étudierons la progression à partir d'un sommet singulier.

$b_1)$ $\Delta_1 = \Delta_2 = 0$

Des tests de disponibilité sont effectués sur les opérations des deux jobs à exécuter. En fonction des résultats de ces tests, plusieurs cas sont à distinguer quant à la détermination des successeurs de v :

<u>1er cas</u> :

Dans le cas où des problèmes de disponibilité se posent dans les deux directions, c'est-à-dire si les opérations des deux jobs ne peuvent démarrer à l'instant $h(v)$ car les ressources associées sont ou vont devenir (trop tôt) indisponibles, le sommet v est dupliqué. Un sommet d'attente $v_a = ((j, 0), (k, 0))$ est ainsi crée, et v_a est l'unique successeur de v. A partir de v_a, la progression est possible dans au moins une direction et $h(v_a)$ est égal au premier instant auquel une des deux opérations peut démarrer.

<u>2ème cas</u> :

Si le problème de disponibilité se pose uniquement dans la direction J_1 (resp. J_2), la progression s'effectue alors le long de la ligne verticale (resp. horizontale), ce qui revient à réaliser les opérations du job J_2 (resp. J_1). Cette progression s'arrête dans les deux cas suivants :

- Soit la progression du job J_1 (resp. J_2) n'est plus possible car un problème de disponibilité survient (cf. 1er cas), lors d'un changement d'opération de J_1.
- Soit la machine nécessaire à la réalisation de l'opération O_{1j} (resp. O_{2k}) devient disponible.

Sans perte de généralité, supposons que c'est le job J_1 qui pose problème (la méthode de progression étant analogue pour le problème de disponibilité sur le job J_2). L'instant de disponibilité $T_{1, h(v)}$ de la machine qui doit exécuter O_{1j} est avant tout calculé. L'opération O_{2k} peut quant à elle être exécutée, ce qui correspond à une progression verticale.

Tant que l'instant de disponibilité $T_{1,\,h(v)}$ du job J_1 n'est pas atteint, la progression verticale se poursuit au fil des opérations du job J_2. A chaque fois que le chemin croise une ligne horizontale, un test de disponibilité est effectué afin de déterminer si l'opération $O_{2k'}$ (k'> k) du job J_2 délimitée par cette ligne horizontale peut démarrer :

- Si l'opération $O_{2k'}$ ne peut pas démarrer et si $T_{1,\,h(v)}$ n'est pas atteint, la progression verticale s'arrête et le sommet d'attente $v'_a = ((j, 0), (k', 0))$ est ajouté comme unique successeur de v (figure 2.12).

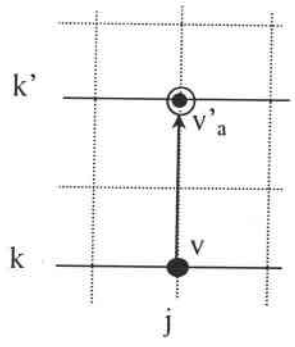
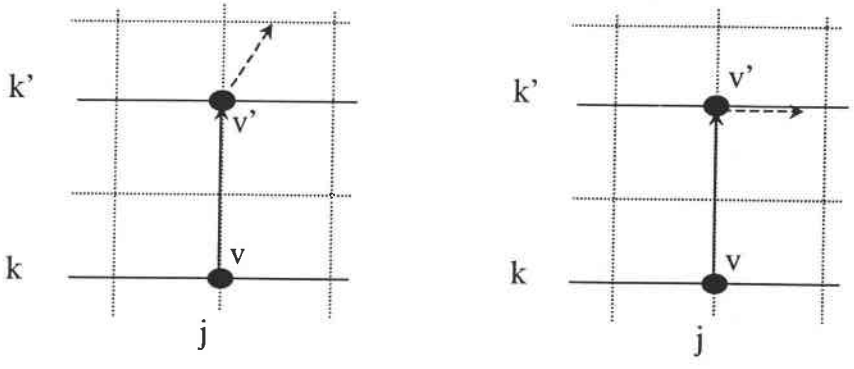


Figure 2.12 : Sommet d'attente comme successeur de v

- Si $T_{1,\,h(v)}$ est exactement atteint, que l'opération $O_{2k'}$ puisse démarrer ou non, la progression verticale s'arrête et le sommet régulier v' = ((j, 0), (k',0)) est ajouté comme unique successeur de v. A partir de v', la progression s'effectuera dans la direction diagonale, si l'exécution de $O_{2k'}$ peut démarrer (figure 2.13 (a)) ou dans la direction du job J_1, sinon (figure 2.13 (b)).



(a) progression diagonale à partir de v' (b) progression horizontale

Figure 2.13 : Sommet régulier comme successeur de v

- Si l'opération $O_{2k'}$ peut démarrer et si $T_{1,\,h(v)}$ doit être atteint après Δ_2' unités de temps d'exécution de l'opération $O_{2k'}$, la progression verticale s'arrête au sommet singulier $s = ((j, 0), (k', \Delta_2'))$. Le sommet régulier v' = ((j, 0), (k',0)) est alors ajouté comme unique successeur de v, et le sommet singulier s est ajouté dans les successeurs de v' (figure 2.14).

Notons qu'à partir du sommet singulier s, la progression peut s'effectuer dans la direction diagonale (l'exécution de l'opération O_{1j} du job J_1 démarre et celle de l'opération $O_{2k'}$ se poursuit).

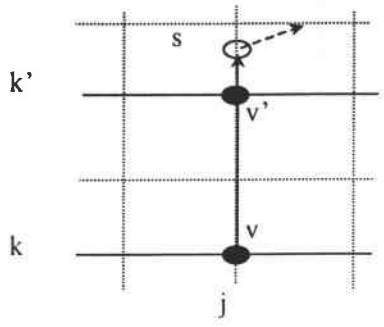


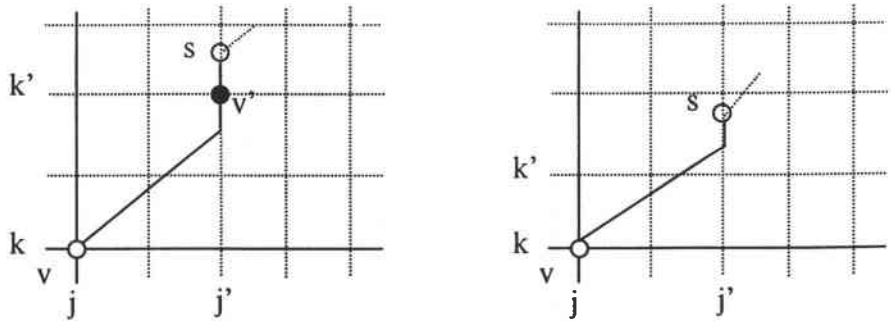
Figure 2.14 : Sommet régulier suivi d'un sommet singulier

3ème cas :

Si aucun problème de disponibilité ne se pose, c'est-à-dire si les opérations des deux jobs peuvent être réalisées dès l'instant h(v), la progression s'effectue dans la direction diagonale. Bien entendu, des tests de disponibilité sont effectués dans la direction du job J_1 (resp. J_2, resp. les deux) à chaque fois que la diagonale rencontre une ligne verticale (resp. une ligne horizontale, resp. un point de croisement), et ce, afin de guider la progression.

La progression diagonale est arrêtée dans les cas suivants :

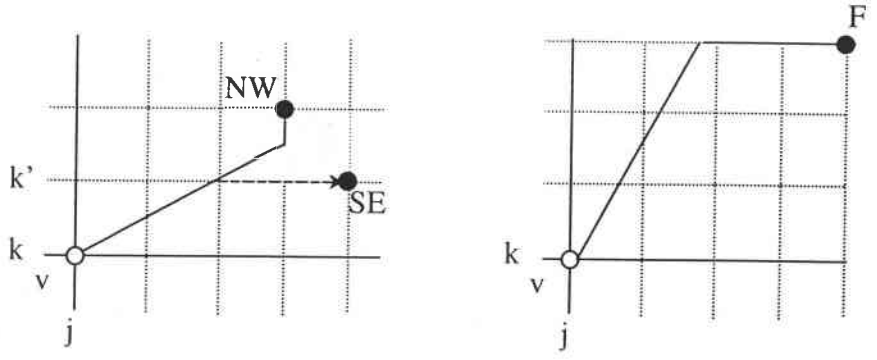
- Si un problème de disponibilité se pose dans une seule direction, nous sommes ramenés au 2ème cas, c'est-à-dire que la progression ne se fait plus que dans le long d'une horizontale ou d'une verticale. Un sommet régulier (figure 2.15 (a)) ou un sommet singulier (figure 2.15 (b)) est ajouté comme unique successeur de v. Notons que cette deuxième situation se produit lorsque la progression dans la direction problématique (exécution du job J_1 sur la figure 2.15 (b)) peut reprendre pendant l'exécution de l'opération du job qui ne pose pas problème (opération $O_{2k'}$ du job J_2 sur la figure 2.15 (b)).



(a) sommet régulier (b) sommet singulier

Figure 2.15 : Changement de direction

- Si des problèmes de disponibilité surviennent dans les deux directions, nous sommes renvoyés au premier cas et le sommet d'attente sur lequel s'arrête la progression est ajouté comme seul successeur de v.
- Si un obstacle de partage de machine est rencontré, comme dans le cas classique, les coins nord-ouest et sud-est de l'obstacle sont ajoutés comme successeurs de v (figure 2.16 (a)).
- Si l'obstacle final est heurté par la diagonale, le point F est, comme pour l'approche géométrique classique, l'unique successeur de v (figure 2.16 (b)).



(a) obstacle de partage de machine (b) obstacle final

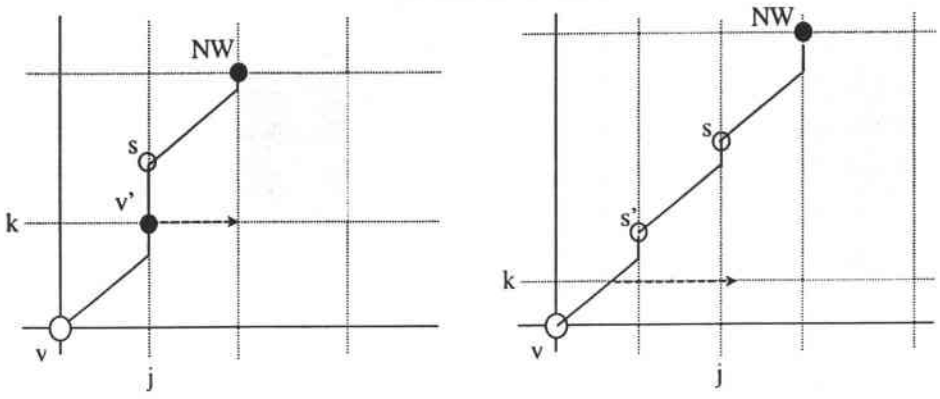
Figure 2.16 : Obstacle rencontré par la diagonale

Il est à noter que dans les deux situations ci-dessus, aucun sommet singulier (intermédiaire) n'est nécessaire dans la mesure où l'exécution de toute opération n'est démarrée que si elle peut s'achever avant une période d'indisponibilité. Néanmoins, des attentes peuvent survenir sur le parcours allant aux coins des obstacles ou au point final F.

Nous avons examiné tous les modes de progression depuis un sommet régulier. Dans le cas d'un sommet d'attente, qui correspond à une duplication d'un sommet régulier, la majorité de ces règles restent valable. La seule différence réside dans le fait qu'à partir d'un sommet d'attente, la progression dans au moins une direction est toujours possible. En d'autres termes, le 1^{er} cas ne peut pas se produire.

$b_2)$ Δ_1 ou $\Delta_2 \neq 0$

Considérons maintenant la progression à partir d'un sommet singulier $s = ((j, \Delta_1), (k, \Delta_2))$. Sans perte de généralité, supposons que s est situé sur une ligne verticale ($\Delta_1 = 0$). A partir de s la progression s'effectue forcément dans la direction diagonale (l'opération O_{2k} est en cours d'exécution, tandis que l'opération O_{1j} démarre). Les règles de progression du $3^{ème}$ cas de la partie b_1) s'appliquent, excepté dans les situations particulières où un obstacle de partage de machine est heurté avant que ne se termine l'opération O_{2k} (figure 2.17).



(a) prédécesseur immédiat (b) prédécesseur lointain

Figure 2.17 : Situations particulières

Sur les exemples illustrés par la figure 2.17, la diagonale partant de s heurte la bordure gauche d'un obstacle. Le coin nord-ouest est automatiquement ajouté comme successeur direct de s. Cependant, le coin sud-est ne peut être obtenu à partir de s. En revanche, il peut être atteint à partir d'un prédécesseur de s (le sommet v' pour l'exemple (a) et v pour l'exemple (b)).

La règle qui s'applique donc à ces situations particulières consiste à ajouter le coin sud-est comme successeur du prédécesseur $p = ((j_p, \Delta_{1p}), (k_p, \Delta_{2p}))$ de s vérifiant l'une des conditions suivantes :

(i) $\Delta_{2p} = 0$ et $k_p = k$
(ii) $k_p < k$

La première condition désigne un point p situé sur la même ligne que le coin sud-est (SE), la seconde, un point situé sur une ligne en dessous de celle où se trouve SE. Notons que pour tous les prédécesseurs p de s, l'abscisse j_p est forcément inférieure à celle de SE. En d'autres termes ses deux conditions suffisent à trouver un prédécesseur de s à partir duquel il est possible d'atteindre SE (dans le pire des cas, p est l'origine O).

Le procédé est analogue pour les coins nord-ouest si le sommet singulier $s = ((j, \Delta_1), (k, \Delta_2))$ est sur une ligne horizontale ($\Delta_2 = 0$) et si un obstacle est heurté (forcément sur sa bordure inférieure) avant la fin de l'exécution de l'opération O_{1j}.

c) Calcul des distances

Soit $v = ((j, \Delta_1), (k, \Delta_2))$ un sommet quelconque appartenant à un chemin du plan avec obstacles, et soit $v' = ((j', \Delta_1'), (k', \Delta_2'))$ un de ses successeurs.

c_1) Distance entre deux sommets de mêmes coordonnées géométriques

Si v' est le sommet v dupliqué, alors la distance entre v et v' correspond à l'attente de la disponibilité de l'un des deux jobs, ce qui est donnée par la formule suivante :

$$d(v, v') = \text{Min}\ \{T_{1v}, T_{2v}\} - h(v)$$

c_2) Distance entre deux sommets de coordonnées géométriques différentes

Sans perte de généralité, supposons que v et v' sont situés sur des lignes horizontales, c'est-à-dire que $\Delta_2 = \Delta_2' = 0$ (figure 2.18).

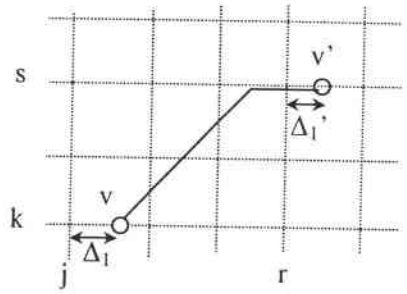


Figure 2.18 : Distance entre deux sommets

Excepté les cas où le sommet v' est le coin sud-est d'un obstacle ou bien le point final F, la distance d(v, v') entre v et v' est donnée par la formule générale suivante :

$$d(v, v') = \sum_{x=j}^{r-1} P_{1x} - \Delta_1 + \Delta_1'$$

- La quantité $s_p = \sum_{x=j}^{r-1} P_{1x}$ représente la distance entre les lignes verticales j et r.
- Si v (resp. v') est un sommet singulier, il est nécessaire d'ôter (resp. d'ajouter) à la distance d, la quantité Δ_1 (resp. Δ_1'), Δ_1 et Δ_1' étant égales à 0 si v et v' sont des sommets réguliers.

Si v' est le coin sud-est d'un obstacle (figure 2.19), la formule précédente n'est pas directement applicable dans la mesure où un problème de disponibilité peut se poser entre les coins sud-ouest (SW) et sud-est (SE) de l'obstacle (ce dont ne tient pas compte la quantité s_p). Il faut donc ajouter à la formule de calcul des distances la quantité $T_{1SW} - h(SW)$, qui est égale à 0 si le passage de SW à SE s'effectue sans attente de disponibilité.

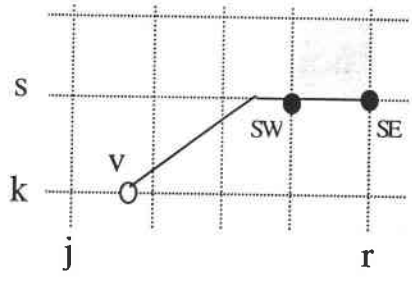


Figure 2.19 : Distance à un coin sud-est

La distance entre v et SE est donc donnée par la formule suivante :

$$d(v, SE) = \sum_{x=j}^{r-1} p_{1x} - \Delta_1 + (T_{1sw} - h(sw))$$

Enfin, si le sommet v' est le sommet final F, des attentes peuvent être nécessaires à chaque croisement du chemin longeant l'obstacle final avec des lignes verticales (figure 2.20).

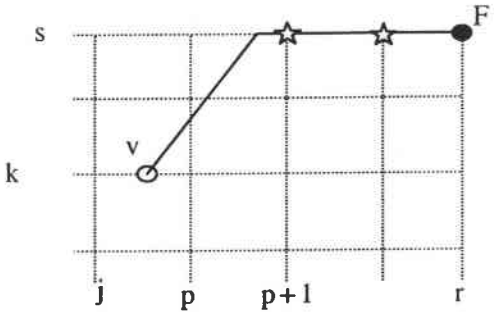


Figure 2.20 : Distance au point final F

Dans ce cas-là la distance entre v et F est donnée par la formule :

$$d(v, F) = \sum_{x=j}^{r-1} p_{1x} - \Delta_1 + \sum_{i=p+}^{r-1} (T_{1i} - h(i)),$$

où : $p^+ = \begin{cases} p & \text{si le chemin heurte l'obstacle final sur la ligne verticale p,} \\ p+1 & \text{si le chemin heurte l'obstacle final entre les lignes p et p+1.} \end{cases}$

Dans le cas où les sommets v et v' considérés sont situés sur des lignes verticales k et s (c'est-à-dire si $\Delta_1 = \Delta_1' = 0$), il suffit de remplacer dans la première formule la quantité $\sum_{x=j}^{r-1} P_{1x}$ par $\sum_{y=k}^{s-1} P_{2y}$ qui exprime la distance entre les lignes horizontales k et s, et (Δ_1, Δ_1') par (Δ_2, Δ_2') soit :

$$d(v, v') = \sum_{y=k}^{s-1} P_{2y} - \Delta_2 + \Delta_2'$$

La formule permettant de calculer la distance entre un point v quelconque et le coin nord-ouest d'un obstacle est :

$$d(v, NW) = \sum_{y=k}^{s-1} P_{2y} - \Delta_2 + (T_{2sw} - h(sw))$$

Enfin, la distance entre v et l'obstacle final F est égale à :

$$d(v, F) = \sum_{y=k}^{s-1} P_{2y} - \Delta_2 + \sum_{i=q+}^{s-1} (T_{2i} - h(i)),$$

où : $q^+ = \begin{cases} q & \text{si le chemin heurte l'obstacle final sur la ligne verticale q,} \\ q+1 & \text{si le chemin heurte l'obstacle final entre les lignes q et q+1.} \end{cases}$

3.2.3 Construction du réseau N = (V, E, d)

Nous avons décrit dans ce qui précède la méthode de progression depuis un sommet. A chaque fois qu'un nouveau sommet v' (c'est-à-dire un successeur d'un sommet v exploré) est généré, il est ajouté à l'ensemble V des sommets du réseau. Un arc liant les deux sommets est créé et le poids de cet arc est égal à la distance d(v, v'). La question qui se pose alors est la suivante : Etant donnés plusieurs sommets du réseau qui n'ont pas encore été explorés, dans quel ordre doit-on les considérer ?

a) Sélection du sommet à explorer

Le premier sommet à explorer est bien sûr le point d'origine O. Ses successeurs sont alors générés et ajoutés à l'ensemble V des sommets à explorer. Les sommets de V sont rangés selon un ordre dépendant de leurs coordonnées dans le plan avec obstacles.

Considérons deux sommets $v = ((j, \Delta_1), (k, \Delta_2))$ et $v' = ((j', \Delta_1'), (k', \Delta_2'))$ appartenant à V. Le sommet v sera rangé avant v' dans l'ensemble V si l'une des conditions suivantes est vérifiée :

(i) $j < j'$ et $k < k'$: v est situé à gauche et en dessous de v'.

(ii) $j < j'$ et $k = k'$: v est situé à gauche de v' sur le plan.

(iii) $k < k'$ et $j = j'$: v est situé en dessous de v' sur le plan.

(iv) $j = j'$, $k = k'$ et ($\Delta_1 < \Delta_1'$ ou $\Delta_2 < \Delta_2'$) : v est un sommet régulier, v' un singulier.

Il se peut également que toutes les coordonnées des points v et v' soient égales. Cela se produit si v est un sommet régulier et v' son sommet dupliqué. Dans ce cas, v est logiquement rangé avant v' dans l'ensemble V.

Enfin, les sommets v et v' sont dits *incomparables* si leurs coordonnées vérifient la relation :

$j < j'$ et $k > k'$ (ou de manière symétrique : $j > j'$ et $k < k'$)

Dans ce cas, le sommet v est situé à gauche de v' mais également au-dessus, dans le plan avec obstacles. Par exemple, les coins sud-est et nord-ouest d'un même obstacle sont incomparables. Il est impossible d'établir un ordre entre deux sommets incomparables v et v'.

Cependant, cela n'est pas problématique dans la mesure où il est impossible d'atteindre le sommet v depuis v' et inversement (à moins d'effectuer une progression en arrière, ce qui créerait des cycles dans le réseau). En revanche, deux sommets incomparables seront insérés dans l'ensemble V en fonction des coordonnées des autres sommets de l'ensemble.

L'ensemble V des sommets à explorer étant partiellement ordonné, le premier sommet de V est celui qui est sélectionné à chaque étape de la construction du réseau. Ce sommet est alors retiré de l'ensemble et les successeurs du sommet, s'ils n'existent pas déjà, sont insérés dans V.

b) Détermination du plus court chemin

Avant l'exploration de tout sommet v, c'est-à-dire avant d'appliquer les règles de progression à v afin d'en déterminer les successeurs, la valeur de sa date au plus tôt h(v) est calculée. Pour ce faire, nous recherchons parmi tous les prédécesseurs v_i de v le sommet tel que la valeur $h_i = h(v_i) + d(v, v_i)$ soit minimale. Cela revient à déterminer, avant de progresser à partir du sommet v, le plus court chemin partant de l'origine O et menant à v. En outre, h(v) est égale à la longueur de ce plus court chemin O-v.

Cette détermination de plus court chemin durant la construction du réseau présente deux avantages : d'une part, cela permet de garantir l'optimalité des chemins empruntés à chaque étape, et d'autre part, cela évite de faire une recherche de plus court chemin après la construction du réseau. En effet, lorsque la progression s'arrête au point final F, le plus court chemin est déjà complètement identifié et sa longueur est égale à h(F).

3.3 Résultats de complexité

Dans la sous-section précédente, nous avons décrit le principe de notre approche de résolution pour le problème d'ordonnancement J, $N_{Cwin} \mid n = 2 \mid C_{max}$. Aussi, nous étudions dans ce qui suit la complexité de l'approche proposée.

L'algorithme 2.1 suivant permet de construire le réseau N = (V, E, d) associé à la recherche de plus court chemin.

Algorithme 2.1 : 2-jobs

Données : Deux jobs et les indisponibilités des machines

Résultat : Ordonnancement optimal et le makespan associé

Etape 1 : *Initialisation*

- $V = \{O = (1, 0), (1, 0))\}$ /* *ensemble ordonné des sommets non explorés* */
- $E = \varnothing$ /* *ensemble des arcs créés* */

Etape 2 : *Construction*

Tant que $V \neq \{F\}$ **Faire**

2.1. Sélectionner le premier sommet s de V et calculer h(s).

2.2. Enlever s de V : $V = V \setminus \{s\}$.

2.3. Appliquer la méthode de progression pour obtenir les successeurs :

Pour chaque successeur v_i **Faire**

- Insérer v_i dans l'ensemble ordonné V : $V = V \cup \{v_i\}$.
- Ajouter l'arc (s, v_i) dans E : $E = E \cup \{(s, v_i)\}$.

Fin Pour.

Fin Tant que.

Etape 4 :
- La valeur du makepsan est donné par la date au plus tôt h(F) de F.
- L'ordonnancement optimal est obtenu par chaînage arrière.

Théorème 2.1.

L'ensemble des sommets et la valuation des arcs construits par l'algorithme *2-jobs* est suffisant pour déterminer un plus court chemin dans le plan, et par conséquent une solution optimale pour le problème d'ordonnancement J, $N_{Cwin} \mid n = 2 \mid C_{max}$.

Preuve.

Il suffit de montrer qu'un plus court chemin dans le plan avec obstacles P est constitué seulement par une succession d'arcs du réseau construit par l'algorithme.

Soit $C_{opt} = (v_0 = O, v_1, v_2, \ldots, v_s = F)$ le chemin optimal construit par l'algorithme 2.1, avec $v_i = (x_i, y_i, h(v_i))$ et considérons un plus court chemin C. Nous supposons que C est choisi de telle sorte que la partie commune entre C_{opt} et C soit maximale. Soit $v = v_k$ le premier point de C_{opt} tel que l'arc (v, v') n'appartient au réseau construit par notre algorithme, v' étant le successeur de v dans le chemin C. Notons que dans le pire des cas, $v = O$. Nous allons montrer que $v = F$.

Si k = s, alors la preuve est terminée. Sinon, k < s, et l'arc (v, v') (et éventuellement le sommet v') n'appartient pas au réseau construit par notre algorithme (figure 2.21).

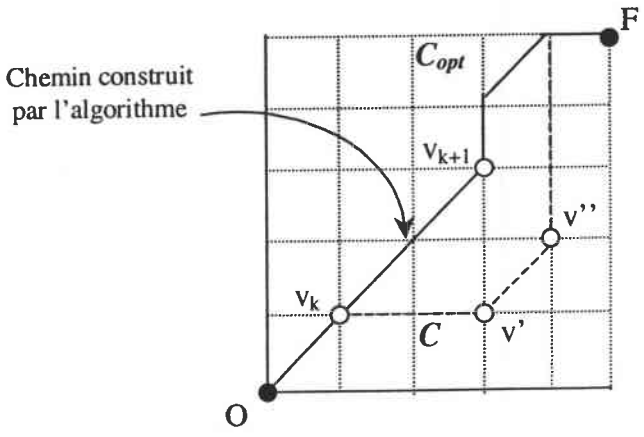


Figure 2.21 : Plus courts chemins

Nous montrons dans ce qui suit que, quelle que soit la nature du successeur v_{k+1} de v_k dans le réseau, le plus court chemin C peut être transformé en un chemin de longueur inférieure ou égale, mais ayant un arc du réseau supplémentaire. Cela permet de contredire l'hypothèse de maximalité sur l'indice k. Et par conséquent v_k ne peut être le premier point du chemin C_{opt} tel que l'arc (v_k, v') n'appartient pas au réseau construit par notre approche

En répétant ce raisonnement (en considérant v_{k+1} et son successeur v'' dans le chemin C, etc.) nous atteignons ainsi le point F et nous pouvons conclure qu'un plus court chemin est uniquement constitué d'une succession d'arcs appartenant au réseau construit par l'algorithme 2.1.

Cas 1 : le successeur v_{k+1} de v_k (dans le réseau) est un coin SE ou NW.

Ce cas se présente si la progression diagonale à partir du sommet v_k ne peut être interrompue par des tâches de maintenance, mais par un obstacle D représentant le partage de machines entre deux opérations. Le chemin C peut alors passer à droite ou à gauche de l'obstacle D. Sans perte de généralité, nous supposons que C passe à droite de l'obstacle D. Soit t le point de croisement de C avec la ligne horizontale H passant par le coin SE de l'obstacle D (figure 2.22).

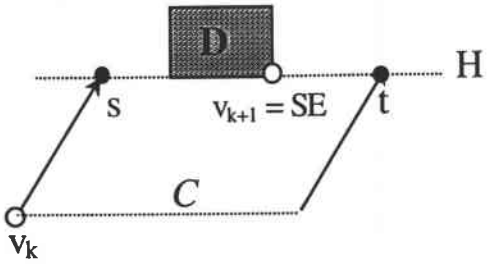


Figure 2.22 : v_{k+1} est un coin sud-est

En remplaçant la partie de C située entre le sommet v_k et le point t par le chemin $v_k \rightarrow s \rightarrow SE \rightarrow t$, où s est le point de croisement entre la diagonale partant de v_k et l'horizontale H, nous obtenons un nouveau chemin qui comporte un arc appartenant au réseau (v_k, SE) en plus. De plus, la longueur de $v_k \rightarrow s \rightarrow SE \rightarrow t$ dans le réseau, est égale à d(v_k, SE) + d(SE, t), qui est une borne inférieure sur la distance entre v_k et t, du fait que la réalisation du job J_1 n'est interrompue que par d'éventuelles tâches de maintenance entre SE et t.

Par conséquent, un nouveau plus court chemin ayant un arc du réseau en plus que C est obtenu, ce qui contredit l'hypothèse de maximalité sur le nombre d'arcs communs entre C et C_{opt}.

Dans le cas où le chemin C passe à gauche de l'obstacle D, la preuve est similaire mais en se basant sur le coin NW de l'obstacle.

Cas 2 : le successeur de v_k est un coin régulier v_{k+1} (différent d'un coin NW ou SE).

Par définition, v_{k+1} est situé à l'intersection d'une ligne horizontale H et d'une ligne verticale V. Si le chemin C passe à droite de v_{k+1} alors C traverse l'horizontale H en un point t_h (figure 2.23). En remplaçant la partie entre v_k et t_h par le chemin $v_k \rightarrow v_{k+1} \rightarrow t_h$, nous obtenons un nouveau chemin avec un arc du réseau en plus, et ce, sans altérer la valeur du makespan, car les opérations du job J_1 entre v_k et t_h, sont exécutées au plus tôt. Ceci est une contradiction avec l'hypothèse de maximalité sur le nombre d'arcs consécutifs de C appartenant au réseau.

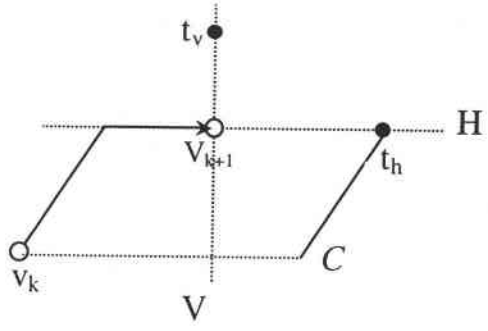


Figure 2.23 : v_{k+1} est un sommet régulier

Si C passe à gauche du point v_{k+1}, la preuve est similaire en considérant cette fois le point de croisement t_v du chemin C avec la verticale V.

Cas 3 : le successeur de v_k est un sommet d'attente v_a.

Le sommet v_a ne peut pas être le sommet dupliqué de v_k. En effet, si v_a est le sommet obtenu en dupliquant v_k, cela signifie qu'en v_k les deux jobs sont indisponibles. Par conséquent, la progression du chemin C à partir de v_k doit au moins attendre la disponibilité de l'un des deux jobs. Cette attente est exactement exprimée par l'arc (v_k, v_a) dans le réseau. Donc, l'arc (v_k, v_a) appartient à C, ce qui contredit l'hypothèse que v_k est le premier sommet de C_{opt} à partir duquel C et C_{opt} divergent.

Le sommet v_a est donc forcément obtenu à la suite d'une progression depuis v_k. Le chemin C peut alors passer à droite ou à gauche de v_a. La preuve, similaire au cas d'un sommet régulier est omise.

Cas 4 : le successeur de v_k est un coin singulier s.

Sans perte de généralité, nous supposons que le coin singulier s se trouve sur une horizontale H. Si le chemin C traverse l'horizontale H en un point t se trouvant sur la droite du sommet s (figure 2.24), alors en remplaçant la partie entre v_k et t par le chemin $v_k \rightarrow u \rightarrow s \rightarrow t$ (u étant le point de croisement de la diagonale partant de v_k avec l'horizontale H), nous obtenons un nouveau chemin avec un arc du réseau en plus, et ce, sans altérer la valeur du makespan, d'où la contradiction.

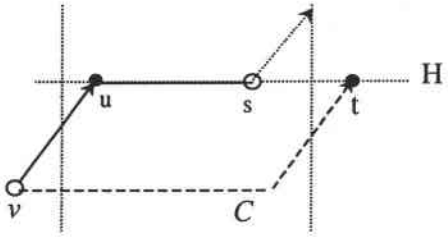


Figure 2.24 : Le chemin C passe à droite de s

Supposons maintenant que le chemin C passe à gauche du point s. Dans ce cas, nous raisonnons par rapport à la verticale V se trouvant juste à droite de s.

Si la progression diagonale à partir de s croise directement la verticale V en un point l_1, alors le chemin C croise nécessairement V en un point c_1 situé au dessus de l_1 (figure 2.25). Dans ce cas, avec les mêmes arguments que pour les cas précédents, nous aboutissons à une contradiction sur la maximalité de C (l'arc (l_1, c_1) peut être ajouté).

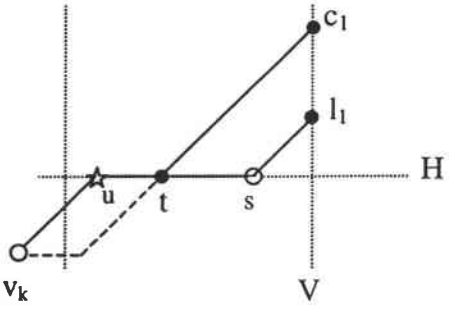


Figure 2.25 : La progression diagonale heurte directement V

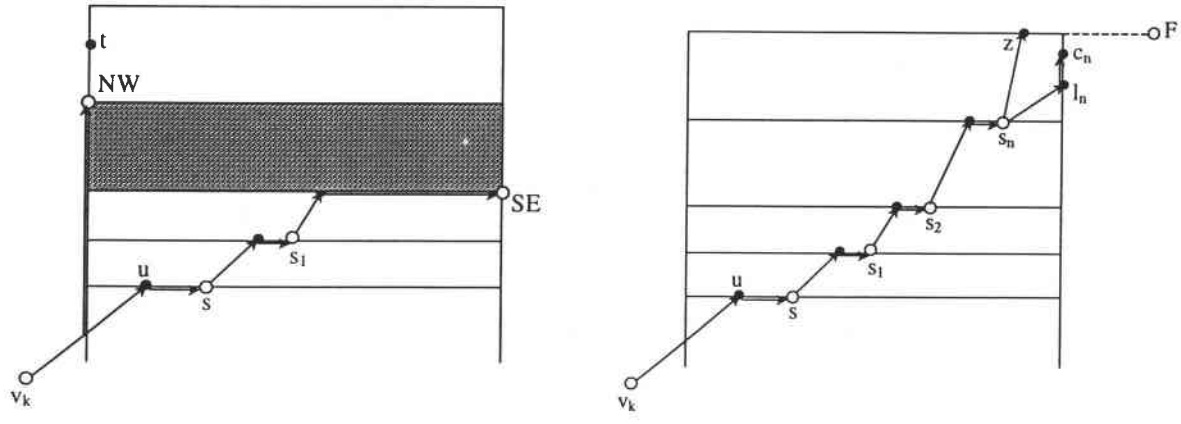
Sinon, la progression diagonale à partir de s s'arrête à cause de la rencontre d'un obstacle (figure 2.26.a) ou un problème de disponibilité sur le job J_2 (figure 2.26.b).

Dans le premier cas, le chemin C ne passe pas à droite de l'obstacle en question car sinon l'hypothèse de maximalité sur C serait mise en cause. En effet, puisque le chemin C doit dans ce cas heurter l'obstacle et passer par le coin SE, un arc de réseau peut être ajouté à C, d'où la contradiction. Donc, C passe à gauche de l'obstacle et l'utilisation de la preuve du Cas 1, en raisonnant sur le coin NW, permet encore d'aboutir à une contradiction.

Dans le cas d'un problème de maintenance sur le job J_2, un nouveau sommet s_1 est ajouté comme successeur direct de s. Soit le raisonnement ci-dessus s'applique à s_1, soit un sommet s_2 est ajouté comme successeur direct de s_1 etc.

En répétant ce raisonnement, nous arrivons à un point s_n à partir duquel la progression diagonale heurte la bordure supérieure de l'obstacle finale en un point z ou bien la verticale V en un point l_n :

- Dans le premier cas, en remplaçant la partie de C entre v et z par la succession d'arcs $v_k \rightarrow s \rightarrow s_1 \rightarrow \ldots \rightarrow s_n \rightarrow z$ nous arrivons à une contradiction. Notons que le point z est considéré car le chemin C croise nécessairement la bordure supérieure de l'obstacle finale en un point à gauche de z.
- Dans le deuxième cas, la considération de la verticale V et la succession d'arc $v_k \rightarrow s \rightarrow s_1 \rightarrow \ldots \rightarrow s_n \rightarrow l_n \rightarrow c_n$, où l_n (resp. c_n) est le point de croisement de la diagonale (chemin C) partant de s_n avec V, conduit à une contradiction.



(a) obstacle rencontré (b) problèmes de disponibilité

Figure 2.26 : La progression diagonale s'arrête

Notons que le cas où le coin singulier se trouve sur la même horizontale que le sommet v_k, est un cas dégénéré du cas traité précédemment et la preuve qui est similaire est omise.

En conclusion, dans tous les cas de figure, la maximalité du nombre d'arcs consécutifs communs aux chemins C et C_{opt} est mise en cause. Par contradiction, il n'existe pas de premier point v_k à partir duquel les chemins C et C_{opt} divergent et le chemin C est seulement composé d'une succession d'arcs du réseau N.

CQFD.

Théorème 2.2.

Le problème d'ordonnancement J, $N_{Cwin} \mid n = 2 \mid C_{max}$ est polynomial et sa complexité est au plus égale à $O(Ks^4)$,

où K est le maximum sur toutes les machines du nombre tâche de maintenance,
et $s = \max \{n_1, n_2\}$.

Preuve.

La recherche du plus court chemin de l'origine O à un sommet v dans le réseau étant calculée avant que le sommet v soit exploré, la complexité de l'approche dépend du temps nécessaire pour explorer tous les sommets.

Les sommets du réseau sont les coins réguliers, singuliers et d'attente. Une borne supérieure sur le nombre de coins singuliers qu'on peut générer est $O(n_1 \times n_2)$ qui représente le nombre d'obstacles possibles. Le nombre de coins d'attente pouvant être généré par l'exploration définie par l'algorithme *2-jobs* est proportionnel à $O(n_1 \times n_2)$ représentant le nombre de croisements possibles entre les lignes horizontales et verticales. Cependant, pour calculer une borne supérieure sur les sommets singuliers, nous devons calculer le nombre de sommets générés à partir d'un sommet quelconque $v = ((x, \Delta_1), (y, \Delta_2))$. A partir de v, avant d'obtenir le point F, au pire des cas, la diagonale passe successivement par une verticale et une horizontale, et sur chaque passage, nous pouvons ajouter au plus un sommet. Ainsi, le sommet v peut générer au plus $(n_1 - x) + (n_2 - y)$ sommets qui est borné par $(n_1 + n_2)$.

Par conséquent, la cardinalité de V est plus égale à $(n_1 \times n_2)\ (4 + n_1 + n_2)$. Le nombre de tests de disponibilité effectué à partir d'un sommet est au plus égal à $(n_1 + n_2) + K \max\{n_1, n_2\}$ (le premier terme correspondant aux passages diagonaux, le second aux passages horizontaux et verticaux).

Ainsi, le nombre d'opérations élémentaires effectuées par l'algorithme est pire des cas égal $(K\ s^4)$ où $s = \max\{n_1, n_2\}$.

3.4 Extensions de l'approche

3.4.1 Fonctions objectif régulières

L'algorithme précédent peut être modifié pour prendre en compte des fonctions objectif régulière $f(C_1, C_2)$ où C_i est la date de fin du job J_i. La façon de construire le réseau reste la même. La seule modification concerne le calcul de la valeur du critère chaque fois que la bordure supérieure ou à droite de l'obstacle finale est heurtée. Pour ce faire, nous considérons tout arc (x, y) du réseau pour lequel le passage diagonal heurte l'obstacle final en un point A.

Si A se trouve sur la bordure droite alors :

$$C_1 = PCC(x) + d(x, A), \text{ et}$$
$$C_2 = C_1 + d(A, F).$$

Si par contre le point A se trouve sur la bordure supérieure alors :

$$C_2 = PCC(x) + d(x, A), \text{ et}$$
$$C_1 = C_2 + d(A, F).$$

Puisque d'une part le calcul des distances entre A et F prend un temps proportionnel à $K \max\{n_1, n_2\}$, et d'autre part le nombre d'arcs (x, y) heurtant l'obstacle final est au pire des cas égal au nombre de sommets du réseau qui est proportionnel à s^3. Nous en déduisons alors le résultat suivant :

Théorème 2.3.

Le problème d'ordonnancement J, $N_{Cwin} \mid n = 2 \mid f$ est polynomial pour toute fonction objectif régulière et sa complexité est au plus $O(K\, s^4)$.

3.4.2 Prise en compte de contraintes additionnelles

L'approche proposée pour la résolution du problème d'ordonnancement J, $N_{Cwin} \mid n = 2 \mid f$, peut être modifiée de manière à prendre en compte des contraintes additionnelles, à savoir des contraintes de précédence, ainsi que des dates de disponibilité sur les opérations.

a) Contraintes de précédence

Pour l'intégration de contraintes de précédence, il est possible d'appliquer à notre approche la modification introduite par Brucker et Jurisch [1993] pour l'approche géométrique classique. Nous rappelons qu'elle consiste à interdire certains passage lors de la progression des chemins par l'ajout de barrières dans le plan (cf. figure 2.6).

Mais puisque notre méthode de résolution effectue des tests de disponibilité à chaque exploration de sommet, nous pouvons directement intégrer les contraintes de précédence dans ces tests. Ainsi, si pour deux opérations en conflit O_{1j} et O_{2k} l'ordre $O_{2k} \prec O_{1j}$ est imposé (c'est-à-dire que O_{2k} doit être réalisée avant O_{1j}), il suffit de calculer, dans un premier temps, la date de disponibilité T_{2v} du job prioritaire. Dans un second temps, le test de disponibilité dans la deuxième direction est réalisé (pour déterminer T_{1v}) non pas à partir de la date au plus tôt h(v), mais à partir de la valeur $T = T_{2v} + p_{2k}$, où v est le sommet régulier situé au croisement des lignes j et k. De cette façon, l'exécution de l'opération O_{1j} ne peut démarrer qu'après que O_{2k} est achevée.

b) Dates de disponibilité

De la même manière, il est facile de prendre en compte des contraintes de disponibilité sur les opérations. Pour cela, il suffit d'effectuer les différents tests de disponibilité de notre approche, non pas à partir de la date au plus tôt h(v) de tout sommet v considéré, mais plutôt à partir de la dates de disponibilité des opérations concernées (figure 2.27).

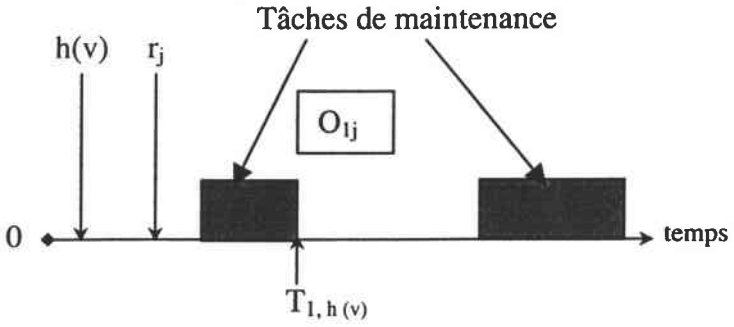


Figure 2.27 : Ajustement de la date au plus tôt

La modification à apporter à notre méthode de résolution consiste donc à ajuster la valeur de départ avant tout test de disponibilité dans une direction. Pour une opération O_{1j} du job J_1, le test de disponibilité au point $v = ((j, 0), (k, \Delta_2))$ s'effectuera à partir de la valeur h_1 telle que : $h_1 = \text{Max } \{h(v), r_j\}$.

Notons que cette modification ne fait pas augmenter la complexité de l'algorithme que nous proposons, contrairement à l'introduction d'un troisième axe (de manière permanente) dans l'approche géométrique classique (cf. figure 2.7) qui fait croître considérablement le nombre de chemins possible.

Nous pouvons donc énoncer le résultat suivant :

Théorème 2.4.

Le problème d'ordonnancement J, $N_{Cwin} \mid n = 2, prec, r_i \mid f$ est polynomial pour toute fonction objectif régulière et sa complexité est au plus $O(K\, s^4)$.

Notons que dans le cas où les machines sont continuellement disponibles, l'approche que nous proposons permet de résoudre le problème $J \mid n = 2, prec, r_i \mid f$ en au plus $O(s^4)$ étapes, ce qui améliore de manière significative le résultat proposé par Brucker et Jurisch [1993] (leur approche étant en $O(s^6)$).

3.5 Exemple

Nous terminons ce chapitre par un exemple simple (noté E1) à 4 machines illustrant l'application de l'approche géométrique temporisée pour la résolution d'un problème d'ordonnancement de type job shop à deux jobs avec contraintes de disponibilité. L'objectif est la minimisation du makespan.

Soient deux jobs J_1 et J_2, disponibles respectivement aux dates $r_1 = 3$ et $r_2 = 4$, définis par les gammes opératoires suivantes :

J_1 : $M_1(2)\ M_2(3)\ M_3(2)\ M_4(2)$

J_2 : $M_3(3)\ M_2(1)\ M_4(2)\ M_1(2)$

Et considérons les périodes d'indisponibilité :

- M_1 : [12, 14], qui signifie que la machine M_1 est indisponible entre les dates 12 et 14,
- M_2 : [4, 6]
- M_3 : [10, 13]
- M_4 : [7,9].

La figure 2.28 suivante présente le plus court chemin obtenu par application de l'approche géométrique temporisée à l'exemple considéré.

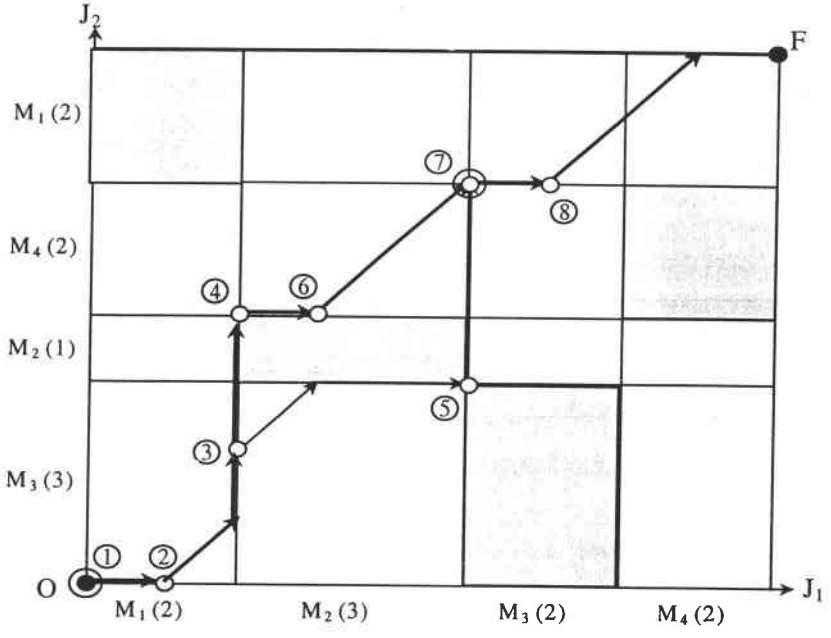


Figure 2.28 : Plus court chemin pour l'exemple E1

Le premier sommet considéré est l'origine O. Les jobs J_1 et J_2 n'étant pas disponibles à l'instant 0, le sommet O est dupliqué. Un sommet d'attente, noté 1, est créé et sa date au plus tôt est égale à Min $(r_1, r_2) = 3$.

A l'instant 3, la progression ne peut s'effectuer que dans la direction horizontale. La première opération de J_1 est réalisée jusqu'à la date 4 à laquelle le job J_2 devient disponible. Le sommet singulier 2 (avec $h(2) = 4$) est donc créé. A partir de 2, la progression s'effectue dans la direction diagonale (jusqu'à la fin de la première opération du job J_1), puis horizontale (la machine nécessaire à l'exécution de la deuxième opération de J_1 étant indisponible jusqu'à la date 6). Un nouveau sommet singulier 3 est ajouté comme successeur du sommet 2. Sa date au plus tôt est $h(3) = 6$.

La progression diagonale à partir de 3 heurte un obstacle. Les coins nord-ouest (sommet 4) et sud-est (sommet 5) sont alors ajoutés comme successeurs du sommet 2. Les dates au plus tôt associées sont $h(4) = 8$ et $h(5) = 9$.

Le sommet 4 est ensuite exploré. La machine 4 étant indisponible à l'instant 8 et jusqu'à l'instant 9, un sommet singulier 6 (avec h(6) = 9) est ajouté comme (unique) successeur de 4.

Avant l'exploration du sommet 6, il faut revenir à celle du sommet 5. Puisque seule l'opération du job J_2 peut démarrer à la date h(5) = 9, la progression s'effectue dans la direction verticale. Un sommet 7 est alors ajouté comme successeur de 5 et ce sommet est atteint à la date 12. Notons que ce sommet est créé dans la mesure où la progression verticale n'est plus possible (la machine M_1 étant indisponible à la date 12).

L'exploration suivante est celle du sommet 6. La progression diagonale partant de ce sommet aboutit au sommet 7 déjà existant. Par ce chemin, le sommet 7 est atteint à la date 11.

Le sommet 7, dont la date au plus tôt est fixée à 11 est ensuite exploré. Les deux jobs étant indisponibles, le sommet est dupliqué. Un sommet d'attente encore noté 7 et dont la date au plus tôt est 13 est ainsi créé.

La progression s'effectue alors dans la direction du job J_1 jusqu'à la date de disponibilité du job J_2, en l'occurrence 14. Le sommet singulier 8, avec h(8) = 14, est ajouté comme unique successeur de 7.

A partir du sommet singulier 8, la progression diagonale aboutit au point final F et sa date au plus tôt qui est égale à la valeur du makespan est h(F) = 17.

Le réseau associé à la recherche du plus court chemin dans le plan avec obstacles de la figure 2.28 est représenté sur la figure 2.29 suivante :

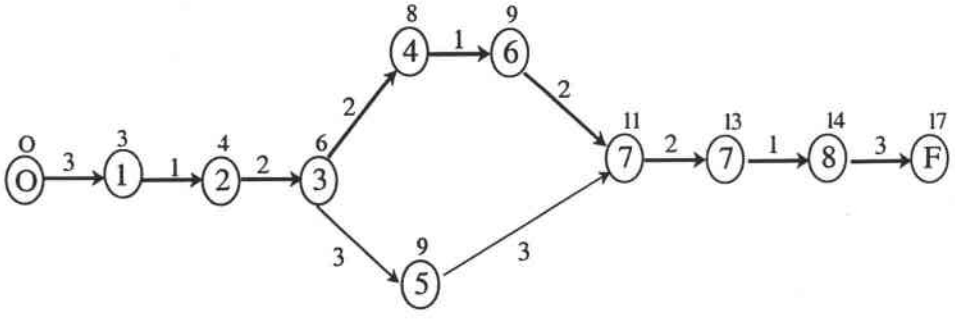


Figure 2.29 : Réseau associé à la recherche du plus court chemin

Le plus court chemin trouvé est C_{opt} = (O, 1, 2, 3, 4, 6, 7, 8, F). Il est à noter que le chemin C = (O, 1, 2, 3, 5, 7, 8, F) est aussi optimal. De plus, ce dernier comporte moins de sommets que le chemin *Copt*. Cependant, le chemin obtenu par l'algorithme 2-jobs est tel que tous ses sommets sont atteints au plus tôt (notamment le sommet 7), c'est-à-dire tel que toute opération se termine au plus tôt. Nous verrons dans le chapitre 4 de cette thèse l'intérêt de garantir l'optimalité des chemins intermédiaires liant l'origine à tout point du chemin C_{opt}.

4. Conclusion

Nous avons étudié dans ce chapitre la complexité du problème d'ordonnancement de type job shop à deux jobs, dans le cas où sont prises en compte des périodes d'indisponibilité des machines.

Après avoir décrit l'approche géométrique classique qui permet de résoudre le problème du job shop à deux jobs sans contraintes de disponibilité, nous avons proposé une extension de l'approche géométrique à notre problème. Cette extension est basée d'une part sur une nouvelle caractérisation des sommets, sur l'introduction de sommets additionnels, et surtout, sur la prise en compte du temps durant l'ordonnancement. Pour cela, des tests devant être effectués à chaque étape de la progression de l'ordonnancement, ont été introduits pour prendre en compte les périodes d'indisponibilité des machines.

L'algorithme polynomial en fonction du nombre d'opérations et du nombre d'indisponibilités que nous avons développé permet de résoudre de manière optimale, le problème d'ordonnancement du job shop à deux jobs, dans le cas d'opérations strictement non-préemptives, et ce, pour tout critère régulier. Par ailleurs, la méthode proposée permet d'intégrer aisément d'autres contraintes, et notamment des dates de disponibilité des opérations.

Dans les chapitres suivants, nous utilisons cet algorithme polynomial dans l'élaboration de méthodes de résolution approchées (au chapitre 3) et exactes (au chapitre 4) des problèmes généraux, c'est-à-dire à plus de deux jobs.

Chapitre 3

Méthodes de Résolution Approchées

Ce troisième chapitre est consacré au développement de méthodes de résolution approchées pour les problèmes d'ordonnancement généraux, c'est-à-dire à plus de deux jobs, du flow shop et du job shop dans lesquels sont prises en compte des contraintes de disponibilité des machines. La deuxième section du chapitre est dédiée au modèle d'atelier de type flow shop. Deux variantes du problème strictement non-préemptif y sont étudiées. La première est caractérisée par des périodes d'indisponibilité fixes, la seconde se place dans le contexte où les tâches de maintenance correspondant aux indisponibilités des machines doivent apparaîtrent dans des fenêtres temporelles. Deux heuristiques permettant la minimisation de tout critère régulier, pour chacune des variantes, sont présentées. La troisième section de ce chapitre est consacrée au modèle d'atelier de type job shop. Les indisponibilités y sont supposées fixes et les deux méthodes de résolution proposées dans la deuxième section sont généralisées.

Une partie des résultats développés dans ce chapitre a été présentée aux conférences internationales ORP³ (Operational Research Peripatetic Postgraduate Programmme, Paris, 2001) et IEEE SMC'01 (System Man and Cybernetics, Tucson, Arizona, 2001). Ceux de la deuxième section font, en outre, l'objet d'un article accepté pour publication dans un numéro spécial de EJOR (European Journal of Operational Research), dédié à la conférence ORP³.

1. Introduction

Dans le chapitre précédent, nous avons étudié la complexité du problème d'ordonnancement de type job shop à deux jobs avec la prise en compte de contraintes de disponibilité des machines. Un algorithme polynomial a été proposé pour la minimisation de toute fonction objectif régulière, et ce, dans le cas d'opérations strictement non-préemptives. A partir de la résolution par cet algorithme polynomial de sous-problèmes à deux jobs, nous développons dans ce troisième chapitre des méthodes approchées pour les problèmes d'ordonnancement généraux, c'est-à-dire à plus de deux jobs, du flow shop et du job shop.

La section suivante du chapitre est dédiée au système d'atelier de type flow shop. Nous y étudions deux variantes du problème strictement non-préemptif. Dans la première de ces variantes, qui est communément étudiée dans la littérature, les périodes d'indisponibilité sont supposées connues à l'avance et fixes. Deux approches de résolution sont proposées : la première est basée sur un algorithme de liste, la seconde utilise la résolution de sous-problèmes à deux jobs au moyen de l'algorithme polynomial développé au chapitre précédent. Afin d'en améliorer les performances, ces deux approches sont couplées à des méta-heuristiques. Dans la seconde variante du problème étudiée, les tâches de maintenance correspondant aux indisponibilités des machines sont supposées flexibles, ce qui signifie que leurs positions ne sont pas fixées a priori et peuvent être optimisées. Les heuristiques proposées dans le cas de tâches de maintenance fixes sont alors modifiées de manière à prendre en compte cette dernière hypothèse. Des résultats d'expériences permettant de comparer les performances des différentes approches de résolution sont ensuite commentés.

Dans la troisième section de ce chapitre, nous considérons le problème d'ordonnancement général de type job shop avec la prise en compte de contraintes de disponibilité des machines, dans le cas où les périodes d'indisponibilité sont fixes. Les méthodes de résolution proposées à la deuxième section, à savoir l'algorithme de liste et l'heuristique basée sur l'ordonnancement de paires de jobs, sont généralisées et des expériences sont menées pour attester de l'efficacité des approches proposées.

2. Problème de type flow shop

Nous étudions dans cette section le problème d'ordonnancement de type flow shop avec la présence de périodes d'indisponibilité sur les machines. Comme l'indique l'état de l'art dressé dans le premier 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.

Dans ce qui suit, nous supposons que les périodes d'indisponibilité sont dues à des interruptions planifiées des machines selon un calendrier régissant par exemple des pauses de personnel ou des activités de maintenance. Nous considérons en premier lieu que les arrêts des machines sont connus à l'avance et fixes, puis nous introduisons, en second lieu, une nouvelle hypothèse permettant d'optimiser les activités liées à la maintenance, dans laquelle les positions des différentes tâches de maintenance sont supposées flexibles.

2.1 Tâches de maintenance fixes

Nous étudions dans cette sous-section le problème d'ordonnancement de type flow shop strictement non-préemptif, en supposant, comme dans la majorité des ouvrages traitant de contraintes de disponibilité dues à une activité de maintenance, que les tâches de maintenance sont connues à l'avance et fixes. Après une présentation du problème, nous présentons deux heuristiques permettant sa résolution. Si l'objectif considéré dans la présentation est la minimisation du makespan, il est cependant possible d'appliquer les approches proposées à la considération de tout critère régulier. La première de ces heuristiques est une approche gloutonne qui consiste à placer les jobs l'un après l'autre selon l'ordre exprimé par une séquence de traitement. La seconde heuristique est basée sur la résolution de plusieurs problèmes à deux jobs, au moyen de l'algorithme polynomial développé dans le deuxième chapitre de la thèse.

2.1.1 Description du problème et propriétés

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

- Un ensemble de n travaux $\mathfrak{I} = \{J_1, J_2, \ldots, J_n\}$ doit être réalisé par un ensemble de m machines $M = \{M_1, M_2, \ldots, M_m\}$.
- Chaque travail J_i, composé d'une séquence linéaire 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 opération à la fois et chaque opération O_{ij} nécessite une seule machine à la fois, durant p_{ij} unités de temps.
- Plusieurs périodes d'indisponibilité correspondant à des tâches de maintenance préventive peuvent se produire sur chacune des machines. Les durées et dates de réalisation de ces tâches sont connues à l'avance et fixes.
- 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 tâche de maintenance, ni par celle d'une autre opération.
- L'objectif est de déterminer les séquences de passage des opérations sur les machines de telle sorte que le makespan soit minimal.

D'après la notation concernant les contraintes de disponibilité des machines introduite par Schmidt [2000], le problème d'ordonnancement peut être noté F, $N_{Cwin} \parallel C_{max}$, où N_{Cwin} signifie que les périodes d'indisponibilité sont arbitrairement distribuées sur les machines.

Le problème d'ordonnancement classique du flow shop, c'est-à-dire sans indisponibilité des machines, étant NP-difficile au sens fort dès que le nombre de machines est supérieur à deux, il en résulte que le problème que nous étudions ici est également NP-difficile au sens fort. D'autre part, il a été montré par Kubiak et al. [2002] qu'il n'existait pas d'heuristique à performance garantie pour la minimisation du makespan dans un flow shop à deux machines si plus d'une indisponibilité par machine étaient considérées. Ce résultat, qui souligne la difficulté de notre problème, nous a incité à élaborer des méthodes de résolution approchées.

Des résultats de dominance existent pour les problèmes d'ordonnancement du flow shop classique. En effet, pour les problèmes à deux (et même à trois) machines, nous savons qu'il est suffisant de rechercher une solution optimale parmi les ordonnancements de permutation, c'est-à-dire tels que la séquence de traitement des jobs est la même sur toutes les machines (règle de Johnson [1954]). Pour un flow shop à *m* machines avec comme fonction objectif le makespan, il est suffisant de considérer les ordonnancements tels que la séquence de passage des jobs est la même sur les deux premières machines, ainsi que sur les deux dernières Campbell, Dudeck et Smith [1970].

Cependant, l'introduction de périodes d'indisponibilité sur les machines infirme ces règles de dominance. La proposition 3.1 (resp. 3.2) suivante montre que la règle de Johnson (resp. CDS) ne s'applique plus en présence de contraintes de disponibilité.

Proposition 3.1

Les ordonnancements de permutation ne sont pas dominants pour le problème d'ordonnancement F2, $N_{Cwin} \parallel C_{max}$, dès lors qu'il existe au moins une période d'indisponibilité par machine.

Preuve

Considérons une instance E_1 à trois jobs avec les durées opératoires du tableau 3.1 suivant:

Tableau 3.1 : Durées opératoires

	M_1	M_2
J_1	1	2k+2
J_2	k	2
J_3	k+1	1

et supposons qu'il existe une période d'indisponibilité $[S_{j1}, T_{j1}]$ par machine, avec les dates de début S_{j1} et de fin T_{j1} (j = 1, 2) suivantes :

M_1: [k+1, k+2],

M_2: [2k+4, 2k+5],

où k>2.

Pour cette instance, l'unique solution optimale consiste à ordonnancer les jobs selon l'ordre (J_1, J_2, J_3) sur la première machine, puis (J_1, J_3, J_2) sur la deuxième machine, pour un makespan égal à 2k+7 (figure 3.1). Tout autre ordonnancement conduit à un makespan supérieur.

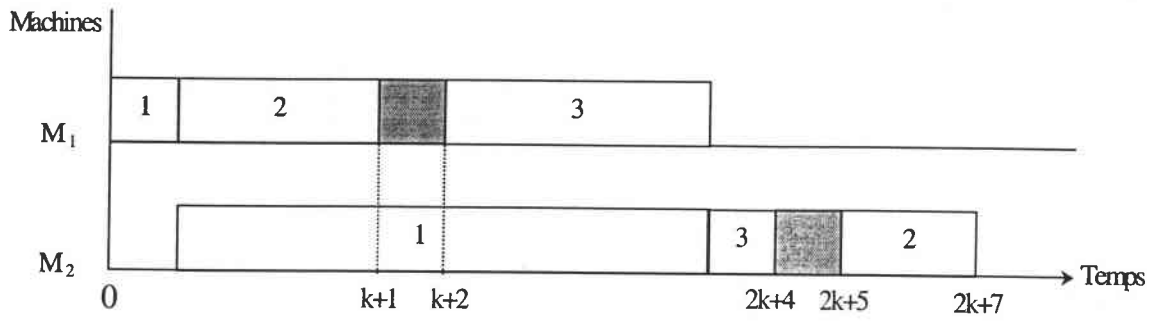


Figure 3.1 : Solution optimale pour l'instance E_1

Proposition 3.2

Les ordonnancements de permutation ne sont pas dominants pour le problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$. De plus, il se peut qu'il n'existe aucune solution optimale telle que le même ordre soit observé sur les deux premières machines M_1 et M_2, ainsi que sur les deux dernières M_{m-1} et M_m, dès lors qu'il existe au moins une période d'indisponibilité sur les deux premières et sur les deux dernières machines.

Preuve

Considérons l'instance E_2 à 3 machines et 3 jobs dont les durées opératoires sont données par le tableau 3.1 suivant:

Tableau 3.2 : Durées opératoires

	M_1	M_2	M_3
J_1	1	2k+2	3
J_2	k	4	1
J_3	k+1	1	1

Nous supposons que toutes les machines ont une seule période d'indisponibilité dont les dates sont les suivantes :

M_1: [k+1, k+2], qui signifie que la machine M_1 est indisponible entre les dates k+1 et k+2,

M_2: [2k+4, 2k+5],

M_3: [2k+5, 2k+6],

avec k>2.

Pour cette instance, l'unique solution optimale consiste à ordonnancer les jobs selon les ordres (J_1, J_2, J_3) sur la machine M_1, puis (J_1, J_3, J_2) sur la machine M_2 et finalement (J_3, J_1, J_2) sur la machine M_3, pour un makespan égal à 2k+10. Tout autre ordonnancement conduit à un makespan plus long.

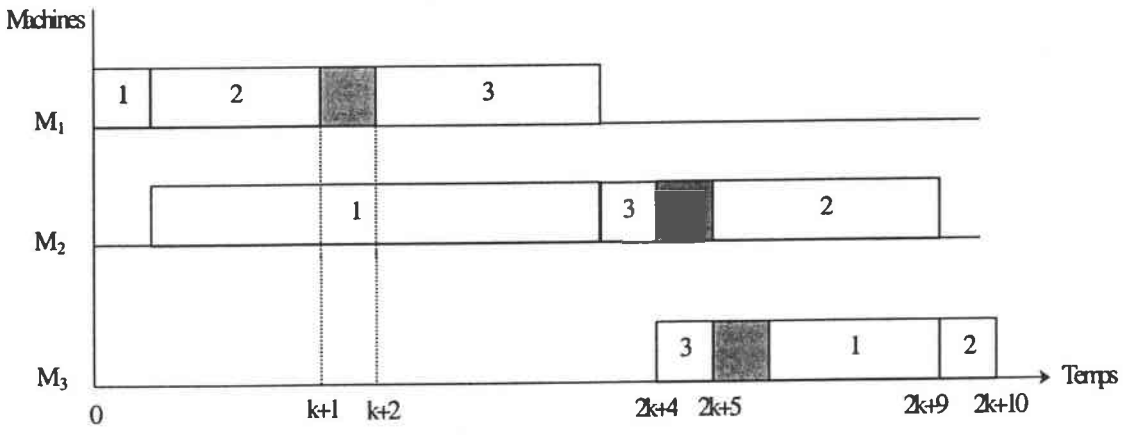


Figure 3.2 : Solution optimale pour l'instance E_2

Ces deux propositions indiquent que la recherche des solutions optimales au problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$ ne peut se restreindre à un sous-ensemble particulier. Par conséquent, toutes les méthodes de résolution doivent être capables de construire des ordonnancements de non-permutation en autorisant les doublements éventuels d'opérations d'une machine à l'autre. Cette dernière remarque souligne encore la difficulté du problème étudié.

2.1.2 Heuristique gloutonne

Nous décrivons dans cette sous-partie une première heuristique permettant de résoudre le problème d'ordonnancement décrit précédemment. Il s'agit d'une heuristique gloutonne, notée HG, qui place les jobs l'un après l'autre en suivant une séquence qui représente un ordre de traitement des jobs. L'ordre de priorité considéré par HG est unique pour toutes les machines. Cependant, la séquence d'opérations obtenue par application de l'heuristique peut être différente d'une machine à l'autre.

La construction d'une solution à partir d'une séquence donnée est du type algorithme de liste (Carlier et Chrétienne [1988]) où les opérations sont considérées machine par machine dans l'ordre de la séquence définie sur les jobs, elle est détaillée dans ce qui suit. Nous présentons ensuite deux méta-heuristiques utilisées pour l'optimisation de la séquence d'entrée de l'heuristique HG.

2.1.2.1 Méthode de résolution

La construction des différents ordonnancements s'effectue sur une machine après l'autre. Par exemple, considérons une séquence $s = (J_1, J_2, J_3)$, HG place d'abord la première opération du job J_1, puis la première du job J_2 et enfin la première du job J_3. Lorsque toutes les opérations sont ordonnancées sur la première machine, l'heuristique HG procède de la même manière pour la deuxième machine et ainsi de suite.

Toute opération déjà placée l'est de manière définitive. Par ailleurs, l'heuristique HG essaie d'ordonnancer les opérations au plus tôt et repart donc de l'instant zéro, ou plus précisément de la date de fin de l'opération précédente sur la gamme du job concerné, à chaque fois qu'elle tente d'insérer une opération. Une opération peut donc être placée sur une machine strictement avant (c'est-à-dire à gauche de) une opération déjà ordonnancée. Les ordonnancements construits sont donc des ordonnancements actifs. Par ailleurs, cette méthode de construction peut être considérée comme un cas particulier des algorithmes de listes (Carlier et Chrétienne [1988]).

K_j tâches de maintenance sont imposées sur la machine M_j, ce qui décompose l'horizon de planification de chaque machine M_j en K_j+1 sous-intervalles I_{j1}, I_{j2},..., I_{j,K_j+1}. La longueur d'un intervalle I_{jh} est notée L_{jh}.

Pour construire un ordonnancement réalisable, HG procède alors selon les étapes suivantes. Une séquence de priorité entre les jobs $(J_1,\ldots, J_n)$ est examinée. A chaque fois qu'un job J_i est sélectionné, HG détermine le premier sous-intervalle sur l'horizon de planification de la machine M_j, dans lequel l'opération O_{ij} du job J_i peut être insérée au plus tôt, sans recouvrir une tâche de maintenance ou une opération déjà ordonnancée. Après chaque insertion, la longueur de l'intervalle d'accueil est mise à jour. Le cas échéant, un intervalle de plus est créé.

Considérons la séquence s = (J_1, J_2, J_3) pour expliquer ces étapes, en supposant que la machine M_j est soumise à deux tâches de maintenance (figure 3.3).

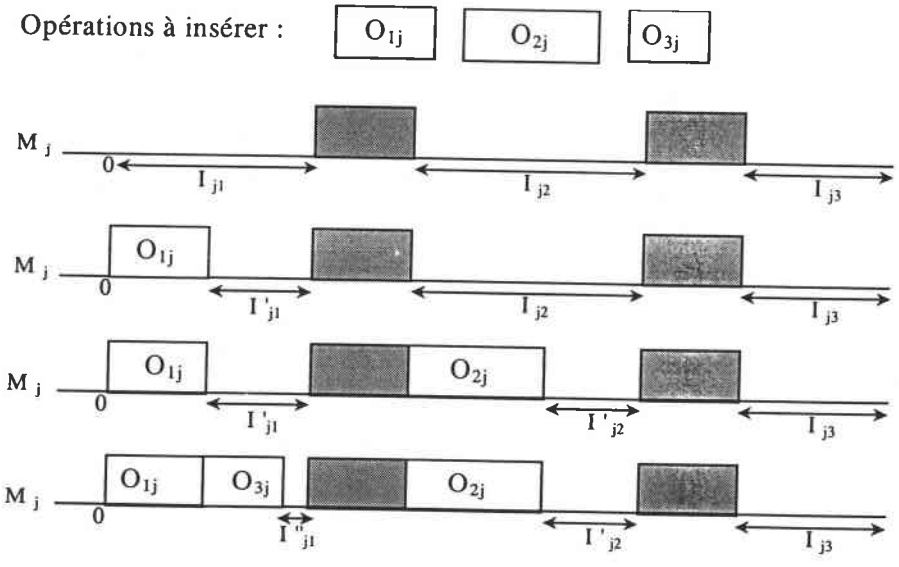


Figure 3.3 : Insertion des opérations

Sur la figure 3.3, l'opération O_{1j} du premier job J_1 de la séquence peut être insérée dans l'intervalle I_{j1} puisque sa durée ne dépasse pas la longueur L_{j1}. Cependant, l'opération O_{2j} du job J_2 ne peut être réalisée au plus tôt que dans l'intervalle I_{j2}, les opérations étant strictement non-préemptives (troisième étape de la figure).

A la quatrième étape, l'opération O_{3j} du job J_3 peut être insérée dans le premier des nouveaux intervalles I'_{j1} dont la longueur est égale à $L_{j1} - p_{1j}$, où p_{1j} est la durée opératoire de l'opération O_{1j}.

Notons que l'ordre de priorité (J_1, J_2, J_3) entre les jobs, donné par la séquence s, conduit à une séquence d'opérations différente, à savoir (J_1, J_3, J_2), sur la machine M_j. Sur la machine suivante M_{j+1}, l'ordre de priorité considéré est également s = (J_1, J_2, J_3), et la séquence d'opérations obtenue peut être différente, comme l'indique la figure 3.4. L'ordonnancement final sera donc bien un ordonnancement de non-permutation, même si la séquence d'entrée (c'est-à-dire l'ordre de priorité considéré) est le même pour toutes les machines.

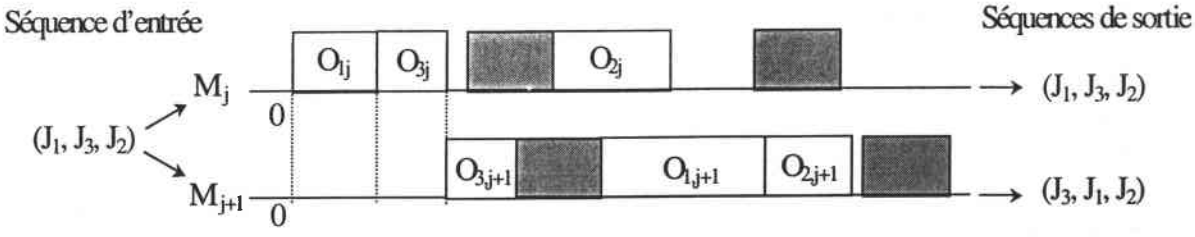


Figure 3.4 : Séquences de sortie différentes

L'algorithme 3.1 suivant permet de résoudre de manière approchée le problème strictement non-préemptif F, $N_{Cwin} \parallel C_{max}$ dans le cas où les périodes d'indisponibilité sont supposées fixes.

Algorithme 3.1 : HG

Données : Une séquence d'entrée s. Sans perte de généralité, soit s = (J_1, J_2,..., J_n) cette séquence

Résultat : Ordonnancement et makespan associé

Début *Construction*

Pour chaque machine M_j, j allant de 1 à *m* **Faire**

/ étape d'insertion */*

Pour chaque job J_i, i allant de 1 à *n* **Faire**

- Trouver l'intervalle d'accueil de l'opération O_{ij}
- Mettre à jour les intervalles d'accueil

Fin Pour.

Fin Pour.

Fin

2.1.2.2 *Optimisation de la séquence d'entrée*

Il est évident que les performances de l'algorithme de résolution proposé ci-avant dépendent fortement de la séquence de traitement des jobs. Deux méthodes d'optimisation sont donc couplées (séparément) à l'heuristique gloutonne pour en optimiser la séquence d'entrée : un algorithme génétique et une recherche tabou.

a) Algorithme génétique

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 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 [1995]).

Le codage naturel employé par notre algorithme génétique est un codage indirect, de permutation, où un chromosome est une chaîne de longueur n représentant un ordre de priorité entre les jobs. A partir de cet ordre, l'algorithme HG construit la solution associée à chaque chromosome et évalue son makespan. Bien que l'ordre de priorité soit le même pour toutes les machines, la manière dont procède HG pour la construction des ordonnancements, c'est-à-dire une machine après l'autre, implique que les solutions obtenues ne sont pas nécessairement des ordonnancements de permutation. Le schéma de reproduction employé est le schéma de Davis [1991], qui consiste à remplacer une partie de la population à chaque itération.

Les paramètres propres de l'algorithme génétique ont quant à eux été choisis après plusieurs séries d'expériences :

- La population initiale est générée aléatoirement. Sa taille est fixée à deux fois le nombre de jobs.
- L'opérateur classique de croisement d'ordre en un point 1X est employé et appliqué à des paires de chromosomes, dont la probabilité d'être tirés est inversement proportionnelle à la valeur de leur makespan. La probabilité de croisement pour les chromosomes sélectionnés est fixée à 0.9. Au total, $\frac{n}{2}$ croisements sont effectués, ce qui permet de créer n nouveaux individus.
- Après le croisement, une mutation peut s'effectuer sur les chromosomes obtenus avec une probabilité égale à 0.05. La mutation consiste à échanger les positions de certains jobs au sein des séquences sélectionnées. Les chances de mutation relativement élevées des chromosomes introduisent des variations dans la population, garantissant ainsi la possibilité d'explorer une plus vaste région de l'espace des solutions.
- Les $\frac{n}{2}$ meilleurs individus parmi ceux qui ont été générés, sont ensuite introduits dans la population courante, remplaçant ainsi les $\frac{n}{2}$ moins bons chromosomes de cette population.
- Enfin, l'algorithme génétique est stoppé après 500 itérations.

Afin de juger des performances de cet algorithme génétique, nous avons choisi de le confronter à une autre méta-heuristique, qui n'est pas une heuristique de population mais une méthode d'amélioration par voisinage : la recherche tabou.

b) Recherche tabou

L'origine et le principe de fonctionnement de la recherche tabou ont été présentés dans le premier chapitre de la thèse. Nous décrivons donc uniquement les paramètres employés pour notre application.

La solution initiale qui est la séquence (représentant l'ordre de priorité entre les jobs) à optimiser est générée aléatoirement. L'algorithme HG construit alors l'ordonnancement réalisable associé à cette séquence, et calcule son makespan.

Le voisinage V(s) d'une séquence s est défini comme étant l'ensemble des séquences obtenues en permutant les positions de certaines paires de jobs J_i et J_j, initialement placés aux positions i et j. Les paires (J_i, i) et (J_j, j) sont alors introduites dans la liste tabou T de telle sorte que le job J_i ne puisse retourner à la position i ou le job J_j à la position j, avant $|T|$ itérations. La taille de la liste est fixée, après expérimentation, au nombre de jobs.

L'algorithme HG évalue toutes les permutations décrivant le voisinage et la séquence qui permet d'obtenir l'ordonnancement ayant le meilleur makespan est sélectionnée comme nouvelle séquence courante. Le critère d'aspiration global qui outre-passe le statut tabou d'une permutation lorsque son action conduit à une nouvelle meilleure solution est adopté. Finalement, la recherche tabou est stoppée après 500 itérations.

c) Algorithmes finaux

L'algorithme 3.2 suivant décrit l'approche de résolution finale, notée HG-AG, combinant l'heuristique gloutonne HG et l'algorithme génétique :

Algorithme 3.2 : HG-AG

Données : Paramètres de l'algorithme génétique, données du problème (temps opératoires, maintenance)

Résultat : Meilleur ordonnancement S* et son makespan C^*_{max}

Etape 1 : *Initialisation*

- Générer aléatoirement une population initiale P
- Initialiser $C^*_{max} \leftarrow +\infty$

Etape 2 : *Evaluation des séquences de la population*

Pour toute séquence s de la population **Faire**

- Appliquer l'algorithme HG à la séquence s. Soit S l'ordonnancement obtenu et C_{max} son makespan
- **Si** $C_{max} < C^*_{max}$ **Alors**

  $S^* \leftarrow S$, et $C^*_{max} \leftarrow C_{max}$

  Fin Si.

Fin Pour.

Etape 3 : *Génération d'une nouvelle population*

- Sélectionner des séquences de la population
- Appliquer les opérateurs de croisement et de mutation
- Introduire les meilleures séquences obtenues à la place d'autres moins bonnes de la population

Etape 4 : *Test d'arrêt*

Si le nombre maximum de générations est atteint **Alors**

Retourner la solution S* et son makespan C^*_{max}

Fin Si.

Sinon Aller à l'étape 2

L'approche finale, notée HG-RT, combinant l'heuristique gloutonne HG et la recherche tabou est donnée par l'algorithme 3.3 suivant :

Algorithme 3.3 : HG-RT

Données : Paramètres de la recherche tabou, données du problème (temps opératoires, maintenance)

Résultat : Meilleur ordonnancement S* et son makespan C^*_{max}

Etape 1 : *Initialisation*

- Générer aléatoirement une séquence initiale s
- Initialiser les structures de la recherche tabou
- Initialiser $C^*_{max} \leftarrow +\infty$

Etape 2 : *Evaluation de la séquence*

- Appliquer l'algorithme HG à la séquence s. Soit S l'ordonnancement obtenu et C_{max} son makespan
- **Si** $C_{max} < C^*_{max}$ **Alors**

  Sauvegarder S comme meilleure solution actuelle, et mettre à jour C^*_{max}

  Fin Si.

Etape 3 : *Détermination de la meilleure solution voisine*

- Appliquer le mouvement de permutation pour générer la meilleure séquence s' dans le voisinage de s
- Mettre à jour les structures de la recherche tabou
- $s \leftarrow s'$

Etape 4 : *Test d'arrêt*

Si le nombre maximum d'itérations est atteint **Alors**

Retourner la solution S* et son makespan C^*_{max}

Fin Si.

Sinon Aller à l'étape 2

2.1.2.3 Résultats numériques

Comparé au grand nombre de benchmarks existant dans la littérature pour le problème d'ordonnancement classique du flow shop, peu d'instances sont dédiées à l'ordonnancement sous contraintes de disponibilité des machines. Les études numériques existantes traitent essentiellement de problèmes à deux machines, dans le cas d'opérations sécables. Il n'existe pas, du moins à notre connaissance, d'instances pour le problème d'ordonnancement strictement non-préemptif F, $N_{Cwin} \parallel C_{max}$.

Afin d'évaluer les performances des approches finales HG-AG et HG-RT, des expériences ont été menées sur des instances générées aléatoirement. Le choix de générer ces instances, ainsi que la manière de le réaliser sont motivés par le souhait de se rapprocher de situations susceptibles d'être rencontrées dans des problèmes industriels réels. En effet, la plupart des benchmarks de la littérature consacrés au flow shop classique (auxquels nous aurions pu ajouter des périodes d'indisponibilité) présente des durées opératoires très dispersées, entre 1 et 100, ce qui n'est pas usuel dans la réalité des entreprises.

Nous présentons dans ce qui suit quelques résultats numériques, pour trois classes d'instances, comportant chacune cinq instances. Le nombre de jobs pour chaque classe est respectivement 10, 10, 20. Le nombre de machines est égal à 5 pour la première classe et 10 pour les deux autres. La durée de chaque opération est tirée aléatoirement dans l'ensemble $\{50, 60, \ldots, 140, 150\}$.

Nous supposons que l'horizon de planification n'est pas trop long, et nous réduisons donc nos expériences au cas de deux périodes d'indisponibilité exactement par machine. Les positions des tâches de maintenance sont générées aléatoirement, cependant nous ne considérons que les schémas tels que pour chaque machine, la première tâche de maintenance se produit après au moins une opération et la seconde avant au moins une opération. En effet, sans cette précaution, les périodes d'indisponibilité n'influeraient pas sur l'ordonnancement et le problème serait classique.

Les tâches de maintenance sont supposées être des tâches de durée moyenne. La durée d'une tâche de maintenance sur une machine est donc donnée par la moyenne des temps opératoires des opérations exécutées sur cette machine.

Le tableau 3.3 présente le makespan initial, le meilleur makespan, le makespan moyen, ainsi que le plus mauvais makespan trouvés par l'algorithme HG-AG en dix exécutions indépendantes de chaque instance. Dans la colonne CPUsecs, figurent les temps mis par l'algorithme pour atteindre la meilleure valeur.

Nous pouvons déduire du tableau 3.3 que l'algorithme génétique développé est assez stable, en ce sens que la différence entre le meilleur makespan et le plus mauvais pour dix exécutions est relativement faible. D'une exécution à l'autre, les solutions trouvées ne sont donc pas aberrantes. Pour ce qui est du temps d'exécution, la meilleure solution est obtenue dans la majorité des cas en un temps assez court.

Tableau 3.3 : Résultats de 10 exécutions indépendantes par HG-AG

Problème	n	m	initial	mauvais	moyen	meilleur	CPUsecs
RA1	10	5	1954	1794	1773	1752	0.2
RA2	10	5	2149	1867	1848	**1836**	0.1
RA3	10	5	2171	1889	1868	1824	0.2
RA4	10	5	2136	1976	1953	**1918**	0.1
RA5	10	5	2163	1825	1817	1809	0.3
RA6	10	10	2918	2674	2634	**2588**	0.3
RA7	10	10	3231	2768	2739	**2719**	0.4
RA8	10	10	3211	2653	2636	**2601**	0.1
RA9	10	10	2741	2481	2462	2441	0.3
RA10	10	10	2786	2468	2467	2465	0.2
RA11	20	10	4089	3579	3526	3409	2.4
RA12	20	10	4216	3730	3718	3686	1.7
RA13	20	10	4285	3752	3708	3672	0.8
RA14	20	10	3972	3691	3609	3532	3.5
RA15	20	10	4226	3784	3716	3679	0.4

Le tableau 3.4 présente le makespan initial, le meilleur makespan, le makespan moyen, ainsi que le plus mauvais makespan obtenus en dix exécutions indépendantes de chaque instance RA1 à RA15 par l'algorithme HG-RT.

Tableau 3.4 : Résultats de 10 exécutions indépendantes par HG-RT

Problème	n	m	initial	mauvais	moyen	meilleur	CPUsecs
RA1	10	5	2014	*1804*	1769	**1744**	0.1
RA2	10	5	2203	*1921*	1889	1859	0.1
RA3	10	5	2214	*1915*	1864	**1811**	0.4
RA4	10	5	2146	*2001*	1977	1929	0.2
RA5	10	5	2093	*1933*	1848	**1807**	0.3
RA6	10	10	2938	*2699*	2660	2629	0.7
RA7	10	10	3081	*2814*	2780	2741	0.5
RA8	10	10	2969	*2695*	2656	2618	0.1
RA9	10	10	2726	*2581*	2504	**2437**	0.6
RA10	10	10	2716	*2566*	2510	**2459**	0.3
RA11	20	10	3979	*3582*	3468	**3369**	3.1
RA12	20	10	4266	*3830*	3763	**3641**	1.9
RA13	20	10	4088	*3823*	3754	**3668**	2.1
RA14	20	10	4072	*3691*	3599	**3506**	2.1
RA15	20	10	4216	*3904*	3748	**3646**	1.2

Nous pouvons déduire du tableau 3.4 que l'algorithme de recherche tabou est légèrement moins stable que l'algorithme génétique, les variations de la solution d'une exécution à l'autre étant plus importantes. Les temps d'exécution sont comme pour l'algorithme génétique relativement court.

Excepté les problèmes de plus grandes tailles, pour lesquels la recherche tabou a trouvé de meilleures solutions, nous ne pouvons affirmer qu'une des méthodes d'optimisation domine complètement l'autre. Aussi, serait-il intéressant de mener des expériences supplémentaires, en changeant les paramètres et en augmentant le nombre d'itérations.

Dans la littérature consacrée à l'ordonnancement sous contraintes de disponibilité des machines, les études sont essentiellement dédiées à la minimisation du makespan. L'un des avantages des algorithmes développés dans cette sous-section est qu'ils permettent de traiter d'autres fonctions objectif régulières. Il suffit pour cela de modifier l'heuristique gloutonne HG en évaluant le critère considéré, à chaque fois que tous les jobs associés à une séquence donnée, sont ordonnancés. Le lecteur trouvera ainsi, dans Aggoune et al. [2001], d'autres résultats d'expériences menées en considérant pour objectif la minimisation de la somme pondérée des retards des jobs.

2.1.3 Heuristique basée sur la résolution de sous-problèmes à deux jobs

Nous présentons dans ce qui suit une seconde heuristique, que nous notons H2, permettant la résolution du problème d'ordonnancement strictement non-préemptif F, $N_{Cwin} \parallel C_{max}$, dans le cas où les périodes d'indisponibilité sont fixes. Cette heuristique est basée sur la résolution de sous-problèmes à deux jobs au moyen de l'algorithme polynomial développé dans le second chapitre de la thèse. Comme pour les algorithmes HG-AG et HG-RT de la section précédente, le principe de fonctionnement de H2 est applicable à la considération de tout critère régulier.

2.1.3.1 Méthode de résolution

Etant donnée une séquence de traitement des jobs $s = (J_1, J_2, \ldots, J_n)$, l'heuristique H2 construit l'ordonnancement associé à s de la manière suivante :

Les jobs J_1 et J_2 sont ordonnancés de manière optimale par l'algorithme polynomial *2-jobs* du chapitre précédent. Cet algorithme développé pour la résolution du problème d'ordonnancement du job shop à deux jobs, permet évidemment d'ordonnancer de manière optimale des paires de jobs dans un système de type flow shop. La complexité de l'algorithme est alors au plus égale à $O(Km^4)$, où m est le nombre de machines de l'atelier et K le nombre maximal de tâches de maintenance sur une même machine.

Une fois l'ordonnancement optimal obtenu, les opérations des jobs J_1 et J_2 sont fixées comme des nouvelles tâches de maintenance sur toutes les machines. L'algorithme *2-jobs* est ensuite appliqué aux jobs J_3 et J_4, en considérant les tâches de maintenance additionnelles. Les dates d'exécution des opérations données par l'ordonnancement optimal des jobs J_3 et J_4 permettent de fixer deux tâches supplémentaires de maintenance sur chaque machine, et le procédé est ainsi réitéré jusqu'à ce que tous les jobs de la séquence soient traités.

Notons que si le nombre total de jobs à ordonnancer est impair, les opérations du dernier job sont insérées au plus tôt sur chaque machine selon le procédé d'insertion utilisé par l'heuristique gloutonne de la sous-section précédente.

L'heuristique H2 basée sur l'ordonnancement successif de paires de jobs est donnée par l'algorithmes 3.4 suivant :

Algorithme 3.4 : H2

Données : Une séquence d'entrée s. Sans perte de généralité, soit $s = (J_1, J_2, \ldots, J_n)$ cette séquence.

Résultat : Ordonnancement et makespan associé

Début *Construction*

$k \leftarrow 1$

Tant que $k < n$ **Faire**

/ Ordonnancement optimal des deux jobs */*

Appliquer l'algorithme *2-jobs* à (J_k, J_{k+1})

/ Mise à jour de la maintenance */*

Pour chaque machine M_j, j allant de 1 à m **Faire**

Fixer les opérations O_{kj} et $O_{k+1, j}$ comme nouvelles périodes d'indisponibilité

Fin Pour.

$k \leftarrow k + 2$

Fin Tant que.

Si $k = n$ **Alors** */* Cas où n est impair */*

/ Ordonnancer le job J_n */*

Pour chaque machine M_j, j allant de 1 à m **Faire**

Insérer au plus tôt l'opération O_{nj}

Fin Pour.

Fin Si.

Fin

2.1.3.2 *Intégration d'une recherche tabou*

Les performances de l'heuristique H2 dépendent fortement de l'ordre de traitement des jobs. La recherche tabou ayant donné des résultats légèrement meilleurs que ceux de l'algorithme génétique aux expérimentations précédentes (du moins pour les instances de plus grandes tailles), nous choisissons de l'ajouter à H2 pour en optimiser la séquence d'entrée. Les paramètres de la recherche tabou employée sont décrits ci-après.

La séquence initiale représentant l'ordre de priorité entre les jobs, est générée aléatoirement. A partir de l'ordre donné par cette séquence, les jobs sont regroupés par paires (les deux premiers jobs, puis les deux suivants, etc.) et l'algorithme *2-jobs* est appliqué successivement à ces paires, jusqu'à ce que tous les jobs soient ordonnancés.

Le voisinage V(s) d'une séquence s est défini comme étant l'ensemble des séquences obtenues en permutant les positions des jobs J_i et J_j initialement placés aux positions i et j, et n'appartenant pas à une même paire. En effet, l'ordonnancement de deux jobs par l'algorithme *2-jobs* étant optimal, il ne dépend pas de l'ordre de considération de ces jobs. Notons que cette restriction nous permet de diminuer légèrement le nombre d'évaluations à effectuer dans le voisinage de s, chose que n'aurait pas permis un algorithme génétique.

Les couples (J_i, i) et (J_j, j) sont ensuite introduits dans la liste tabou T de telle sorte que le job J_i (resp. J_j) ne puisse retourner à la position i (resp. j) avant $|T|$ itérations. La taille de la liste est fixée au nombre total de jobs à ordonnancer.

Toutes les permutations définissant le voisinage sont alors évaluées et la séquence qui permet d'obtenir le meilleur ordonnancement, est sélectionnée comme nouvelle séquence. Le critère d'aspiration global qui modifie le statut tabou d'une permutation lorsque son application permet d'atteindre une nouvelle meilleure solution, est au besoin appliqué. Et la recherche tabou est finalement stoppée après 500 itérations.

2.1.3.3 *Résultats numériques*

Nous rapportons dans cette sous-section les résultats d'expériences numériques obtenus par application de l'heuristique finale H2-RT combinant l'heuristique H2 à la recherche tabou décrite ci-dessus. Les tests ont été effectués sur les instances RA1 à RA15 générées aléatoirement afin de pouvoir comparer les performances de l'heuristique H2 à celles de l'heuristique gloutonne HG, proposée dans la sous-section précédente.

Le tableau 3.5 suivant présente le makespan initial, le pire makespan, le makespan moyen, ainsi que le meilleur makespan trouvés par l'heuristique H2-RT en dix exécutions indépendantes de chacune des instances. Les résultats apparaissent dans le premier groupe de colonnes (H2-RT), tandis que le second groupe de colonnes (H1) contient les makespan des meilleures solutions obtenues par application de l'heuristique gloutonne couplée, soit à l'algorithme génétique (HG-AG), soit à la recherche tabou (HG-RT).

Tableau 3.5 : Résultats de 10 exécutions indépendantes par H2-RT

Problème	H1		H2-RT				
	meilleur	CPU secs	initial	mauvais	moyen	meilleur	CPU secs
RA1	1744	0.1	2022	1784	1726	**1712**	0.3
RA2	1836	0.1	2186	1851	1840	**1816**	0.2
RA3	1811	0.4	2211	1829	1825	**1784**	0.2
RA4	1918	0.1	2179	1965	1937	**1909**	0.1
RA5	1807	0.3	1907	1812	1795	**1767**	0.5
RA6	2588	0.3	2958	2656	2598	**2567**	0.7
RA7	2719	0.4	2921	2773	2724	**2671**	0.4
RA8	2601	0.1	2885	2637	2613	2601	0.2
RA9	2437	0.6	2643	2462	2448	**2413**	0.7
RA10	2459	0.3	2776	2489	2478	**2456**	1.6
RA11	3369	3.1	3909	3401	3366	**3344**	5.8
RA12	3641	1.9	4103	3730	3695	**3586**	4.6
RA13	3668	2.1	4007	3777	3698	**3625**	2.4
RA14	3506	2.1	3876	3568	3534	**3493**	3.9
RA15	3646	1.2	4095	3814	3698	**3629**	3.7

Les résultats figurant au tableau 3.5 indiquent clairement que l'heuristique basée sur la résolution de sous-problèmes à deux jobs H2-RT domine l'approche basée sur l'heuristique gloutonne HG. En effet, pour toutes les instances testées sauf une, le meilleur makespan trouvé par H2-RT est strictement inférieur au makespan donné par application de l'heuristique intégrant HG (pour l'instance RA8, il y a égalité). Par ailleurs, les makespan moyens obtenus par H2-RT sont relativement proches des valeurs de H1, et même inférieurs pour certaines instances. Les temps d'exécution de H2 sont cependant légèrement supérieurs à ceux de H1. Sans aucun doute, ceci est dû à la construction des réseaux qu'effectue l'algorithme *2-jobs* utilisé par H2-RT. En effet, à chaque fois qu'une paire de job est ordonnancée, les opérations des deux jobs sont fixées en tant que tâches additionnelles de maintenance. A l'appel suivant de la procédure *2-jobs*, le nombre de tâches de maintenance considérées a donc été augmenté, ce qui génère forcément un plus grand nombre de sommets dans le réseau obtenu.

Puisque la recherche tabou est appliquée pour l'optimisation des séquences d'entrée des deux heuristiques HG et H2 avec les mêmes paramètres, nous pouvons en déduire que la différence de performance entre les deux méthodes réside dans la construction des ordonnancements. Bien que légèrement plus coûteuse en temps, l'approche basée sur la résolution exacte de sous-problèmes à deux jobs est donc plus performante que l'approche par insertion d'opérations.

2.2 Tâches de maintenance flexibles

Nous supposons dans cette sous-section que les positions des différentes périodes d'indisponibilité correspondant à des tâches de maintenance ne sont pas fixes et peuvent être optimisées durant la procédure d'ordonnancement. Nous commençons par expliquer dans ce qui suit l'origine et l'intérêt de l'introduction de ces tâches de maintenance flexibles. Puis, nous proposons des modifications des heuristiques HG et H2 de la sous-section précédente de sorte à prendre en compte cette nouvelle hypothèse. Les différents résultats, présentés pour la minimisation du makespan, sont là encore, applicables à d'autres critères réguliers.

2.2.1 Description du problème et intérêt pratique

Le contexte communément étudié dans la littérature dédiée à l'ordonnancement (déterministe et statique) sous contraintes de disponibilité des machines consiste à supposer que les dates de réalisation des différentes tâches de maintenance sont fixes (figure 3.5).

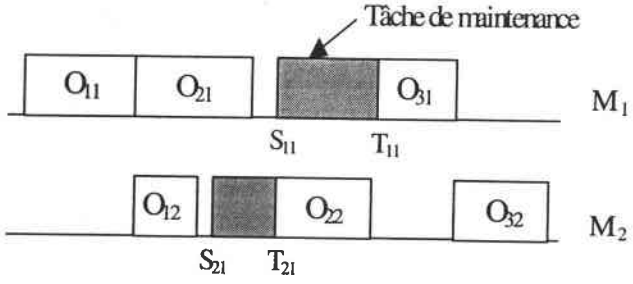


Figure 3.5 : Tâches de maintenance fixes

Sur la figure 3.5, S_{jh} (resp. T_{jh}) représente la date de début (resp. de fin) de réalisation de la $h^{ème}$ tâche de maintenance sur la machine M_j. L'opération strictement non-préemptive O_{31} (resp. O_{22}) ne peut être réalisée par la machine M_1 (resp. M_2), qu'après la réalisation de la tâche de maintenance de la machine M_1 (resp. M_2). Des temps morts apparaissent donc éventuellement entre la date à laquelle se termine l'exécution de l'opération O_{21} (resp. O_{12}) et la date de début S_{11} (resp. S_{21}) de la tâche de maintenance.

Cette dernière remarque nous a amené à introduire une seconde variante du problème, afin d'optimiser le placement des activités de maintenance. Dans cette dernière, les dates de réalisation des tâches de maintenance sont supposées flexibles (figure 3.6), et doivent être déterminées durant la procédure d'ordonnancement.

Une fenêtre temporelle est donc allouée à l'exécution de chaque tâche de maintenance. Cette hypothèse se rapproche d'ailleurs des situations industrielles, telle que l'obligation de planifier une tâche de maintenance sur une machine, toutes les 300 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 aussi tôt que possible, de manière à poursuivre avec une machine dont la probabilité de panne est plus faible.

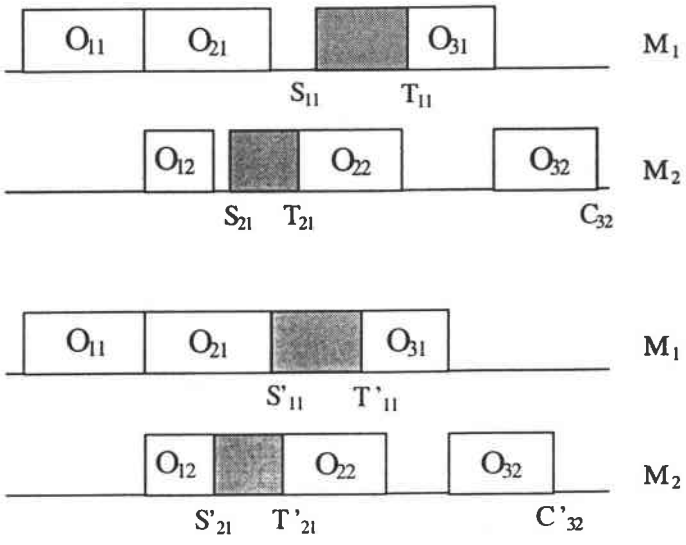


Figure 3.6 : Tâches de maintenance flexibles

Sur la figure 3.6, l'opération strictement non-préemptive O_{31} (resp. O_{22}) ne peut être réalisée par la machine M_1 (resp. M_2) qu'après la tâche de maintenance. Cependant, nous pouvons combler le temps mort entre la date de fin d'exécution de l'opération O_{21} (resp. O_{12}) et celle de début de la tâche de maintenance S_{11} (resp. S_{21}), en décalant cette tâche vers la gauche. Ainsi, l'opération O_{31} (resp. O_{22}) peut commencer, et donc finir, plus tôt. Et finalement, l'exécution de l'opération O_{32} peut être achevée à la date $C'_{32} < C_{32}$.

Le problème d'ordonnancement de type flow shop strictement non-préemptif avec la prise en compte de contraintes de disponibilité flexibles se définit de la manière suivante :

- Un ensemble de n travaux $\mathfrak{J} = \{J_1, J_2, \ldots, J_n\}$ doit être réalisé par un ensemble de m machines $M = \{M_1, M_2, \ldots, M_m\}$.
- Chaque travail J_i, composé d'une séquence linéaire 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 opération à la fois et chaque opération O_{ij} nécessite une seule machine à la fois, durant p_{ij} unités de temps.
- Plusieurs périodes d'indisponibilité correspondant à des tâches de maintenance préventive peuvent se produire sur chacune des machines. Les durées des tâches sont connues à l'avance et fixes. Chaque tâche de maintenance doit être réalisée à l'intérieur d'une fenêtre temporelle qui lui est associée.
- Les opérations sont strictement non-préemptives.
- L'objectif est de déterminer les séquences d'entrée des opérations sur les machines de telle sorte que le makespan soit minimal.

Nous complétons la notation de Schmidt [2000] utilisée jusqu'à présent, en désignant par *flex* l'hypothèse de flexibilité sur les périodes d'indisponibilité. Le problème d'ordonnancement peut ainsi être noté F, $N_{Cwin\text{-}flex} \parallel C_{max}$, où $N_{Cwin\text{-}flex}$ signifie que les périodes d'indisponibilité sont arbitrairement distribuées sur les machines et flexibles.

Le problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$ étant NP-difficile au sens fort, il en résulte que le problème F, $N_{Cwin\text{-}flex} \parallel C_{max}$ étudié ici est également NP-difficile au sens fort. En effet, le modèle où les tâches de maintenance sont fixes est un cas limite du modèle où les tâches de maintenance sont flexibles, pour lequel les fenêtres temporelles correspondent exactement aux durées des tâches de maintenance associées. Les propriétés caractérisant le problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$ s'appliquent donc au problème F, $N_{Cwin\text{-}flex} \parallel C_{max}$, à savoir :

- Il n'existe pas d'heuristique à performance garantie pour le problème d'ordonnancement F, $N_{Cwin\text{-}flex} \parallel C_{max}$.
- Les ordonnancements de permutation, ainsi que les ordonnancements pour lesquels les séquences d'opérations sont les mêmes sur les deux premières machines ainsi que sur les deux dernières, ne sont pas dominants.

2.2.2 Modification des approches de résolution

Nous décrivons dans ce qui suit comment adapter les méthodes de résolution proposées dans la sous-section 2.1, de sorte que soit prise en compte l'hypothèse de flexibilité sur l'activité de maintenance. En effet, sous réserve de quelques modifications, les heuristiques HG et H2 utilisées pour résoudre le problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$ permettent de traiter le problème F, $N_{Cwin\text{-}flex} \parallel C_{max}$, en utilisant le fait que la position des tâches de maintenance n'est pas fixée. Les structures des deux approches de résolution proposées dans le cas de tâches de maintenance fixes, c'est-à-dire la construction d'ordonnancements réalisables à partir d'ordres de priorité et l'intégration de méta-heuristiques pour optimiser ces ordres, peuvent donc être conservées.

2.2.2.1 Approche par insertion d'opérations

a) Modification de l'heuristique gloutonne

Dans le cas où les tâches de maintenance sont flexibles, la construction de l'ordonnancement à partir d'une séquence est analogue au cas des tâches de maintenances fixes (cf. figure 3.3), avec cependant une étape supplémentaire (figure 3.7).

Les tâches de maintenance sont a priori calées à droite dans les fenêtres temporelles qui leur sont associées. Lorsque toutes les opérations sont insérées sur une machine donnée, HG essaie de réduire, autant que faire se peut, les temps morts apparus devant chaque tâche de maintenance. Cette action est réalisée, en considérant ces tâches dans l'ordre chronologique et en les décalant au plus tôt à l'intérieur de leurs fenêtres temporelles.

Puis, les opérations succédant aux tâches de maintenance sont décalées vers la gauche autant que le permettent les dates de fin des opérations des mêmes jobs sur la machine précédente.

Ainsi, les dates de fin d'exécution des opérations décalées (qui donnent les dates de début au plus tôt des opérations sur la machine suivante) sont mises à jour.

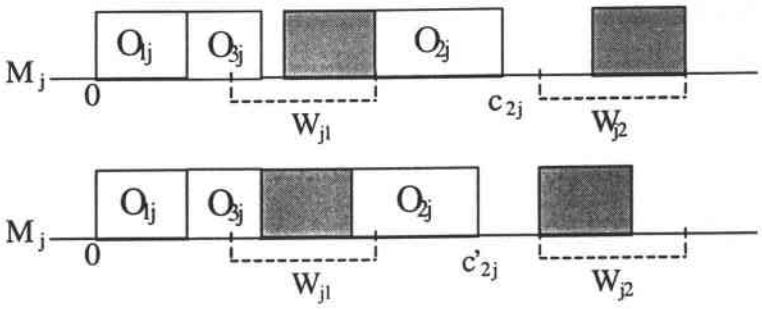


Figure 3.7 : Etape supplémentaire

Sur la figure 3.7, toutes les opérations sont insérées sur la machine M_j entre les différentes tâches de maintenance (nous pouvons supposer qu'il existe une première tâche de maintenance qui commence et se termine à la date 0), qui sont toutes calées à droite au sein de leurs fenêtres temporelles associées W_{j1} et W_{j2}.

L'étape supplémentaire dans la construction de l'ordonnancement associé à la séquence $s = (J_1, J_2, J_3)$ consiste à décaler les deux tâches de maintenance (de durées non nulles) ainsi que l'opération O_{2j} autant que possible. Par conséquent, l'exécution de l'opération $O_{2,j+1}$ sur la machine suivante pourra démarrer à la date $c'_{2j} < c_{2j}$.

L'algorithme 3.5 suivant permet de résoudre, de manière approchée, le problème d'ordonnancement strictement non-préemptif F, $N_{Cwin\text{-}flex} \parallel C_{max}$. Pour faire la distinction avec l'algorithme utilisé dans le cas de tâches de maintenance fixes, désigné maintenant par HG (MF) et non plus par HG, nous notons cet algorithme HG (MD), où MD signifie que la maintenance est placée dynamiquement.

Algorithme 3.5 : HG (MD)

Données : Une séquence d'entrée s. Sans perte de généralité, soit $s = (J_1, J_2, \ldots, J_n)$ cette séquence

Résultat : Ordonnancement et makespan associé

Début *Construction*

Pour chaque machine M_j, j allant de 1 à *m* **Faire**

/ étape d'insertion */*

Pour chaque job J_i, i allant de 1 à *n* **Faire**

- Trouver l'intervalle d'accueil de l'opération O_{ij}
- Mettre à jour la longueur de l'intervalle d'accueil

Fin Pour.

/ étape de décalage */*

Décaler successivement à gauche chaque tâche de maintenance de la machine M_j et les opérations lui succédant

Fin Pour.

Fin

Afin d'optimiser les séquences d'entrée considérées par l'algorithme HG (MD), nous appliquons les méta-heuristiques utilisées dans le contexte de tâches de maintenance fixes, à savoir un algorithme génétique et une recherche tabou. Par ailleurs, nous choisissons de conserver les paramètres propres aux méta-heuristiques, de manière à pouvoir comparer les résultats obtenus par HG (MD) à ceux de HG (MF).

b) Résultats numériques

Nous présentons dans ce qui suit les résultats d'expériences numériques obtenus par application des approches finales, notées AG-MD et RT-MD, combinant l'heuristique HG (MD) à l'algorithme génétique et à la recherche tabou respectivement. Les tests ont été effectués sur les instances RA1 à RA15 présentées dans la sous-section 2.1.2.3. Nous ajoutons uniquement que les fenêtres temporelles allouées aux tâches de maintenance sont supposées relativement modestes, et nous autorisons un décalage maximal de 100 unités de temps pour chaque tâche de maintenance.

Le second groupe de colonne du tableau 3.6 donne le meilleur makespan atteint par l'algorithme AG-MD en dix exécutions indépendantes de chaque instance, et le temps d'exécution associé. Le premier groupe de colonnes présente les meilleurs résultats que nous avions obtenus dans le cas où les maintenances étaient supposées fixes (AG-MF). Notons que l'algorithme AG-MD est lancé à partir de la population initiale de l'algorithme AG-MF.

Tableau 3.6 : Résultats de 10 exécutions indépendantes par AG-MD

Problème	n	m	initial	AG-MF		AG-MD	
				meilleur	CPUsecs	meilleur	CPUsecs
RA1	10	5	1954	1752	0.2	1617	0.2
RA2	10	5	2149	1836	0.1	1700	0.4
RA3	10	5	2171	1824	0.2	1732	0.9
RA4	10	5	2136	1918	0.1	1807	0.3
RA5	10	5	2163	1809	0.3	**1662**	0.2
RA6	10	10	2918	2588	0.3	2440	0.1
RA7	10	10	3231	2719	0.4	2481	0.7
RA8	10	10	3211	2601	0.1	2397	1.0
RA9	10	10	2741	2441	0.3	**2220**	0.8
RA10	10	10	2786	2465	0.2	2296	0.9
RA11	20	10	4089	3409	2.4	3288	2.7
RA12	20	10	4216	3686	1.7	3459	3.2
RA13	20	10	4285	3672	0.8	3458	2.9
RA14	20	10	3972	3532	3.5	3368	4.0
RA15	20	10	4226	3679	0.4	3426	4.6

Le résultat intéressant qui ressort du tableau 3.6, implique le second groupe de colonnes. Ce dernier montre en effet que la prise en compte de périodes d'indisponibilité flexibles permet d'améliorer la valeur du makespan de manière significative. Les temps d'exécution de AG-MD sont dans l'ensemble supérieurs à ceux de AG-MF.

Clairement, pour toutes les instances, l'introduction de fenêtres temporelles pour l'activité de maintenance permet d'améliorer les résultats obtenus en supposant que les tâches de maintenance sont fixes.

Le tableau 3.7 présente le meilleur makespan, ainsi que le temps d'exécution associé, pour dix exécutions indépendantes de chaque instance par l'algorithme RT-MD. Dans le premier groupe de colonnes figurent les résultats pour le modèle avec maintenance fixe (RT-MF).

Tableau 3.7 : Résultats de 10 exécutions indépendantes par RT-MD

Problème	n	m	initial	RT-MF		RT-MD	
				meilleur	CPUsecs	meilleur	CPUsecs
RA1	10	5	2014	1744	0.1	**1484**	0.6
RA2	10	5	2203	1859	0.1	1700	0.1
RA3	10	5	2214	1811	0.4	1732	0.7
RA4	10	5	2146	1929	0.2	**1799**	0.2
RA5	10	5	2093	1807	0.3	1677	0.2
RA6	10	10	2938	2629	0.7	**2438**	0.4
RA7	10	10	3081	2741	0.5	2481	0.9
RA8	10	10	2969	2618	0.1	2397	1.7
RA9	10	10	2726	2437	0.6	2253	1.8
RA10	10	10	2716	2459	0.3	2296	0.4
RA11	20	10	3979	3369	3.1	**3224**	4.6
RA12	20	10	4266	3641	1.9	**3456**	3.2
RA13	20	10	4088	3668	2.1	**3438**	3.0
RA14	20	10	4072	3506	2.1	3368	1.7
RA15	20	10	4216	3646	1.2	**3399**	4.9

Pour toutes les instances, la prise en compte d'une activité de maintenance dynamique (RT-MD) donne également une amélioration significative des résultats obtenus avec une maintenance fixe (RT-MF), ce qui confirme bien l'intérêt de considérer des fenêtres temporelles.

Comme dans le cas de tâches de maintenance fixes, les résultats obtenus en utilisant la recherche tabou sont légèrement meilleurs que ceux de l'algorithme génétique, ce dernier étant cependant moins coûteux.

Enfin, une combinaison des deux méthodes aurait probablement donné de meilleurs résultats, mais l'important ici était de montrer l'intérêt d'introduire la flexibilité dans le processus de maintenance comme un moyen de compenser la contrainte de non-préemption stricte. Notons que l'algorithme HG (MD) permet, comme HG (MF), de traiter d'autres fonctions objectif régulières (Aggoune et al. [2001]).

2.2.2.2 *Approche par résolution de sous-problèmes à deux jobs*

Nous proposons dans ce qui suit une adaptation de l'approche basée sur l'ordonnancement de paires de jobs H2, au contexte où les tâches de maintenance sont supposées flexibles. La nouvelle heuristique sera notée H2 (MD) et celle utilisée dans le cas de tâches de maintenance fixes sera notée H2 (MF) (plutôt que H2).

Dans l'heuristique gloutonne HG (MD) décrite précédemment, le décalage des différentes tâches de maintenance sur une machine donnée s'effectue une fois que toutes les opérations sont placées sur cette machine. Dans le cas de la résolution de sous-problèmes à deux jobs, les ordonnancements sur toutes les machines sont effectués de manière quasi-simultanée. L'étape de décalage des tâches de maintenance doit donc s'effectuer avant que ne soient examinés tous les jobs de la séquence de traitement.

Aussi, proposons-nous de décaler une tâche de maintenance précédée d'un temps mort sur une machine, si aucune des opérations restant à ordonnancer sur cette machine ne peut combler ce temps mort (figure 3.8).

Considérons la séquence de priorité s = (J_1, J_2,..., J_6). Les jobs J_1 et J_2 sont ordonnancés en premier au moyen de l'algorithme *2-jobs* du chapitre 2. Nous rappelons que la seconde étape de l'heuristique H2 consiste alors à fixer sur chaque machine les opérations des jobs J_1 et J_2, comme de nouvelles tâches de maintenance, de sorte que *2-jobs* soit appliqué aux jobs J_3 et J_4 en considérant l'activité maintenance initiale, ainsi que les tâches de maintenance additionnelles.

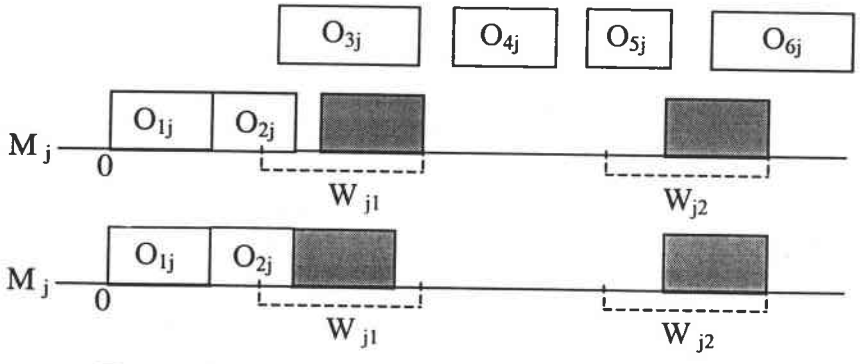


Figure 3.8 : Décalage d'une tâche de maintenance

Sur la figure 3.8, les opérations O_{1j} et O_{2j} ont été placées sur la machine M_j. Cependant, toutes les opérations restant à ordonnancer sur cette machine sont de durées supérieures au temps mort situé entre la fin de l'opération O_{2j} et le début de la première tâche de maintenance (initiale). En décalant cette tâche de maintenance autant que possible à l'intérieur de la fenêtre temporelle qui lui est associée, nous arrivons à combler ce temps mort. En outre, cela permet à la prochaine opération à placer sur cette machine, de démarrer éventuellement plus tôt. Notons que le décalage de cette seule tâche de maintenance peut également permettre d'insérer plus d'opérations entre les deux tâches de maintenance.

Des tests sont donc ajoutés à l'algorithme H2 (MF), après chaque étape de placement d'opérations pour effectuer les différents décalages et ainsi prendre en compte l'hypothèse de flexibilité sur la maintenance.

L'algorithme 3.6 suivant présente l'heuristique basée sur la résolution de sous-problèmes à deux jobs et qui permet de résoudre de manière approchée le problème d'ordonnancement strictement non-préemptif F, $N_{Cwin\text{-}flex} \parallel C_{max}$.

Algorithme 3.6 : H2 (MD)

Données : Une séquence d'entrée s. Sans perte de généralité, soit $s = (J_1, J_2, \ldots, J_n)$ cette séquence

Résultat : Ordonnancement et makespan associé

Début *Construction*

$k \leftarrow 1$

Tant que $k < n$ **Faire**

/ Ordonnancement optimal des deux jobs */*

Appliquer l'algorithme *2-jobs* à (J_k, J_{k+1})

/ Mise à jour de la maintenance */*

Pour chaque machine M_j, j allant de 1 à *m* **Faire**

Fixer les opérations O_{kj} et $O_{k+1,j}$ comme nouvelles périodes d'indisponibilité

/ Etape de décalage */*

Pour chaque tâche de maintenance initiale **Faire**

Si la tâche de maintenance n'est pas fixée **Alors**

Si aucune insertion ne sera possible devant la tâche **Alors**

- Décaler à gauche la tâche de maintenance

- Fixer la tâche de maintenance

Fin Si.

Fin Si.

Fin Pour.

Fin Pour.

$k \leftarrow k + 2$

Fin Tant que.

Si $k = n$ **Alors** */* Cas où n est impair */*

/ Ordonnancer le job J_n */*

Pour chaque machine M_j, j allant de 1 à *m* **Faire**

Insérer au plus tôt l'opération O_{nj}

Fin Pour.

Fin Si.

Fin

Afin d'optimiser l'ordre de traitement des jobs utilisé par l'heuristique H2 (MD), nous choisissons de coupler cette dernière à une recherche tabou. Les paramètres de la recherche tabou sont les mêmes que ceux employés dans le cas de tâches de maintenance fixes (cf. sous-section 2.1.3.2).

2.2.3 Comparaison des deux approches de résolution

Nous présentons dans ce qui suit les résultats d'expériences numériques obtenus par application de l'heuristique finale combinant l'heuristique H2 (MD) à la recherche tabou, sur les instances RA1 à RA15. L'algorithme final est noté H2-RT (MD), et nous désignons par H2-RT-MF l'heuristique finale utilisée dans le cas de tâches de maintenance fixes (précédemment notée H2-RT). Le décalage maximal pour chaque tâche de maintenance est encore fixé à 100 unités de temps. Les résultats obtenus sont ensuite comparés aux valeurs atteintes par les algorithmes AG-MD et RT-MD.

Le tableau 3.8 présente le meilleur makespan, ainsi que le temps d'exécution associé, pour dix exécutions indépendantes de chaque instance par l'algorithme H2-RT-MD. Dans le premier groupe de colonnes figurent les résultats pour le modèle avec maintenance fixe (H2-RT-MF).

Tableau 3.8 : Résultats de 10 exécutions indépendantes par H2-RT-MD

Problème	n	m	initial	H2-RT-MF		H2-RT-MD	
				meilleur	CPUsecs	meilleur	CPUsecs
RA1	10	5	2022	1712	0.3	1457	0.9
RA2	10	5	2186	1816	0.2	1700	0.2
RA3	10	5	2211	1784	0.2	1721	1.6
RA4	10	5	2179	1909	0.1	1742	1.0
RA5	10	5	1907	1767	0.5	1662	0.5
RA6	10	10	2958	2567	0.7	2425	1.9
RA7	10	10	2921	2671	0.4	2450	0.8
RA8	10	10	2885	2601	0.2	2397	1.3
RA9	10	10	2643	2413	0.7	2223	1.2
RA10	10	10	2776	2456	1.6	2296	1.4
RA11	20	10	3909	3344	5.8	3201	5.5
RA12	20	10	4103	3586	4.6	3417	5.0
RA13	20	10	4007	3625	2.4	3412	2.9
RA14	20	10	3876	3493	3.9	3356	3.7
RA15	20	10	4095	3629	3.7	3354	6.4

Comme cela était le cas pour l'approche basée sur l'heuristique gloutonne, il ressort du tableau 3.8 que la prise en compte de tâches de maintenance flexibles permet d'améliorer de manière significative les valeurs atteintes dans le cas de tâches de maintenance fixes. Cependant, la prise en compte de fenêtres temporelles par l'heuristique H2 semble plus coûteuse que cela ne l'était pour l'heuristique gloutonne HG, notamment sur les instances de plus petites tailles.

Dans le tableau 3.9 suivant nous résumons les résultats obtenus par les méthodes de résolution basées sur l'heuristique gloutonne et sur la résolution de sous-problèmes à deux jobs. Dans le premier groupe de colonnes (H1-MD) apparaissent les meilleures valeurs atteintes par la première approche, et qui sont soit celles de l'algorithme AG-MD soit celles de RT-MD. Le second groupe de colonnes (H2-RT-MD) reprend les valeurs du tableau précédent.

Tableau 3.9 : Comparaison des deux méthodes de résolution

Problème	H1-MD		H2-RT-MD	
	meilleur	CPU secs	meilleur	CPU secs
RA1	1484	0.6	**1457**	0.9
RA2	1700	0.1	1700	0.2
RA3	1732	0.7	**1721**	1.6
RA4	1799	0.2	**1742**	1.0
RA5	1662	0.2	1662	0.5
RA6	2438	0.4	**2425**	0.9
RA7	2481	0.7	**2450**	0.8
RA8	2397	1.0	2397	1.3
RA9	**2220**	0.8	2223	1.2
RA10	2296	0.4	2296	1.3
RA11	3224	4.6	**3201**	5.5
RA12	3456	3.2	**3417**	5.0
RA13	3438	3.0	**3412**	2.9
RA14	3368	1.7	**3356**	3.7
RA15	3399	4.9	**3354**	6.4

Les résultats figurant au tableau 3.9 indiquent que l'heuristique basée sur la résolution de sous-problèmes à deux jobs domine encore l'approche basée sur l'heuristique gloutonne. La différence entre les différents résultats obtenus est du même ordre que dans le cas de tâches de maintenance fixes. Par ailleurs, les temps d'exécution de l'algorithme H2-RT-MD sont aussi globalement plus élevés que ceux de l'algorithme H1.

En conclusion, nous avons présenté dans cette section deux méthodes de résolution approchées permettant de traiter le problème d'ordonnancement strictement non-préemptif du flow shop avec prise en compte de contraintes de disponibilité des machines. Ces approches ont été appliquées dans le contexte de tâches de maintenance fixes et adaptées à la considération de tâches de maintenance flexibles. Clairement, l'introduction de cette dernière hypothèse permet d'optimiser l'activité de maintenance et donc d'obtenir de meilleurs résultats.

Dans la section suivante, ainsi que pour la suite de ce manuscrit, nous supposerons que les périodes d'indisponibilité sont fixes.

3. Problème de type job shop

Nous considérons dans cette section le problème d'ordonnancement de type job shop avec la présence de périodes d'indisponibilité sur les machines. A notre connaissance, il n'existe pas dans la littérature d'étude théorique portant sur ce problème.

Après une description du problème d'ordonnancement dans la sous-section suivante, nous proposons deux méthodes de résolution approchées, analogues à celles présentées pour le problème du flow shop. D'une part, il est possible de développer une heuristique gloutonne qui ordonnance les jobs l'un après l'autre selon une séquence de traitement. D'autre part l'heuristique H2-RT permettant de résoudre le problème d'ordonnancement F, $N_{Cwin} \parallel C_{max}$ est directement applicable au modèle du job shop. L'algorithme *2-jobs*, qui constitue la base de cette approche, permet en effet de résoudre de manière optimale le problème d'ordonnancement de type job shop à deux jobs avec contraintes de disponibilité des machines. Notons que si l'objectif considéré est encore la minimisation du makespan, les approches proposées sont applicables à d'autres critères.

3.1 Description du problème

Le problème d'ordonnancement de type job shop strictement non-préemptif avec la prise en compte de contraintes de disponibilité (fixes) sur les machines se définit de la manière suivante :

- Un ensemble de n travaux $\mathfrak{J} = \{J_1, J_2, \ldots, J_n\}$ doit être réalisé par un ensemble de m machines $M = \{M_1, M_2, \ldots, M_m\}$.
- Chaque travail J_i est constitué d'une séquence linéaire de n_i opérations $\{O_{i1}, O_{i2}, \ldots, O_{i,n_i}\}$.
- Chaque machine ne peut réaliser qu'une opération à la fois et chaque opération O_{ij} nécessite une seule machine à la fois, durant p_{ij} unités de temps.
- Plusieurs périodes d'indisponibilité correspondant à des tâches de maintenance préventive se produisent sur chacune des machines. Les durées et dates de réalisation de ces tâches sont connues à l'avance et fixes.
- Les opérations sont supposées strictement non-préemptives.
- L'objectif est de déterminer les séquences d'entrée des opérations sur les machines de telle sorte que le makespan soit minimal.
- Le problème d'ordonnancement peut être noté J, $N_{Cwin} \parallel C_{max}$ (Schmidt [2000]).

Le problème F, $N_{Cwin} \parallel C_{max}$, étudié à la section précédente, étant un cas particulier du problème J, $N_{Cwin} \parallel C_{max}$, il en résulte que le problème considéré ici est NP-difficile au sens fort dès que le nombre des machines est supérieur à deux. Par ailleurs, il n'existe pas d'heuristique à performance garantie pour la minimisation du makespan (Kubiak et al. [2002]), si plus d'une indisponibilité par machine sont considérées.

3.2 Méthodes de résolution

Nous présentons dans cette sous-section, une première méthode de résolution approchée pour le problème d'ordonnancement J, $N_{Cwin} \parallel C_{max}$, basée sur l'ordonnancement successif des jobs, traités un par un, selon un ordre de priorité. Nous choisissons de noter cette approche AL, pour algorithme de liste. La deuxième méthode, proposée pour résoudre de manière approchée le problème J, $N_{Cwin} \parallel C_{max}$, étant l'algorithme H2-RT de la section précédente, nous ne réitérons pas sa description.

Le principe de fonctionnement de l'algorithme de liste AL est analogue à celui de l'heuristique gloutonne HG développée dans la section consacrée au flow shop.

Etant donnée une séquence représentant un ordre de priorité entre les jobs, AL permet de construire l'ordonnancement réalisable associé à cette séquence. Cette construction ne s'effectue pas sur une machine après l'autre (comme c'est le cas pour l'heuristique HG), mais job par job. En effet, les jobs à ordonnancer ayant chacun une gamme opératoire différente, la construction des ordonnancements se doit de respecter ces gammes. A chaque fois qu'un job J_i est sélectionné dans une séquence de traitement $s = (J_1, J_2,\ldots, J_n)$, AL ordonnance la première opération de la gamme opératoire de J_i, puis la seconde et ainsi de suite. Une fois que toutes les opérations du job J_i sont ordonnancées (c'est-à-dire seulement une fois que le job J_i est ordonnancé), le job suivant J_{i+1} de la séquence de traitement, est considéré.

Les règles qui régissent l'insertion des différentes opérations sont les mêmes que celles définies pour l'heuristique gloutonne HG, à savoir :

- Toute opération déjà placée l'est de manière définitive.
- AL essaie d'ordonnancer les opérations au plus tôt et repart de la date de fin de l'opération précédente sur la gamme du job concerné, à chaque fois qu'elle tente d'insérer une opération.

Les ordonnancements obtenus sont donc des ordonnancements actifs, dans la mesure où une opération peut être placée sur une machine strictement à gauche d'une opération ordonnancée antérieurement, c'est-à-dire que nous ne laissons pas d'intervalle vide où une opération pourrait être avancée sans en retarder d'autres.

K_j tâches de maintenance sont imposées sur chacune des machines. L'horizon de planification de toute machine M_j est ainsi décomposé en K_j+1 sous-intervalles $I_{j1}, I_{j2},\ldots, I_{j,K_j+1}$, la longueur d'un intervalle I_{jh} étant notée L_{jh}.

Pour insérer une opération O_{ij} d'un job J_i considéré, AL détermine le premier sous-intervalle sur l'horizon de planification de la machine M_j, dans lequel O_{ij} peut être insérée au plus tôt, sans recouvrir une tâche de maintenance ou une opération déjà ordonnancée. Après chaque insertion, la longueur de l'intervalle d'accueil est mise à jour et un intervalle de plus est créé, le cas échéant.

L'algorithme 3.7 suivant permet de résoudre de manière approchée le problème strictement non-préemptif J, $N_{Cwin} \parallel C_{max}$.

Algorithme 3.7 : AL

Données : Une séquence d'entrée s. Sans perte de généralité, soit s = (J_1, J_2,..., J_n) cette séquence

Résultat : Ordonnancement et makespan associé

Début *Construction*

Pour chaque job J_i, i allant de 1 à n **Faire**

/ étape d'insertion */*

Pour chaque opération O_{ij}, j allant de 1 à n_i **Faire**

- Trouver l'intervalle d'accueil de l'opération O_{ij}
- Mettre à jour la longueur de l'intervalle d'accueil

Fin Pour.

Fin Pour.

Fin

Les performances de l'algorithme ci-dessus dépendant fortement de l'ordre de traitement des jobs, nous proposons de le coupler à une recherche tabou permettant d'optimiser la séquence d'entrée. Nous choisissons d'employer les mêmes paramètres que ceux utilisé par l'heuristique finale H2-RT, afin de pouvoir comparer les deux heuristiques AL et H2.

3.3 Résultats numériques

Nous comparons dans cette sous-section les performances de l'heuristique finale AL-RT combinant l'algorithme AL et la recherche tabou, à celles de H2-RT, sur les instances RA1 à RA15 utilisées jusqu'à présent. Les gammes opératoires des jobs sont générées aléatoirement.

Le tableau 3.10 suivant présente les meilleurs makespan, les makespan moyen, ainsi que les temps d'exécution associés pour dix exécutions de chacune des instances. Le premier groupe de colonnes contient les résultats obtenus par l'heuristique AL-RT, et dans le second groupe de colonnes apparaissent les valeurs atteintes par H2-RT.

Tableau 3.10 : Comparaison des deux heuristiques

Problème	AL-RT			H2-RT		
	moyen	meilleur	CPUsecs	moyen	meilleur	CPUsecs
RA1	1642	1582	0.4	1581	**1502**	0.9
RA2	1693	1583	0.3	1680	**1529**	0.4
RA3	1845	1784	0.1	1846	1784	0.7
RA4	1689	1646	0.2	1589	**1526**	0.5
RA5	1667	1643	1.0	1643	**1544**	0.9
RA6	2458	2290	0.5	2254	**2120**	1.1
RA7	2472	2311	0.5	2364	2311	0.4
RA8	2283	2225	0.2	2239	**2088**	0.9
RA9	2199	2113	0.7	2110	**2046**	0.7
RA10	2383	2326	0.1	2345	**2268**	0.4
RA11	3453	3329	2.7	3095	**2999**	4.9
RA12	3644	3601	3.2	3355	**3257**	3.4
RA13	3441	3222	4.1	3216	**3098**	4.6
RA14	3499	3407	3.3	3393	**3292**	4.1
RA15	3503	3376	2.5	3305	**3155**	3.5

De manière évidente, il apparaît dans le tableau 3.10 que l'approche H2-RT est plus performante que AL-RT. Pour la majorité des instances testées (excepté RA3 et RA7, pour lesquelles il y a égalité), l'écart entre les meilleures valeurs atteintes par les deux approches est, en effet, assez important. En ce qui concerne les temps d'exécution, H2-RT est dans l'ensemble plus coûteuse que AL-RT.

La recherche tabou, étant appliquée avec les mêmes paramètres pour les deux approches, nous pouvons en déduire que les interprétations précédentes s'appliquent non pas aux approches globales (heuristiques + recherche tabou) mais directement aux heuristiques AL et H2.

4. Conclusion

Nous avons présenté dans ce chapitre des méthodes de résolution approchées pour les problèmes d'ordonnancement généraux, c'est-à-dire à plus de deux jobs, de type flow shop et job shop avec prise en compte de contraintes de disponibilité des machines.

Pour le problème du flow shop, deux variantes du problème strictement non-préemptif ont été abordées. En premier lieu, nous nous sommes placés dans le contexte de périodes d'indisponibilité dues à des tâches de maintenance fixes. Deux méthodes de résolution approchées ont été développées et comparées : d'une part, une heuristique gloutonne qui ordonnance les jobs l'un après l'autre, sur chaque machine, selon un ordre de priorité, et d'autre part, une heuristique basée sur la résolution par l'algorithme polynomial du chapitre précédent, de sous-problèmes à deux jobs. En l'absence de benchmarks spécifiques dans la littérature, les tests sur les deux approches ont été menés sur des instances générées aléatoirement. Clairement, l'heuristique utilisant l'algorithme polynomial donne de meilleurs résultats que l'approche gloutonne.

Afin d'optimiser l'activité de maintenance correspondant aux indisponibilités des machines, une seconde variante du problème a été proposée. Elle consiste à allouer une fenêtre temporelle à l'exécution de chaque tâche de maintenance. Des résultats d'expériences ont nettement montré l'intérêt d'introduire cette flexibilité dans le processus de maintenance.

Le problème d'ordonnancement du job shop avec prise en compte de périodes d'indisponibilité fixes sur les machines a ensuite été traité. Les méthodes approchées développées pour le flow shop ont été adaptées au job shop, puis testées. L'heuristique basée sur la résolution de sous-problèmes à deux jobs a, là encore, donné les meilleurs résultats.

Enfin, les performances des approches proposées ne pourront être validées qu'après la connaissance des valeurs optimales pour les instances testées. C'est ce qui fait l'objet du chapitre suivant, dans lequel nous proposons des méthodes de résolution exactes pour les problèmes d'ordonnancement de type flow shop et job shop avec la prise en compte de périodes d'indisponibilité des machines.

Chapitre 4

Méthodes de Résolution Exactes

Nous proposons dans ce chapitre des méthodes de résolution exactes, à savoir des procédures par séparation et évaluation, pour les problèmes d'ordonnancement du flow shop et du job shop, avec la prise en compte de contraintes de disponibilité des machines. Le graphe disjonctif est employé pour la modélisation des différents nœuds de l'arborescence et une variante de l'algorithme polynomial, développé dans le second chapitre de la thèse, est proposée pour le calcul des bornes inférieures. Les résultats d'expériences effectuées sur les instances générées au troisième chapitre, permettent de montrer l'efficacité des approches exactes présentées ainsi que celle des méthodes heuristiques du chapitre précédent.

1. Introduction

Nous avons proposé et comparé dans le chapitre précédent, plusieurs heuristiques originales permettant de résoudre de manière approchée les problèmes d'ordonnancement de type flow shop et job shop, en présence de périodes d'indisponibilité des machines. Cependant, l'étude de ces problèmes serait incomplète sans la confrontation des heuristiques proposées avec des méthodes de résolution exactes. En effet, avant d'essayer d'améliorer les solutions approchées aux différents problèmes étudiés, il est souhaitable de connaître les valeurs optimales.

Dans ce dernier chapitre, nous proposons des procédures par séparation et évaluation pour les problèmes d'ordonnancement étudiés au chapitre précédent à savoir ceux du flow shop et du job shop à m machines, avec la prise en compte de contraintes de disponibilité. Le job shop étant plus général, nous consacrons notre étude à ce modèle. Par ailleurs, nous choisissons, pour objectif la minimisation du makespan. Cependant, les algorithmes proposés sont, comme dans les deux chapitres précédents, généralisables à la considération de tout critère régulier.

La section suivante est dédiée à la description de la procédure par séparation et évaluation générale. La modélisation du problème, le schéma de séparation des nœuds, ainsi que les différentes bornes utilisées pour leur évaluation, y sont présentés. En particulier, nous proposons une borne inférieure basée sur la résolution de sous-problèmes à deux jobs avec contraintes de précédence et dates de disponibilité sur les opérations à ordonnancer. Cette résolution se fait évidemment au moyen d'une extension de l'algorithme polynomial développé dans le deuxième chapitre de cette thèse.

Dans la troisième section de ce chapitre, nous présentons et commentons les résultats d'expériences menées sur les instances générées aléatoirement au chapitre précédent. En particulier, nous comparons les meilleures solutions atteintes par les différentes heuristiques aux solutions optimales déterminées par les procédures par séparation et évaluation, et ceci, pour tous les problèmes traités.

2. Procédure par séparation et évaluation

Nous développons dans cette section une procédure par séparation et évaluation (PSE) permettant de résoudre de manière exacte le problème d'ordonnancement de type job shop en présence de périodes d'indisponibilité des machines.

Le principe des procédures par séparation et évaluation a été introduit dans le premier chapitre. Nous présentons donc dans ce qui suit les paramètres propres de la PSE que nous employons, à savoir :

- Le mode de représentation du problème.
- La manière dont sont effectuées les séparations de nœuds.
- Les sélections immédiates proposées.
- Les bornes utilisées pour l'évaluation des différents nœuds de l'arborescence.

2.1 Modélisation

Les procédures par séparation et évaluation les plus efficaces pour la résolution de problèmes d'ordonnancement de type job shop sont basées sur la représentation par le graphe disjonctif, introduite par Roy et Sussmann [1964]. Aussi, choisissons-nous d'employer cette modélisation dans notre PSE.

Etant donnée une instance du problème J, $N_{Cwin} \parallel C_{max}$, le graphe disjonctif $G = (V, C \cup D)$, est défini de la manière suivante :

V est l'ensemble des nœuds, chacun d'eux représentant une opération d'un job à ordonnancer. Les deux nœuds particuliers que sont la source O et le puits * sont ajoutés à l'ensemble V. C est l'ensemble des arcs conjonctifs représentant les contraintes de précédence entre les opérations d'un même job. Le partage d'une machine entre deux opérations est modélisé par un arc disjonctif, D étant l'ensemble de ces arcs. Tout arc sortant d'un nœud est pondéré par la durée opératoire de l'opération correspondante, mis à part les arcs partant de O qui sont pondérés par 0, ce qui signifie que tous les jobs sont disponibles à l'instant 0.

Afin de représenter les différentes périodes d'indisponibilité des machines, des jobs fictifs sont introduits. Chaque job fictif correspond à l'activité de maintenance à réaliser sur une machine.

Considérons par exemple une machine M_j sur laquelle K tâches de maintenance sont à exécuter. Du fait que chaque tâche de maintenance h_{jk} (k = 1,..., K) doit être réalisée à des dates connues à l'avance et fixes, un ordre de passage existe entre ces tâches. Ainsi, le job fictif correspondant à la machine M_j est composé de K nœuds, chacun d'eux modélisant une tâche de maintenance h_{jk}, et ces nœuds sont reliés par des arcs conjonctifs selon leur ordre de passage. Un arc conjonctif allant du nœud h_{jk} au nœud $h_{j,k+1}$ est pondéré par la durée de la tâche de maintenance h_{jk}.

Pour exprimer le fait qu'une tâche de maintenance h_{jk} doit nécessairement démarrer à la date S_{jk}, deux arcs conjonctifs (O, h_{jk}) et (h_{jk}, O) entre l'origine O et h_{jk} sont introduits. Le premier arc (O, h_{jk}) de poids S_{jk} indique que la tâche de maintenance doit démarrer au moins à la date S_{jk}, tandis que le second arc (h_{jk}, O) ayant un poids égal à $(-S_{jk})$ empêche la tâche de maintenance de démarrer après la date S_{jk}.

La figure 4.1 suivante présente la modélisation de deux tâches de maintenance $h_{j1} = [10, 15]$ et $h_{j2} = [50, 70]$ à réaliser sur une machine M_j.

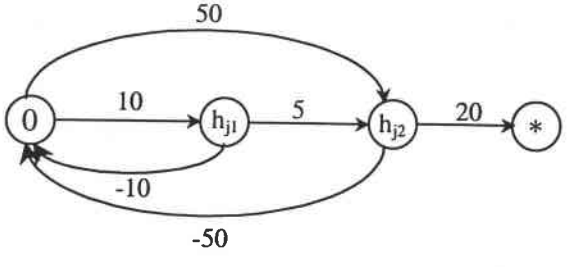


Figure 4.1 : Modélisation de l'activité de maintenance

Remarque 4.1 : La modélisation proposée permet de prendre en compte de la flexibilité sur l'activité de maintenance. Il suffit pour cela de modifier les valuations des arcs liant l'origine et les tâches de maintenance, de sorte à intégrer les limites d'exécutions de ces tâches.

En effet, considérons l'exemple de la figure 4.1 et supposons que les tâches h_{j1} et h_{j2} puissent éventuellement être avancées de 5 unités de temps. Pour h_{j1}, qui peut alors démarrer entre les dates 5 et 10, l'arc (O, h_{j1}) sera pondéré par 5, tandis que l'arc (h_{j1}, O) restera pondéré par -10. Les poids des arcs (O, h_{j2}) et (h_{j2}, O) seront respectivement égaux à 45 et -50 (figure 4.2).

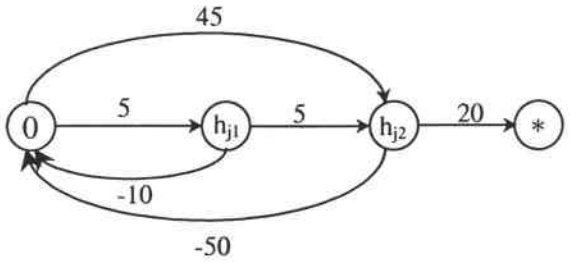
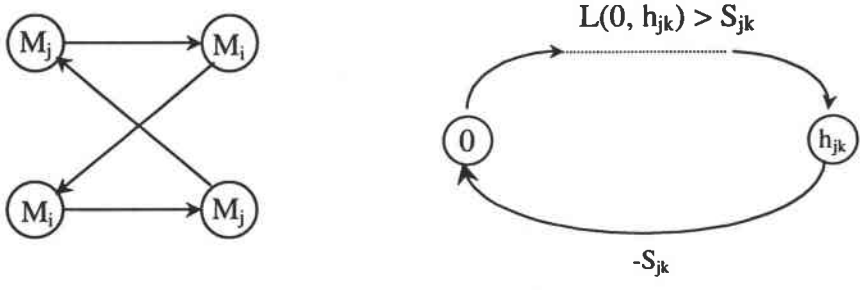


Figure 4.2 : Prise en compte de flexibilité

Une solution réalisable du problème d'ordonnancement J, $N_{Cwin} \parallel C_{max}$ (ou J, $N_{Cwin\text{-}flex} \parallel C_{max}$) est obtenue en choisissant une orientation pour chaque arc disjonctif, de telle sorte que le graphe résultant ne contienne aucun circuit de longueur strictement positive. En effet, si un tel circuit P existe alors :

(i) Soit P ne contient aucun arc de poids négatif, ce qui exprime une incompatibilité dans le séquencement des opérations (figure 4.3 (a)).

(ii) Soit P contient un arc e_l de poids $(-S_{jk})$ négatif. Dans ce cas, l'origine O appartient nécessairement au circuit et est reliée à une tâche de maintenance h_{jk} par l'arc e_l. Puisque la longueur de P est strictement positive, la longueur $L(O, h_{jk})$ du chemin allant de O à h_{jk} est forcément strictement supérieure à S_{jk}, ce qui signifie que h_{jk} n'est pas réalisée à sa date de début S_{jk} (figure 4.3 (b)).



(a) problème de séquencement (b) problème de maintenance

Figure 4.3 : Circuits de longueur strictement positive

L'orientation de tous les arcs disjonctifs étant fixée (on parle alors de graphe disjonctif arbitré), la valeur du makespan associée à la solution ainsi modélisée est donnée par la longueur du plus long chemin allant de l'origine O au puits *.

La procédure par séparation et évaluation consiste à construire une arborescence où chaque nœud r correspond à une solution partielle du problème J, $N_{Cwin} \parallel C_{max}$. Cette solution est modélisée par un graphe disjonctif $G_r = (V, C \cup DF_r)$, dans lequel l'orientation de tout arc disjonctif $e \in DF_r$ est fixée. Initialement, l'arborescence contient un seul nœud : la racine. Aucune orientation d'arc n'est fixée pour ce nœud, qui représente ainsi toutes les solutions du problème.

Les successeurs d'un nœud sont obtenus par la fixation de l'orientation de plusieurs arcs disjonctifs supplémentaires. L'exploration d'un nœud est arrêtée si toutes les disjonctions sont fixées ou s'il peut être montré que le nœud ne contient pas de solution strictement meilleure que la meilleure solution courante. Dans le premier cas une solution réalisable est obtenue, tandis que le second cas se vérifie au moyen de bornes.

2.2 Schéma de séparation

Soit r un nœud de l'arborescence et notons S(r) l'espace de toutes les solutions qui ont en commun les arcs disjonctifs fixés de l'ensemble DF_r.

Le schéma de séparation appliqué à r consiste en premier lieu à choisir une machine M_k. Les opérations à exécuter sur cette machine ne doivent pas toutes être ordonnées, c'est-à-dire que certaines d'entre elles doivent être connectées par des arcs disjonctifs. La seule exception concerne les tâches de maintenance qui seraient connectées à des opérations par des arcs disjonctifs, bien qu'elles soient liées entre elles par des arcs conjonctifs. Les L opérations non-ordonnées (dont peuvent faire partie des tâches de maintenance) définissent ainsi une *quasi-clique de disjonction K*.

Une fois la machine M_k sélectionnée et la quasi-clique de disjonction *K* identifiée, l'ensemble S(r) est alors séparé en L sous-ensembles disjoints $S(r_1), S(r_2), \ldots, S(r_L)$ tels que :

- $\bigcup_{i=1}^{L} S(r_i) = S(r)$.
- $S(r_i)$ est l'ensemble des solutions associées au graphe disjonctif $G_{r_i} = (V, C \cup DF_{r_i})$, où $DF_r \subset DF_{r_i}$.

Un ensemble $S(r_i)$ est obtenu à partir de S(r) en imposant que l'opération $O_{ij} \in K$ soit ordonnancée avant toutes les opérations $O_{rs} \in K \setminus \{O_{ij}\}$. Ainsi, le nœud r a au maximum L successeurs. Par ailleurs, pour chaque nœud r_i le graphe disjonctif associé possède L arcs disjonctifs fixés de plus que le graphe de r.

Le choix d'employer ce mode de séparation est motivé par l'intérêt de fixer un grand nombre de disjonctions d'un nœud de l'arborescence à ses successeurs. Ceci permet d'augmenter la qualité de la procédure par séparation et évaluation, pour les raisons suivantes :

- Un grand nombre de successeurs ne sont pas réalisables, car leur graphe disjonctif associé comportent des circuits, et ne sont pas à explorer.
- La valeur de la borne inférieure augmente plus rapidement, en particulier lorsque celle-ci est basée sur la résolution de sous-problèmes à deux jobs, ce qui sera détaillé dans la sous-section suivante.

Dans notre application, la machine sélectionnée pour effectuer la séparation est celle dont le nombre d'arcs disjonctifs restants à fixer, c'est-à-dire la cardinalité de la quasi-clique de disjonction, est maximal.

Notons que d'autres schémas de séparation existent pour des approches utilisant le graphe disjonctif, notamment la séparation basée sur le concept de blocs développée par Grabowski et al. [1986] et améliorée par Brucker et al. [1994].

2.3 Fixation des disjonctions

2.3.1 Dates de disponibilité et durées de latence

A chaque nœud r de l'arborescence, des informations nécessaires au calcul des bornes inférieures ainsi qu'à la détermination des arcs disjonctifs à orienter, sont calculées. Ces informations, relatives aux nœuds du graphe disjonctif G_r, c'est-à-dire à chaque opération, sont la date de disponibilité et la durée de latence.

La date de disponibilité r_{ij} d'une opération O_{ij} est la date au plus tôt à laquelle peut démarrer O_{ij}. Cette date peut être directement calculée par le plus long chemin allant de la source O au nœud O_{ij} du graphe disjonctif, ce qui est donné par la formule récursive :

- $r_O = 0$.
- $r_{ij} = \max\{r_{i,j-1} + p_{i,j-1}, \max_{Pred(O_{ij})} \{r_{st} + p_{st}\}\}$, où $Pred(O_{ij})$ est l'ensemble des opérations ordonnancées avant O_{ij}.

La durée de latence q_{ij} de l'opération O_{ij} est la longueur du plus long chemin allant de la fin de l'exécution de O_{ij} au puits $*$. Cette valeur est calculée par la formule récursive suivante :

- $q_* = 0$.
- $q_{ij} = \max\{q_{i,j+1} + p_{i,j+1}, \max_{Succ(O_{ij})} \{q_{st} + p_{st}\}\}$, où $Succ(O_{ij})$ est l'ensemble des opérations ordonnancées après O_{ij}.

Notons que si la date de disponibilité du puits $*$ n'est pas obtenue par la formule de calcul des r_{ij}, cela signifie que le graphe disjonctif considéré contient un circuit de longueur positive.

2.3.2 Sélections immédiates

Proposée pour la première fois par Carlier et Pinson [1989] pour le cas du job shop classique, la sélection immédiate est une procédure qui permet de fixer directement l'orientation de certains arcs disjonctifs. Elle est basée sur des comparaisons entre la borne supérieure BS du makespan et des valeurs faisant intervenir les dates de disponibilité et durées de latence des opérations.

Soient O_{ij} et O_{st} deux opérations du graphe disjonctif G_r, associé à un nœud r de l'arborescence. L'orientation de l'arc disjonctif liant les deux opérations est fixée dans la direction $O_{ij} \rightarrow O_{st}$ (resp. $O_{st} \rightarrow O_{ij}$), si la condition (i) (resp. (ii)) est vérifiée :

(i) $r_{st} + p_{st} + p_{ij} + q_{ij} \geq BS$

(ii) $r_{ij} + p_{ij} + p_{st} + q_{st} \geq BS$

Si les deux conditions sont simultanément vérifiées, alors le nœud r ne contient pas de solution optimale.

D'autre part, lorsque l'orientation $O_{ij} \rightarrow O_{st}$ est sélectionnée, la date de disponibilité de l'opération O_{st} et la durée de latence de O_{ij} sont mises à jour, c'est-à-dire : $r_{st} = \max (r_{st}, r_{ij} + p_{ij})$ et $q_{ij} = \max (q_{ij}, q_{st} + p_{st})$. De même si l'orientation inverse est sélectionnée, les valeurs r_{ij} et q_{st} sont ainsi mises à jour : $r_{ij} = \max (r_{ij}, r_{st} + p_{st})$ et $q_{st} = \max (q_{st}, q_{ij} + p_{ij})$.

A ces tests s'ajoutent des sélections immédiates relatives à l'activité de maintenance. Considérons une machine M_j et la quasi-clique de disjonction associée K, dont fait partie une tâche de maintenance h_{jk}. L'orientation de l'arc disjonctif liant une opération $O_{ij} \in K$ et h_{jk} est fixée dans la direction $h_{jk} \rightarrow O_{ij}$, si la condition suivante est vérifiée :

(iii) $r_{ij} + p_{ij} > S_{jk}$.

La valeur de r_{ij} est alors mise à jour : $r_{ij} = \max (r_{ij}, T_{jk})$, où T_{jk} est la date de fin d'exécution de la tache de maintenance h_{jk}.

2.4 Evaluation des nœuds

2.4.1 Borne supérieure

Avant que ne commence la construction de l'arborescence, une borne supérieure BS est calculée. Elle est donnée par la meilleure valeur (en l'occurrence le plus petit makespan) obtenue par application des heuristiques proposées au chapitre précédent. Cette borne supérieure est donc calculée une première fois alors qu'aucune orientation d'arc disjonctif n'a été fixée.

Cependant, durant l'exploration des branches de l'arborescence, de plus en plus d'arcs disjonctifs sont fixés, ce qui réduit progressivement l'espace des solutions. Afin d'en améliorer la valeur, nous choisissons donc de recalculer BS en tenant compte des nouvelles contraintes de précédence, et ce, à chaque exploration de nœud.

Ce calcul est effectué au moyen d'une heuristique d'insertion quasi-analogue à celles du chapitre précédent. Cette heuristique consiste en effet à ordonnancer au plus tôt les différentes opérations, tout en respectant les périodes d'indisponibilité des machines.

La différence avec les algorithmes de liste proposés au chapitre 3 réside dans le fait que les opérations ne sont pas insérées selon un ordre de priorité entre les jobs, mais selon l'ordre donné par une numérotation des sommets du graphe disjonctif G_s, associé au nœud de l'arborescence s considéré, compatible avec la fonction 'rang' du graphe.

En effet, puisque le graphe G_s est sans circuits de longueur strictement positive, il existe une partition $P = P_1 \cup P_2 \cup \ldots \cup P_h$ de l'ensemble de ces sommets telle que tous les arcs du graphe vont d'un sommet de niveau l à un sommet de niveau l', avec $l < l'$. Par suite, il est possible d'établir une numérotation des sommets de G_s. Notons que les arcs de poids négatifs ne sont pas pris en compte dans cette numérotation.

2.4.2 Bornes inférieures

A chaque exploration d'un nœud s de l'arborescence, des bornes inférieures sont calculées. La première borne $LB1 = r_{ij} + p_{ij} + q_{ij}$ est déterminée durant le calcul des dates de disponibilité et durées de latence associées à s. Plus précisément, s'il existe une opération O_{ij} telle que $r_{ij} + p_{ij} + q_{ij} \geq BS$, alors le nœud n'est pas exploré.

La seconde borne LB2 est basée sur la résolution de sous-problèmes particuliers à deux jobs, connus sous le nom de *relaxations à deux jobs*. Cette méthodologie a été initialement développée par Brucker et Jurisch [1993] pour le problème du job shop classique, puis reprise par Jurisch [1995] pour le job shop flexible. Une relaxation à deux jobs est définie de la manière suivante :

Soient deux jobs J_1 et J_2 composés de n_i opérations O_{i1}, O_{i2},..., O_{in_i} (i = 1,2). A chaque opération O_{ij} sont associées une date de disponibilité r_{ij} et une durée de latence q_{ij}. Des contraintes de précédences, dues à l'orientation d'arcs disjonctifs, existent entre certaines opérations O_{1i} et O_{2j} des deux jobs partageant une même machine.

La borne inférieure LB2 consiste alors à déterminer la solution optimale au problème suivant :

$$PM = \begin{cases} \min \max\left\{C_{ij} + q_{ij} \mid i = 1,2;\ j = 1,\ldots,n_i\right\} \\ \text{Sous les contraintes :} \\ \quad C_{ij} - p_{ij} \geq r_{ij} \\ \quad \text{Toutes les contraintes de précédence sont satisfaites} \\ \quad C_{ij} \geq 0 \end{cases}$$

où C_{ij} est la date de fin d'exécution de l'opération O_{ij}.

En effet, soient s un nœud de l'arborescence et G_s le graphe disjonctif associé à ce nœud. Des orientations d'arcs disjonctifs sont fixées entre plusieurs sommets de G_s.

En particulier, si nous considérons deux jobs J_1 et J_2, plusieurs types d'arcs disjonctifs fixés sont à distinguer :

- Les arcs liant des opérations des deux jobs, définissant ainsi des contraintes de précédence entre ces opérations.
- Les arcs arrivant aux opérations des deux jobs (et venant d'autres jobs), fixant ainsi des dates de disponibilités.
- Les arcs partant des opérations des deux jobs (et allant vers d'autres jobs), permettant de définir les durées de latence.

La figure 4.4 suivante illustre ces trois types d'arcs disjonctifs orientés pour deux jobs J_1 et J_2 composés chacun de 4 opérations.

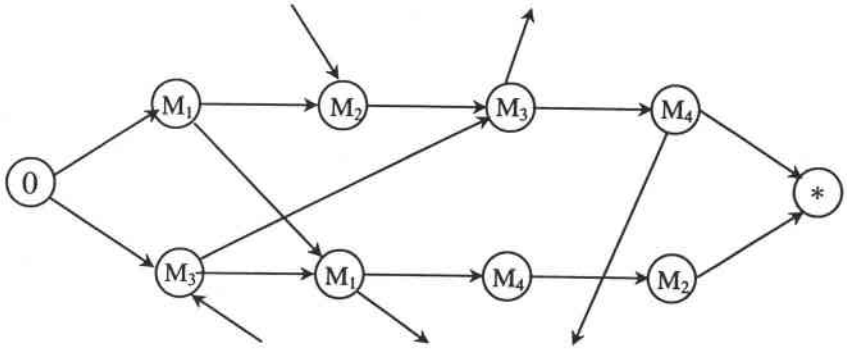


Figure 4.4 : Arcs orientés impliquant deux jobs

Résoudre le problème PM revient alors à trouver une orientation pour les arcs disjonctifs non fixés qui lient les opérations des deux jobs, tout en respectant les contraintes établies par les arcs déjà orientés, de telle sorte que la quantité $C^* = \max_{O_{ij}} \{C_{ij} + q_{ij}\}$ soit minimale.

L'introduction des durées de latence q_{ij} dans la fonction objectif C^* a pour but d'intégrer les arcs sortant des opérations des deux jobs. Si aucun arc de ce type n'existe, le problème est équivalent à minimiser le makespan. En revanche, le fait que notre schéma de séparation soit basé sur la fixation simultanée de plusieurs arcs disjonctifs, permet d'accroître la valeur de la borne inférieure LB2 dans la mesure où cela fait augmenter plus rapidement les durées de latence.

Le problème PM peut être résolu de manière polynomiale par une variante de l'algorithme *2-jobs* proposé au second chapitre de cette thèse. Cette variante consiste en effet à résoudre dans un premier temps le problème PM sans tenir compte des durées de latence des opérations ($q_{ij} = 0$) et en mémorisant des points intermédiaires correspondant à la fin de toutes les opérations qu'il n'était pas nécessaire de retenir dans l'algorithme initial. L'algorithme *2-jobs* modifié est donc appliqué, et un réseau N, dont la longueur du plus court chemin est égale au makespan, est ainsi construit.

Dans un second temps, une recherche est effectuée pour déterminer si d'autres chemins de N permettent d'améliorer la fonction objectif du problème PM.

L'algorithme 4.1 suivant permet de résoudre le problème PM et donc de trouver une borne inférieure LB2.

Algorithme 4.1 : LB2

Données : Deux jobs et toutes les contraintes

Résultat : Valeur optimale C* du problème PM

Etape 1 : *Initialisation*

- Appliquer l'algorithme 2-jobs modifié pour construire un réseau de travail initial N
- $C^* = +\infty$

Etape 2 : *Recherche du plus court chemin*

2.1. Calculer le plus court chemin P du réseau N. Soit C_{ij} la date de fin de toute opération O_{ij}, en suivant ce chemin

2.2. Déterminer l'opération critique O_{rs} telle que $C_{rs} + q_{rs} = \max_{O_{ij}} \{C_{ij} + q_{ij}\}$.

Sans perte de généralité, O_{rs} est une opération du job J_1

2.3. **Si** $C_{rs} + q_{rs} < C^*$ **Alors** $C^* = C_{rs} + q_{rs}$

2.4. Déterminer l'arc $e = (v_1, v_2)$ de P croisant la verticale V^{rs} (dans le plan) associée à la fin de O_{rs}

Etape 3 : *Condition d'arrêt*

3.1. Déterminer l'ensemble E des arcs du réseau N qui croisent V^{rs}

3.2. **Si** $E \neq \varnothing$ **Alors** Calculer $C^\circ = \min_{E} C_{ij}$

Sinon Arrêter

3.3. **Si** $C^\circ + q_{ij} < C^*$ **Alors** Aller à l'étape 4

Sinon Arrêter

Etape 4 : *Mise à jour du réseau N*

- Supprimer le nœud v_1 du réseau N
- Aller à l'étape 2

A l'étape 2.2 de l'algorithme, l'opération O_{rs}, appelée *opération critique*, pour laquelle la quantité $C = \{C_{rs} + q_{rs}\}$ est maximale est identifiée. Sans perte de généralités, nous supposons dans ce qui suit que l'opération critique appartient au job J_1. Notons qu'à la première itération de l'algorithme, C est affectée à la valeur C* de la fonction objectif du problème PM. Nous avons ainsi une solution réalisable de PM que nous essayons de faire décroître.

La durée de latence q_{rs} étant fixée, l'amélioration de C passe obligatoirement par la diminution de la quantité C_{rs}. Dans la représentation dans le plan avec obstacles, la fin d'exécution de O_{rs} correspond au croisement d'un arc e = (v_1, v_2) du plus court chemin, avec la ligne verticale V^{rs} (figure 4.5).

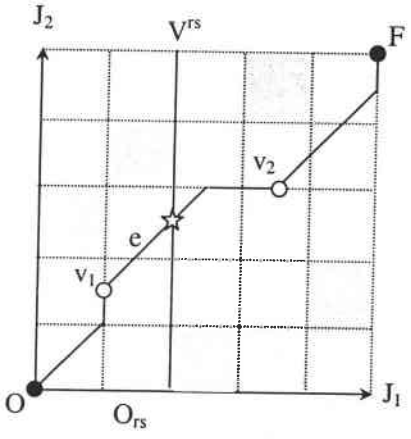


Figure 4.5 : Opération critique dans le plan

L'étape 3 consiste ensuite à déterminer si pour un autre chemin du réseau construit par l'algorithme *2-jobs*, l'opération O_{rs} peut s'achever avant la date C_{rs}, ce qui revient à évaluer les dates de fin de l'opération O_{rs} sur l'ensemble E des arcs qui croisent la ligne V^{rs} dans le plan (figure 4.6).

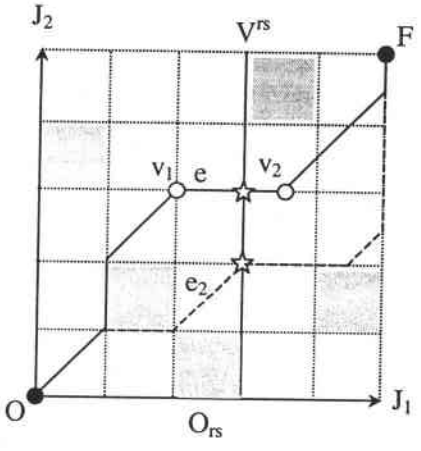


Figure 4.6 : Arcs croisant V^{rs}

Sur la figure 4.6, deux arcs e et e_2 croisent la verticale V^{rs} correspondant à la fin de l'opération critique O_{rs}. L'arc e appartient au plus court chemin allant de O à F et C_{rs} est la date de fin d'exécution de l'opération O_{rs} en empruntant ce plus court chemin. Cependant, si nous empruntons le chemin auquel appartient l'arc e_2, nous obtenons une date de fin de l'opération O_{rs} inférieure à C_{rs}.

Il est évident que si aucun arc croisant V^{rs} n'existe ($E = \varnothing$), comme cela est le cas sur la figure 4.5, C* est optimale. En revanche, si $E \neq \varnothing$ trois situations sont possibles :

- Soit la valeur C ne peut pas être améliorée et dans ce cas C* est optimale.
- Soit C peut être améliorée et la nouvelle valeur $C^\circ + q_{rs}$ est inférieure à C*. Dans ce cas, nous nous intéressons au plus court chemin du réseau qui n'emprunte pas le sommet v_1 (étape 4 de l'algorithme).
- Soit C peut être améliorée mais la nouvelle valeur $C^\circ + q_{rs}$ est supérieure à C*. Dans ce cas C* est optimale, du fait que la nouvelle valeur de la fonction objectif C' obtenue après suppression de v_1 est telle que : $C' \geq C^\circ + q_{rs} \geq C^*$.

Notons que ce dernier cas ne se produit pas au premier passage à l'étape 2.

Lemme 4.1

La valeur $C = \{C_{rs} + q_{rs}\}$ obtenue à l'étape 2.2 est optimale si l'une des conditions suivantes est satisfaite :

(i) Le chemin $P(O, v_1)$ du plan avec obstacles allant de O à v_1 ne heurte aucun obstacle.

(ii) $P(O, v_1)$ ne passe que par les bordures inférieures des obstacles.

Preuve

Il suffit de démontrer que sous l'une des conditions (i) ou (ii), l'opération critique O_{rs} ne peut être achevée avant la date C_{rs}.

Si la condition (i) est satisfaite, alors le chemin $P(O, v_1)$ ne passe que par des sommets singuliers, des sommets d'attentes, et des sommets réguliers différents de coins nord-ouest et sud-est des obstacles. Un minorant de la date de fin C_{rs} est égal à la somme S des durées des opérations du job J_1 précédant O_{rs}, à laquelle s'ajoutent des attentes de disponibilités éventuelles, dont la durée totale est notée A.

Sur le chemin $P(O, v_1)$, la somme des longueurs des segments horizontaux et diagonaux est exactement égale à S. Les segments verticaux correspondent, quant à eux, aux attentes de disponibilité.

Puisque les opérations du job J_1 sont ordonnancées dès que les machines nécessaires à leur exécution sont disponibles, la somme des longueurs des segments verticaux est exactement égale à A. Par conséquent, il n'existe aucun autre chemin dans le plan qui permette d'améliorer la valeur C_{rs}.

Si la condition (ii) est satisfaite, nous supposons sans perte de généralité qu'aucun problème de disponibilité ne se pose sur le chemin $P(O, v_1)$ car sinon les arguments précédents sont utilisés. $P(O, v_1)$ est donc seulement constitué de segments horizontaux et diagonaux, ce qui signifie que le job J_1 n'est jamais interrompu. Par conséquent, aucun autre chemin du plan avec obstacles ne permet d'améliorer la valeur C_{rs}. ■

Remarque 4.2 : Dans le cas où l'opération critique appartient au job J_2, le raisonnement est analogue. La différence réside dans l'étape 2.4 de l'algorithme, où nous nous intéressons à l'arc de P croisant l'horizontale correspondant à la fin de l'exécution de l'opération critique. La condition suffisante (i) du lemme 4.1 reste valable, et (ii) doit être remplacée par :

(ii)' $P(O, v_1)$ ne passe que par les bordures gauches des obstacles.

Dans ce qui suit, nous montrons que l'algorithme 4.1 est suffisant pour déterminer une solution optimale au problème PM et nous donnons sa complexité.

Théorème 4.1

L'algorithme 4.1 permet de résoudre le problème PM de manière polynomiale, et sa complexité est au plus égale à $O(Ks^6)$, où $s = \max\{n_1, n_2\}$ et K est le nombre maximal de tâches de maintenance sur une même machine.

Preuve

L'algorithme 4.1 s'arrête si l'ensemble d'arcs E est vide ou si l'une des conditions du lemme 4.1 est vérifiée. Dans le premier cas, tout chemin de O à F passe obligatoirement par l'arc e déterminé à l'étape 2.4. Par conséquent la valeur C* de la fonction objectif de PM ne peut pas être améliorée. Dans le deuxième cas, le lemme 4.1 permet de conclure que la valeur C* est optimale.

Si aucune de ces situations ne se produit, la seule manière d'améliorer C* est d'empêcher le plus court chemin de passer par le sommet v_1, ce qui est réalisé à l'étape 4 de l'algorithme. Par conséquent, l'algorithme 4.1 permet d'obtenir une solution optimale au problème PM.

La construction du réseau de travail N à l'étape 1 de l'algorithme prend un temps proportionnel à $(K\ s^4)$. De plus, le nombre de fois qu'un plus court chemin est déterminé (étape 2) est majoré par le nombre de sommets du réseau N, qui est proportionnel à $O((n_1 \times n_2)\ (n_1 + n_2))$. Puisque la recherche du plus court chemin dans un réseau acyclique prend un temps proportionnel à la cardinalité du nombre d'arcs du réseau, la complexité totale de l'algorithme 4.1 est au plus égale à $O(K\ s^4 + ((n_1\ n_2)^2\ (n_1 + n_2)^2)$. ■

Concernant les problèmes de flow shop, une troisième borne inférieure, notée LB3 et basée uniquement sur les données des problèmes à résoudre, peut être calculée a priori. Cette borne inférieure qui n'évolue pas durant la construction de l'arborescence est définie de la manière suivante.

La charge des machines, égale à la somme des durées opératoires des *n* jobs, est connue à l'avance et K périodes d'indisponibilité se produisent sur chacune de ces machines. Nous rappelons que les périodes d'indisponibilité sont placées de sorte qu'il existe des opérations avant la première période et après la $K^{ème}$. Par ailleurs, il est possible d'évaluer la date de début R_j (au plus tôt) de l'exécution des jobs sur chaque machine M_j, soit :

$$R_j = \min_{1 \le i \le n_{ij}} \{ \sum_{k=1}^{j-1} P_{ik} \}$$

Connaissant la date R_j et la charge $Ch(M_j)$ de la machine M_j, nous relaxons le problème en supposant que les opérations à réaliser sont sécables, c'est-à-dire qu'une opération peut être interrompue par une tâche de maintenance, puis reprise sans pénalité dès que la machine est à nouveau disponible.

En relaxant également les dates de disponibilité des opérations, nous pouvons effectuer une répartition de la charge $Ch(M_j)$ sur la machine M_j entre les périodes d'indisponibilité, comme l'illustre la figure 4.7 suivante.

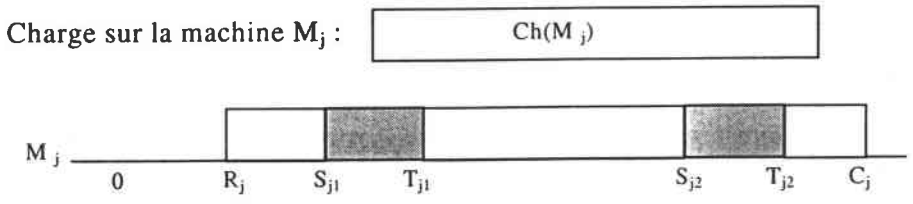


Figure 4.7 : Répartition de la charge

Nous obtenons ainsi une date de fin d'exécution au plus tôt des opérations de la machine M_j, notée C^j, et C^j est une borne inférieure de la date de fin d'exécution des opérations dans le cas strictement non préemptif. De manière plus formelle :

$$C^j = Ch(M_j) + R_j + \sum_{k=1}^{K} (T_{jk} - S_{jk})$$

Reste ensuite à évaluer une borne inférieure de la date de fin d'exécution des travaux. Il suffit pour cela d'ajouter à C^j la quantité Q_j telle que :

$$Q_j = \min_{1 \le i \le n_{ij}} \{ \sum_{k=j+1}^{m} P_{ik} \}$$

Finalement, la borne LB3 est donnée par le maximum sur toutes les machines de ces dates de fin de travaux, soit :

$$LB3 = \max_{1 \le j \le m_{ij}} (C^j + Q_j)$$

Dans le cas du job shop, les quantités t_j et Q_j ne peuvent être obtenues directement, les gammes opératoires étant différentes de l'ordre (M_1, M_2, M_m). Aussi, ne garderons-nous que les quantités C^j, calculées à partir de l'instant zéro, dans la formule de calcul de LB3, ce qui revient à ajouter à la charge de chaque machine, les durées des périodes d'indisponibilité.

2.5 Algorithme de séparation et évaluation

Les différentes procédures intervenant dans la procédure par séparation et évaluation ayant été définies aux sous-sections précédentes, nous pouvons maintenant présenter l'algorithme général qui permet de résoudre le problème d'ordonnancement J, $N_{Cwin} \parallel C_{max}$:

Algorithme 4.2 : PSE

Données : N jobs et des contraintes de disponibilité des machines

Résultat : Solution optimale S* et son makespan C_{max}*

Etape 1 : *Initialisation*

- Déterminer la borne supérieure BS et calculer LB3
- C_{max}* = BS

Etape 2 : *Sélections Immédiates*

2.1. Pour chaque opération O_{ij} calculer r_{ij} et q_{ij}

2.2. Appliquer la sélection immédiate basée sur les r_{ij} et q_{ij}

2.3. Appliquer la sélection immédiate relative à la maintenance

2.4. Répéter 2.1 et 2.2 jusqu'à ce que l'on ne puisse plus fixer de disjonctions

Etape 3 : *Evaluation*

3.1. Calculer la borne supérieure BS1, en tenant compte des arcs orientés.

Si BS1 < BS **Alors** BS = BS1

3.2. Calculer les bornes inférieures LB1 et LB2

3.3. **Si** max (LB1, LB2, LB3) ≥ BS **Alors** Arrêter l'exploration de la branche

Etape 4 : *Séparation*

4.1. Sélectionner le nœud à séparer

Si tous les nœuds sont des feuilles **Alors** Arrêter

4.2. Sélectionner la machine M_j sur laquelle fixer de nouvelles disjonctions

Si M_j n'existe pas **Alors**

- Déterminer le makespan C_{max} associé à la solution trouvée S
- **Si** $C_{max} < C_{max}$* **Alors** C_{max}* = C_{max} et S* = S

Fin Si

4.3. Effectuer la séparation et Aller à l'étape 2

---

3. Résultats numériques

Nous présentons dans cette section les résultats d'expériences numériques, obtenus par application de la procédure par séparation et évaluation décrite précédemment aux instances RA1 à RA15. En premier lieu sont donnés les résultats pour les problèmes de type flow shop, puis nous présentons et commentons, en second lieu, les expériences menées pour les problèmes de job shop.

3.1 Flow shop

Le tableau 4.1 présente (dans la colonne C_{max}*) les valeurs optimales pour les instances RA1 à RA15, dans le cas du flow shop. Les temps d'exécution, ainsi que le nombre de nœuds générés y sont également donnés. La colonne BS-init rappelle les meilleurs makespan obtenus au chapitre précédent et qui constituent les bornes supérieures de départ. Les valeurs figurant dans la colonne BI-init correspondent à la borne inférieure de départ LB3, pour chaque instance.

Tableau 4.1 : Valeurs optimales pour le flow shop

Problème	n	m	PSE				
			BS-init	BI-init	C_{max}*	CPUsecs	Nœuds
RA1	10	5	**1712**	1448	1712	95.42	893
RA2	10	5	1816	1616	1738	228.81	1758
RA3	10	5	**1784**	1702	1784	39.32	341
RA4	10	5	1909	1632	1795	155.46	1427
RA5	10	5	1767	1552	1643	219.37	1536
RA6	10	10	2567	2236	2324	10384.20	205088
RA7	10	10	2671	2162	2399	6854.24	126591
RA8	10	10	2601	2018	2376	8586.43	150673
RA9	10	10	2413	1972	2188	12313.86	212410
RA10	10	10	2456	2054	2314	996.84	16925
RA11	20	10	3344	2986	3179	151893.75	1070851
RA12	20	10	3586	3192	3342	283151.7	2163279
RA13	20	10	3625	3062	3468	583903.59	4385116
RA14	20	10	3493	3188	3313	273746.12	2083208
RA15	20	10	3629	3228	3537	12154.43	954116

Nous pouvons déduire du tableau 4.1 que la procédure par séparation et évaluation proposée est efficace en ce sens que toutes les problèmes de type flow shop avec indisponibilités des machines ont pu être résolus.

Pour les instances à 5 machines et 10 jobs, les solutions optimales ont été trouvées en des temps d'exécution et des nombres de nœuds générés relativement faibles. En revanche, les performances de l'algorithme sont moindres pour résoudre les instances de taille 10×10 et les temps d'exécution deviennent prohibitifs sur les problèmes de plus grande taille.

Les résultats figurant dans le tableau 4.1 permettent également de montrer l'efficacité des heuristiques développées au chapitre précédent. En effet, pour deux des instances testées, l'heuristique basée sur la résolution de sous-problèmes à deux jobs a atteint la solution optimale. Par ailleurs, les écarts relatifs entre les meilleures solutions heuristiques et les solutions exactes n'excèdent pas 7.1 % pour les instances de taille 5×10, 10.2 % pour les instances 10×10, et 11% pour les instances 10×20.

Nous pouvons donc déduire de ce qui précède que nos méthodes heuristiques restent un moyen très efficace d'aborder les problèmes de type flow shop étudiés dans cette thèse, dans la mesure où elles ont permis d'obtenir de bonnes solutions et en des temps d'exécution très faibles.

3.2 Job shop

Le tableau 4.2 présente les valeurs optimales pour le job shop des instances RA1 à RA15, ainsi que les temps d'exécution et le nombre de nœuds générés pour atteindre ces valeurs.

Tableau 4.2 : Valeurs optimales pour le job shop

Problème	n	m	PSE				
			BS-init	BI-init	C_{max}*	CPUsecs	Nœuds
RA1	10	5	1502	1308	1417	25.43	189
RA2	10	5	1529	1308	1468	198.30	768
RA3	10	5	1784	1452	1713	317.37	1231
RA4	10	5	1526	1332	1481	139.92	644
RA5	10	5	1544	1332	1435	83.91	503
RA6	10	10	2120	1476	1843	2479.14	27354
RA7	10	10	2311	1452	2094	4704.38	52178
RA8	10	10	2088	1368	1801	3150.52	40825
RA9	10	10	2046	1332	1772	5585.28	65690
RA10	10	10	2268	1344	1948	4139.91	48317
RA11	20	10	2999	2376	2613	116808.17	831673
RA12	20	10	3257	2442	2991	145330.49	1024580
RA13	20	10	3098	2452	2775	416971.06	2910458
RA14	20	10	3292	2508	2910	328961.84	2276416
RA15	20	10	3155	2518	2836	194687.20	1352492

Les conclusions qui se dégagent du tableau 4.2 sont analogues à celles mises en évidence pour les problèmes de type flow shop.

En premier lieu, toutes les instances testées ont été résolues par la procédure par séparation et évaluation, ce qui confirme l'efficacité de la méthode et notamment celle des bornes utilisées.

Les temps d'exécution ainsi que le nombre de nœuds générés pour atteindre les solutions optimales des problèmes de plus petite taille sont faibles. Pour les instances de plus grande taille, ces valeurs sont, comme dans le cas du flow shop, beaucoup plus importantes.

Les écarts relatifs entre solutions heuristiques et solutions optimales sont majorés par 7 % pour les problèmes de taille 5×10, par 14.1 % pour les problèmes à 10 machines et 10 jobs et par 12.9 % pour les problèmes de taille 10×20, ce qui confirme l'efficacité de l'heuristique basée sur la résolution de sous-problèmes à deux jobs.

En conclusion, les faibles temps de calculs mis par les méthodes approchées du chapitre précédent pour atteindre des valeurs relativement proches des optima indiquent que les heuristiques proposées permettent de traiter efficacement les problèmes de type job shop avec contraintes de disponibilité étudiés dans cette thèse.

Remarque 4.3 : Les performances de la procédure par séparation et évaluation proposée sont dues non seulement aux bornes utilisées dans l'évaluation des nœuds, mais aussi aux conditions de sélections immédiates impliquant les tâches de maintenance. En effet, en l'absence de périodes d'indisponibilité des machines, la PSE n'avait résolu aucune des instances de grande taille au bout d'une semaine de calculs.

Cette dernière remarque indique que les algorithmes développés dans ce chapitre sont dédiés aux problèmes d'ordonnancement sous contraintes de disponibilité des machines. Plus le nombre de périodes d'indisponibilité est grand, plus la PSE élimine de branches dans l'arborescence. Les instances à 10 machines et 20 jobs correspondent donc aux problèmes de taille limite que nous pouvons espérer résoudre de manière exacte, en un temps acceptable, à moins de considérer plus de deux périodes d'indisponibilité par machines.

De manière évidente, ces conclusions ne s'appliquent aux heuristiques développées au chapitre précédent qui sont d'autant plus performantes que les problèmes à résoudre sont moins contraints.

4. Conclusion

Nous avons proposé dans ce chapitre des procédures par séparation et évaluation permettant de résoudre de manière exacte les problèmes d'ordonnancement du flow shop et du job shop, en présence de périodes d'indisponibilités des machines.

Le modèle de graphe disjonctif a été utilisé pour représenter les différents nœuds des arborescences. En particulier, nous avons proposé une manière originale de prendre en compte les périodes d'indisponibilité des machines, en introduisant des travaux fictifs composés des tâches de maintenance à réaliser. Cette modélisation permet en outre d'introduire de la flexibilité sur les périodes d'indisponibilité.

Des extensions de l'algorithme polynomial, développé au second chapitre de la thèse, ont été utilisées pour le calcul des bornes inférieures, tandis qu'une heuristique d'insertion permettait de mettre à jour rapidement les bornes supérieures.

Les expérimentations ont été menées sur les instances générées au troisième chapitre. Les résultats obtenus attestent d'une part de l'efficacité des algorithmes proposés dans ce chapitre, et d'autre part, indiquent que les méthodes de résolution approchées du chapitre précédent sont performantes.

Conclusion Générale et Perspectives

1.Conclusions

Nous avons étudié dans cette thèse les problèmes d'ordonnancement dans les ateliers de production avec la prise en compte de contraintes de disponibilité des machines. En particulier, nous nous sommes intéressés aux modèles du flow shop et du job shop, dans le contexte déterministe et statique d'indisponibilités dues à des activités de maintenance sur les machines. Nous avons supposé tout au long de la thèse que les opérations à réaliser étaient strictement non-préemptives, ce qui signifie qu'une fois commencée, l'exécution d'une opération ne pouvait être interrompue ni par la réalisation d'une tâche de maintenance, ni par celle d'une autre opération. Des méthodes de résolution approchées et exactes ont été développées pour la résolution des problèmes d'ordonnancement de type flow shop et job shop, en présence de périodes d'indisponibilité des machines. En outre, nous avons établi des résultats de complexité pour les problèmes de type job shop à deux travaux, avec un nombre quelconque de machines.

Dans le premier chapitre, nous avons mené une étude bibliographique portant sur l'ordonnancement sous contraintes de disponibilité, les notions de base, les modèles classiques d'ateliers ainsi que les principales méthodes de résolution, ayant été décrits au préalable. Cette étude a montré d'une part que, comparé à la littérature dédiée aux problèmes classiques d'ordonnancement, les ouvrages traitant de contraintes de disponibilité des machines ne sont pas très nombreux. D'autre part, nous avons mis en évidence qu'il n'existait pas, du moins à notre connaissance, d'articles consacrés au modèle du flow shop à plus de deux machines, ainsi qu'à celui du job shop.

Le second chapitre est consacré au problème d'ordonnancement de type job shop avec uniquement deux travaux à ordonnancer, ainsi que des contraintes de disponibilité des machines. Après y avoir présenté l'approche géométrique qui permet de résoudre de manière exacte le problème classique du job shop à deux jobs, nous avons proposé une extension de cette approche à notre problème. Cette extension, appelée approche géométrique temporisée, est un algorithme polynomial original permettant la prise en compte de contraintes de disponibilité des machines. Elle est basée sur l'introduction et la définition de nouveaux sommets, ainsi que sur une méthode de progression dynamique, tenant compte du temps, d'un sommet à ses successeurs. Par ailleurs, il a été démontré que l'approche géométrique temporisée permettait la résolution du problème d'ordonnancement à deux jobs, pour tout critère régulier et en présence de contraintes additionnelles.

Dans le troisième chapitre, nous avons élaboré des méthodes de résolution approchées pour les problèmes d'ordonnancement généraux, c'est-à-dire à plus de deux jobs, du flow shop et du job shop. Tous les algorithmes y figurant ont été développés pour la minimisation du makespan. Cependant, les approches proposées sont applicables à la considération de tout critère régulier.

Dans la première partie du chapitre, consacrée au flow shop, deux hypothèses ont été adoptées pour la prise en compte des périodes d'indisponibilité des machines.

Dans un premier temps, nous avons supposé que les tâches de maintenance étaient fixes, et deux méthodes de résolution ont été proposées pour résoudre de manière approchée le problème d'ordonnancement. La première de ces méthodes est une heuristique gloutonne qui ordonnance les jobs machine par machine et l'un après l'autre selon une séquence de traitement. Afin d'optimiser cette séquence, cette heuristique a été couplée à un algorithme génétique, puis à une recherche tabou. La seconde méthode de résolution proposée est basée sur la résolution exacte de sous-problèmes à deux jobs, au moyen de l'algorithme polynomial développé au second chapitre.

Dans un second temps, nous avons supposé que les positions de certaines tâches de maintenance pouvaient être optimisées durant la procédure d'ordonnancement. Des fenêtres temporelles indiquant les limites des dates d'exécution des différentes tâches de maintenance ont ainsi été introduites et les deux méthodes de résolution développées pour le cas d'indisponibilité fixes ont été modifiées de sorte à tenir compte de ces fenêtres temporelles.

Dans la deuxième partie du chapitre, dédiée au job shop, nous avons également développé et comparé deux méthodes de résolution, analogues à celles du flow shop : d'une part un algorithme de liste ordonnançant les jobs un par un selon un ordre de traitement lui-même optimisé par une recherche tabou, et d'autre part une heuristique ordonnançant les jobs deux par deux, au moyen de l'algorithme polynomial du deuxième chapitre.

En l'absence de benchmarks dédiés aux problèmes étudiés, dans la littérature, nous avons choisi de tester toutes les méthodes de résolution développées sur des instances générées aléatoirement. Deux conclusions sont nettement mises en évidence par les résultats de ces expériences. La première conclusion est que l'introduction de flexibilité dans le processus de maintenance, au moyen de fenêtres temporelles, permet d'améliorer sensiblement les résultats obtenus en considérant des tâches de maintenance fixes. La seconde est que les heuristiques basées sur la résolution exacte de sous-problèmes à deux jobs dominent de manière significative les approches gloutonnes qui consistent à ordonnancer les jobs l'un après l'autre.

Notons que si l'introduction de tâches de maintenance flexibles n'a pas été développée pour le job shop, elle est pourtant directement applicable à ce modèle et conduit évidemment à des améliorations significatives des résultats. Par ailleurs, en dépit de l'absence de benchmarks, nous aurions pu évaluer les algorithmes proposés en déterminant des bornes inférieures des instances testées. Cependant, les bornes classiques basées notamment sur la charge des machines, sont très éloignées des résultats que nous avons pu obtenir et ne permettent donc pas une bonne évaluation des méthodes de résolution proposées. Aussi, avons-nous choisi d'une part de ne pas considérer ces bornes, et d'autre part d'essayer de déterminer les solutions optimales des problèmes traités.

Dans le quatrième et dernier chapitre de cette thèse, nous avons développé des procédures par séparation et évaluation pour les problèmes d'ordonnancement du flow shop et du job shop, en présence de périodes d'indisponibilité des machines. Le fait que les ordonnancements de permutation ne soient pas dominants pour le flow shop, nous a incité à appliquer à ce modèle la procédure par séparation et évaluation proposée pour le job shop.

Dans cette dernière, nous avons choisi de modéliser les différents nœuds des arborescences au moyen du graphe disjonctif. Afin d'intégrer les contraintes de disponibilité des machines, des travaux fictifs composés des tâches de maintenance ont été introduits. Durant la construction des arborescences, les bornes supérieures ont été mises à jour, grâce à une heuristique d'insertion analogue à celles développées au troisième chapitre de la thèse. Pour le calcul des bornes inférieures, l'algorithme polynomial du second chapitre a été modifié de sorte à prendre en compte des dates de disponibilité et des durées de latence sur les opérations.

Les méthodes de résolution proposées ont permis d'atteindre les solutions optimales pour toutes les instances proposées, la taille limite testée étant pour des instances à dix machines, vingt jobs et deux périodes d'indisponibilité par machine. Par ailleurs, il ressort des ces dernières expérimentations que les méthodes de résolution approchées proposées au chapitre précédent sont relativement performantes.

2. Perspectives

Nous envisageons par la suite de poursuivre l'étude des problèmes d'ordonnancement d'ateliers sous contraintes de disponibilité des machines, et ce, dans plusieurs directions de recherche définies notamment par les différents chapitres que compte cette thèse.

Concernant l'étude bibliographique, nous étudierons tout nouvel ouvrage concernant l'ordonnancement sous contraintes de disponibilité des machines, d'abord afin de tenir à jour l'état de l'art développé dans cette thèse, mais aussi pour confronter nos méthodes de résolution à de nouvelles approches qui traiteraient des mêmes problèmes.

Pour ce qui est de la complexité de problèmes d'ordonnancement, nous essaierons à court terme d'étendre l'approche géométrique temporisée à d'autres problèmes à deux travaux. En particulier, nous sommes en train de travailler sur les modifications à apporter à l'approche pour tenir compte d'opérations sécables, puis semi-sécables.

Les méthodes de résolution approchées proposées dans le troisième chapitre seront améliorées. En effet, il est clair que les heuristiques basées sur la résolution de sous-problèmes à deux jobs fournissent les meilleurs résultats. Pour ces heuristiques, l'algorithme polynomial est appliqué séquentiellement à des paires de jobs. Cependant, une application plus judicieuse de cet algorithme conduirait probablement à de meilleures performances. Des efforts seront également dirigés vers l'utilisation des méta-heuristiques et l'amélioration de leurs paramètres. Enfin, nous consacrerons des recherches à l'introduction de périodes d'indisponibilité flexibles. En particulier, nous envisageons de considérer des schémas particuliers de maintenance, notamment où l'activité de maintenance serait régie par des lois de probabilités, exprimant par exemple l'espérance de vie d'outils qu'il conviendra de remplacer, etc. Ainsi, nos études pourront être plus proches de situations réelles d'entreprise.

Pour ce qui est de l'élaboration de méthodes exactes de résolution, nous essayerons d'améliorer les procédures par séparation et évaluation proposées. En particulier, nous chercherons à établir de nouvelles conditions de sélections immédiates qui permettront d'accélérer les recherches en réduisant le nombre de nœuds à explorer.

Enfin, nous envisageons d'étudier la prise en compte de contraintes de disponibilité des machines dans des modèles d'ateliers plus généraux que ceux étudiés dans cette thèse, notamment le flow shop hybride et le job shop flexible. Il s'agira alors de développer des méthodes de résolution spécifiques, se servant éventuellement des approches proposées jusqu'à présent. De manière simultanée, l'intégration de nouvelles contraintes aux problèmes étudiés dans cette thèse sera un axe de recherche privilégié, dans la mesure où l'objectif sous-jacent de l'ensemble de nos travaux reste l'élaboration de méthodes de résolution performantes pour des problèmes concrets rencontrés dans les entreprises.

Références Bibliographiques

[1] Aarts, E.H.L., Van Laarhoven, P.J.M. Lenstra, J.M., and Ulder, N.L.J, (1994). A Computational Study of Local Search Algorithms for Job-shop Scheduling. *ORSA Journal of Computing*, 6, 2, 118-125.

[2] Adams, J. Balas, E., and Zawack, D., (1988). The Shifting Bottleneck Procedure for the Job-shop Scheduling. *Management Science*, 34, 3, 391-401.

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

[4] Aggoune, R., Mahdi, A. H., and Portmann, M. C., (2001). Genetic algorithms for the flow shop scheduling problem with availability constraints. *IEEE International Conference on Systems, Man and Cybernetics*, Tucson (AZ).

[5] Aggoune, R., (2001). Minimizing the Makespan for the Flow Shop Scheduling Problem with Availability Constraints. *Operational Research Peripatetic Post-Graduate Programme*, ORP3, Paris.

[6] Aggoune, R., and Portmann, M. C., (2002). A New Heuristic for the Flow Shop Scheduling Problem with Availability Constraints. *International Symposium on Combinatorial Optimization*, CO'02, Paris.

[7] Aggoune, R., and Portmann, M. C., (2002). A Heuristic Approach for the Job Shop Scheduling Problem with Availability Constraints. *XV Conference of the European Chapter on Combinatorial Optimization*, ECCO XV, Lugano (Switzerland).

[8] Aggoune, R., (2002). Minimizing the Makespan for the Flow Shop Scheduling Problem with Availability Constraints. *European Journal of Operational Research*, special issue devoted to ORP3, accepté.

[9] Agnetis, A., Mirchandani, P. B., Pacciarelli, D., Pacifici, A., (2001). Job Shop Scheduling with Two Jobs and Nonregular Objective Functions, *INFOR*, 39, 3, 227-244.

[10] Akers, S. B., Friedman, J., (1955). A Non-numerical Approach to Production Scheduling Problems. *Operations Research*, 3, 429-442.

[11] Applegate, D., Cook, W. (1991). A Computational Study of the Job-shop Scheduling Problem. *ORSA Journal of Computing*, 3, 149-156.

[12] Baker, K.R. (1974). *Introduction to Sequencing and Scheduling*. Wiley, New York.

[13] Balas, E., Lenstra, J.K., and Vazacopoulos, A., (1995). The One-machine Problem with Delayed Precedence Constraints and its use in Job Shop Scheduling. *Management Science*, 41, 1, 94-109.

[14] Balas, E., and Vazacopoulos, A., (1998). Guided Local Search with Shifting Bottleneck for Job-shop Scheduling. *Management Science*, 44, 2, 262-275.

[15] Bellman, R., (1957). *Dynamic Programming*. Princeton University Press.

[16] Birge, J., and Glazebrook, K. D., (1988). Assessing the effects of machine breakdowns in stochastic scheduling. *Operations Research Letters*, 7, 6, 267-271.

[17] Birge, J., Frenk, J. B. G., Mittenthal, J., and Rinnooy Kan, A. H. G., (1990). Single machine scheduling subject to stochastic breakdowns. *Naval Research Logistics*, 37, 661-677.

[18] Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., and Weglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer, Berlin.

[19] Blazewicz, J., W. Domschke, and E. Pesch (1996). The Job Shop Scheduling Problem: Concentional and New Solution Techniques. *European Journal of Operational Research*, 93, 1-33.

[20] Blazewicz, J., Formanowicz, P., Kubiak, W., Przysucha, M., and Schmidt, G., (2000). Parallel Branch and Bound Algorithms for the Two-machine Flow Shop Problem with Limited Machine Availability. *Bulletin of the Polish Academy of Sciences*, Technical Sciences, 48, 1, 105-115.

[21] Blazewicz, J., Pesch, E., and Sterna, M., (2000). The disjunctive graph machine representation of the job shop scheduling problem, *European Journal of Operational Research*. 127, 2, 1, 317-331.

[22] Blazewicz, J., Breit. J., Formanowicz, P., Kubiak, W., and Schmidt, G., (2001). Heuristic algorithms for the two-machine flowshop Problem with limited machine availability. *Omega Journal*, 29, 599-608.

[23] Braun, O., Schmidt, G., et Sotskov, Y., (2001). Stability of Jonson's Schedule with limited Machine availability. *Actes de la 3e conférence francophone de MOdélisation et de SIMulation (MOSIM'01)*, Troyes, Conception, analyse et gestion des systèmes industriels, SCS European Publishing House, 2, 683-687.

[24] Breit, J., (2000). *Heuristische Ablaufplanungsverfahren für Flowshops und Openshops mit beschränkt verfügbaren Prozessoren*, Ph.D. Thesis, University of Saarland, Saarbrücken.

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

[26] Breit, J., Schmidt, G., and Strusevich, V. A., (2002). Non-Preemptive Two-Machine Open Shop Scheduling with Non-availability Constraints. *Project Management and Scheduling*, Valence, Spain.

[27] Brinkkötter, W., and Brucker, P., (2001). Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments. *Journal of Scheduling*, 4, 53-64.

[28] Brucker, P., (1988). An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40, 353-359.

[29] Brucker, P., (1998). *Scheduling Algorithms*. Springer-Verlag, Berlin Heidelberg.

[30] Brucker, P., Schlie, R., (1990). Job-shop scheduling with multi-purpose machines, *Computing*, 45, 369-375.

[31] Brucker, P., Jurisch, B., (1993). A new lower bound for job-shop scheduling problem, *European Journal of Operational Research*, 64, 156-167.

[32] Brucker, P., Jurisch, B., and Sievers, B., (1994). A Branch and Bound Algorithm for the Job-Shop Scheduling Problem. *Discrete Applied Mathematics*, 49, 109-127.

[33] Brucker, P., Jurisch, B., and Krämer, A., (1994). The Job-shop Problem and Immediate Selection. *Annals of Operations Research*, 50, 73-114.

[34] Brucker, P., and J. Neyer, (1998). Tabu search for the multi-mode job-shop problem, *Operations Research Spektrum*, 20, 21-28.

[35] Campbell, H.G., Dudek, R.A., Smith, M.L., (1970). A heuristic algorithm for the n-job, m-machine sequencing problem. *Management science*, 16, 10, 630-637.

[36] Carlier, J., (1982). The one machine sequencing problem. *European Journal of Operational Research*, 11, 42-47.

[37] Carlier, J., and Chretienne, P. (1988). *Les Problèmes d'ordonnancement*. Masson, Paris, France.

[38] Carlier, J., and Pinson, E. (1989). An Algorithm for Solving the Job Shop Problem. *Management Science*, 35, 164-176.

[39] Carlier, J., and Pinson, E. (1990). A Practical use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem. *Annals of Operations Research*, 78, 2, 146-161.

[40] Carlier, J., and Rebaï, I., (1996). Two Branch and Bound Algorithms for the Permutation Flow Shop Problem. *European Journal of Operational Research*, 90, 2, 238-251.

[41] Cheng, J., Kise. H., and Matsumoto. H., (1997). A branch-and-bound Algorithm with Fuzzy Inference for a Permutation Flowshop Scheduling Problem. *European Journal of Operational Research*, 96, 3, 578-590.

[42] Cheng, T. C. E., and Wang, G., (1999). Two-machine flowshop scheduling with consecutive availability constraints. *Information Processing Letters*, 71, 2, 49-54.

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

[44] Cheng, J., Steiner, G., and Stephenson, P., (2001). A computational study with a new algorithm for the three-machine permutation flow-shop problem with release times, *European Journal of Operational Research*, 130, 3, 1, 559-575.

[45] Colorni, A., Dorigo, M., and Maniezzo, V., (1991). An Investigation of Some Properties of an Ant Algorithm. In *Parallel Problem Solving from Nature* 2, R. Männer and B. Manderick eds, North-Holland: Amsterdam, 509-520.

[46] Conway, R.W., Maxwell, W.L., Miller, L.W. (1967). *Theory of Scheduling*. Addison Wesley, Reading, Mass., USA.

[47] Cook, S. A. (1971) The Complexity of Theorem Proving Procedures, *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, Association of Computing Machinery, New York, 151-158.

[48] Dantzig, G. B., (1951). Application of the simplex method to a transportation problem, in *Activity analysis of production and allocation*, T. C. Koopmans eds., John Wiley & Sons Inc., 359-373.

[49] Dantzig, G. B., Fulkerson, R., Johnson, S., (1954). Solution of a large-scale traveling-salesman problem. *Operational Research* 2, 393-410.

[50] Dauzère-Pérès, S., and Lasserre, J.B., (1993). A Modified Shifting Bottleneck Procedure for Job-shop Scheduling. *International Journal of Production Research*, 31, 4, 923-932.

[51] Davis, L., (1991). *Handbook of Gentic Algorithms*, Van Nostrand Reinhold, New York.

[52] Della Croce, F., Ghirardi, M., and Tadei, R., (2002). An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. *European Journal of Operational Research*. 139, 2, 1, 293-301.

[53] Della Croce, F., Gupta, J. N. D., and Tadei, R., (2002). Minimizing tardy jobs in a flowshop with common due date. *European Journal of Operational Research*, 120, 2, 375-381.

[54] Demeulemeester E. L. and Herroelen, W. L., (2001). *Project Scheduling: A Research Handbook*. International Series in Operations Research and Management Science, Volume 49. Kluwer.

[55] Dorndorf, U., Pesch, E., Phan-Huy, T., (2002). Constraint Propagation and Problem Decomposition: A Preprocessing Procedure for the Job Shop Problem. *XV Conference of the European Chapter on Combinatorial Optimization*, ECCO XV, Lugano, Switzerland.

[56] Drozdowski, M, (1996). Scheduling Multiprocessor Tasks – An Overview. *European Journal of Operational Research*, 94, 215-230.

[57] Epek, O., Okada, M., and Vlach, M., (2002). Nonpreemptive flowshop scheduling with machine dominance. *European Journal of Operational Research*, 139, 2, 245-261.

[58] Espinouse, M. L., Formanowicz, P., and Penz, B., (1999). Minimizing the makespan in the two-machine no-wait flow-shop. *Computers & Industrial Engineering*, 37, 497-500.

[59] Esquirol, P., and Lopez, P. (1999). *L'ordonnancement*. Economica, Paris, France.

[60] Falkenauer E. and S. Bouffouix, (1991). A genetic aigorithm for job shop. *Proceedings of the IEEE Int. Conference on robotics and Automation,* 824-829, Sacramento, CA, April.

[61] Fisher, H., and G.L. Thompson (1963). Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In Muth, J.F. and G.L. Thompson. (eds) *Industrial Scheduling*. Prentice Hall, Englewood Cliffs, New Jersey, Ch 15, 225-251.

[62] Framinan, J. M., Leisten, R., and Ruiz-Usano, R., (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimization, *European Journal of Operational Research*, 141, 3, 559-569.

[63] French, S., (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Horwood, Chichester.

[64] Garey, M. R., Johnson, D. S., and Sethi, R., (1976). The Complexity of Flow Shop and Job-Shop Scheduling, *Mathematics of Operations Researc*h, May, 1, 2, 117-129.

[65] Garey, M.R., Johnson, M.R., (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman.

[66] Garey, M.R., Tarjan, R.E., Wilfong, G.T., (1988). One-processor scheduling with earliness and tardiness penalties, *Mathematics of Operations Research*, 13, 330-348.

[67] Gilmore, P.C., and Gomory, R.E. (1964). Sequencing a one-state Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 12, 655-679.

[68] Glass, C.A., and Potts, C.N. (1996). A Comparaison of Local Search Methods for the Flow Shop Scheduling. *Annals of Operations Research*, 63, 489-509.

[69] Glazebrook, K. D., (1987). Evaluating the effects of machine breakdowns in stochastic scheduling problems. *Naval Research Logistics*, 34, 319-335.

[70] Glover, F., (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8, 1, 156-166.

[71] Glover, F., (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13, 533-549.

[72] Glover, F., and Greenberg, H.J., (1989). New Approach for Heuristic Search : A Bilateral Linkage with Artificial Intelligence. *European Journal of Operational Research*, 39, 119-130.

[73] Glover, F., (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1, 3, 190-206.

[74] Glover, F., (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2, 1, 4-32.

[75] Glover, F., E. Taillard and D. de Werra. (1993). A User's Guide to Tabu Search. *Annals of Operations Research*, 41, 3-28.

[76] Glover, F. (1995). Tabu Search Fundamentals and Uses. *Revised and Expanded, Technical Report*, Graduate School of Business, University of Colorado, Bolder, CO.

[77] Glover, F., and Laguna, M., (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

[78] Goldberg, D. E., Lingle, R., (1985). Alleles, Locl, and the travelling salesman problem. *Proceedings of International Conference on Genetic Aigorithms and their Appiication*, 154-159.

[79] Goldberg, D.E., (1989). *Genetic Algorithms in Search : Optimisation and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

[80] Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs, *Bulletin of the American Mathematical Society*, 64, 275-278.

[81] Gondran, M., and Minoux, M., (1984), *Graphs and Algorithms*, John Wiley and Sons, New York. 21.

[82] Grabowski, J., Nowicki, E., and Zdrzalka, S., (1986). A Block Approach for Single Machine Scheduling with Release Dates and Due Dates. *European Journal of Operational Research*, 26, 2, 278-285.

[83] Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G, (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals Discrete Math.*, 5, 287 - 326.

[84] Graves, G. H., and Lee, C. Y., (1999). Scheduling maintenance and semiresumable jobs on a single machine, *Naval Research Logistic*, 46, 845-863.

[85] Guinet, A., (2000). Efficiency of reductions of job-shop to flow-shop problems. *European Journal of Operational Research*, 125, 3, 469-485.

[86] Hansen, P. (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Presented at the *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

[87] Hardgrave, W.H., Nemhauser, G.L., (1963). A geometric model and a graphical algorithm for a sequencing problem. *Operations Research*, 11 889-900.

[88] Holland, J., (1975). *Adaptive in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[89] Ignall, E., and Schrage, L., (1965). Application of the Branch and Bound Technique to some Flow-Shop Scheduling Problems, *Operations Research*, 13, 400-412.

[90] Jackson, J. R., (1955). Scheduling a Production Line to Minimise Maximum Tardiness. *Research Report 43, Management Science Research Projects*, University of California, Los Angeles, USA.

[91] Jackson, J.R., (1956). An Extension of Johnson's Results on Job Lot Scheduling. *Naval Research Logistic Quarterly*, 1, 61-68.

[92] Jain, A. S, (1998). A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem. *P.h.D Thesis*, University of Dundee, Scotland.

[93] Jain, A. S., and S. Meeran, (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113, 2, 390-434.

[94] Jonhson, S.M. (1954). Optimal two- and three-stage Production Scheduling with Setup Times Included. *Naval Research Logistic Quarterly*, 1, 61-68.

[95] Jurisch, B., (1995). Lower bounds for the job-shop scheduling problem on multi-purpose machines. *Discrete Applied Mathematics*, 58, 145-156.

[96] Khachian, L. G., (1979). A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20, 191-194.

[97] Karmarkar, N., (1984). A new polynomial time algorithm for linear programming. *Combinatorica*, 4, 373-395.

[98] Karp, R. M. (1972) Reducibility Among Combinatorial Problems, in Miller, R. E. and Thatcher, J. W. (eds).*Complexity of Computer Computations*, Plenum Press, New York, 85-104.

[99] Kashyrskikh, K. N., Potts, C. N., and Sevastianov, S. V., (2001). A 3/2-approximation algorithm for two-machine flow-shop sequencing subject to release dates. *Discrete Applied Mathematics*, 114, 1, 3, 255-271.

[100] Kaspi, M., and Montreuil, B., (1988). On the scheduling of identical parallel processes with arbitrary initial processor available time. *Research Report 88-12*, School of Industrial Engineering, Purdue University.

[101] Kirkpatrick, S., Gellatt, C.D., and Vecchi, M.P., (1983). Optimization by Simulated Annealing. *Science*, 220, 671-680.

[102] Kraemer, F., and Lee, C. Y., (1993). Common Due-Window Scheduling. *Production and Operations Management*, 2, 262-275.

[103] Kravchenko, S. A., (2000). Minimizing the number of late jobs for the two-machine unit-time job-shop scheduling problem. *Discrete Applied Mathematics*, 98, 3, 209-217.

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

[105] Kusiak, A., (1990). *Intelligent Manufacturing Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

[106] Kyparisis G. J., and Koulamas, C., (2000). Flow shop and open shop scheduling with a critical machine and two operations per job. *European Journal of Operational Research*, 127, 1, 120-125.

[107] Ladhari, T., Haouri, M., (2002). A branch-and-bound based local search algorithm for the flow shop problem. *XV Conference of the European Chapter on Combinatorial Optimization*, ECCO XV, Lugano, Switzerland.

[108] Lawler, E. L. (1973). Optimal Sequencing of a Single Machine Subject to Precedence Constraints, *Management Science*, 19, 544-546.

[109] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., (1989). Sequencing and scheduling: Algorithms and complexity, *Report 8945/A*, Econometric Institute, Erasmus University Rotterdam.

[110] Lee, C. Y., (1991). Parallel machine scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30, 53-61.

[111] Lee, C. Y., and Liman, S. D., (1992). Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29, 375-382.

[112] Lee C. Y., and Liman, S. D., (1993). Capacitated two-parallel machines scheduling to minimize sum of job completion times. *Discrete Applied Mathematics*, 41, 211-222.

[113] Lee, C. Y., (1996). Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9, 395-416.

[114] Lee, C. Y., (1997). Minimizing the makespan in two-machine flowshop scheduling problem with an availability constraint. *Operations Research Letters*, 20, 129-139.

[115] Lee, C. Y., and Leon, J., (1998). Machine scheduling with rate-modifying activity. *European Journal of Operational Research*, to appear.

[116] Lee, C. Y., (1999). Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114, 420-429, 1999.

[117] Lee, C. Y., and Chen, Z. L., (2000). Scheduling jobs and maintenance activities on parallel machines, *Naval Research Logistics*, 47, 145-165.

[118] Lei, L., and Wong, T. J., (1991). The Minimum Common-Cycle Algorithm for Cycling Scheduling of Two Material Handling Hoists with Time Window Constraints. *Management Science*, 37, 1629-1639.

[119] Li, W., and Cao, J., (1995). Stochastic scheduling on a single machine subject to multiple breakdowns according to different probabilities. *Operations Research Letters*, 18, 81-91.

[120] Liman, S., (1991). Scheduling with Capacities and Due-Dates. *Ph.D. Dissertation*, Industrial and Systems Engineering Department, University of Florida.

[121] Lin, G. H., Yao, E. Y., and He, Y., (1998). Parallel machine scheduling tomaximize the minimum load with nonsimultaneous machine available times. *Operations Research Letters*, 22, 75-81.

[122] Liu, Z., and Sanlaville, E., (1995). Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Applied Mathematics*, 58, 253-280.

[123] Liu, Z., and Sanlaville, E., (1997). Stochastic scheduling with variable profile and precedence constraints, *SIAM Journal on Computing*, 26, 173-187.

[124] Lopez, P., Roubellat, F., (2001). *Ordonnancement de la Production*. Hermes Science, France.

[125] Lorigeon, T., Billaut, J. C., and Bouquard, J. L., (2002), Availability Constraint for a Single Machine Problem with Heads and Tails. *Project Management and Scheduling*, Valence, Spain.

[126] Lu, L., and Posner, M. E., (1993). An NP-hard open shop scheduling problem with polynomial average time complexity, *Mathematics of Operations Research*, 18, 12-38.

[127] Mac Carthy, B., Liu, J., (1993). Adressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31, 59-79.

[128] Mac Maron, G.B., and Florian, M., (1975). On Scheduling with Ready Time and Due Dates to Minimize Maximum Lateness. *Operations Research*, 23, 3, 475-482.

[129] Mati, Y., (2001). On the Complexity of the two-job shop Problems with Multi-purpose Unrelated Machines. *Operational Research Peripatetic Post-Graduate Programmme, ORP3*, Paris, France.

[130] Mati, Y., Rezg, N., Xie, X.L., (2001). A Taboo Search Approach for Deadlock-free Scheduling of Automated Manufacturing Systems. *Journal of Intelligent Manufacturing*, special issue on Metaheuristics, 12, 5/6, 535-552.

[131] Mati, Y., Rezg, N., Xie, X.L., (2002). Greedy Heuristic and Genetic Algorithms for the Multi-resource Shop Scheduling with Resource Flexibility. *Project Management and Scheduling*, Valence, Spain.

[132] Mati, Y., (2002). Les Problèmes d'Ordonnancement dans les Systèmes de Production Automatisés : Modèles, Complexité et Approches de Résolution. *Ph.D. Thesis*, Universite de Metz, France.

[133] Moore, J. M., (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15, 102-109.

[134] Mosheiov, G., (1994). Minimizing the sum of job completion times on capacitated parallel machines. *Mathl. Comput. Modelling*, 20, 91-99.

[135] Mosheiov, G., (2002). Complexity analysis of job-shop scheduling with deteriorating jobs. *Discrete Applied Mathematics*, 117, 1, 3, 195-209.

[136] Nowicki, E., and Smutnicki, C., (1996). A Fast Tabu Search Algorithm for the Permutation Flow Shop Problem. *European Journal of Operational Research*, 91, 160-175.

[137] Nowicki, E., and Smutnicki, C., (1996). A Fast Taboo Search Algorithm for the Job-Shop Problem. *Management Science*, 42, 6, 797-813.

[138] Nowicki, E., (1999). The Permutation Flow Shop with Buffers: A Tabu Search Approach. *European Journal of Operational Research*, 116, 205-219.

[139] Oliver, I. M., D.J.Smith, and J.R.C. Holland, (1987). A study of permutation crossover operators on the travelling salesman problem, *Technical Report*, Texas Instruments Ltd., Dallas, TX.

[140] Osman, I. H. and G. Laporte (1996). Metaheuristics: A Bibliography. *Annals of Operations Research*, 63, 513-623.

[141] Panwalker, S.S., and Iskander, W., (1977). A Survey of Scheduling Rules. *Operations Research*, 25, 1, 45-61.

[142] Park, Y.B., Pegden, C.D., and Enscore, E.E., (1984). A Survey and Evaluation of Static Flowshop Scheduling Heuristics. *International Journal of Production Research*, 22, 127-141.

[143] Péridy, L. Pinson, E., and Rivreau, D., (1998). Enhanced Disjunctive Elimination Rules for the Flow-Shop and Permutation Flow-Shop Problems. *6th International Workshop on Project Management and Scheduling*, Istanbul, 277 – 280.

[144] Pezzella, F., and Merelli, E., (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120, 2, 297-310.

[145] Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, Etats-Unis.

[146] Portmann, M. C., and Vignier, A., (2001). Algorithmes génétiques et ordonnancement, *Ordonnancement de la Production*. Hermes Science, France.

[147] Potts, C. N., and Van Wassenhove L. N. (1982). A Decomposition Algorithm for the Single Machine Tardiness Problem. *Operations Research Letters*, 32, 177-181.

[148] Qi, T., Chen, T., and Tu, F., (1997). Scheduling with the maintenance on a single machine. *Working Paper*, Department of Computer and System Sciences, Nankai University. People's Republic of China.

[149] Reeves, C. R., (1995). A genetic algorithm for flowshop sequencing, *Computers and Research*, 22, 5-14.

[150] Rinnooy Kan, A.H.G., (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Haque.

[151] Roy, B., and Susmann, B., (1964). Les Problèmes d'ordonnancement avec Contraintes Disjonctives. *Note DS n° 9 bis*, SEMA, Montrouge.

[152] Sadfi, C., (2002). Problèmes d'ordonnancement avec minimisation des encours. *P.h.D. Thesis*, GILCO, INP Grenoble, France.

[153] Sanlaville, E., and Schmidt, G., (1998). Machine scheduling with availability constraints. *Acta Informatica*, 35, 795-811.

[154] Schmidt, G., (1984). Scheduling on semi-identical processors. *Z. Oper. Res.*, A28, 153-162.

[155] Schmidt, G., (1988). Scheduling independent tasks with deadlines on semi-identical processors. *Journal of Operational Research Society*, 39, 271-277.

[156] Schmidt, G., (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121, 1-15.

[157] Schrage, L. E., and Baker, K. R., (1978). Dynamic Programming Solution of Sequencing Problems with Precedence Constraints, *Operation Research*, 26, 444-448.

[158] Smith, W. E., (1956). Various Optimizers for Single Stage Production, *Naval Research Logistics Quarterly*, 3, 59-66.

[159] Sotskov, Y.N., (1985). Optimal servicing two jobs with a regular criterion, In: *Automation of Designing Processes*, Minsk, 86-95 (in Russian).

[160] Sotskov, Y.N., (1991). The complexity of shop-scheduling problems with two or three jobs, *European Journal of Operational Research*, 53, 326-336.

[161] Steinhöfel, K., Albrecht, A., and Wong, C. K., (2002). The convergence of stochastic algorithms solving flow shop scheduling. *Theoretical Computer Science*, 285, 1, 101-117.

[162] Szwarc, W., (1960), Solution of the Akers-Friedman scheduling problem. *Operations Research*, 8, 6, 782-788.

[163] Taillard, E. D., (1990). Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, 47, 1, 65-74.

[164] Taillard, E. D., (1990). Robust Taboo Search for the Quadratic Assignment problem. *Parallel Computing*, 17, 443-455.

[165] Taillard, E. D., Gambardella, L.M., Gendreau. M., and Potvin, J.Y., (2001). Adaptive Memory Programming: A Unified View of Metaheuristics. *European Journal of Operational Research*, 135, 1, 1-16.

[166] Tanaev V.S., Gordon V.S., and Shafransky Y.M. (1994). *Scheduling theory. Single-stage systems*. Kluwer Academic Publishers. Dordrecht / Boston / London.

[167] Tanaev V.S., Gordon V.S., and Shafransky Y.M. (1994). *Scheduling theory. Multi-stage systems*. Kluwer Academic Publishers. Dordrecht / Boston / London.

[168] T'kindt V., Billaut J. C., (2000). *L'ordonnancment multicritère*. Presses universitaires de Tours.

[169] Ullman, J. D., (1975). NP-complete scheduling problems, Journal of Computer and System Sciences, 10, 384-393.

[170] Van Laarhoven, P. J. M., E. H. L. Aarts and J. K. Lenstra (1992). Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40, 1, 113-125.

[171] Vairaktarakis, G., and Sahni, S., (1995). Dual criteria preemptive open-shop problems with minimum makespan. *Naval Research Logistics*, 42, 103-121.

[172] Wang, G., and Cheng, T. C. E., (2001). Heuristics for two-machine no-wait flowshop scheduling with an availability constraint, *Information Processing Letters*, 80, 6, 305-309.

[173] Whitley, Y., Starkweather, T., Fuquay, D., (1989). Scheduling problems and travelling salesman : the genetic edge recombination operators. *Proceedings of the 3rd International Conference on genetic Algorithms and their applications*, Morgan Kaufmann,133-140.

Résumé : Ordonnancement d'Ateliers sous Contraintes de Disponibilité des Machines.

Contrairement à la plupart des travaux en ordonnancement d'ateliers pour lesquels les machines sont supposées continuellement disponibles, nous étudions dans cette thèse des problèmes d'ordonnancement en présence de périodes d'indisponibilité des machines. Nous considérons le contexte déterministe d'indisponibilités dues à une activité de maintenance préventive. Deux hypothèses caractérisant les tâches de maintenance sont observées: soit les tâches sont totalement fixes, ce qui est communément étudié dans la littérature, soit elles sont flexibles et leurs positions peuvent être optimisées au sein de fenêtres temporelles. Les modèles d'ateliers étudiés sont ceux du flow shop et du job shop, avec un nombre quelconque de machines et un nombre quelconque d'indisponibilités sur chacune d'entre elles. Les opérations à ordonnancer sont supposées strictement non-préemptives. Des algorithmes polynomiaux originaux sont développés pour des problèmes d'ordonnancement à deux travaux et nous nous servons de ces algorithmes pour concevoir des méthodes de résolution approchées et exactes, pour les problèmes généraux. Les méthodes approchées utilisent des méta-heuristiques et les méthodes exactes reposent sur des procédures par séparation et évaluation. Des résultats d'expériences menées sur des instances générées aléatoirement attestent de l'efficacité des algorithmes proposés.

Mots clefs : Ordonnancement, problèmes d'atelier, contraintes de disponibilité, algorithmes polynomiaux, méta-heuristiques, séparation et évaluation.

Abstract : Machine Scheduling with Availability Constraints.

In contrast to most of papers on machine scheduling, in which machines are supposed to be continuously available, we study in this thesis scheduling problems under limited machine availability. We consider the deterministic case of unavailability periods due to a preventive maintenance activity. To deal with the maintenance tasks, two assumptions are considered: either the maintenance tasks are fixed, which is commonly studied in the literature, or they are flexible and their positions inside some time windows can be optimized. The studied models are flow shop and job shop with an arbitrary number of machines and an arbitrary number of unavailability periods on each of them. Operations are supposed to be strictly non-preemptive. Original polynomial algorithms are developed for problems with only two jobs. Based on these algorithms, we provide approximation and exact solution methods for general problems. Approximation methods use meta-heuristics and exact methods are based on branch and bound procedures. Computational results realized on randomly generated problems show the efficiency of the proposed algorithms.

Keywords : Scheduling, shop problems, availability constraints, polynomial algorithms, meta-heuristics, branch and bound.