Les problèmes d'ordonnancement dans les systèmes de production automatisés : Modèles, complexité et approches de résolution

Yazid Mati

THÈSE

présentée à

L'UNIVERSITÉ DE METZ
Mathématiques, Informatique, Mécanique

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Automatique

soutenue par

Yazid MATI

le 12 juin 2002

Titre

Les Problèmes d'Ordonnancement dans les Systèmes de Production Automatisés : Modèles, Complexité et Approches de Résolution

Directeur de thèse : Xiaolan Xie - Co-directeur : Nidhal Rezg

Jury

Président	Jean-Claude Gentina
Rapporteurs	Jacques Carlier Stéphane Dauzère-Pérès Yannick Frein
	Jean-Pierre Campagne Fabrice Chauvet Christian Proust

THÈSE

présentée à

L'UNIVERSITÉ DE METZ
Mathématiques, Informatique, Mécanique

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Automatique

soutenue par

Yazid MATI

le 12 juin 2002

Titre

Les Problèmes d'Ordonnancement dans les Systèmes de Production Automatisés : Modèles, Complexité et Approches de Résolution

Directeur de thèse : Xiaolan Xie - Co-directeur : Nidhal Rezg

Jury

Président	Jean-Claude Gentina
Rapporteurs	Jacques Carlier
	Stéphane Dauzère-Pérès
	Yannick Frein
	Jean-Pierre Campagne
	Fabrice Chauvet
	Christian Proust

année : 2002

N° attribué par la bibliothèque

THÈSE

présentée à

L'UNIVERSITÉ DE METZ
Mathématiques, Informatique, Mécanique

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Automatique

soutenue par

Yazid MATI

le 12 juin 2002

Titre

Les Problèmes d'Ordonnancement dans les Systèmes de Production Automatisés : Modèles, Complexité et Approches de Résolution

Directeur de thèse : Xiaolan Xie - Co-directeur : Nidhal Rezg

Jury

Président	Jean-Claude Gentina
Rapporteurs	Jacques Carlier
	Stéphane Dauzère-Pérès
	Yannick Frein
Examinateurs	Jean-Pierre Campagne
	Fabrice Chauvet
	Christian Proust

Dédicace

A mes parents dont le sacrifice pour faire de moi ce que je suis aujourd'hui n'a pas d'égal.

A mon frère Mohamed qui a toujours été à mes cotés et qui ne cesse de donner pour faire mon bonheur.

A mes sœurs Tamma et Saïda dont l'intérêt qu'elles me portent ne peut être occulté.

A tous mes frères, sœurs, neveux et nièces.

A tous ceux que j'aime.

Je dédie ce modeste travail.

Remerciements

Le travail de recherche qui est présenté dans ce mémoire à pu voir le jour grâce à l'aide précieuse de tous les membres de ma famille qui m'ont encouragé tout au long de mon séjour en France, en particulier durant mes trois années de thèse. Un grand merci à mes parents qui ont sacrifié tant de leurs forces et de leur temps pour assurer mon éducation et ma réussite. Je n'oublierai jamais les coups de téléphone de mon mentor, mon frère Mohamed, qui a su par ses précieux conseils et son expérience me remonter le moral dans mes périodes difficiles.

Je garderai un souvenir inoubliable des trois années passées à travailler avec Monsieur Xiaolan Xie, professeur à l'Ecole Nationale d'Ingénieurs de Metz qui a dirigé mes travaux de recherche. Je le remercie infiniment pour toute sa confiance et toute sa disponibilité. Sa volonté d'atteindre les objectifs, ses critiques constructives et nos réunions de travail m'ont permet d'enrichir mes connaissances dans les domaines du contrôle et de l'ordonnancement. Je lui exprime ici toute ma reconnaissance et toute ma sympathie.

Je remercie Monsieur Nidhal Rezg, maitre de conférences à l'Université de Metz et co-directeur de ma thèse pour son aide et ses conseils.

Je suis extrêmement honoré d'avoir dans mon jury de thèse et surtout en tant que rapporteur Monsieur Jacques Carlier, professeur à l'Université de Compiègne. Je lui suis reconnaissant pour l'intérêt qu'il a porté à mes travaux malgré son emploi du temps fort chargé.

Je suis également très honoré par la présence de Monsieur Stéphane Dauzère-Pérès, professeur à l'Ecole des Mines de Nantes, dans mon jury de thèse. Je n'oublierai jamais ses encouragements après mon exposé à Aussois et les discussions techniques qui ont suivi. Je lui exprime ici toute ma reconnaissance et toute ma sympathie.

Je remercie Monsieur Yannick Frein, professeur à l'Ecole Nationale Polytechnique de Grenoble de me faire l'honneur de participer à mon jury de thèse et d'accepter la responsabilité de rapporteur. Qu'il trouve ici toute ma reconnaissance.

Toute ma gratitude s'adresse en outre à Messieurs Jean-Pierre Campagne, professeur à l'Ecole des Mines de Saint-Étienne, Fabrice Chauvet, responsable R & D à Bouygues Télecom, Jean-Claude Gentina, professeur à l'Ecole Centrale de Lille et Christian Proust, professeur à l'Université de Tours, pour avoir accepté de participer à mon jury.

Un grand merci aux familles Zeraoulia, Hamani, Guedioura et à tous les membres du service OMRA du Touring Club d'Algérie (Bir Khadem) pour leur aide précieuse.

Je remercie Messieurs Zouhir Bouredji et Hassane Alla, professeur à l'Université Joseph Fourier de Grenoble pour m'avoir accueilli au Laboratoire d'Automatique de Grenoble et pour leur précieux conseils qui m'ont permet de prendre les bonnes décisions.

Je réserve une place spéciale dans mes remerciements à mon ami Riad qui m'a beaucoup aidé et auprès de qui j'ai énormément appris. Je le remercie pour sa gentillesse, son amitié et pour tout le temps qu'il m'a consacré pour améliorer la présentation de ce manuscrit. Un grand merci à toute sa famille pour son accueil chaleureux, en particulier Woihida pour son difficile travail de relecture.

Je ne saurais oublier dans ces remerciements les moments passés en compagnie de mon ami Sif. Je le remercie pour toute sa confiance, ses encouragements et toute son amitié. Les chambres 414 Tchèque et 415 Hollandais, les voyages que nous avons effectué ensemble et d'autres moments inénarrables resteront à jamais graver dans ma mémoire.

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

Table des matières

Liste des figures

Chapitre 1 :

Chapitre 2 :

Chapitre 3 :

Chapitre 4 :

Chapitre 5 :

Chapitre 6 :

Liste des tableaux

Liste des algorithmes

Introduction Générale

Ces dix dernières années, les systèmes de production automatisés tels que les ateliers flexibles (FMSs) ont largement attiré l'attention des chercheurs dans le domaine de la recherche opérationnelle. L'intérêt porté à ces systèmes est largement motivé par le caractère de la flexibilité. Par ailleurs, les caractéristiques des systèmes peuvent être rapidement et facilement modifiées de manière à optimiser les rendements en terme de coûts et de délais de la production.

Un système de production automatisé est composé de machines capables de réaliser différentes tâches. Ces machines sont généralement regroupées en cellules de telle sorte que la réalisation d'un produit puisse être complètement accomplie dans une même cellule. Afin de transporter les produits à l'intérieur des systèmes, les ressources modernes de transport tels que les robots, les véhicules autoguidés (AGVs) et les convoyeurs sont largement utilisés. Pour exploiter au mieux le potentiel de ces équipements, une coordination efficace est primordiale.

Un des problèmes les plus difficiles rencontré dans les systèmes de production est le problème d'ordonnancement. Les différentes technologies introduites dans ces ateliers rendent difficile l'application des modèles classiques d'ordonnancement. Plus précisément, la définition traditionnelle d'un problème d'ordonnancement comme étant le séquencement d'un ensemble de travaux, ou jobs, sur un ensemble de machines sans tenir compte des autres ressources engagées n'a plus de sens dans les systèmes de production automatisés. En effet, dans les ateliers flexibles, la fabrication d'un produit peut nécessiter la coopération de différentes ressources telles que les machines, les opérateurs, les ressources de transport, etc. Ainsi, l'intégration des différentes ressources engagées dans la production est nécessaire à l'élaboration d'un modèle réaliste permettant de modéliser un grand nombre d'ateliers flexibles.

Une autre contrainte négligée par la plupart des travaux de recherche dans le domaine de l'ordonnancement concerne la capacité des stocks tampons. En effet, les modèles classiques d'ordonnancement ne tiennent pas compte des limites de la capacité de stockage entre les machines, la supposant infinie. Pourtant, les nouveaux systèmes de production sont de plus en plus composés de lignes de fabrication ayant des stocks intermédiaires très limités, voire nuls, entre les machines. Par conséquent, il faut se tourner vers de nouveaux modèles permettant d'intégrer cette contrainte.

1. Objectif de la thèse

Les travaux de recherche proposés dans cette thèse portent essentiellement sur les problèmes d'ordonnancement rencontrés dans les ateliers flexibles de production et tiennent compte de nouvelles contraintes, notamment la considération des moyens de transport, des stocks tampons de capacité limitée, etc. Trois objectifs principaux sont fixés, à savoir la proposition de nouveaux modèles d'ordonnancement, l'étude de la complexité de ces problèmes et enfin le développement d'algorithmes pour l'ordonnancement et le contrôle des systèmes étudiés. Les deux premiers objectifs sont considérés dans une optique théorique alors que le troisième a un but pratique. Plus précisément, les trois objectifs peuvent être définis de la manière suivante :

(A) *Nouveaux modèles d'ordonnancement*

Le but est de généraliser les modèles classiques d'ordonnancement et d'élaborer une méthodologie intégrée pour la spécification et le contrôle des systèmes de production. Plus précisément, nous développons des modèles d'ordonnancement permettant d'incorporer les différentes caractéristiques des systèmes de production automatisés. Ces modèles sont basés sur les modèles classiques existants dans la littérature, en particulier sur celui du job shop, auxquels s'ajoutent les propriétés suivantes :

- *Opérations multi-ressources* : Une opération peut nécessiter plusieurs ressources en même temps pour sa réalisation.
- *Multiples unités* : Une ressource peut être disponible en plusieurs exemplaires.
- *Retenir et attendre* : A la fin d'une opération, les ressources utilisées ne sont libérées qu'au passage à l'opération suivante.
- *Flexibilité* : Les ressources nécessaires à la réalisation d'une opération ne sont pas fixes, mais choisies dans un ensemble fixé à priori.
- *Moyens de transport* : Les déplacements des produits semi-finis dans le système ne sont pas négligés et sont réalisés par les moyens de transport adoptés pour le système.

(B) *Complexité des problèmes*

Les études théoriques de complexité des problèmes d'ordonnancement permettent de donner une limite entre les problèmes dits faciles, c'est-à-dire pouvant être résolus de manière polynomiale, et les problèmes dits difficiles pour lesquels aucun algorithme polynomial n'est disponible. Dans cette thèse, chaque fois qu'un nouveau modèle d'ordonnancement est proposé, une étude est consacrée à la complexité de ses cas particuliers et ceci afin de pouvoir situer, en terme de complexité, ces nouveaux problèmes par rapport aux problèmes classiques d'ordonnancement.

(C) Algorithmes d'ordonnancement

Dans le contexte actuel, il est nécessaire pour une entreprise manufacturière de disposer d'outils performants pour l'ordonnancement et le contrôle afin de répondre rapidement aux requêtes des clients, et ainsi améliorer sa compétitivité. C'est la réalisation de tels outils qui constitue le troisième volet de cette thèse. En effet, pour tous les modèles définis par le premier objectif, nous proposons des méthodes heuristiques originales permettant d'obtenir des résultats satisfaisants dans un temps assez court. Les algorithmes développés sont basés sur des résultats de complexité prouvés, en particulier sur les algorithmes polynomiaux établis dans certains des cas étudiés.

2. Organisation de la thèse

Dans le premier chapitre, nous dressons une étude bibliographique des différents problèmes d'ordonnancement des systèmes de production. Nous nous focalisons plus particulièrement sur les problèmes d'ordonnancement les plus proches de nos modèles et qui prennent en compte les différentes contraintes rencontrées dans les ateliers flexibles. Nous rappelons en outre quelques méta-heuristiques de la littérature utilisées pour la résolution de problèmes difficiles d'optimisation combinatoire.

Le mémoire se divise ensuite en deux parties que sont : "les problèmes sans contraintes de flexibilité" et "les problèmes avec flexibilité sur l'exécution des opérations", la première partie étant constituée de deux chapitres, la seconde de trois chapitres.

La première partie est dédiée aux problèmes d'ordonnancement de type job shop multi-ressources avec blocage, sans contraintes de flexibilité. Dans le premier chapitre de cette partie, nous introduisons le modèle d'ordonnancement de type job shop multi-ressources avec blocage, et pour lequel une ressource peut être disponible en plusieurs exemplaires. Nous commençons par définir ce type de modèle, puis nous décrivons les différents problèmes pouvant être modélisés de la sorte. Nous identifions ensuite un cas spécial comportant seulement deux jobs à ordonnancer, qui peut être résolu optimalement au moyen d'un algorithme polynomial. En utilisant cet algorithme, nous proposons une méthodologie originale pour la résolution de différents problèmes d'ordonnancement, et nous l'appliquons au problème traité. Enfin, des résultats numériques montrant l'efficacité de l'approche proposée sont présentés.

Le deuxième chapitre de la première partie est consacré à un cas particulier du modèle présenté dans le chapitre précédent. Ce nouveau problème est caractérisé par le fait que les ressources ne sont disponibles qu'en un seul exemplaire. Dans un premier temps, nous modélisons le problème d'ordonnancement en utilisant une nouvelle extension du graphe disjonctif. Cette modélisation est ensuite utilisée pour résoudre le problème d'ordonnancement associé. Un algorithme de recherche locale, basé sur une recherche tabou et une définition originale de la structure de voisinage, est alors présenté. Des résultats numériques sont proposés pour évaluer l'efficacité de l'algorithme développé. Enfin, une nouvelle modélisation du problème d'ordonnancement étudié, basée sur la programmation linéaire en nombres entiers, est décrite.

Dans la seconde partie de la thèse, nous nous intéressons à l'étude des problèmes d'ordonnancement avec flexibilité des ressources. Dans ce type de problèmes, les ressources nécessaires à l'exécution d'une opération ne sont pas fixes mais choisies dans un ensemble fixé à priori.

Dans le premier chapitre de cette partie, le système d'atelier considéré est celui du job shop flexible avec machines non-reliées. Nous présentons tout au long du chapitre une étude de complexité de quelques cas particuliers à deux jobs. Tout d'abord, nous montrons que le problème de la minimisation de tout critère objectif régulier est NP-difficile. Les résultats restent valables même si la préemption des opérations est permise. Ensuite, nous proposons un algorithme pseudo-polynomial pour trouver les ordonnancements optimaux. Nous nous focalisons en outre sur une variante du problème à deux jobs, pour laquelle nous développons un algorithme polynomial pour la minimisation de tout critère régulier. Cet algorithme est ensuite généralisé pour traiter le cas de la préemption des opérations.

Le second chapitre de la partie est consacré à l'étude des problèmes d'ordonnancement de type job shop multi-ressources avec flexibilité des ressources. Dans un premier temps, nous généralisons l'algorithme polynomial proposé dans le chapitre précédent pour résoudre le problème à deux jobs de ce modèle. Une heuristique développée pour résoudre le problème général un nombre quelconque de jobs, sur la base de cet algorithme et la méthodologie décrite en première partie, est ensuite présentée. Des résultats des nombreuses expériences numériques testant l'efficacité de l'approche sont rapportés.

Dans le dernier chapitre de cette partie, nous généralisons le modèle du chapitre précédent en introduisant la propriété "retenir et attendre" relative à la libération des ressources à la fin des opérations. Nous commençons par étudier la complexité du problème à deux jobs, et nous identifions deux cas particuliers pouvant être résolus polynomialement. Dans le premier cas, la flexibilité des ressources est considérée sur les deux jobs, et les temps opératoires ne dépendent pas de la flexibilité sélectionnée. Dans le deuxième cas, un seul job est flexible, et les durées opératoires dépendent fortement de la flexibilité choisie. Par ailleurs, la méthodologie introduite dans la première partie est utilisée pour le développement des méthodes heuristiques de résolution du problème général. Là encore, des résultats expérimentaux sont présentés pour valider les différentes approches proposées.

Chapitre 1 Etat de l'art

Dans ce chapitre, nous considérons dans un premier temps, les problèmes d'ordonnancement des systèmes de production. Dans un second temps, nous dressons un état de l'art sur la recherche tabou et les algorithmes génétiques. Les deux méthodes seront utilisées pour la résolution des problèmes abordés dans cette thèse. Afin de mettre en evidence notre contribution, nous rappellons les travaux traités dans la littérature. Seuls sont traités les problèmes statiques d'ordonnancement, pour lesquels les durées des opérations à réaliser et les relations de précédence sont connues. Notre étude se restreint également aux problèmes où les ressources sont toujours disponibles ; on ne considère ni des pannes, ni des opérations de maintenance sur les ressources.

1. Introduction

Nombreux sont les problèmes qui peuvent être modélisés et résolus en utilisant des techniques d'optimisation telles que la programmation linéaire et dynamique, notamment dans le domaine de l'industrie manufacturière (Stecke [1985]). Une des principales difficultés rencontrées dans ce type d'industrie réside dans la manière de coordonner l'ensemble des ressources engagées dans la production, pour satisfaire la demande des clients, tout en respectant les délais annoncés. C'est ce qui constitue le problème d'ordonnancement, qui se définit formellement de la façon suivante : étant donné un ensemble de ressources (machines, personnels, outils, etc.) permettant d'exécuter un ensemble de travaux, ou jobs, le problème d'ordonnancement consiste à trouver les dates de début de réalisation des opérations constituant les travaux, en tenant compte de la disponibilité et de la capacité des ressources engagées, et ceci dans le but de faire fonctionner le système de façon optimale. On peut par exemple chercher à minimiser la durée de réalisation des travaux. Ce critère, connu sous le nom de *makespan*, est celui que l'on retrouve le plus souvent dans la littérature liée aux travaux d'ordonnancement. D'autres fonctions objectifs telles que la minimisation de la somme des retards, la minimisation des coûts d'utilisation des ressources, etc., sont également traitées.

La recherche dans le domaine de l'ordonnancement a évolué de manière considérable durant ces 40 dernières années que ce soit sur le plan théorique ou sur le plan pratique. L'intérêt croissant porté aux problèmes d'ordonnancement tient du fait que l'une des principales préoccupations du milieu industriel est l'augmentation de l'efficacité et du rendement, en termes de coûts et de délais de livraison de la production. De plus, l'ordonnancement trouve des applications diverses dans tous les secteurs économiques tels que l'industrie (Pinedo [1995]), l'informatique (Blazewicz et al. [1996]), l'administration, etc.

Ainsi, avec cet intérêt accrû, s'est développée une littérature abondante dédiée à l'ordonnancement. De nombreux ouvrages de référence parmi lesquels Conway et al. [1967], Baker [1974], Rinnooy Kan [1976], French [1982], Carlier et Chrétienne [1988], Pinedo [1995], Blazewicz et al. [1996], Brucker [1998], Esquirol et Lopez [1999], Lopez et Roubellat [2001], consacrés à la résolution de divers problèmes, ont été publiés.

Dans ce chapitre, nous considérons les problèmes d'ordonnancement des systèmes de production. En particulier, nous présentons les travaux liés aux problèmes étudiés dans cette thèse, dans le sens où ils prennent en compte les mêmes caractéristiques de système et les mêmes contraintes. Il est à noter que seuls sont traités, les problèmes statiques d'ordonnancement, pour lesquels les durées des opérations et les relations de précédence sont connues. Notre étude se restreint également aux problèmes où les ressources sont toujours disponibles ; on ne considère ni pannes, ni opérations de maintenance sur les ressources.

Dans une première partie, nous présentons les problèmes classiques d'ordonnancement des systèmes de production ainsi que les problèmes particuliers sur lesquels se porte notre étude. Nous traitons dans un premier temps, les systèmes classiques du flow shop et du job shop. Un état de l'art, constitué des principaux résultats de complexité et des meilleures méthodes heuristiques ou exactes développées pour la résolution des problèmes, est dressé. Nous considérons par la suite les problèmes avec contraintes de moyens de transport, en particulier ceux utilisés dans les cellules robotisées. Puis, nous nous intéressons aux problèmes avec flexibilité sur les opérations. Pour ce dernier cas, les systèmes de production considérés sont:

le flow shop hybride, le job shop flexible, et le job shop avec flexibilité des ressources. Etant donné la multitude des études dédiées au flow shop hybride, seuls les meilleurs résultats sont rappelés. Pour les deux autres types de systèmes, nous présentons de manière détaillée les différentes approches existantes dans la littérature.

Dans un deuxième temps, nous traitons les problèmes d'ordonnancement des systèmes de production utilisant des chariots filoguidés ainsi que les problèmes d'ordonnancement avec contraintes de blocage. Contrairement à d'autres problèmes pour lesquels la littérature est abondante, les approches proposées pour résoudre les problèmes sous contrainte de blocage sont rares. Nous les rappelons ainsi que leur champ d'application.

Dans la seconde partie de ce chapitre, nous nous intéressons aux méta-heuristiques. En particulier, nous dressons un état de l'art qui porte sur la recherche tabou, ainsi que sur les algorithmes génétiques. Ces deux méthodes sont utilisées pour la résolution de plusieurs problèmes rencontrés.

2. Ordonnancement de systèmes de production

2.1. Problème du flow shop

Le problème du flow shop, ou atelier à cheminement unique est un problème classique d'atelier dans lequel un ensemble de n jobs (ou travaux) doit être réalisé sur un ensemble de m machines, en passant successivement sur toutes les machines, dans un même ordre. Les machines sont disponibles sur tout l'horizon de planification et elles 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 souvent étudié dans la litérature est le flow shop de permutation (*prmu*) qui se présente lorsque la séquence des jobs visitant une machine est la même pour toutes les machines.

D'un point de vu théorique, les problèmes de type flow shop, à quelques cas particuliers près, sont NP-difficiles. Pour le cas particulier à deux machines $F2 \mid prmu \mid C_{max}$, une solution optimale peut être donnée en un temps polynomial en utilisant la règle de Johnson [1954]. Un autre cas spécial noté $F2 \mid nwt \mid C_{max}$ se présente lorsque toutes les opérations du même job doivent passer sans attente (*nwt*). Ce problème est résolu polynomialement par l'algorithme de Gilmore et Gomory [1964], qui transforme le problème en un problème de voyageur de commerce avec des distances particulières. Ce dernier problème est équivalent à $Fm \mid block \mid C_{max}$ pour lequel la capacité de stock entre les machines est nulle.

En ce qui concerne le cas général avec plusieurs machines $Fm \| C_{max}$, plusieurs méthodes exactes ont été proposées (Carlier et Rebaï [1996], Cheng et al. [1997]). 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 [1996]. Récemment, Nowicki [1999] a dévelopé une recherche tabou dans le cas où la capacité du stock entre les machines est limitée. Une comparaison des différentes méthodes de recherche locale proposées a été réalisée par Glass et Potts [1996].

2.2. Problème du 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 $J2||C_{max}$ par un algorithme polynomial. Le problème à deux jobs $J \mid n = 2 \mid f$ 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 quel critère régulier. Nous décrivons dans ce qui suit l'approche géométrique, en considérant la minimisation de makespan.

2.2.1. Approche géométrique

Le problème $J \mid n = 2 \mid C_{max}$ consiste à ordonnancer deux jobs J_i (i =1,2), qui se décomposent en une séquence d'opérations $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$, où n_i est le nombre d'opérations du job J_i. Chaque opération doit être exécutée sur une seul machine sans interruption, et nécessite p_{ij} unités de temps. Une machine peut réaliser au plus une opération à la fois. L'objectif est de trouver une séquence d'entrée des opérations sur les machines afin de minimiser le makespan. Ce problème est équivalent à la recherche d'un plus court chemin dans le plan muni d'obstacles, où les obstacles représentent les conflits de machines entre les opérations des deux jobs. Plus précisément,

- Chaque job est représenté sur un axe avec n_i intervalles, selon sa gamme opératoire. Chaque intervalle correspond à une opération O_{ij} et a une longueur égale au temps opératoire de l'opération (figure 1.1).

- Les intervalles O_{1j} et O_{2k} constituent un obstacle si les deux opérations correspondantes partagent la même machine.

- La bordure supérieure et à droite du rectangle défini par le point d'origine O et le point final F est considéré comme l'obstacle final.

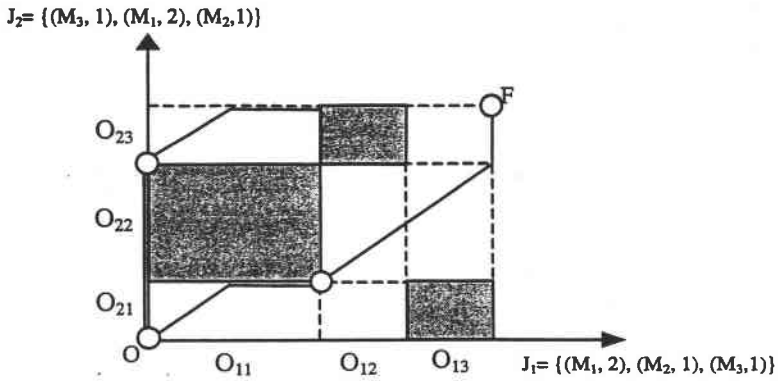


Figure 1.1 : Plus court chemin dans le plan

Une solution réalisable du problème $J \mid n = 2 \mid C_{max}$ correspond à un chemin de l'origine O au point final F. Le chemin est composé des segments horizontaux (job J_1 uniquement en exécution), verticaux (job J_2 uniquement en exécution) et diagonaux (les deux jobs progressent en même temps sur des machines différentes). Puisque l'on considère que les opérations doivent s'exécuter sans préemption, les chemins doivent éviter l'intérieur des obstacles, qui correspondent à une utilisation conjointe d'une machine par les opérations des deux jobs. La longueur d'un segment diagonal est égale à la longueur de sa projection sur l'un des axes, et indique le temps nécessaire pour exécuter simultanément les deux jobs.

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 $N = (V, E, d)$ approprié. V est l'ensemble des sommets du réseau, composé de l'origine O, du point final F et des coins sud-est (SE) et nord-ouest (NW) des obstacles. Chaque sommet v_i a au plus deux successeurs directs obtenus en progressant diagonalement à partir de v_i jusqu'à heurter un obstacle D. Si l'obstacle D est l'obstacle final, alors le sommet F est le seul successeur direct de v_i, sinon les coins nord-ouest C_{NW} et sud-est C_{SE} de l'obstacle D sont les deux successeurs directs de v_i (figure 1.2). Enfin, la distance entre le sommet v_i et son successeur C_{SE} (resp. C_{NW}) est égale à la projection sur l'axe x (resp. y) des segments liants les deux points.

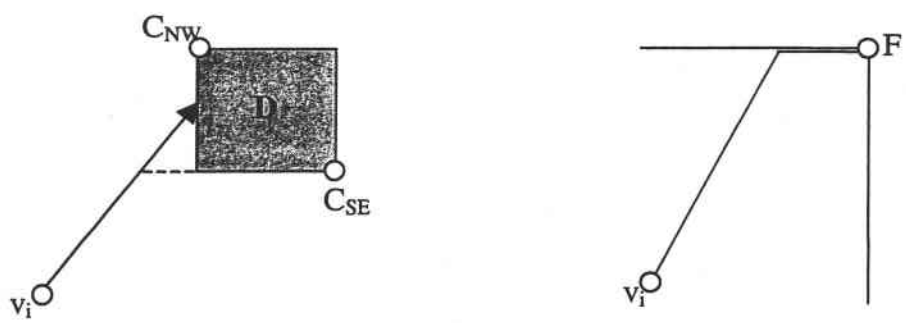


Figure 1.2 : Successeurs directs d'un sommet v_i

Il a été montré par Brucker [1988] que le réseau N pouvait être construit en $O(r \log r)$ étapes, où r est le nombre d'obstacles, égal au plus à $O(n_1\, n_2)$, où n_i est le nombre d'opérations du job J_i. D'autre part, le plus court chemin dans un graphe acyclique peut être obtenu en $O(r)$ étapes. En conséquence, la complexité du problème $J \mid n = 2 \mid C_{max}$ est $O(r \log r)$. Ces mêmes résultats restent vrais pour le problème de job shop multiprocesseur noté $JMPT \mid n = 2 \mid f$, pour lequel la réalisation d'une opération peut nécessiter en même temps plusieurs machines (processeurs) (Brucker [1998]).

2.2.2. *Autres variantes de l'approche géométrique*

Depuis les travaux de Sotskov [1985] et Brucker [1988], nombreuses ont été les généralisations de l'approche géométrique. La première extension a été proposée par Sotskov [1991] 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 allant horizontalement à l'intérieur de l'obstacle jusqu'à atteindre la bordure droite sont les deux successeurs de v_i. Cette nouvelle approche donne les solutions optimales pour tout critère objectif régulier et a une complexité $O(n^3_{max})$, où $n_{max} = \max\{n_1, n_2\}$.

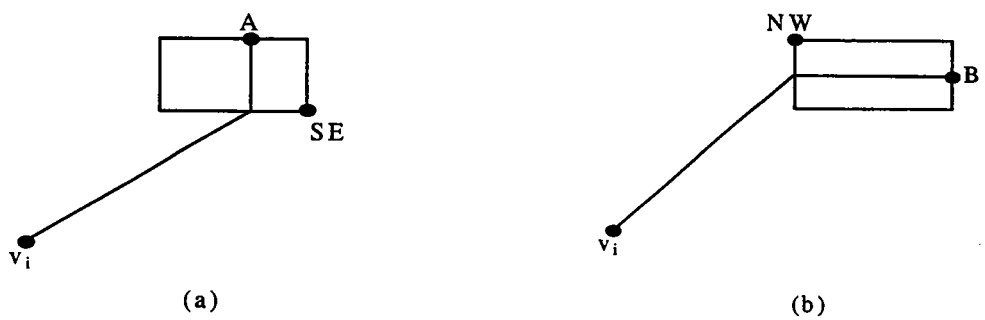


Figure 1.3 : Nouveaux successeurs d'un sommet v_i

Brucker et Jurisch [1993] ont proposé une autre extension pour le problème du job shop à deux jobs, avec des contraintes de précédence et des dates de disponibilités $J \mid n = 2 ; prec ; r_i \mid \max(c_{ij} + q_{ij})$, où c_{ij} est la date de fin de l'opération O_{ij}, et q_{ij} est la durée de latence "*tail*". Ils ont généralisé la représentation dans le plan en ajoutant un troisième axe de temps pour permettre de représenter les contraintes de disponibilités des jobs. Cette extension a été utilisée pour le calcul des bornes inférieures dans la méthode par séparation et évaluation (Brucker et al. [1994]), développée dans le cas général de plusieurs jobs. La complexité de l'approche est $O((n_1\, n_2)^3)$.

Pour résoudre le cas général, différentes 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 (Applegate et Cook [1990], Carlier et Pinson [1990], 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 ont été proposées pour approcher la solution optimale. 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 détail sur toutes les méthodes appliquées au modèle job shop, nous renvoyons le lecteur intéressé aux articles de Jain et Meeran [1999], Blazewicz et al. [1996], Aarts et al. [1994].

2.3. Cellules robotisées

Une cellule de production robotisée se compose de plusieurs machines arrangées autour d'un robot de manutention utilisé pour transporter les produits entres les machines (figure 1.4). La capacité de stock entre deux machines est très limitée voire nulle. Le problème d'ordonnancement des cellules robotisées dérive des problèmes du flow shop et du job shop, et consiste à prendre en compte les déplacements des jobs dans le système.

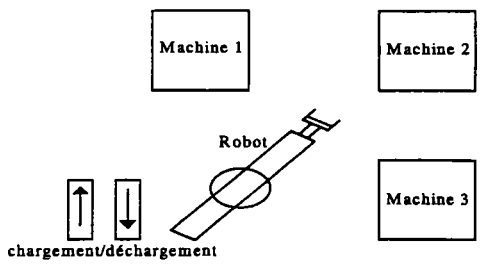


Figure 1.4 : Cellule robotisée

En raison des nombreuses applications pratiques de ce modèle (Kamoun et al. [1999]), plusieurs études y ont été consacrées en particulier pour la minimisation du temps de cycle dans une production cyclique. Dans les travaux de Sethi et al. [1992], les auteurs ont proposé une méthode pour déterminer la séquence optimale des mouvements du robot pour le cas de deux machines fabriquant un seul produit. Des extensions de l'algorithme de Gilmore et Gomory [1964] ont été développées pour ordonnancer polynomialement le problème du flow shop à deux machines sous des hypothèses particulières (Crama et Van de Klundert [1997], Hall et al. [1997]). Un schéma de classification des problèmes d'ordonnancement dans les cellules robotisées est donné dans Hall et al. [1997]. Récemment, Abadi et al. [2000] ont considéré un problème de flow shop avec blocage et ont proposé des heuristiques ainsi que des méthodes optimales pour la minimisation du temps de cycle. Enfin, Kamoun et Sriskandarajah [1993] ont montré que le problème devenait NP-difficile à partir de 3 machines.

Le cas des ordonnancements acycliques est aussi considéré dans la littérature. Legendran et Sriskandarajah [1996] ont fourni un exemple pour lequel un ordonnancement acyclique domine tous les ordonnancements cycliques. King et al. [1993] ont proposé une procédure de séparation et évaluation pour le flow shop avec un seul robot. Langston [1987] a proposé des algorithmes à performance garantie pour le cas d'un flow shop flexible avec deux étages et un seul robot entre les deux étages. Hurink et Knust [1999] ont développé une heuristique en deux phases pour le problème de job shop avec un robot de manutention, qui consiste à résoudre d'une manière disjointe le problème d'ordonnancement sur les machines et la gestion des déplacements du robot. Dans la première phase, le problème de job shop est résolu par une recherche tabou. En se basant sur l'ordonnancement trouvé, des fenêtres de temps sont

attribuées à chaque opération afin de déterminer la bonne séquence d'utilisation du robot. Ces séquences sont obtenues en résolvant un problème de voyageur de commerce avec fenêtres temporelles (Gendreau et al. [1997]). Hurink et Knust [2001] ont étudié le problème de minimisation du makespan dans un flowshop avec un seul robot pour transporter les produits entre les machines. Enfin, on trouve les travaux de Knust [1999], où les problèmes avec moyens de transport sont étudiés.

2.4. Problèmes avec flexibilité des ressources

Dans ce type de problèmes, la machine nécessaire pour exécuter une opération n'est pas connue à priori mais doit être choisie dans un ensemble donné. Ce modèle est introduit dans les systèmes de production dans le but d'augmenter leurs flexibilités et ainsi palier les perturbations qui peuvent se produire. Trois types de modèles sont considérés ci-après : le flow shop hybride, le job shop flexible, et le job shop avec flexibilité de ressources.

2.4.1. Flow shop hydride

Dans un problème de type flow shop hybride, on considère des étages constitués d'un ensemble de machines capables de réaliser les mêmes opérations. Les différents jobs doivent passer dans l'atelier dans le même ordre, et chaque job doit passer sur chaque étage. Le problème d'ordonnancement consiste alors à trouver pour chaque job, la machine exécutant l'opération associée à chaque étage. Les applications pratiques modélisées de cette manière sont nombreuses, on peut en trouver quelques unes dans (chapitre 10 de Lopez et Roubellat [2001]). Le flow shop en étant un cas particulier, la majorité des problèmes d'ordonnancement de flow shop hybride sont NP-difficiles. En effet, les premiers travaux réalisés par Gupta [1988] s'intéressant à des ateliers composés de deux étages ont montré que la minimisation de la plus grande date de fin est NP-difficile au sens fort même dans ce cas, dès qu'un étage a plus d'une machine. Depuis, plusieurs méthodes approchées basées sur la règle de Johnson [1954] ont été proposées.

Pour le cas général de plusieurs étages, les méthodes par séparation et évaluation ont été largement utilisées (Dessouky et al. [1998], Moursli et Pochet [2000]). Par ailleurs, de nombreuses heuristiques ont également été proposées pour résoudre ce problème (Linn et Zhang [1999]). Par exemple, une recherche tabou améliorée a été développée par Nowicki et Smutnick [1998]. Le lecteur trouvera dans Negenman [2001], une comparaison entre les heuristiques de recherche locale appliquées à ce problème et un état de l'art détaillé dans l'article de Linn et Zhang [1999].

2.4.2. Job shop flexible

Le job shop flexible est une extension du problème classique de job shop. Dans ce type de problème, une opération nécessite exactement une machine pour être réalisée et cette machine peut être choisie dans un ensemble défini à priori. Le problème d'ordonnancement consiste alors à affecter une machine à chaque opération et déterminer la séquence des opérations sur les machines obtenues, afin de minimiser un critère objectif donné. Deux cas sont traités dans la littérature ; le cas où les durées opératoires sont les mêmes pour toutes les machines pouvant exécuter une opération (machines reliées), et le cas où la durée dépend de la machine

choisie (machines non-reliées). Une grande variété de problèmes pratiques peut être traitée en utilisant ce modèle, par exemple l'optimisation des opérations des grues (Mastrolilli et Gambardella [2000]).

La première étude a été initiée par les travaux de Brucker et Schlie [1990], qui ont proposé une extension de l'approche géométrique permettant de résoudre de façon optimale le problème $JMPM \mid n = 2 \mid f$, avec n'importe quel critère objectif régulier. Dans leur nouvelle approche géométrique, la notion d'obstacles est élargie et la définition de successeurs directs d'un sommet du réseau est modifiée. La complexité de l'approche est $O(r^3)$, où r est le nombre d'obstacles.

L'approche a plus tard été étendue par Jurisch [1995] pour résoudre le problème $JMPM \mid n = 2, prec ; r_i \mid L_{max}$. La complexité de cette nouvelle approche est $O(n^5{}_{max})$, où $n_{max} = \max \{n_1, n_2\}$. L'algorithme de résolution utilise la méthodologie introduite dans Brucker et Jurisch [1993], pour le problème $J \mid n = 2; prec; r_i \mid C_{max}$, c'est-à-dire la représentation à trois axes désignant respectivement la gamme du premier job, celle du second et l'axe de temps.

Le problème de job shop flexible devient NP-difficile à partir de deux machines et trois jobs $JMPM2 \mid n = 3 \mid C_{max}$. Pour résoudre le cas général à plusieurs jobs noté $JMPM||C_{max}$, une méthode de séparation et évaluation est développée par Jurisch [1992]. La détermination des bornes inférieures est basée sur la relaxation du problème à deux jobs $JMPM \mid n = 2, prec; r_i \mid L_{max}$ qui est résolu polynomialement par l'algorithme proposé par Jurisch [1995]. Hurink et al. [1994] ont utilisé une extension du graphe disjonctif pour développer une heuristique de recherche tabou. Deux voisinages permettant de générer une solution voisine ont été proposés par les auteurs qui se sont basés sur la notion de block introduite pour la première fois par Grabowski et al. [1986]. Il a été démontré que le deuxième voisinage était connexe, ce qui est une propriété importante pour la convergence des méthodes de recherche locale. Des benchmarks ont été générés aléatoirement en utilisant les données du problème de job shop (Fisher et Thomson [1963] ; Lawrence [1984]), et les solutions trouvées ont été comparées avec les bornes inférieures proposées dans Jurisch [1992]. Cette approche est caractérisée par le fait que l'affectation des machines aux opérations et le séquencement sur les machines sont traités séparément.

Une approche en deux phases a été proposée par Paulli [1995] pour ordonnancer un système de production manufacturier dans lequel le nombre de jobs pouvant être exécuté en même temps est limité. Dans la première phase, les machines sont affectées aux opérations en utilisant des règles de priorité, et le problème devient alors un job shop classique. La représentation par un graphe disjonctif est ensuite utilisée pour développer une heuristique de recherche tabou permettant de trouver les meilleures séquences d'entrée. Les deux phases sont répétées et une opération appartenant au chemin critique est à chaque fois réaffectée.

Dans le cas où les durées opératoires ne sont pas égales, deux types d'approches ont été proposés dans littérature : les approches hiérarchiques et intégrées. Le premier type d'approche est basé sur l'idée de décomposer le problème afin d'en réduire la complexité. Ce type d'approche est naturel pour le *JMPM* dans la mesure où les problèmes d'affectation et de recherche de la meilleure séquence d'entrée peuvent être séparés.

Brandimarte [1993] a été le premier à utiliser l'approche par décomposition pour le *JMPM* avec comme critère la minimisation du makespan. L'approche hiérarchique est basée sur la

décomposition du problème en un problème d'affectation des machines et un problème d'ordonnancement des jobs. L'auteur résout donc d'abord le problème d'affectation en utilisant des règles de priorités existantes, avant de se concentrer sur le problème de job shop. La réaffectation des machines aux opérations sur le chemin critique, étant calculée à chaque intervalle de temps prédéfini, le problème de job shop est ainsi résolu à nouveau. La recherche tabou est utilisée en adoptant pour définir le voisinage la permutation de deux opérations. L'heuristique finale est testée sur des exemples générés aléatoirement. L'heuristique a de plus été modifiée pour la minimisation de la *somme des retards pondérés* (*total weighted tardiness*).

Dauzère-Pérès et Paulli [1994] considèrent la minimisation du makespan et proposent des conditions suffisantes permettant d'assurer que le déplacement des opérations n'augmente pas la valeur du makespan. Basée sur ces conditions, une approche de recherche tabou est développée pour résoudre le problème du job shop une fois que l'affectation des machines aux opérations est fixée. Les résultats trouvés par leur méthode sont comparables à ceux de Hurink et al. [1994]. Une autre approche basée sur la recherche tabou est proposée par Barnes et Chambers [1996].

Le second type d'approche que sont les approches intégrées, consiste à résoudre simultanément les problèmes d'affectation et de séquencement. Les méthodes proposées sont basées là encore sur les méthodes de recherche locale. La méthode proposée par Dauzère-Pérès et Paulli [1997] repose sur une nouvelle façon de déplacer une opération sur le chemin critique, pour laquelle la réaffectation et le reséquencement ne sont pas différenciés (Dauzère-Pérès [1994]). Dans le cas où les opérations *j* et *k* utilisent la même machine que l'opération *i*, un reséquencement est réalisé. Si par contre les machines sont différentes, on a alors un mouvement de réaffectation (figure 1.5). Basée sur une extension du graphe disjonctif, le voisinage ainsi construit est connexe. Ce voisinage est intégré pour l'exploration de l'espace des solutions dans la recherche tabou. Les résultats obtenus sont meilleurs que ceux donnés dans Hurink et al. [1994], Dauzère-Pérès et Paulli [1994].

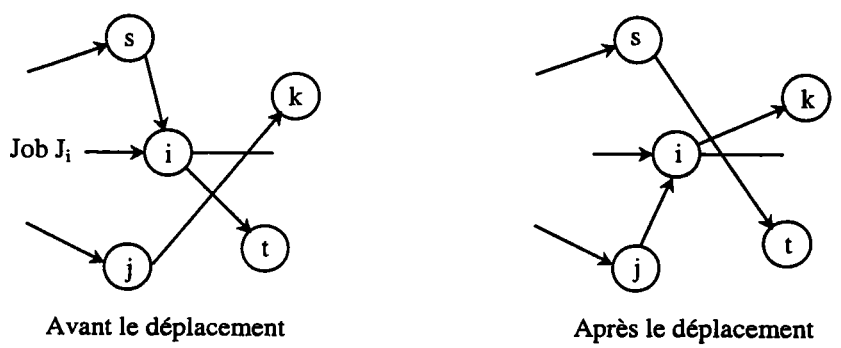


Figure 1.5 : Déplacement d'une opération

Mastrolilli et Gambardella [2000] proposent une autre approche intégrée basée sur la recherche tabou. Cette approche utilise le graphe disjonctif pour la représentation des solutions, ainsi que deux types de voisinages dont le premier est connexe. La contribution de cette étude est la diminution de la taille du voisinage. Cette approche améliore les solutions proposées dans les études précédentes, et est connue comme étant la meilleure heuristique pour résoudre le problème d'ordonnancement de type $JMPM||C_{max}$.

2.4.3. Job shop avec flexibilité des ressources

Le modèle de job shop avec flexibilité de ressources (Dauzère-pérès et al. [1998]) appelé aussi job shop multi-mode (Brucker et Neyer [1998]) est une généralisation directe du modèle de job shop flexible. Il combine le modèle de job shop multiprocesseur noté *MPT* job shop, c'est-à-dire., un job shop dans lequel une opération peut nécessiter plusieurs ressources en même temps (Drozdowski [1996]), et le modèle du job shop flexible (figure 1.6). Dans l'exemple de la figure 1.6, la ressource R_2 est disponible avec une seule unité alors qu'il y a deux unités des ressources R_1 et R_3. Trois alternatives sont possibles pour exécuter l'opération O_{ij} ; le premier ensemble candidat contient trois ressources R_1, R_2 et R_3. Dans le second ensemble, O_{ij} a besoin de deux types de ressources, mais avec deux unités de R_1. Enfin, le dernier ensemble candidat est composé de deux unités de R_3 et une unité de R_2.

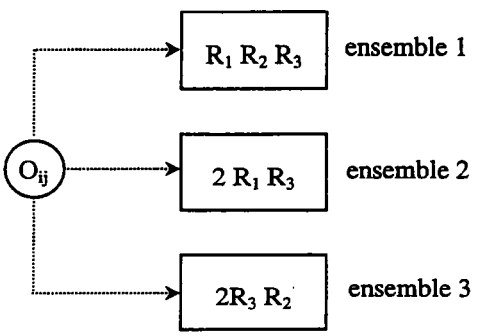


Figure 1.6 : Ensemble des ressources candidates pour une opération

Plus précisément, dans le job shop avec flexibilité de ressources, un ensemble $A_{ij} = \{A_{ij}^1, A_{ij}^2, ..., A_{ij}^{m_{ij}}\}$ est associé à chaque opération, où $A_{ij}^k \subset \Re$, $\Re$ étant l'ensemble des ressources. Le problème consiste alors à affecter à chaque opération O_{ij} un ensemble de ressources $A_{ij}^k \in A_{ij}$. Ce problème a été étudié par Dauzère-Pérès et al. [1998] ainsi que par Brucker et Neyer [1998]. Dans la première étude, une généralisation du graphe disjonctif est proposée. L'approche développée par Dauzère-Pérès et Paulli [1997] est étendue, à travers une recherche tabou avec un voisinage vérifiant la propriété de connectivité. Dans la seconde étude, la méthode utilisée est basée sur le graphe disjonctif et généralise l'approche proposée par Hurink et al. [1994]. Il est à noter que ce problème comme tout problème d'ordonnancement d'atelier peut être vu comme un cas particulier du problème d'ordonnancement de projet avec contraintes de ressources (RCPSP) où plusieurs modes sont autorisés (voir par exemple Brucker et al. [1999]).

2.5. Problèmes avec moyens de transport

De nombreuses études sont consacrées dans la littérature aux problèmes d'ordonnancement avec prise en compte des moyens de transport. Ceci s'explique par la nouvelle implémentation dans les systèmes de production qui tendent à utiliser, de plus en plus, plusieurs moyens pour faciliter le transport des différents types de produits sans intervention d'opérateurs humains. Parmi ces moyens de transport, on trouve les véhicules filoguidés (*AGVs*) (Kusiak [1990]). Deux modèles d'*AGVs* sont utilisés en pratique: les *AGVs* centralisés et les *AGVs* distribués. Dans le premier modèle, un *AGV* est dédié à un job du début à la fin de sa réalisation. Après la fin de l'exécution du job, l'*AGV* revient à la station de chargement/déchargement pour être affecté à un autre job. Dans le second modèle, chaque

machine a son propre *AGV* qui est seulement utilisé pour transporter les produits à la machine suivante. L'importance d'intégrer les moyens de transports dans les systèmes flexibles de production (*FMSs*) est soulignée par Raman et al. [1986], Greenwood [1988], Borez [1989], Han et Mcginnis [1989], Kouvelis [1992]. Le réseau de routage des *AGVs* d'un atelier flexible donné dans Langevin et al. [1996] est représenté par la figure 1.7. Les points ronds représentent les points de chargement et de déchargement des produits semi-finis. Les *AGVs* peuvent se déplacer dans les deux sens entre deux points particuliers. Ainsi, des collisions peuvent se produire si les déplacements des deux *AGVs* ne sont pas correctement planifiés.

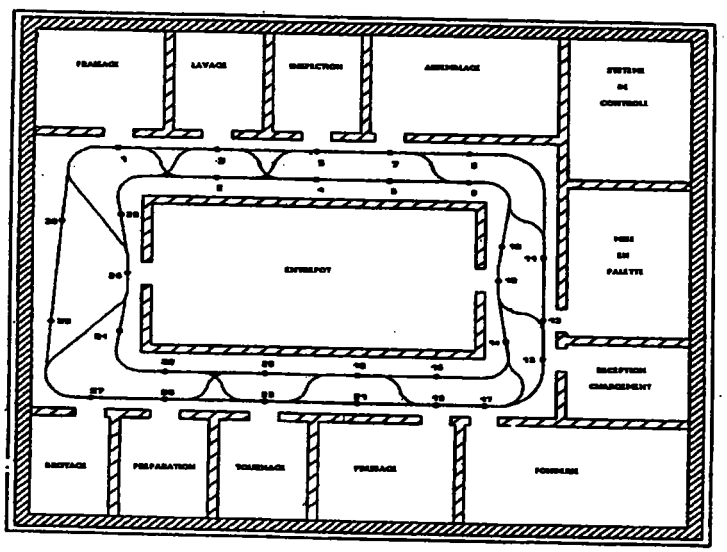


Figure 1.7 : Agencement du réseau des *AGVs* dans une *FMS*

Dans Bilge et Ulusoy [1995], le problème d'ordonnancement simultané d'*AGVs* identiques et de machines est considéré sous des hypothèses particulières : (i) une capacité du stock entre les machines est supposée infinie, (ii) un *AGV* ne retourne pas à la station de chargement/déchargement après la fin de l'exécution d'un job, (iii) la préemption des actions des *AGVs* n'est pas permise, (iv) le routage des *AGVs* est connu à l'avance. Dans un premier temps, un modèle de programmation non linéaire a été développé pour l'ordonnancement des *AGVs* et les jobs. Le modèle mathématique n'étant pas exploitable à cause de la taille du problème et de la non-linéarité des contraintes, une procédure itérative en deux-phases a été proposée. Cette procédure opère de la manière suivante. Dans la première phase, le problème d'ordonnancement est résolu sans tenir compte des *AGVs* et une fenêtre de temps est associée à chaque opération selon la solution trouvée. Dans la deuxième phase, une heuristique appelée «*sliding time windows*» inspirée de l'approche de Ferland et Fortin [1989], est appliquée pour trouver les déplacements des *AGVs* respectant les fenêtres de temps. Si l'heuristique n'arrive pas à trouver une solution réalisable, les fenêtres de temps sont mises à jours et l'heuristique est relancée.

Ulusoy et al. [1997] ont considéré une *FMS* avec les mêmes hypothèses que les travaux de Bilge et Ulusoy [1995]. Une approche basée sur les algorithmes génétiques (*AGs*) a été proposée pour minimiser le makespan. L'algorithme génétique utilise une représentation directe, dans laquelle figurent le séquencement sur les machines et l'affectation des *AGVs* aux jobs. Un opérateur de croisement de type uniforme est considéré et la permutation entre deux gènes d'un chromosome est choisie comme opérateur de mutation. La population initiale est déterminée aléatoirement. Des bornes inférieures sont proposées pour tester l'efficacité des solutions trouvées par l'*AG*. L'algorithme génétique est testé sur les instances proposées dans

Bilge et Ulusoy [1995] et sur d'autres instances générées aléatoirement. Les résultats obtenus montrent que cette approche améliore considérablement les résultats de Bilge et Ulusoy [1995].

Langevin et al. [1996] ont étudié un système de production avec deux *AGVs* bidirectionnels. Comme les deux *AGVs* peuvent utiliser le même chemin en même temps, des collisions peuvent se produire. Pour résoudre simultanément le problème d'affectation, d'ordonnancement et la détermination du routage pour les *AGVs*, une méthode de programmation dynamique visant à minimiser le makespan est introduite et des règles de dominance sont développées. Des extensions permettant la prise en compte d'autres critères objectifs, sont aussi discutées. L'approche développée est testée sur un exemple pratique schématisé par la figure 1.7.

Pour les travaux de Lee et Dicesare [1994], deux types de *FMSs* sont considérés : des *AGVs* centralisés et des *AGVs* distribués. Une modélisation par des réseaux de Petri permettant d'intégrer le problème d'ordonnancement sur les machines et celui de la supervision des *AGVs* est proposée. Basée sur le modèle trouvé, une heuristique de recherche aveugle appelée algorithme A^*, est utilisée pour approcher la solution optimale dans le graphe de marquage du réseau de Petri. Une approche très similaire est proposée dans Sun et al. [1994].

D'autres travaux basés sur la modélisation mathématique du problème ont aussi été proposés (voir par exemple Blazewicz et Fincke [1994]). On trouve les travaux de Blazewicz et al. [1991] qui ont développé une approche par programmation dynamique pour résoudre le problème d'ordonnancement d'un *FMS* fabriquant des produits pour des hélicoptères. Krishnamurthy et al. [1993] ont considéré la minimisation du makespan dans le cas d'un réseau de routage bidirectionnel des *AGVs*. Un algorithme de génération de colonnes permettant de trouver des routes sans conflit pour les *AGVs* a été mis au point. Un autre algorithme pour la détermination des routes sans conflit est proposé dans Kim et Tanchoco [1991]. Dans Chen [1996], un modèle de programmation linéaire en nombre entier est proposé pour minimiser la somme des coûts de transport, l'utilisation des véhicules et le coût de stockage. Le problème est résolu au moyen d'une approche par relaxation lagrangienne. Enfin, des règles de priorité testées par simulation sont proposées dans Sabuncuoglo et Hommertzheim [1992], Sabuncuoglo et Hommertzheim [1993].

2.6. Problèmes avec contraintes de blocage

Les changements apparus progressivement dans les systèmes de production modernes rendent difficile l'application directe des modèles classiques d'ordonnancement. En effet, les responsables de production tendent à automatiser de plus en plus la production en utilisant plusieurs types de ressources automatisées telles que les machines de contrôle numérique, les systèmes de stockage automatisés, etc. Aussi, les moyens automatisés de transport tels que les robots, les véhicules filoguidés et les convoyeurs, sont introduits pour transporter les produits à fabriquer dans le système. La tendance à construire des lignes de fabrication avec des stocks intermédiaires très limités entre les machines est également une contrainte qui limite les applications des modèles classiques d'ordonnancement.

La présence de cette variété de ressources dans le système complique considérablement le problème, et une mauvaise coordination de ces ressources conduit à des situations indésirables dans le système de production. Ces situations sont connues sous le nom de problème de

blocage. Une situation de blocage est illustrée par la figure 1.8. Le premier produit P_1 s'exécutant sur la machine M_1 doit passer sur la machine M_3, le produit P_3 s'exécutant sur la machine M_3 doit progresser vers la machine M_2, et enfin le produit P_2 actuellement sur la machine M_2 doit aller sur la machine M_1. Ainsi, chaque produit est en attente d'une machine qui est retenue par un autre produit. Si les machines et le robot ne peuvent pas manipuler (exécuter) plus d'une opération à la fois, cette attente circulaire cause un blocage des produits. Dans cette situation, le déblocage de la situation ne peut être résolu que par la mise en place d'un stock intermédiaire ou par l'intervention d'un opérateur humain.

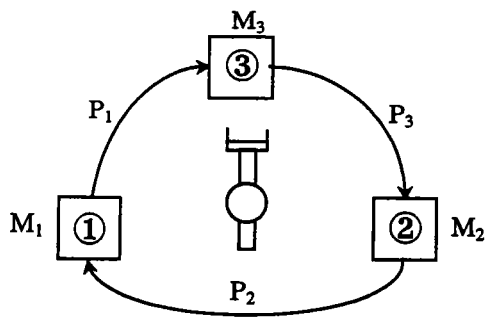


Figure 1.8 : Blocage sur le routage de produits

Les études des problèmes de blocage ont été menées en premier sur les systèmes d'exploitation informatiques. Coffman et al. [1971] définissent quatre conditions caractérisant l'existence de blocages : (1) exclusion mutuelle, (2) non préemption, (3) condition retenir et attendre et (4) attente circulaire.

Dans les systèmes de production, les deux premières conditions sont généralement satisfaites. Le blocage ne se produit pas dans les modèles traditionnels tels que le flow shop et le job shop, car la troisième condition n'est pas remplie. En effet, dans ces systèmes, une machine exécutant une opération est libérée dès la fin de la durée opératoire. Malheureusement, la contrainte *retenir et attendre* est présente dans les systèmes de production utilisant des moyens de transport et pour lesquels les capacités des stocks entre les machines sont souvent faibles voire nulles.

Deux approches sont utilisées pour résoudre les situations de blocage. La première approche consiste à résoudre d'abord le problème d'ordonnancement sans tenir compte des situations de blocage, et ensuite d'appliquer des stratégies de prévention, de recouvrement et de contrôle en temps réel. La deuxième approche consiste à intégrer le problème de blocage à la construction des ordonnancements.

2.6.1. Stratégies de détection, prévention et évitement de blocage

En présence de situations de blocage, il est important de caractériser ces situations afin d'interdire au système de les atteindre (deadlock prevention / avoidance problem) ou alors de faire une correction a posteriori pour sortir de ces situations (deadlock recovery – correction a posteriori). En effet, même si un ordonnancement sans blocage est conçu, tout changement imprévu apparu dans le système tel que la panne d'une machine, peut engendrer une difficulté à son application.

La détection de blocage est un problème de reconnaissance où il s'agit de répondre par vrai si l'état actuel du système mène à un état de blocage et par faux dans le cas contraire. La prévention consiste à définir pour chaque état du système, une politique d'allocation de ressources afin d'éliminer les blocages. Cette politique est nécessairement statique ; ainsi une fois établie, nous sommes sûrs que le système ne peut atteindre les états indésirables. En revanche, le contrôle en temps réel permet de définir les allocations de ressources admissibles et interdites. La correction a posteriori consiste alors à choisir parmi toutes les allocations admissibles, celle qui satisfait l'objectif fixé. Plusieurs méthodes de prévention et d'évitement de blocage ont été proposées ces dernières années. Ces méthodes reposent essentiellement sur les outils de la théorie des graphes et les réseaux de Petri (Murata, [1989], David et Alla [1989], Proth et Xie [1996]).

Banaszak et Krogh [1990] ont considéré un système de production de type job shop dans lequel chaque opération nécessite une seule machine, qui reste occupée jusqu'au passage à l'opération suivante. Une politique de contrôle en temps réel basée sur l'algorithme de Banker pour les systèmes d'exploitation y est définie. Des améliorations de cet algorithme sont proposées dans Hsieh et Chang [1994], Xing et al. [1996].

Dans Ezpeleta et al. [1995], les auteurs ont généralisé le modèle proposé par Banaszak et Krogh [1990] en introduisant la flexibilité de routage. Une politique sous-optimale fondée sur les propriétés des siphons a ensuite été développée. Chu et Xie [1997] ont proposé une méthode de contrôle basée sur les propriétés des siphons et la programmation linéaire en nombre entier.

Dans les travaux de Wysk et al. [1991], Cho et al. [1995], la modélisation est basée sur la théorie de graphe. Les situations de blocage dans des *FMSs* sont caractérisées et des techniques d'évitement de blocage en temps réel sont proposées. Fanti et al. [1996,1997,1998] ont introduit deux graphes orientés pour décrire les relations entre le routage des jobs et les ressources. En utilisant ces graphes, les auteurs ont formulé des stratégies de contrôle pour l'évitement de blocage.

Viswanadham et al. [1990] ont proposé des stratégies de prévention en temps réel. Ces stratégies s'appuient sur le graphe de marquage du réseau de Petri modélisant le système. La théorie des automates est utilisée dans Reveliotis et al. [1997] pour modéliser le système étudié dans lequel une opération peut nécessiter plusieurs ressources en même temps. Les auteurs proposent des politiques d'évitement de blocage basées sur l'analyse des capacités de ressources.

Ces approches de détection et prévention de blocage permettent au système d'opérer sans situations de blocage. Cependant, elles ne tiennent pas compte des temps opératoires des opérations, et ainsi la solution au problème d'ordonnancement proposée peut être très éloignée de la solution optimale. C'est pour cette raison qu'il est important de pouvoir résoudre simultanément les problèmes de blocage et de séquencement sur les ressources. En effet, nous pensons que la séparation des problèmes d'ordonnancement et de détection/prévention du blocage peut conduire à des mauvaises performances.

Nous illustrons la supériorité d'une approche intégrée par rapport à une approche hiérarchique par l'exemple du job shop composé de deux jobs J_1 et J_2 qui doivent être réalisés sur cinq machines M_1, M_2, ..., M_5. Le job J_1 passe successivement sur les machines M_1, M_2 et M_3 avec les durées opératoires : 1 sur M_1, 1 sur M_2 et 10 sur M_3. Le job J_2 passe sur les machines M_4,

M_3, M_2 et M_5 avec les durées opératoires : 2 sur M_4, 1 sur M_3, 1 sur M_2 et 11 sur M_5. il n'existe pas de stocks tampons entre les machines. L'approche hierarchique résout d'abord le problème d'ordonnancement du job shop classique pour J_1 et J_2, et ensuite applique des politiques de détection et de prévention. Pour cet exemple, la solution optimale est :

$$S_{11} = 0, S_{12} = 1, S_{13} = 3, S_{21} = 0, S_{22} = 2, S_{23} = 3, S_{24} = 4,$$

où S_{ik} est la date du début de l'opération k du job J_i. Le makespan est égal à $C_{max} = 15$.

Pour le contrôle en temps réel avec la politique de prévention, les opérations O_{11} et O_{12} sur les machines M_1 et M_2, peuvent démarrer à l'intant $S_{11} = 0$ et $S_{12} = 1$, respectivement. L'opération O_{21} du job J_2 démarre sans générer des blocages à l'intant $S_{21} = 0$. Cependant, à l'instant $S_{22} = 2$, la progression du job J_2 de la machine M_4 à la machine suivante M_3, est interdite par la politique de prévention, car sinon une situation du blocage avec J_1 sur la machine M_2 et J_2 sur M_3 est atteinte. Par conséquent, la progression du job J_2 de M_4 à M_3 est retardée jusqu'à l'instant $t = 13$ qui correspond à la fin de l'exécution du job J_1. La nouvelle valeur du makespan, est égale à $C_{max} = 26$. Si nous considérons une approche intégrée, la solution optimale est :

$$S_{11} = 0, S_{12} = 4, S_{13} = 5, S_{21} = 0, S_{22} = 2, S_{23} = 3, S_{24} = 4.$$

Le makespan de cette solution est égal à $C_{max} = 15$.

Cet exemple montre la supériorité des approches intégrées par rapport aux approches hiérarchiques. C'est pourquoi, plusieurs auteurs se sont concentrés leur étude sur les approches résolvant le problème du blocage durant la construction des ordonnancements. L'objectif de la section suivante est de présenter les différentes approches intégrées développées.

2.6.2. Approches intégrées

Dans ce type d'approches, les situations de blocage sont prises en compte durant la construction des ordonnancements. Ramaswamy and Joshi [1996] ont considéré une cellule de production dans laquelle un ensemble de trois machines sont arrangées autour d'un robot qui permet de charger/décharger et de transporter les produits d'une machine à l'autre. Des modèles de programmation linéaire en variables mixtes ont été proposés pour modéliser les différents systèmes suivants : (i) avec des stocks tampons de capacité illimitée et sans moyen de transport. Dans ce cas, le problème est équivalent à l'ordonnancement d'un job shop ; (ii) sans stock tampon et sans moyen de transport ; (iii) sans stock tampon et avec prise en compte du robot ; (iv) avec des stocks tampons de capacité 1 et sans moyen de transport. Dans ces modélisations, les stocks tampons et les robots sont représentés comme des machines. Les problèmes sont résolus par un logiciel de programmation linéaire, et les résultats obtenus pour des problèmes de taille modeste montrent que ces modèles ne sont pas exploitables pour des cas pratiques.

Lee et DiCesare [1994] ont étudié des systèmes avec présence de véhicules filoguidés et flexibilité sur les opérations, c'est-à-dire qu'une opération peut être exécutée par plusieurs machines. Un modèle de réseaux de Petri intégrant les déplacements des *AGVs* et le routage pour les produits est proposé. Basée sur le graphe de marquage d'un modèle en réseaux de

Petri, une méthode de recherche aveugle appelée algorithme A^* (Pearl [1984]) est développée. Cependant, la fonction heuristique proposée pour évaluer un nœud du graphe de marquage dans l'algorithme A^* ne garantit pas l'optimalité de l'ordonnancement obtenu.

Sun et al. [1994] présente une approche similaire à celle proposée par Lee et DiCesare [1994]. Le système étudié est composé de plusieurs machines et d'un ensemble d'*AGVs*, et un job est composé d'une séquence d'opérations qui doivent être réalisées selon un ordre donné à priori. Les chemins utilisés par un *AGV* pour transporter un produit d'une machine à l'autre sont connus. Dans un premier temps, le système est modélisé au moyen des réseaux de Petri. Ensuite, une heuristique de recherche dans le graphe de marquage est développée. Cette heuristique est basée sur une modification de l'algorithme A^*.

Xiong et Zhou [1998] ont considéré un problème d'ordonnancement dans un système de production de semi-conducteurs. Le système correspond à un job shop multi-ressources dans lequel une opération peut nécessiter plusieurs ressources en même temps. Les réseaux de Petri sont utilisés pour modéliser le comportement de système. Pour résoudre le problème, deux heuristiques basées sur l'algorithme A^* sont proposées et utilisent le graphe du marquage de réseau de Petri obtenu. Les performances des deux heuristiques sont comparées sur des exemples pratiques. Une autre variante de l'algorithme A^* utilisant une forme quadratique pour la fonction heuristique d'évaluation des nœuds a été proposée par Jeng et al. [1996]. Cette variante est appliquée à l'ordonnancement d'un système de production contenant des opérations d'assemblage.

Dans les travaux de Damasceno et Xie [1998, 1999], Damasceno [1999], un problème d'ordonnancement plus général appelé job shop multi-ressources avec blocage est étudié. Ce problème généralise le modèle de *MPT* job shop (Drozdowski [1996]) à travers l'ajout de la contrainte *retenir et attendre* relative à la libération des ressources après la fin d'une opération. La présence de cette contrainte additionnelle conduit à l'existence de situations de blocage. Deux modèles sont considérés: le cas où toutes les ressources sont disponibles avec une seule unité et le cas où chaque ressource est disponible avec une quantité fixée. Pour le premier modèle, une heuristique gloutonne qui ordonnance les jobs un après l'autre est proposée. Cette heuristique est basée sur un algorithme de programmation dynamique où les situations de blocage sont vérifiées par l'utilisation des siphons dans le réseau de Petri. Cette heuristique est testée sur les exemples de Ramaswamy and Joshi [1996] et sur de nouvelles instances générées aléatoirement. Dans le cas où les ressources sont disponibles en plusieurs unités, la programmation dynamique proposée pour le premier modèle est généralisée de manière à prendre en compte les caractéristiques du nouveau modèle. Cette extension est ensuite utilisée pour développer l'heuristique qui traite les jobs un après l'autre. Celle-ci opère en essayant d'insérer les opérations au plus tôt, tout en assurant l'absence de blocage. L'heuristique obtenue est testée sur de nouvelles instances générées aléatoirement.

3. Méta-heuristiques

Comme nous l'avons souligné précédemment, la plupart des problèmes d'ordonnancement sont NP-difficiles, et il est difficile voire impossible de trouver des méthodes exactes qui soient efficaces en terme de temps d'exécution. C'est pourquoi les approches proposées et décrites dans les sections précédentes sont basées sur l'utilisation des méthodes heuristiques. Ces méthodes ont pour but de résoudre les problèmes difficiles de manière approchée mais en un temps acceptable.

Nous présentons dans cette section, une classe d'heuristiques appelées méta-heuristiques, qui ont reçu une grande attention des chercheurs durant les 20 dernières années. Parmi les techniques qui ont prouvé leur efficacité dans la résolution de problème d'optimisation combinatoire, nous pouvons citer les *algorithmes génétiques* (Holland [1975]), la *recherche dispersée* (Glover [1977]), le *recuit simulé* (Kirkpatrick [1983]), la *recherche tabou* (Glover [1986]) et la *colonie des fourmis* (Colorni et al. [1992]). Le succès de ces méthodes tient du fait qu'elles permettent d'intégrer les différentes contraintes pratiques, sont faciles à implémenter, et proposent des solutions de bonne qualité. Dans ce qui suit, nous décrivons en détail deux méta-heuristiques utilisées dans nos travaux de recherche : la recherche tabou et les algorithmes génétiques.

3.1. Recherche tabou

L'idée de la recherche tabou a été proposée dans les années soixante dix par Glover [1977], mais la forme et le terme de la recherche tabou utilisée actuellement ont été proposés pour la première fois par Glover [1986]. L'idée de base a également été esquissée par Hansen [1986].

3.1.1. Utilisation de la mémoire

Une méta-heuristique peut se définir comme un assemblage de plusieurs heuristiques simples. Parmi les méta-heuristiques utilisées pour résoudre les problèmes d'optimisation, la recherche tabou est la seule qui a été présentée comme fonctionnant avec une mémoire, ou plus exactement un ensemble de mémoires (Glover [1997]). Les structures de mémoires dans la recherche tabou sont basées sur quatre principes : le passé récent, les fréquences, la qualité et l'influence.

La mémoire basée sur un passé récent (*Recency Based Memory RBM*) représente la manière d'identifier les attributs tabous et de déterminer l'état tabou d'une solution contenant ces attributs. La mémoire basée sur les fréquences, qui fournissent un type d'information complémentaire à l'information donnée par la mémoire *RBM*, consiste à enregistrer les fréquences d'apparition des mouvements, afin d'aider à la sélection des bons mouvements.

Le troisième principe qu'est la qualité, réfère à la capacité de différencier les solutions visitées durant la recherche. Dans ce contexte, la mémoire peut être utilisée pour identifier les éléments qui sont communs aux bonnes solutions (Rochat et Taillard [1995]). D'une manière opérationnelle, la qualité devient un fondement pour l'apprentissage dans la recherche tabou, dans le sens où des incitations permettent d'encourager les actions (mouvements) qui conduisent aux bonnes solutions (stratégies d'intensification), et des pénalités découragent les actions qui conduisent aux mauvaises solutions (stratégies de diversification).

Enfin, le principe de l'influence permet de considérer les impacts des choix faits durant la recherche. L'enregistrement des informations sur l'influence des choix des solutions (mouvements) donne à la recherche tabou un autre niveau d'apprentissage (critère d'aspiration par influence, Glover et al. [1993]).

Deux structures de mémoires sont utilisées dans la recherche tabou, la mémoire explicite et la mémoire attributive. La première enregistre des solutions complètes, plus précisément une élite de solutions visitées durant la recherche. Cette mémoire est souvent utilisée pour implémenter les stratégies d'intensification du long terme de la recherche avec tabous. La deuxième mémoire (attributive) enregistre seulement les informations portant sur des attributs de solutions qui changent d'une solution à une autre. Par exemple, pour le problème de la recherche d'un arbre de poids minimum, les attributs peuvent représenter les arêtes ajoutées, les arêtes enlevées, etc. Pour le problème du voyageur de commerce, l'indice d'une ville peut être utilisé comme attribut.

3.1.2. Fondements de la recherche tabou

La recherche tabou (*TS*) est une méta-heuristique qui guide les procédures de recherche locale dans l'exploration de l'espace des solutions, en allant au-delà de l'optimum local. C'est une méthode qui procède par améliorations successives. La recherche tabou tire son origine dans l'algorithme de descente. Cet algorithme est une heuristique où à chaque étape est exploré un échantillon du voisinage V^* d'une solution courante s (V^* peut être égal au voisinage tout entier ou bien seulement à une partie). La meilleure solution s' du voisinage est retenue si $f(s') < f(s)$, dans le cas d'un problème de minimisation d'une fonction f. Si aucune solution meilleure s' n'est trouvée, alors le test d'arrêt de l'exploration est activé.

L'inconvénient de la méthode de descente est la possibilité d'aboutir à une solution (optimum local) loin de l'optimum global. Pour éviter d'être « piégé » par un optimum local, (*TS*) peut accepter une solution s' obtenue à partir de s même si $f(s') > f(s)$ (toujours pour un problème de minimisation). Cependant, ceci augmente le risque de cyclage lors de l'exploration. Par conséquent, une liste *TL* des solutions déjà visitées, nommée *liste tabou* est construite afin d'empêcher certaines modifications.

Les bases de la recherche tabou se présentent comme suit :

Etant donnée une fonction f à optimiser sur l'ensemble X, (*TS*) commence comme toute procédure de recherche locale par construire une solution initiale, qui peut être de mauvaise qualité, puis progresse itérativement d'une solution à une autre jusqu'à ce qu'un critère d'arrêt soit satisfait. A chaque solution $x \in X$, on associe un voisinage $N(x) \subset X$, et une solution $x' \in N(x)$ obtenue depuis x, par une opération appelée *mouvement*.

TS va au-delà de l'optimum local en modifiant sa stratégie (en ce voisinage) et en cherchant dans un autre voisinage $N^*(x)$. Ceci est réalisé à l'aide d'une structure de mémoire spéciale appelée *"Liste tabou (TL)"* qui détermine $N^*(x)$ et organise la manière dont l'espace devra être exploré. Une solution de $N^*(x)$ est admise dans la structure mémoire si elle a déjà été rencontrée durant la recherche, en l'empêchant ainsi d'appartenir à $N^*(x)$ durant la ou les itérations suivantes, en la classifiant *tabou.*

3.1.3. Paramètres de la méthode

La recherche tabou modifie localement une solution de manière itérative, tout en gardant une trace de ces modifications pendant un certain laps de temps, afin d'éviter de réaliser les modifications inverses susceptibles de mener à des solutions déjà visitées (phénomène de cyclage). Ces modifications définissent une restriction appelée *restriction tabou* qui permet de dire si la modification actuelle est permise ou interdite. La restriction tabou représente donc certaines conditions qui permettent de décider de l'admissibilité d'une solution $x' \in N(x)$.

Les modifications classées interdites par la restriction tabou sont mémorisées dans des listes qui interdisent l'usage de certaines modifications pour les itérations futures, d'où l'appellation de *liste tabou*. La durée (représentée par le nombre d'itérations) pendant laquelle ces modifications restent interdites (tabous) est appelée *taille de la liste* ou *tenure* (Glover et al. [1993]). Les listes tabous peuvent être explicites ou attributives suivant les restrictions utilisées. Le choix d'un type convenable pour la liste tabou dépend du problème traité. Il a été montré de manière empirique que la taille de liste donnant de bons résultats augmentait avec la taille du problème. Donner une taille assez petite à la liste (*TL*) augmente la possibilité de cyclage, alors que lui donner une taille assez grande diminue la qualité de la solution.

Le dernier paramètre caractérisant la recherche tabou est donné par les *conditions de niveaux d'aspiration*. Ce sont des critères qui permettent d'accepter une solution (modification) tabou si elle est meilleure que la meilleure solution connue jusqu'à là. La méthode devient ainsi plus flexible et l'état tabou d'une solution (modification) n'est pas absolu. Par conséquent, le critère d'aspiration fournit un seuil d'attractivité qui permet de dire si la solution obtenue est admissible ou non. Il existe deux types d'aspiration, l'*aspiration de déplacement* (*move aspiration*) et l'*aspiration attributive* (*attribut aspiration*). Le premier type élimine la classification tabou d'un mouvement, alors que le second n'élimine que l'état tabou actif d'un attribut. Pour plus de détails sur ces paramètres, le lecteur pourra se référer aux articles de Glover [1989, 1990], Glover et Laguna [1997].

Les paramètres cités sont considérés de deux façons différentes caractérisées par deux types de mémoires : la *mémoire à court terme* (*short term memory STM*) et la *mémoire à long terme* (*long term memory LTM*). La première (*STM*) est la plus utilisée dans l'algorithme (*TS*) : elle garde la trace des attributs des solutions déjà visitées dans un passé récent. Le mécanisme de cette mémoire ainsi que des exemples illustratifs sont donnés dans Glover [1990] et Glover [1995].

Un grand nombre d'implémentations de la recherche tabou incorporant seulement la mémoire à court terme (*STM*) suffisent à donner des solutions de haute qualité à de nombreux problèmes. Pour d'autres, les composantes de la mémoire à long terme sont nécessaires. Contrairement à (*STM*), cette mémoire consiste à enregistrer les informations portant sur des attributs des solutions visitées dans un passé lointain.

Deux principales composantes de la mémoire à long terme sont les stratégies d'intensification et les stratégies de diversification. Ces stratégies sont implémentées en utilisant des mémoires explicites ou attributives permettant d'enregistrer les fréquences d'apparition des solutions (mouvements). La première stratégie est basée sur la modification des règles de choix, par la favorisation des combinaisons de mouvements jugés bons (Glover [1995]). La seconde,

comme son nom l'indique, consiste à générer des solutions qui se composent d'attributs remarquablement différents des précédents. En d'autres termes, une stratégie de diversification est désignée pour guider la recherche vers de nouvelles régions de l'espace des solutions. Les approches de diversification utilisées dans la recherche tabou sont détaillées dans Glover [1995].

3.2. Les algorithmes génétiques

Contrairement à la recherche tabou qui progresse itérativement d'une solution à l'autre, les algorithmes génétiques sont des méthodes de populations qui agissent sur un ensemble de solutions. Les algorithmes génétiques ont été proposés pour la première fois par Holland [1976], qui s'est inspiré de la reproduction naturelle pour développer une méthode de recherche robuste pouvant être appliquée aux problèmes complexes d'optimisation combinatoire. Cette méthode peut être vue comme une méthode avec mémoire pour laquelle la mémoire est constituée par la population de solutions (Taillard et al. [2001]).

La première étape de tout algorithme génétique est la génération d'une *population initiale*. Cette population peut être générée aléatoirement ou au moyen d'une heuristique. Le but est de construire des solutions régulièrement réparties dans l'espace de solutions. A chaque génération, l'algorithme génétique essaye d'améliorer un ensemble de solutions issues de la population précédente en tentant de garder leurs meilleures caractéristiques, et ceci afin d'obtenir des populations de solutions ayant une qualité croissante.

Dans un premier temps, une représentation génétique ou *codage* des solutions du problème doit être déterminée. L'un des codages les plus utilisés dans la littérature est le codage binaire Goldberg [1989]. Cependant, des applications récentes des algorithmes génétiques utilisent d'autres codages, ainsi à titre d'exemple, une permutation de valeurs entières est utilisée pour le problème du voyageur de commerce (Oliver et al. [1987]). Chaque individu se voit attribuer une fonction mérite *fitness* qui permet de mesurer la force et l'importance d'un individu.

A chaque génération, le *critère de sélection* permet de prendre des individus dans la population courante avec des probabilités proportionnelles à la force des individus. Les individus sélectionnés appelés *parents* sont ensuite utilisés pour générer des individus *enfants* de la population suivante. Ces individus sont obtenus en appliquant aux parents un opérateur de *croisement* qui permet de combiner deux individus avec l'espoir de transmettre leurs meilleures caractéristiques à au mois un des *enfants*. Plusieurs croisements existent dans la littérature dont le plus simple est le croisement en un point présenté par Goldberg [1989].

Enfin, l'opérateur de *mutation* qui consiste à introduire des petites modifications sur la population en changeant une ou plusieurs caractéristiques d'un chromosome (c'est-à-dire une solution) aléatoirement choisi, est appliqué. Le but de cet opérateur est d'assurer qu'aucun espace de solutions n'a une probabilité nulle d'être visité.

4. Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les problèmes d'ordonnancement des systèmes de production. En particulier, nous nous sommes consacrés aux travaux liés aux problèmes étudiés dans cette thèse, dans le sens où ils considèrent les mêmes hypothèses de travail.

Il est important de souligner que contrairement aux modèles d'ordonnancement classiques pour lesquels un nombre important de travaux ont été réalisés, les modèles prenant en compte la variété des ressources engagées dans la production ne sont pas abondamment étudiés. Nous avons cité deux problèmes d'ordonnancement avec la présence d'une part des contraintes de blocage et d'autre part, des contraintes de flexibilités sur les ressources. Ces deux contraintes sont très réalistes dans les systèmes de production automatisés, et constituent les contraintes principales des problèmes d'ordonnancement présentés dans cette thèse.

Dans la suite du manuscrit, nous adopterons des approches intégrées pour la prise en compte des contraintes de blocage et de flexibilité pour l'exécution des opérations. Une des motivations qui nous a amenée à adopter des approches intégrées est qu'elles permettent de prendre en compte simultanément toutes les caractéristiques du modèle. En effet, les approches qui traitent le problème de l'ordonnancement indépendamment de la contrainte de blocage ne permettent pas de réduire la complexité.

Première Partie

Les Problèmes d'Ordonnancement sans Contraintes de Flexibilité

Dans cette partie, nous nous intéressons à la résolution des problèmes d'ordonnancement de type job shop multi-ressources avec la prise en compte du blocage, et ceci sans tenir compte de la flexibilité sur l'exécution des opérations. Dans un premier temps, nous étudions le problème dans le cas de la présence des ressources multiples. Pour ce modèle, nous proposons des algorithmes polynomiaux pour des cas particuliers de deux jobs, et nous développons une méthode heuristique pour le cas général d'un nombre quelconque de jobs. Dans un second temps, nous nous concentrons sur le cas où toutes les ressources sont unitaires. Pour ce dernier modèle, nous proposons une heuristique de recherche locale ainsi que des modélisations mathématiques du problème de contrôle et d'ordonnancement.

Chapitre 2

Job Shop Multi-Ressources avec Prise en Compte du Blocage : Cas des Ressources Multiples

Dans ce chapitre, nous présentons une méthode heuristique originale pour l'ordonnancement et le contrôle des systèmes de production fabriquant des produits dont les caractéristiques sont les suivantes. Tout d'abord, la gamme opératoire de chaque produit est linéaire, et chaque opération peut nécessiter la présence simultanée de plusieurs ressources. Chaque ressource peut être disponible en plusieurs unités. De plus, les ressources nécessaires pour une opération ne peuvent pas être libérées jusqu'au début de l'opération suivante de la gamme. Cette dernière caractéristique est connue dans la littérature sous le nom de la propriété de *retenir et attendre* ou aussi *blocking*. La méthode heuristique proposée est basée sur un nouvel algorithme polynomial pour résoudre le cas de deux jobs, et utilise une méthodologie originale pour le cas général d'un nombre quelconque de jobs. La méthode est améliorée avec une recherche tabou. Les expériences réalisées montrent que cette méthode est très compétitive par rapport aux approches existantes.

Les différents résultats apparaissant dans ce chapitre ont été présentés dans quelques conférences internationales: MCPL-00 (2 nd Conference on Management and Control of Production and Logistics) [2000] à Grenoble, France, IEPM-01 (International Conference on Industrial Engineering and Production Management), au Québec, Canada, MIC-01 (4th Metaheuristics International Conference) [2001], à Porto, Portugal. Ils ont fait l'objet d'un article publié dans IEEE Transactions on ROBOTICS and AUTOMATION.

1. Introduction

Les nouvelles technologies introduites dans les systèmes de production modernes rendent les applications des modèles classiques d'ordonnancement très difficiles. Afin de mettre en œuvre un modèle d'ordonnancement qui colle à la réalité des ateliers flexibles de production (*FMSs*), certains aspects issus du terrain doivent être ajoutés aux modèles d'ordonnancement classiques. L'un des principaux aspects que nous étudions dans ce chapitre concerne la libération des ressources après la fin d'une opération, appelé propriété *retenir et attendre*.

Nous avons vu dans le chapitre 1, que la présence de cette propriété dans les *FMSs* conduit à des problèmes de blocage, et qu'il existe deux types d'approches pour résoudre les problèmes d'ordonnancement. Dans le premier type d'approches appelées *approches hiérarchiques*, le problème d'évitement de blocage et le problème de séquencement sont résolus séparément. Les approches du deuxième type appelées *approches intégrées* résolvent en même temps les deux problèmes. Dans ce chapitre, nous proposons une méthode heuristique intégrée permettant de tenir compte des situations de blocage durant la construction des ordonnancements.

Dans la deuxième section, nous donnerons une description du problème de *job shop multi-ressources avec blocage* avec une formulation mathématique du problème d'ordonnancement dans la section 3. En se basant sur des exemples illustratifs, nous montrons dans la section 4 que le modèle proposé est un modèle générique pour la représentation et l'ordonnancement des systèmes industriels. Dans la section 5, nous étudions un cas particulier du job shop multi-ressources avec blocage : le cas de deux jobs. Tout d'abord, nous commençons par donner une nouvelle caractérisation des situations de blocage. Ensuite, nous donnerons une nouvelle généralisation de l'approche géométrique pour résoudre le problème de minimisation du makespan. Nous généralisons le résultat pour tout critère objectif régulier. Dans la section 6, nous étudions le cas général du job shop multi-ressources avec blocage avec un nombre quelconque de jobs. Tout d'abord, nous présenterons la nouvelle méthodologie utilisée pour résoudre le problème d'ordonnancement, et nous montrerons que cette méthodologie change le problème de la caractérisation du blocage. Partant d'une nouvelle caractérisation du blocage, nous généraliserons l'algorithme polynomial pour proposer une heuristique gloutonne, qui est couplée dans la section 7 à la recherche tabou pour donner l'heuristique finale permettant de résoudre le problème d'ordonnancement étudié. La section 8 présente des résultats numériques obtenus sur des instances issues de la littérature. Dans la section 9, nous examinerons l'extension de l'heuristique proposée pour optimiser d'autres critères objectifs.

2. Description du problème

Dans cette section, nous introduisons le modèle d'ordonnancement de type *Job Shop Multi-ressources avec Blocage* (*JSMB*). Ce modèle est introduit pour fournir un modèle générique pour modéliser les systèmes de production automatisés.

Le problème *JSMB* est défini de la manière suivante. Un ensemble de ressources $\mathcal{R}=\{R_1, R_2,..., R_r, ..., R_m\}$ où m est le nombre de types de ressources, est disponible pour réaliser un ensemble de N jobs $\mathcal{J} = \{J_1, J_2,..., J_i,..., J_N\}$. Chaque ressource R_r est disponible en Q_r exemplaires. La réalisation du job J_i nécessite une séquence linéaire d'opérations (aussi

appelée *gamme opératoire*) $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$ qui doivent être réalisées suivant un ordre défini par le processus de fabrication, où n_i est le nombre d'opérations du job J_i. Une opération O_{ik} nécessite une durée opératoire p_{ik} connue à l'avance, et les temps de préparation entre les opérations sont, soit négligeables, soit inclus dans les durées opératoires (indépendants de la séquence sur la ressource). La préemption des opérations n'est pas autorisée. Cela signifie qu'une opération ne peut pas être interrompue une fois qu'elle est démarrée. De plus, une opération O_{ik} peut avoir besoin de plusieurs ressources $R_{ik} \subset \mathfrak{R}$ pour être réalisée (*multi-ressources*), avec plusieurs unités B_{ik}^r de chaque ressource $R_r \in R_{ik}$ (*multiple unités*).

La dernière caractéristique du modèle concerne la libération des ressources après la fin d'une opération, appelée propriété *retenir et attendre* (Mati et al. [2001b, 2001c]) ou *blocking* (Hall et Sriskandarajah [1996]). Cette propriété signifie qu'après la fin d'une opération O_{ik} d'un job J_i, les ressources dans R_{ik} sont encore retenues par le job J_i jusqu'à ce que les ressources nécessaires pour l'opération suivante sur sa gamme deviennent toutes disponibles.

Dans ce chapitre, nous adoptons la forme suivante pour représenter la gamme opératoire d'un job générique $\{(R_{i1},B_{i1},p_{i1}),(R_{i2},B_{i2},p_{i2}),\ldots,(R_{in_i},B_{in_i},p_{in_i})\}$ où B_{ik} est le vecteur $[B_{ik}^1,\ldots,B_{ik}^r,\ldots,B_{ik}^m]^t$ des besoins en ressource B_{ik}^r de l'opération O_{ik}. Dans la suite, par souci de simplification de l'écriture, nous utilisons aussi une forme plus compacte pour faciliter la représentation de la gamme opératoire d'un job. Par exemple, $J_i = \{(2M_1, M_2, 2), (2M_2, 3)\}$ signifie que le job J_i est constitué de deux opérations. La première opération nécessite deux unités de la ressource M_1 et une unité de M_2, et sa durée opératoire est 2. La deuxième opération nécessite deux unités de la ressource M_2, et sa durée opératoire est 3. Sous la forme générique, le job J_i s'écrit $J_i = \{(\{M_1, M_2\}, [2, 1]^t, 2), (\{M_2\}, [0, 2]^t, 3)\}$.

En présence de la contrainte *retenir et attendre*, des situations indésirables peuvent se produire dans le système de production si les ressources ne sont pas bien coordonnées. Ces situations sont connues sous le nom du *blocage* (Coffmann [1971], Cho et al. [1992]). Par exemple, pour une cellule robotisée, une situation du blocage est atteinte si le robot tente de charger un produit sur une machine, qui est elle-même en train de travailler sur un autre produit. Dans cette situation, le robot ne peut pas charger le produit sur la machine car elle n'est pas disponible, et la machine a besoin du robot pour décharger son produit.

Le problème d'ordonnancement d'un *JSMB* consiste alors à déterminer la meilleure séquence d'entrée σ_r des opérations dans chaque ressource $R_r \in \mathfrak{R}$ et les dates de début S_{ik} pour toutes les opérations afin que le comportement du système soit sans blocage, tout en minimisant une fonction objectif donnée.

On remarque que le problème d'ordonnancement est NP-difficile car il est une généralisation du problème de job shop classique. En revanche, à cause des contraintes additionnelles de notre modèle, en particulier la contrainte de *retenir et attendre*, le problème *JSMB* est plus difficile à résoudre. Selon la notation utilisée dans Brucker [1998], décrivant les problèmes d'ordonnancement en trois champs, notre problème du job shop multi-ressources avec la prise en compte du blocage sera noté *JMPT | blocking | f*.

3. Formulation Mathématique

Nous considérons dans cette formulation la minimisation de la durée totale de l'ordonnancement C_{max} (*makespan*). Comme nous l'avons cité précédemment, le problème d'ordonnancement de *JSMB* consiste d'une part, à déterminer quand exécuter les opérations d'un produit J_j, c'est-à-dire., déterminer les dates de début S_{ik} des opérations O_{ik}, et d'autre part à fixer une séquence d'entrée sur chaque ressource afin de minimiser le makespan et éviter les blocages. Le problème à résoudre s'écrit:

$$
(P)\quad \begin{cases}
Min \quad C_{\max} & & \\
\text{sous les contraintes :} & & \\
S_{i,k+1} - S_{ik} \geq p_{ik} & \forall\, 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i - 1 & (1) \\
C_{\max} \geq S_{in_i} + p_{in_i} & \forall\, 1 \leq i \leq N & (2) \\
\sum_{(i,k)} B_{ik}^{r}\, 1\{S_{ik} \leq t < S_{i,k+1}\} \leq Q_r & \forall \textit{ la période } t\,, \forall r & (3) \\
\text{les séquences d'entrée } \sigma_r \text{ ne conduisent pas au blocage} & & (4) \\
S_{ik} \geq 0 & \forall\, 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i &
\end{cases}
$$

Les contraintes (1) assurent la relation de précédence entre les opérations d'un job. Les contraintes (2) définissent la date de fin de l'exécution d'un job. Les contraintes de capacité de ressources sont exprimées par les contraintes (3). En effet, à cause de la contrainte de *retenir et attendre*, à chaque période de temps t, chaque job J_i contenant une opération O_{ik} tel que $S_{ik} \leq t < S_{i,k+1}$, utilise soit :

i) B_{ik}^{r} unités de la ressource R_r pour l'opération O_{ik} si $S_{ik} \leq t < S_{ik} + p_{ik}$.

ii) Soit retient B_{ik}^{r} si $S_{ik} + p_{ik} \leq t < S_{i,k+1}$.

Les contraintes (4) concernant l'absence de blocage doivent assurer à chaque moment, l'absence des cycles de type *retenir et attendre*. De tels cycles se produisent par exemple à l'instant t, si et seulement si, un job J_1 retient la ressource R_1 et attend la ressource R_2, et le job J_2 retient la ressource R_2 et attend la libération de la ressource R_3, ainsi de suite jusqu'au job J_u qui retient la ressource R_u et attend la libération de R_1 pour qu'il puisse progresser.

Dans la figure 2.1, Il est clair qu'aucun job induit dans le cycle ne peut progresser si chaque ressource est disponible avec une seule unité et si la capacité du stock est nulle.

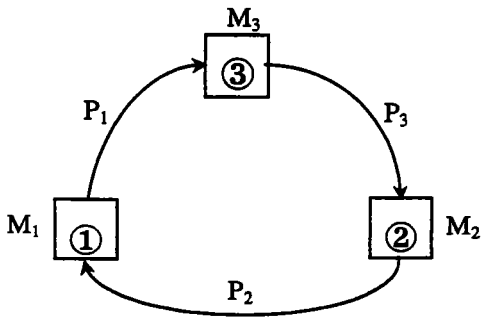


Figure 2.1 : Cycle de type retenir et attendre

Dans le cas général, la contrainte (4) exprimant l'absence de blocages ne peut pas être exprimée par des variables de décision, sans faire de distinction entre les exemplaires de la même ressource. Nous verrons dans le chapitre 3, que l'absence de blocage est décrit explicitement par des relations mathématiques pour un cas particulier pour lequel chaque ressource est disponible avec un seul exemplaire.

Remarque 2.1 : Les contraintes (1)-(3) n'impliquent pas l'absence de blocage. Pour illustrer cette remarque, nous considérons un exemple de deux jobs qui doivent être réalisés sur deux types de ressources, chacune disponible avec une seule unité. Les gammes opératoires des jobs sont les suivantes :

$$J_1 = \{(M_1, 1), (M_2, 1)\}$$
$$J_2 = \{(M_2, 1), (M_1, 1)\}$$

L'ordonnancement $\{S_{11} = 0, S_{12} = 1, S_{21} = 0, S_{22} = 1\}$ satisfait les contraintes de précédence et de capacité de ressources, c'est-à-dire que les contraintes (1) et (3) sont satisfaites. Cependant, cet ordonnancement conduit à une situation du blocage à l'instant 1. En effet, à l'instant $t = 1$, le job J_1 a besoin de la ressource M_2 avant de libérer la ressource M_1, alors que. le job J_2 a besoin de M_1 pour libérer la ressource M_2. Les deux jobs sont impliqués dans un cycle de type *retenir et attendre.*

4. Modélisation des systèmes industriels

Dans cette section, nous allons montrer que le modèle du job shop multi-ressources avec blocage introduit dans la section précédente, est un modèle générique permettant de représenter les différents problèmes d'ordonnancement avec une grande variété de ressources et de contraintes industrielles. Ainsi, plusieurs problèmes d'ordonnancement proposés dans la littérature peuvent être représentés par le modèle générique *JSMB*.

4.1. Job shop classique

Le job shop classique est défini par un ensemble de machines disponibles pour exécuter un ensemble de jobs, où la gamme opératoire de chaque job J_j est la suivante :

$$J_j = \{(M_{j1}, p_{j1}), (M_{j2}, p_{j2}), \ldots, (M_{j,n_j}, p_{j,n_j})\}$$

Dans ce cas, les seules ressources sont les machines et les capacités des stocks tampons entre les machines sont supposées illimitées. Pour modéliser cette dernière contrainte, nous introduisons une opération fictive O_{jk} après chaque opération, pour laquelle l'ensemble des ressources nécessaires est vide ($R_{jk} = \varnothing$) et le temps opératoire est nul ($p_{jk} = 0$). En utilisant la notation générique de la gamme opératoire, le modèle de *JSMB* équivalent peut être écrit de la manière suivante :

$$J_j = \{(M_{j1}, p_{j1}), (\varnothing, 0), (M_{j2}, p_{j2}), (\varnothing, 0), \ldots, (M_{j,n_j}, p_{j,n_j})\}.$$

4.2. Systèmes multiprocesseurs

Ces systèmes sont des jobs shops mais chaque opération peut nécessiter plusieurs ressources simultanément (Drozdowski [1996]). Dans ce cas, la gamme opératoire d'un job est décrite par :

$$J_j = \{(R_{j1}, p_{j1}), (\varnothing, 0), (R_{j2}, p_{j2}), (\varnothing, 0), \ldots, (R_{j,n_j}, p_{j,n_j})\},$$

où R_{jk} est l'ensemble de ressources nécessaire pour l'opération O_{jk}.

4.3. Systèmes de production avec AGVs

Ces systèmes sont des jobs shops qui utilisent des *AGVs* comme moyens de transport. La gamme opératoire de chaque job est définie par :

$$J_j = \{(AGV, M_{j1}, p_{j1}), (AGV, M_{j2}, p_{j2}), \ldots, (AGV, M_{j,n_j}, p_{j,n_j})\}.$$

L'ensemble des ressources utilisées est composé d'un ensemble de machines et des *AGVs*. Un *AGV* est nécessaire tout au long de la réalisation d'un job, et il est occupé à partir du début de la première opération et n'est libéré qu'à la fin de la dernière opération. Avec le modèle *JSMB*, la gamme opératoire d'un job est :

$$J_j = \{(AGV, M_{j1}, p_{j1}), (AGV, q_{12}), (AGV, M_{j2}, p_{j2}), (AGV, q_{23}), \ldots, (AGV, M_{j,n_j}, p_{j,n_j})\},$$

où q_{ik} est le temps nécessaire pour transporter le job J_j de la machine M_{ji} à la machine suivante sur la gamme M_{jk}.

4.4. Job shop sans encours

Connu aussi sous le nom de job shop avec blocage (Hall et Sriskandarajah [1996]), le job shop sans encours est une extension du job shop, pour lequel il n'existe pas de stock intermédiaire. Dans ce cas, la machine exécutant une opération est retenue jusqu'à ce que la machine suivante de la gamme soit disponible. Ce problème est un cas particulier du *JSMB*, où une opération nécessite seulement une machine pour être réalisée. La gamme opératoire est alors la même, et se note :

$$J_j = \{(M_{j1}, p_{j1}), (M_{j2}, p_{j2}), \ldots, (M_{j,n_j}, p_{j,n_j})\}.$$

4.5. Ligne de production

Les lignes de production sont des modèles de type flow shop, pour lesquelles les machines sont séparées par des stocks tampons de capacité très limitée (Pinedo [1996]). Ces stocks sont utilisés pour réguler la cadence de production. Nous notons par M_1, M_2, ..., M_m les machines et s_1, s_2, ..., s_{n-1} les stocks tampons de capacité c_1, c_2, ..., c_{n-1}.

Avec ces notations, la gamme opératoire d'un job dans le modèle *JSMB* est représentée par :

$$J_j = \{(M_1, p_{j1}), (s_1, 0), (M_2, p_{j2}), (s_2, 0),\ldots, (M_m, p_{jm})\}.$$

Dans cette représentation, le stock s_i est considéré comme une ressource disponible avec une quantité c_i.

4.6. Cellule robotisée

Comme nous l'avons vu dans le chapitre 1, ces cellules sont composées d'un ensemble de machines alimentées par un robot R (Sethi et al. [1992]). Il n'existe pas de zones de stockage et le robot est utilisé pour toute opération de manutention. La gamme opératoire d'un job dans le modèle *JSMB* est décrite par :

$$J_j = \{(R, \Delta_{01}), (M_{j1}, p_{j1}), (R, \Delta_{12}), (M_{j2}, p_{j2}), (R, \Delta_{23}),\ldots, (M_{j,nj}, p_{j,nj})\},$$

où Δ_{ik} est le temps nécessaire au robot pour décharger le produit de la machine M_{ji} et le charger sur la machine suivante M_{jk}.

4.7. Autres applications

D'autres applications peuvent être trouvées dans le secteur industriel (Hall et Sriskandarajah [1996]), ainsi que dans d'autres secteurs comme l'ordonnancement et le contrôle des déplacements des trains (Cordeau et al.[1998]), rencontré par les compagnies ferroviaires. Nous discuterons en détail sur ce dernier problème ainsi que sa modélisation dans le chapitre 3.

5. Le problème JSMB à deux jobs

Comme nous l'avons précisé, le problème de *JSMB* d'un nombre quelconque de jobs est NP-difficile. En revanche, il existe un cas particulier qui peut être résolu de façon optimale ; le cas d'ordonnancement à deux jobs noté *JMPT* | $n = 2$, *blocking* | *f*.

Dans cette section, nous proposons une extension de l'approche géométrique (Akers et Friedman [1955], Szwarc [1960], Hardgrave et Nemhauser [1963]) décrit dans le chapitre 1, pour ordonnancer et contrôler le job shop multi-ressources avec blocage en présence de seulement deux jobs, noté par la suite par *2-JSMB*. Cette extension est une méthode intégrée qui permet de traiter les situations de blocage durant la construction des ordonnancements. Elle diffère de l'approche géométrique classique en deux points : le premier point concerne la

définition des obstacles, et le deuxième point concerne la manière de construire le réseau $N = (V, E, d)$.

5.1. La nature des obstacles

Dans l'approche géométrique classique développée pour le problème du job shop à deux jobs, les obstacles sont définis par les conflits de ressources entre les opérations des deux jobs. De plus, un chemin réalisable dans le plan est composé par les segments horizontaux, verticaux et diagonaux, qui contourne l'intérieur des obstacles.

Afin de résoudre le problème d'évitement des situations de blocage qui apparaissent dans le problème *2-JSMB*, des obstacles additionnels doivent être ajoutés. Ces obstacles ont pour objectif, d'une part, d'empêcher la violation de la capacité des ressources, et d'autre part, d'interdire les chemins partant de l'origine O et arrivant au point final F, de passer par les états du blocage.

Pour ce faire, nous proposons un algorithme de *programmation dynamique en chaînage arrière*. Cet algorithme utilise les variables booléennes $I(k_1, k_2)$, $k_1=1, 2,\ldots, n_1+1$, $k_2=1, 2,\ldots, n_2+1$ définies de la manière suivante :

$$I(k_1,k_2)=\begin{cases} 1, & \text{si les opérations } O_{1k_1} \text{ et } O_{2k_2} \text{ sont en conflit de ressources} \\ & \text{ou si le blocage est inévitable partant de l'état } (k_1,k_2), \\ 0, & \text{sinon.} \end{cases}$$

L'état $I(k_1, k_2)$ signifie que les opérations O_{1k_1} et O_{2k_2} sont en cours de réalisation. Il est à noter que, $I(k_1, k_2)$ détermine non seulement les états de blocage immédiats, mais aussi les états de blocage futurs appelés états de *blocage imminents* (*impending deadlock*). Ce dernier type de blocages représente une situation pour laquelle, un état de blocage sera atteint quel que soit le choix des séquences d'entrée sur les ressources, pour les opérations restantes.

En utilisant ces variables booléennes, tous les obstacles sont déterminés en utilisant l'équation récursive qui s'écrit:

$$I(k_1,k_2)=\begin{cases} 1, & \text{si } B^r_{1k_1} + B^r_{2k_2} > Q_r \text{ pour une ressource r,} \\ I(k_1,k_2+1) \wedge I(k_1+1,k_2), & \text{sinon.} \end{cases} \qquad (5)$$

où,

Q_r : est le nombre de ressources de type R_r,

B^r_{ik} : est le nombre de ressources R_r nécessaires pour l'opération O_{ik},

$\wedge$: l'opérateur binaire de conjonction "et".

Par convention :

- $I(k_1, n_2+1) = 0$, pour tout $k_1=1, 2,\ldots, n_1+1$.
- $I(n_1+1, k_2) = 0$, pour tout $k_2=1, 2,\ldots, n_2+1$.

L'algorithme de programmation dynamique que nous proposons est basé sur l'équation récursive (5), et détermine tous les états $I(k_1, k_2)$ par induction sur $k = k_1 + k_2$. Cet algorithme peut être décrit comme suit:

Algorithme 2.1 : (*Détection du blocage et de conflit des ressources*)

1. *Initialisation* : $I(k_1, n_2+1)=0$, $\forall k_1=1, 2,..., n_1+1$ et $I(n_1+1, k_2)=0$, $\forall k_2=1, 2,..., n_2+1$.
2. ***Pour*** $k = n_1 + n_2$ *jusqu'à* 2 ***Faire***
   Pour *tout* (k_1, k_2) *tel que* $k_1 + k_2 = k$, $k_1 \leq n_1$, $k_2 \leq n_2$ ***Faire***
   Déterminer $I(k_1, k_2)$ *en appliquant l'équation récursive (5)*
   Fin Pour.
   Fin Pour.

Fin Algorithme.

Exemple 2.1 : Pour illustrer l'algorithme précédent, nous considérons l'instance de 2-*JSMB* avec les gammes opératoires suivantes:

$$J_1=\{(R_1, 1), (R_1, R_2, 1), (R_3, 1)\}$$
$$J_2=\{(R_3, R_4, 1), (R_1, 1), (R_3, 1)\}.$$

Toutes les ressources sont disponibles avec une seule unité. Les obstacles donnés par l'algorithme de programmation dynamique sont illustrés par la figure 2.2.

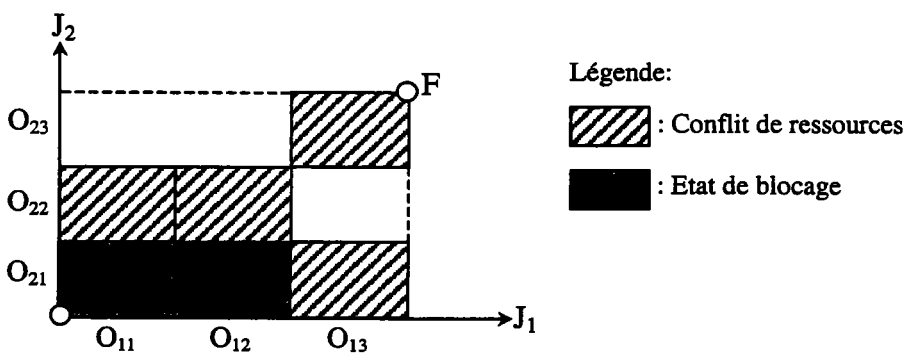


Figure 2.2 : Obstacles construits par la programmation dynamique

Dans la figure 2.2, le rectangle défini par les opérations O_{11} et O_{21} est admissible en terme de capacité de ressources, mais si le système entre dans l'état (O_{11}, O_{21}), un blocage est atteint quel que soit l'ordonnancement des opérations restantes. Cet état est un état de blocage imminent, par conséquent il doit être interdit.

Théorème 2.1

L'algorithme proposé permet de déterminer tous les obstacles relatifs au partage des ressources et aux situations de blocage. De plus, tous les obstacles peuvent être déterminés dans un temps proportionnel à $O(n_1 \times n_2)$.

Preuve

La démonstration est donnée par récurrence sur $k_1 + k_2 = n_1+1 + n_2+1, n_1+1 + n_2, ..., 2$. Il est clair que, la propriété est vraie pour $k_1 + k_2 = n_1+1 + n_2+1$. Supposons que cette propriété est vraie pour tout $k_1 + k_2 \geq k$, c'est-à-dire (k_1, k_2) est un état de blocage ou de partage de ressources si et seulement si $I(k_1, k_2) = 1$, et considérons un état (k_1, k_2) tel que $k_1 + k_2 = k - 1$. Si la capacité des ressources est violée pour l'état (k_1, k_2), alors la propriété est vraie car $I(k_1, k_2) = 1$. Sinon, $I(k_1, k_2) = I(k_1+1, k_2) \wedge I(k_1, k_2+1)$. Puisqu'il existe seulement deux jobs, soit le job J_1 progresse en premier vers son opération suivante, soit c'est J_2 qui progresse en premier. L'état suivant est soit (k_1+1, k_2) ou (k_1, k_2+1). Par conséquent, l'état (k_1, k_2) est un état de blocage si et seulement si, les deux indices $I(k_1+1, k_2)$ et $I(k_1, k_2+1)$ sont des états de blocage ou de partage de ressources. Par réduction, l'état (k_1, k_2) est un état de blocage si et seulement si, $I(k_1+1, k_2) = 1$ et $I(k_1, k_2+1) = 1$, c'est-à-dire si et seulement si $I(k_1, k_2) = 1$. Donc, la propriété est vraie pour tout état (k_1, k_2) tel que $k_1+k_2 = k - 1$ et le théorème est prouvé. En plus, Le nombre d'opérations élémentaires effectuées par l'algorithme 2.1 (dans l'étape 2) est proportionnel à $O(n_1 \times n_2)$. ■

5.2. Construction du réseau

Comme pour l'approche géométrique classique, une solution réalisable du problème *2-JSMB* correspond au plus court chemin partant de l'origine O et arrivant au point final F. Ce chemin est composé de segments horizontaux (seulement une opération de J_1 est réalisée), verticaux (seulement une opération de J_2 est réalisée) et diagonaux (les deux jobs sont réalisés simultanément). La longueur d'un segment diagonal est égale à la projection sur l'un des axes.

A la différence du cas de job shop classique, les passages sur les bordures droite et supérieure de tout obstacle sont interdits à cause de la contrainte *retenir et attendre* (figure 2.3). En effet, sur la bordure supérieure (resp. droite), les ressources relatives à l'opération O_{2k_2} (resp. O_{1k_1}) sont encore retenues puisque le job J_2 (resp. J_1) n'a pas encore commencé l'exécution de l'opération O_{2,k_2+1} (resp. O_{1,k_1+1}).

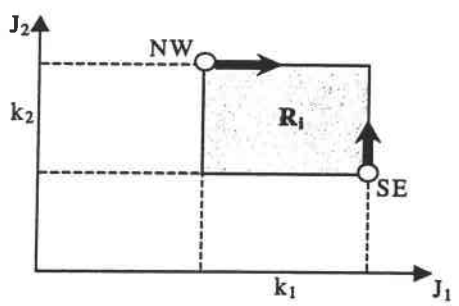


Figure 2.3 : Passages interdits

Comme nous l'avons mentionné dans le Chapitre 1, l'étape principale dans la détermination d'un ordonnancement optimal, en utilisant l'approche géométrique, consiste à déterminer le réseau $N = (V, E, d)$, dans lequel le plus court chemin doit être déterminé. Cette étape transforme la recherche du plus court chemin dans le plan à un problème du plus court chemin sans contraintes dans le réseau N.

Evidemment, la considération de tous les coins des rectangles formés par O_{1k_1} et O_{2k_2} comme des nœuds de $N = (V, E, d)$ garantit l'optimalité du chemin de O à F. Malheureusement, ceci augmente le nombre des nœuds (arcs) du réseau, et rend la détermination du plus court chemin très long. Par conséquent, la détermination des coins suffisants pour déterminer le plus court chemin est cruciale.

De la même manière que dans le cas du job shop classique, les nœuds du réseau correspondent au coins *NW* et *SE* de tous les obstacles. Concernant la définition des successeurs directs d'un nœud, il a été montré dans le cas du job shop classique (Brucker [1988]), que seuls les coins *NW* et *SE* de l'obstacle heurtés diagonalement sont suffisants pour garantir l'optimalité. Ceci n'est plus vrai quand la contrainte de *retenir et attendre* est introduite. Plus précisément, dans le problème *2-JSMB*, le nombre des successeurs d'un nœud v_i peut être supérieure à 2. Dans notre cas, tous les coins *NW* (resp. *SE*) des obstacles pouvant être obtenus de v_i en allant diagonalement et après verticalement (resp. horizontalement), doivent être considérés comme successeurs directs de v_i (figure 2.4). Enfin, l'exploration d'un nœud s'arrête si le passage diagonal heurte un obstacle ou si l'obstacle final défini par l'origine O et le point final F est atteint.

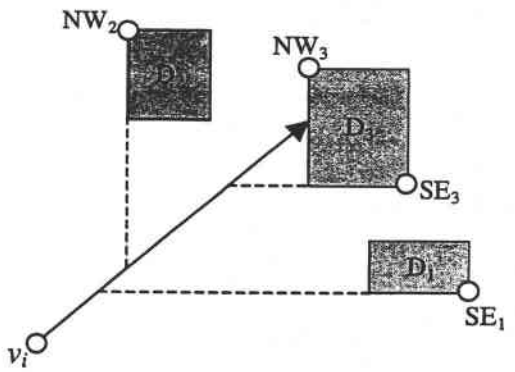


Figure 2.4 : Successeurs directs de v_i

La définition de la longueur $d(v, v')$ de tout arc (v, v') est similaire à celle du cas de job shop classique, c'est-à-dire qu'elle est égale à la projection sur l'axe X (resp. Y) si v' est obtenu de v en allant au début diagonalement et ensuite horizontalement (resp. verticalement). Dans la figure 2.4, SE_1, NW_2, NW_3 et SE_3 sont les successeurs directs de v_i.

La définition suivante récapitule et définie explicitement le réseau $N = (V, E, d)$.

Définition 2.1 : Le réseau $N = (V, E, d)$ pour résoudre le problème *2-MJSB* est défini de la manière suivante:

a) L'ensemble des nœuds est composé de l'origine O, le point final F, et des coins *NW* et *SE* de tous les obstacles (k_1, k_2) tel que $I(k_1, k_2) = 1$,

b) Il existe un arc allant du nœud v_i à un coin *SE* (resp. *NW*) v_j, s'il est possible d'atteindre v_j à partir de v_i en allant au début diagonalement et ensuite horizontalement (resp. verticalement),

c) La longueur de l'arc entre le nœud $v_i = (x_i, y_i)$ et un coin *SE* (resp. *NW*) $v_j = (x_j, y_j)$ est $d(v_i, v_j) = x_j - x_i$ (resp. $d(v_i, v_j) = y_j - y_i$).

Nous montrerons à présent que les nœuds du réseau obtenus par la construction définie précédemment sont suffisants pour déterminer la solution optimale du problème *2-JSMB*. Ceci est donné par le théorème suivant.

Théorème 2.2

Le plus court chemin allant de l'origine O et arrivant au point final F dans le réseau construit $N = (V, E, d)$ correspond au plus court chemin dans le plan avec obstacles définis dans la section 4.1. Par conséquent, il correspond à une solution optimale du problème *2-JSMB*.

Preuve

De la définition du réseau N, tout chemin allant de l'origine O au point final F dans N correspond à un chemin P de O à F dans le plan interdisant l'intérieur, la bordure droite et supérieure de tout obstacle. Le chemin P est composé alors des arcs illustrés par la figure 2.5.

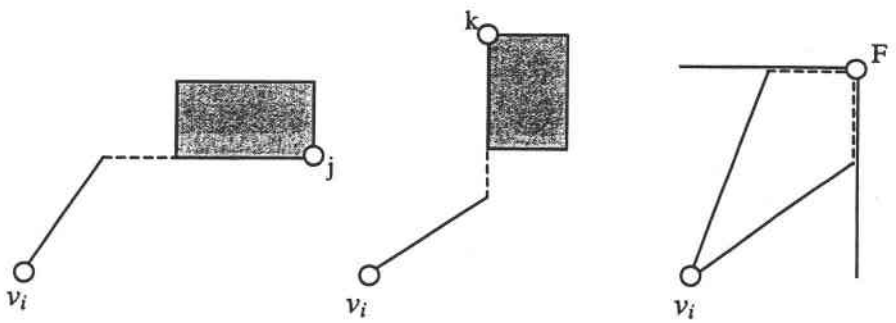


Figure 2.5 : Arcs du réseau N

Dans ce qui suit, nous montrons qu'il existe un plus court chemin dans le plan avec obstacles, qui est composé seulement d'une séquence d'arcs de la figure 2.5 appelés les *arcs du réseau.*

Parmi tous les plus courts chemins dans le plan avec obstacles, nous considérons un chemin P^* ayant un nombre maximum d'arcs du réseau consécutifs en démarrant de O. Si P^* est composé seulement d'arcs du réseau, alors la démonstration est terminée. Sinon, soit v_i le premier nœud à partir duquel l'arc n'est pas un arc de réseau. Avant de continuer, nous examinons d'abord les différentes manières, pour lesquelles un chemin réalisable P contourne les obstacles.

Sur la figure 2.6, il est clair qu'il existe deux manières de contourner un obstacle D, puisqu'un chemin est composé seulement de segments horizontaux, verticaux et diagonaux. La première manière est de croiser l'horizontale l correspondant à la bordure inférieure de D par un point sur la droite du coin SE noté j. La deuxième manière est de croiser la verticale l' sur la gauche de D par un point se trouvant sur le segment supérieur au coin NW noté k. Nous appelons la première manière par *contournement-SE* et la seconde par *contournement-NW.*

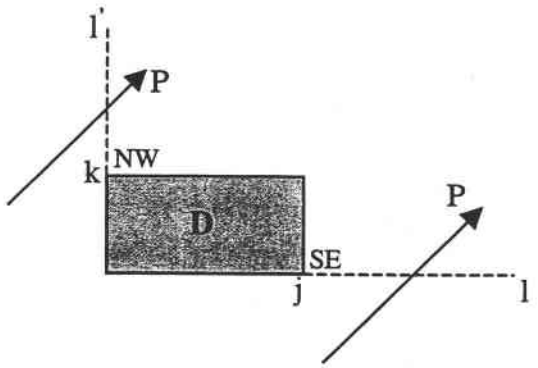


Figure 2.6 : Deux manières de contourner un obstacle

Appelons D_1 l'obstacle heurté en partant diagonalement du nœud v_i. Nous supposons que D_1 est contourné en utilisant un *contournement-SE*. La preuve pour la deuxième manière est similaire et par conséquent est omise. Soit l_1 l'horizontale correspondant à la bordure inférieure de D_1. Cette horizontale est croisée par la diagonale (v_i, s) en un point s_1 et par le chemin P^* en un point t_1 (figure 2.7).

Nous avons alors deux cas de figure qui se présentent :

Cas 1 : le chemin (s_1, t_1) est un segment réalisable évitant tous les obstacles. Dans ce cas, nous pouvons remplacer la partie de P^* entre v_i et t_1 par le segment (v_i, s_1, t_1) sans augmenter la longueur de P^*. Ainsi, un nouveau chemin ayant un arc du réseau en plus (v_i, s_1, SE_1) à partir du nœud v_i est crée. Ceci contredit l'hypothèse de l'optimalité de P^*.

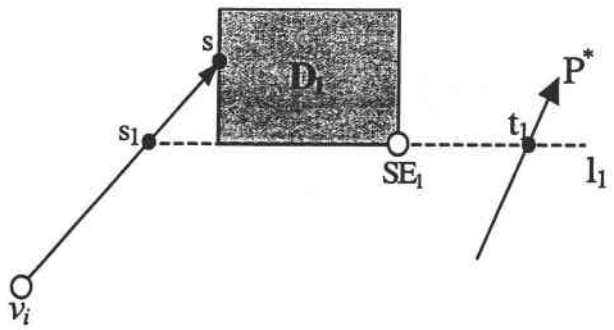


Figure 2.7 : Cas 1

Cas 2 : le chemin (s_1, t_1) n'est pas réalisable. Alors, il existe des obstacles entre s_1 et t_1 (figure 2.8). Soit D_2 le premier obstacle rencontré en allant horizontalement de s_1 à t_1.

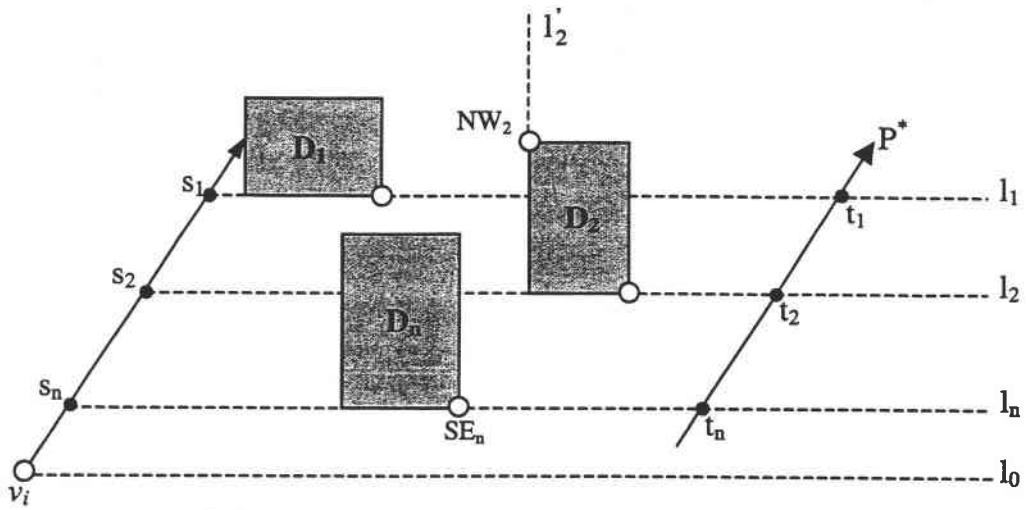


Figure 2.8 : Cas 2

Il est clair que, le chemin P^* contourne D_2 par un *contournement-SE*. Dans le cas contraire, P^* croise la verticale l'_2 en un point sur le segment supérieur de NW_2, qui est au-delà de l'horizontale l_1. Ainsi P^* ne peut pas croiser l_1 au point t_1. Ceci est une contradiction et prouve que P^* contourne D_2 par un *contournement-SE*. Appelons l_2 l'horizontale correspondant à la bordure inférieure de D_2. Il est croisé par la diagonale en un point s_2 et par le chemin P^* au point t_2. Evidemment, la hauteur de l_2 est inférieure à celle de l_1, et supérieure à celle de l_0 qui correspond à v_i car sinon, P^* progresse en dessous de l_2 en démarrant de v_i, ce qui est une contradiction.

Si le segment (s_2, t_2) n'est pas réalisable, alors il existe un obstacle D_3 sur (s_2, t_2). En répétant la démarche ci-dessus, nous retrouvons à la fin un obstacle D_n ayant les propriétés suivantes:

(i) L'horizontale l_n correspondant à la bordure inférieure de D_n, est située entre l_1 et l_0.

(ii) D_n est contourné par un *contournement-SE*.

(iii) Le segment (s_n, t_n) est réalisable.

En remplaçant la partie de P^* située entre v_i et t_n par le segment (v_i, s_n, t_n), nous obtenons un nouveau chemin avec un arc du réseau (v_i, s_n, t_n) en plus, et ceci sans altérer la longueur du chemin. Ceci contredit l'optimalité de P^*.

En résumé, l'hypothèse de l'optimalité de P^* est remise en cause dans tous les cas. Par contradiction, le point v_i sur P^* n'existe pas et P^* est composé seulement par des arcs du réseau. ■

5.3. Complexité du problème 2-JSMB

Dans cette section, nous donnons la complexité de l'algorithme géométrique pour résoudre le problème 2-*JSMB*. Cette complexité dépend de la programmation dynamique développée, de la construction du réseau $N = (V, E, d)$ et de l'algorithme utilisé pour rechercher le plus court chemin.

Le nombre de successeurs d'un nœud particulier v_i dépend des durées opératoires des opérations. Une borne supérieure du nombre des successeurs de v_i est $(n_1 + n_2)$. En plus, il existe au plus $n_1 \times n_2$ obstacles dans le plan. Par conséquent, nous avons le résultat suivant :

Lemme 2.1

La construction du réseau $N = (V, E, d)$ prend un temps proportionnel à $O(n_1 \times n_2 \times (n_1 + n_2))$.

En utilisant le théorème 2.1 et le lemme 2.1, le théorème 2.3 donne la complexité dans le pire des cas de l'algorithme géométrique pour le problème du 2-*JSMB*.

Théorème 2.3

Le problème d'ordonnancement $JMPT \mid n = 2, blocking \mid C_{max}$ est polynomial et sa complexité est $O(n_1 \times n_2 \times (n_1 + n_2))$.

Preuve

La détermination du conflit de ressources et des situations de blocage en utilisant la programmation dynamique prend un temps $O(n_1 \times n_2)$. Le réseau $N = (V, E, d)$ peut être déterminé en $O(n_1 \times n_2 \times (n_1 + n_2))$. La détermination d'un plus court chemin dans un réseau acyclique prend un temps proportionnel à $O(|E|+|V|)$. Dans le pire des cas, $|V| = n_1 \times n_2$ et $|E| = |V| \times (n_1 + n_2)$. Ce qui signifie que, la complexité de l'algorithme est $O(n_1 \times n_2 \times (n_1+n_2))$. ■

Remarque 2.2 : Le théorème 2.3 donne la complexité dans le pire de cas. Des améliorations sont envisageables dans la construction du réseau N. Par exemple, du fait que les bordures droite et supérieure d'un obstacle sont interdites, tous les obstacles adjacents peuvent être considérés comme un seul obstacle, et ainsi le nombre de nœuds (arcs) du réseau peut être réduit.

5.4. Cas d'une fonction objectif régulière

Nous considérons dans cette section la minimisation d'un critère régulier arbitraire $f(C_1, C_2)$, où C_i est la date de fin de la dernière opération du job J_i. Le théorème 2.2 sur l'optimalité du réseau reste vrai. La seule différence concerne le calcul de la valeur du critère. Comme dans (page 184 de Brucker [1998]), nous avons besoin de considérer tous les arcs (x, y) dans le réseau construit N pour lesquels le chemin correspondant à P_{xy} heurte en un point u, l'obstacle final défini par l'origine O et le point final F.

Soient :

$d(O, x)$ la longueur du plus court chemin allant de O à x,

$d(x, u)$, $d(u, y)$, $d(y, F)$ la longueur du plus court chemin allant de x à u, de u à y et de y à F, respectivement.

La valeur de la fonction objectif de l'ordonnancement correspondant au chemin de l'origine O au point final F est $f(C_1, C_2)$ avec,

$C_1 = d(O, x) + d(x, u)$ et $C_2 = C_1 + d(u, y) + d(y, F)$, si u est sur la bordure droite.
$C_2 = d(O, x) + d(x, u)$ et $C_1 = C_2 + d(u, y) + d(y, F)$, si u est sur la bordure supérieure.

Dans le pire de cas, il existe $n_1 \times n_2$ arcs de ce type, et le calcul de C_1 et C_2 est proportionnel à $O(\max(n_1, n_2))$. Par conséquent, nous conclurons que :

Théorème 2.4

Le problème d'ordonnancement *JMPT* / $n = 2$, *blocking* / f est polynomial pour tout critère régulier et sa complexité est au plus $O(n_1 \times n_2 \times (n_1 + n_2))$.

6. Approche gloutonne pour le cas général

Dans cette section, nous présentons notre approche heuristique pour résoudre le problème *JSMB* avec plusieurs jobs. Cette approche est une méthode gloutonne basée sur l'algorithme géométrique proposé pour le cas de deux jobs.

6.1. Méthodologie suivie

Dans l'approche gloutonne que nous proposons, les jobs à ordonnancer sont considérés un après l'autre suivant une séquence fixée. A l'étape initiale, les deux premiers jobs sur la séquence sont ordonnés optimalement en utilisant l'algorithme polynomial proposé dans la section 5. En utilisant l'ordonnancement obtenu, la notion du *job composé* représentant les jobs déjà examinés, est introduite. Le but du job composé est de regrouper dans une même opération, les opérations pouvant être exécutées simultanément. A l'étape suivante, le job composé et le nouveau job à examiner définissent une instance particulière à deux jobs qui peut être résolue par une *modification de l'algorithme géométrique* proposé qui sera présentée dans la section 6.3. Une fois que tous les jobs de l'ensemble $\mathcal{J}$ sont examinés, la séquence d'entrée des opérations dans les ressources, ainsi que la valeur du makespan peuvent être déterminées à partir du *job composé* final.

L'approche gloutonne proposée est résumée par l'algorithme suivant :

Algorithme 2.2

1. *Choisir une séquence de jobs. Sans perte de généralité, soit J_1, J_2,..., J_n. cette séquence.*
2. *Initialiser $J_{com} \leftarrow J_1$.*
3. ***Pour** $i = 2$ jusqu'à N **Faire***
   3.1. Résoudre le problème 2-JSMB avec J_{com} et J_i.
   3.2. Construire le job composé J' de J_{com} et J_i.
   3.3. $J_{com} \leftarrow J'$.
   ***FinPour**,*
4. *L'ordonnancement final et la valeur C_{max} peuvent être déterminés à partir du job composé J_{com}.*

Fin Algorithme.

6.2. Construction du job composé

Après l'obtention du makespan optimal pour le cas à deux jobs, le job composé peut être défini en trois étapes :

(i) Représentation de la solution optimale par le diagramme de Gantt,

(ii) Décomposition de l'intervalle $[0, C_{max}]$ en des intervalles $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{u-1}, t_u]$ de manière à ce qu'aucune opération ne démarre ou termine à l'intérieur d'un intervalle. En d'autre termes, les dates du début des opérations $S_{ik} \in \{t_0, t_1, \ldots, t_u\}$ pour tout job J_i et toute opération O_{ik},

(iii) Attribution à chaque intervalle une opération du job composé. Par conséquent, le nombre d'opérations du job composé est égal au nombre d'intervalles crées u. De plus, la durée opératoire de l'opération est égale à la longueur de l'intervalle associé $[t_{k-1}, t_k]$, c'est-à-dire, $t_k - t_{k-1}$, et les ressources nécessaires à son exécution sont les ressources occupées dans l'intervalle associé.

Remarque 2.3 : Pour des raisons qui seront expliquées dans la suite, une fois que le job composé est obtenu, il est considéré comme un job simple de type job shop multi-ressources avec la contrainte *retenir et attendre*. Ceci, veut dire que, l'ordre d'entrée des jobs déjà examinés sur les ressources ne sera pas remis en cause dans les étapes suivantes.

Exemple 2.2 : Pour illustrer la construction du job composé, nous considérons deux jobs avec les gammes opératoires suivantes :

$$J_{com} = J_1 = \{(R_1, 1), (R_2, 1)\}$$
$$J_i = J_2 = \{(R_2, 1), (R_3, 2)\}$$

L'algorithme géométrique donne l'ordonnancement optimal $\{S_{11} = 0, S_{12} = 1, S_{21} = 0, S_{22} = 1\}$ avec un makespan $C_{max} = 3$. Suivant les dates du début et de fin des quatre opérations, l'intervalle $[0, C_{max}]$ est décomposé en trois sous-intervalles [0, 1], [1, 2] et [2, 3] (voir le diagramme de Gantt de la figure 2.9). Le job composé est alors $J' = \{(R_1, R_2, 1), (R_2, R_3, 1), (R_3, 1)\}$.

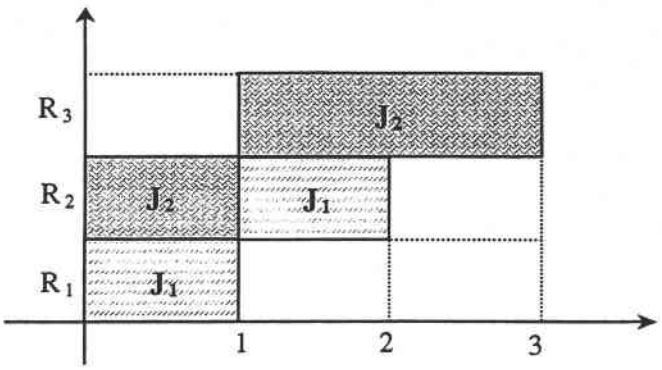


Figure 2.9 : Diagramme de Gantt

6.3. Algorithme géométrique modifié

L'objectif de cette section est de présenter une modification de l'algorithme géométrique proposé dans la section 5, pour résoudre le problème d'ordonnancement entre le job composé J_{com} et un nouveau job J_i. Cette modification est une conséquence directe de la notion d'opération composée. Afin d'atteindre cet objectif, nous commençons d'abord par étendre la notion d'obstacle introduite dans la section 5.1.

6.3.1. Différence entre un job composé et un job simple

Avant de présenter les modifications introduites dans l'algorithme géométrique, nous illustrons par les deux exemples suivants, les différents problèmes qui peuvent se produire avec le job composé.

Exemple 2.3 : Nous considérons une instance à trois jobs J_1, J_2 et J_3 avec les gammes opératoires suivantes :

$$J_1 = \{(R_3, 2), (R_1, 1), (R_2, 1)\}$$
$$J_2 = \{(R_3, 2), (R_2, R_3, 1), (R_1, 1)\}$$
$$J_3 = \{(R_4, 1), (R_2, 3), (R_3, 1)\}$$

Les ressources R_1 et R_4 sont disponibles avec une seule unité, alors qu'il existe deux unités de type R_2, et deux unités de type R_3. Les jobs sont considérés dans l'ordre J_1, J_2, J_3.

L'application de l'algorithme géométrique entre les jobs J_1 et J_2 conduit au job composé :

$$J_{com} = \{(O_{11}O_{21},2),(O_{12}O_{22},1),(O_{13}O_{23},1)\} = \{(2R_3,2),(R_1R_2R_3,1),(R_1R_2,1)\}.$$

Nous notons que, le passage de la deuxième opération (O_{12} O_{22}) du job composé J_{com} à l'opération suivante (O_{13} O_{23}) se fait en avançant d'abord le job J_1 à l'opération O_{13}, et ensuite laisser le job J_2 progresser à l'opération O_{23}. L'inverse n'est pas possible.

Pour ordonnancer le job composé J_{com} et J_3, les obstacles et le plus court chemin allant de O à F sont représentés par la figure 2.10. Nous notons qu'au niveau du point q, le système est encore dans l'état ($O_{com,2}$, O_{32}), c'est-à-dire., (O_{12}, O_{22}, O_{32}).

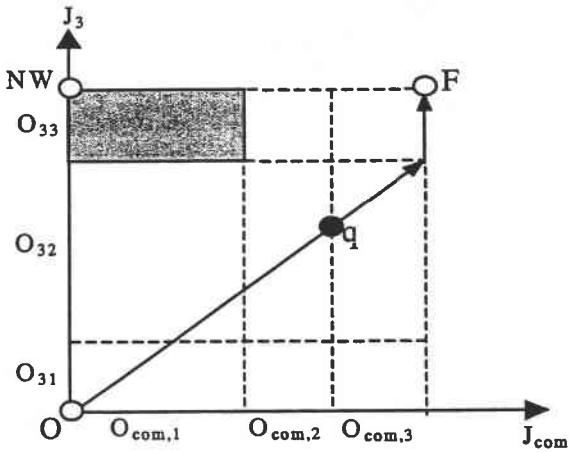


Figure 2.10 : Problèmes rencontrés avec le job composé

Pour pouvoir continuer à progresser du point q, J_{com} doit passer à sa troisième opération tandis que J_3 reste à sa seconde opération O_{32}. La seule manière de progresser alors à l'état ($O_{com,3}$, O_{32}) est de laisser le job J_1 progresser à l'opération O_{13}. Ceci n'est plus possible puisque la ressource R_2 nécessaire pour démarrer O_{13} est maintenant retenue par l'opération O_{32} du job J_3.

Remarque 2.4 : Les croisements de type ($O_{com,k}$, $O_{ik'}$) à ($O_{com,k+1}$, $O_{ik'}$) qui sont possibles lorsque J_{com} est considéré comme un job simple, peuvent ne pas être réalisables. Ceci est dû essentiellement au fait que les ressources retenues par une opération $O_{ik'}$ du job additionnel (J_3 dans l'exemple 2.3) peuvent être nécessaires pour la progression du job composé.

Remarque 2.5 : Le phénomène précédent ne se produit pas dans les systèmes où toutes les ressources sont disponibles en un seul exemplaire. En effet, s'il n'est pas possible de progresser de l'état ($O_{com,k}$, $O_{ik'}$) alors l'opération $O_{com,k}$ partage des ressources avec $O_{ik'}$. Par conséquent, l'état ($O_{com,k}$, $O_{ik'}$) est interdit par la programmation dynamique avec chaînage arrière.

Exemple 2.4 : Considérons maintenant une autre instance de *JSMB* avec trois jobs définis comme suit :

$$J_1 = \{(R_1, 1), (2R_2, 1)\}$$
$$J_2 = \{(R_2, 1), (R_3, 1)\}$$
$$J_3 = \{(R_3, 1), (R_1, 1)\}$$

Les ressources sont disponibles avec les quantités $Q_1 = 1$, $Q_2 = 3$, $Q_3 = 1$, et les jobs sont considérés pour l'ordonnancement dans l'ordre J_1, J_2, J_3.

L'algorithme géométrique appliqué à J_1 et J_2 fournit le job composé :

$$J_{com} = \{(R_1, R_2, 1), (2R_2, R_3, 1)\}.$$

Ensuite, le job composé est considéré avec le nouveau job J_3. Les obstacles sont donnés par la figure 2.11. A partir de cette représentation, il est clair que les jobs J_{com} et J_3 ne peuvent pas être réalisés en parallèle, et la valeur du makespan est alors égale à 4.

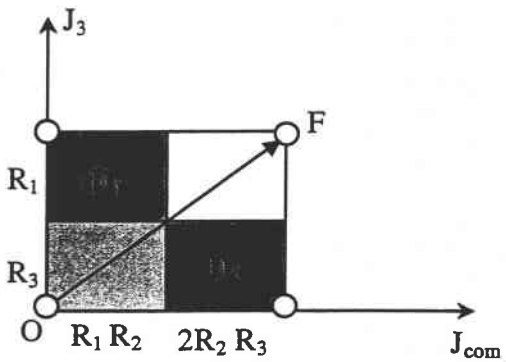


Figure 2.11 : Représentation géométrique de l'exemple 2.4

Le rectangle ($O_{com,1}$, O_{31}) qui correspond à l'état ($O_{com,1}$, O_{31}) ou d'une manière équivalente à l'état (O_{11}, O_{21}, O_{31}) est interdit par l'algorithme de détection de blocage. Ceci signifie que l'état ($O_{com,1}$, O_{31}) est un état de blocage si le job J_{com} est considéré comme un job simple, puisqu'il n'existe pas de conflit des ressources entre $O_{com,1}$ et O_{31}. En revanche, il est possible de progresser de l'état ($O_{com,1}$, O_{31}) à l'état ($O_{com,2}$, O_{32}) en utilisant les étapes de franchissements suivantes : au départ, on progresse le job J_1 à l'opération O_{12}, ce qui est possible car $Q_2 = 3$. Ensuite, on avance le job J_3 à l'opération O_{32} qui devient possible puisque la ressource R_1 est libérée par O_{11}, et enfin, le job J_2 progresse à l'opération O_{22} qui devient possible puisque la ressource R_3 est libérée par O_{31}.

Cette règle de progression permet de progresser le système de l'état (O_{11}, O_{21}, O_{31}) à l'état (O_{12}, O_{22}, O_{32}) et par conséquent permet de réaliser les trois jobs. En conséquence, il existe une manière d'exécuter les jobs J_1, J_2 et J_3 en démarrant de l'état ($O_{com,1}$, O_{31}). Ceci implique que ce dernier état n'est pas un état du blocage. Dans ce cas, le plus court chemin a une longueur égale à 2, illustré par la figure 2.11.

Remarque 2.6 : A la différence du cas à deux jobs simples dans lequel le passage en diagonale de l'état (O_{1,k_1}, O_{2,k_2}) à (O_{1,k_1+1}, O_{2,k_2+1}) passe toujours par (O_{1,k_1+1}, O_{2,k_2}) ou (O_{1,k_1}, O_{2,k_2+1}), le passage en diagonale (O_{com,k_1}, O_{ik_2}) à (O_{com,k_1+1}, O_{i,k_2+1}) n'a pas besoin de passer par (O_{com,k_1+1}, O_{ik_2}) ou (O_{com,k_1}, O_{i,k_2+1}), et peut être réalisé directement en laissant progresser quelques jobs de J_{com}, puis en laissant le job J_i progresser et enfin en laissant progresser le reste des jobs de J_{com}. En conséquence, quelques états de blocage quand le job composé est considéré comme un job simple, sont actuellement admissibles.

6.3.2. *Nouvelle caractérisation du blocage*

Il résulte des deux exemples 2.3 et 2.4 que nous venons de présenter, qu'un job composé ne peut pas être considéré comme un job simple puisqu'il conduit à des situations différentes de blocage. Le problème de détection du blocage lorsque nous ordonnançons un job composé et un job simple, consiste à vérifier, en démarrant d'une opération composée O_{com,k_1} et d'une opération O_{2,k_2} du job simple, si nous aboutissons à une situation de blocage inévitable.

Pour développer la nouvelle procédure de détection de blocage, nous introduisons les indicateurs suivants relatifs aux passages diagonaux :

$$P_x(k_1,k_2)=\begin{cases}1, & \text{si le passage de l'état } (O_{com,k_1-1}, O_{i,k_2}) \text{ à} \\ & \text{l'état } (O_{com,k_1}, O_{i,k_2}) \text{ n'est pas possible,} \\ 0, & \text{sinon.}\end{cases}$$

$$P_y(k_1,k_2)=\begin{cases}1, & \text{si le passage de l'état } (O_{com,k_1}, O_{i,k_2-1}) \text{à} \\ & \text{l'état } (O_{com,k_1}, O_{i,k_2}) \text{ n'est pas possible,} \\ 0, & \text{sinon.}\end{cases}$$

$$P_{xy}(k_1,k_2)=\begin{cases}1, & \text{si le passage de l'état } (O_{com,k_1-1}, O_{i,k_2-1}) \text{à} \\ & \text{l'état } (O_{com,k_1}, O_{i,k_2}) \text{ n'est pas possible,} \\ 0, & \text{sinon.}\end{cases}$$

L'introduction des indicateurs $P_{xy}(k_1, k_2)$ est motivée par la remarque 2.6. Ces indicateurs déterminent l'admissibilité des passages immédiats horizontaux (P_x), verticaux (P_y) et diagonaux (P_{xy}).

Le calcul de ces indicateurs est un problème de détection de blocage qui est NP-difficile dans le cas général. Pour éviter l'explosion combinatoire dans la vérification du blocage, nous introduisons l'hypothèse suivante :

Hypothèse : *Une séquence de passage des opérations est associée à chaque opération du job composé*

Dans l'algorithme glouton, les séquences de passage des opérations sont mises à jour à chaque étape. Après l'ordonnancement d'un nouveau job J_j, le nouveau job composé est déterminé, et les séquences d'une opération composée sont obtenues en insérant l'opération du job J_j dans la séquence actuelle de l'opération $O_{com,k}$.

Remarque 2.7 : Le calcul de $P_x(k_1, k_2)$ est réalisé en vérifiant les blocages en laissant progresser les jobs de J_{com} suivant la séquence associée à O_{com,k_1}. L'indicateur $P_y(k_1, k_2)$ est déterminé en vérifiant la capacité des ressources de l'état (O_{com,k_1}, O_{i,k_2}). Enfin, le calcul de $P_{xy}(k_1, k_2)$ est similaire à celui de $P_x(k_1, k_2)$ en insérant l'opération O_{i,k_2} dans la séquence associée à O_{com,k_1}.

Comme dans le cas de l'algorithme de programmation dynamique à chaînage arrière déjà proposé dans la section 5, nous considérons les indicateurs suivants pour caractériser les états de blocage et de partage de ressources.

$$I(k_1,k_2)=\begin{cases}1, & \text{si les opérations } O_{1k_1} \text{ et } O_{2k_2} \text{ sont en conflit de ressources}\\ & \text{ou une situation de blocage est inévitable partant de l'état } (k_1,k_2),\\ 0, & \text{sinon.}\end{cases}$$

En utilisant les indicateurs P_x, P_y, P_{xy}, la nouvelle équation récursive de la programmation dynamique est :

$$I(k_1,k_2)=\begin{cases}1, & \text{si } B^r_{com,k_1} + B^r_{i,k_2} > Q_r \text{ pour une ressource } R_r,\\ \big(I(k_1,k_2+1) \vee P_y(k_1,k_2+1)\big) & \\ \wedge\ \big(I(k_1+1,k_2) \vee P_x(k_1+1,k_2)\big) & \\ \wedge\ \big(I(k_1+1,k_2+1) \vee P_{xy}(k_1+1,k_2+1)\big), & \text{sinon.}\end{cases}$$

où : $\wedge$ (resp. $\vee$) est l'opérateur binaire de conjonction (resp. disjonction). Par convention $I(k_1, n_2+1) = I(n_{com}+1, k_2) = 0$, $\forall\ k_1, k_2$, où n_{com} est le nombre d'opérations du job composé J_{com}.

En résumé, la méthode modifiée de détection des blocages procède ainsi :

- Détermination des trois indicateurs P_x, P_y et P_{xy} en utilisant la remarque 2.7.
- Détermination de la matrice d'indicateurs I pour déterminer les conflits de ressources et les situations du blocage.

Remarque 2.8 : La détermination des matrices d'indicateurs P_x, P_y et P_{xy}, prend un temps proportionnel à $O(n_2 \times n_{com})$. Pour le cas de la matrice d'indicateurs I, la nouvelle modification examine tous les états entre le job composé et le nouveau job. Ces états sont en nombre de $n_2 \times n_{com}$. En conséquence, la complexité de la nouvelle caractérisation du blocage et de conflit des ressources est $O(n_2 \times n_{com})$. Nous notons que, n_{com} est au plus égal à $\sum_{i=1}^{l} n_i$, où l est le nombre des jobs déjà examinés.

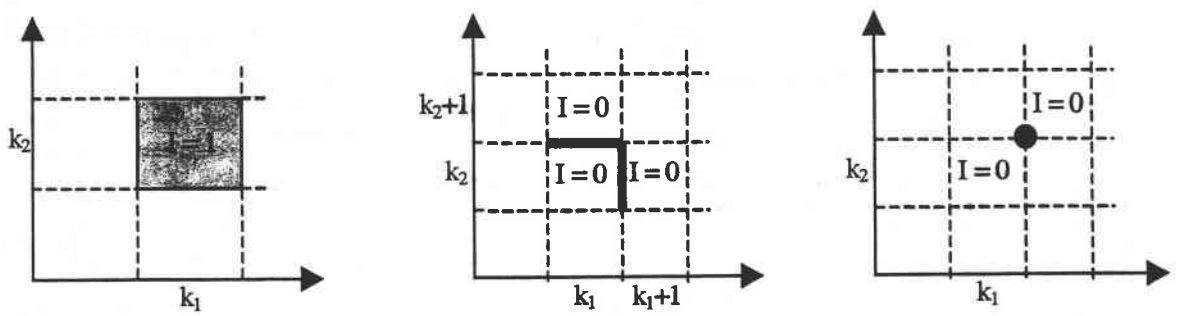
6.3.3. Construction des nouveaux obstacles

Trois types d'obstacles représentés par la figure 2.12 doivent être considérés :

- Les obstacles *2-dimension* notés 2-*D* correspondants aux intervalles O_{com,k_1} et O_{i,k_2} tel que $I(k_1, k_2) = 1$. Ces obstacles 2-*D* correspondent aux situations de conflit des ressources et des situations de blocage,

- Les obstacles *1-dimension* notés 1-*D* correspondants aux bordures droite et supérieure d'un rectangle (k_1, k_2) tels que $I(k_1, k_2) = I(k_1, k_2+1) = 0$ et $P_y(k_1, k_2+1) = 1$ ou $I(k_1, k_2) = I(k_1+1, k_2) = 0$ et $P_x(k_1+1, k_2) = 1$. Ces obstacles 1-*D* sont dus à la remarque 2.6. Ils n'existent pas si J_{com} est considéré comme un job simple.

- Les obstacles *0-dimension* notés 0-*D* correspondants aux coins *NE* des rectangles (k_1, k_2) tels que $I(k_1, k_2) = I(k_1+1, k_2+1) = 0$ et $P_{xy}(k_1+1, k_2+1) = 1$. De tels obstacles, interdissent le passage diagonal dans le rectangle (k_1+1, k_2+1).



(a) obstacle 2-D (b) obstacle 1-D (c) obstacle 0-D

Figure 2.12 : Trois types d'obstacles

Les figures 2.13 et 2.14 montrent les nouveaux obstacles pour les exemples 2.3 et 2.4.

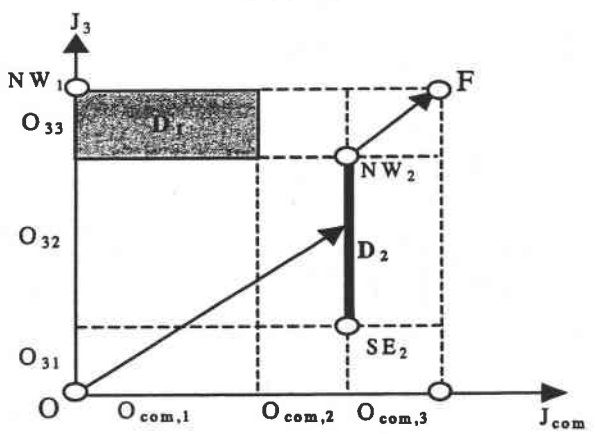


Figure 2.13 : Nouvelle représentation géométrique de l'exemple 2.3

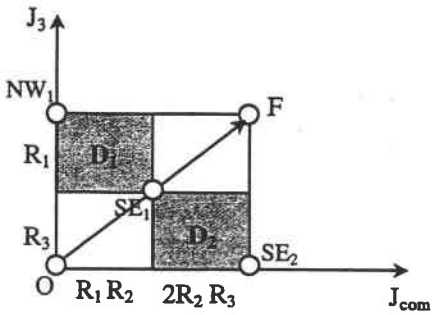


Figure 2.14 : Nouvelle représentation géométrique de l'exemple 2.4

Pour l'exemple 2.3, un obstacle de type 1-*D* est introduit puisqu'il n'est pas possible de progresser de l'état ($O_{com,2}$, O_{32}) à ($O_{com,3}$, O_{32}). Pour l'exemple 2.4, le rectangle ($O_{com,1}$, O_{31}) n'est pas un état du blocage puisque le passage diagonal de ($O_{com,1}$, O_{31}) à ($O_{com,2}$, O_{32}) est admissible.

Remarque 2.9 : Les points suivants doivent être pris en compte dans la construction du réseau :

- Les bordures supérieure, inférieure et l'intérieur des obstacles de type 2-*D* sont interdites.
- L'obstacle 1-*D* est un cas dégénéré des obstacles de type 2-*D*.
- Les coins *NW* et *SE* des obstacles 1-*D* et 2-*D* sont des nœuds du réseau.
- L'obstacle 0-*D* apparaît toujours comme l'adjonction de deux obstacles (figure 2.15). En conséquence, si un coin *NW/SE* des obstacles 1-*D* et 2-*D* est un obstacle 0-*D*, alors les coins correspondants *NW/SE* sont interdits et n'appartiennent pas à l'ensemble des nœuds du réseau.

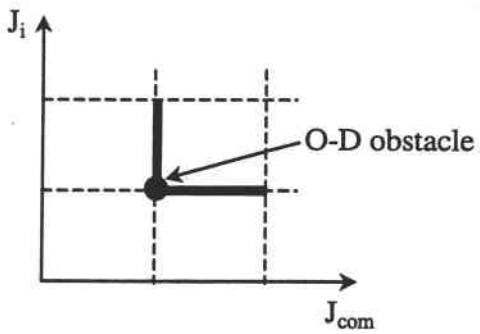
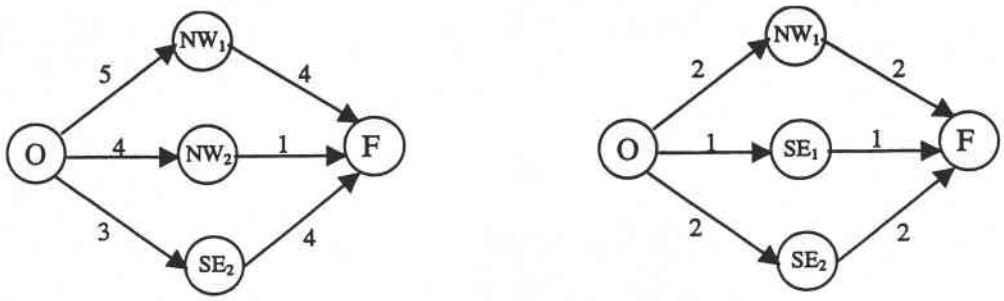


Figure 2.15 : Obstacles *O-D*

Pour les deux exemples 2.3 et 2.4, les nouveaux réseaux obtenus sont donnés par la figure 2.16.



(a) réseau de l'exemple 2.3 (b) réseau de l'exemple 2.4

Figure 2.16 : Les réseaux pour les exemples 2.3 et 2.4

6.4. Amélioration de l'ordonnancement

D'après la discussion précédente, la notion du job composé permet de réduire la complexité du problème d'ordonnancement. Malheureusement, il pose le problème de l'optimalité en raison de l'intégration des opérations $\{O_1, O_2, \ldots, O_{i'k'}, O_p\}$ dans une seule opération. En effet, une opération O_{i,k_2} du nouveau job J_i ayant un conflit de ressources avec une opération $O_{i',k'}$ du job composé peut pénaliser les autres opérations si O_{i,k_2} est ordonnancée avant $O_{i',k'}$. Autrement dit, l'ordonnancement trouvé par l'algorithme 2.2 peut être un ordonnancement avec délai.

Pour obtenir un ordonnancement sans délai, nous commençons d'abord par extraire les séquences d'entrée σ_r des opérations dans chaque ressource R_r. Cette séquence est unique et peut être déterminée facilement à cause de l'hypothèse sur le job composé qui associé une séquence à chaque opération composée.

Ensuite, nous ordonnançons les opérations O_{ik} pour tous les jobs J_j une après l'autre suivant la séquence d'entrée σ_r. Nous notons que lors de la considération d'une opération O_{jk}, toutes les opérations qui partagent une ressource avec O_{jk} et qui précèdent O_{jk} dans la séquence d'entrée sont déjà ordonnancées. Par conséquent, on peut déterminer la date de libération de toutes les ressources R_r. Soit *Release(r, n)* la date à laquelle n unités de ressources du type r deviennent disponibles. La date de début au plutôt de l'opération O_{jk} respectant la séquence d'entrée σ_r peut alors être calculée comme suit:

$$S'_{jk} = \max\left\{S'_{j,k-1} + p_{j,k-1}, \mathrm{Max}_{r \in R_{jk}}\left\{\mathrm{Re}\,lease\left(r, B^r_{jk}\right)\right\}\right\} \qquad (6)$$

Le nouvel algorithme combinant l'algorithme glouton et l'amélioration introduite dans cette section, peut être décrit comme suit :

Algorithme 2.3 : *(Algorithme glouton suivant la séquence de job π)*

1. $J_{com} \leftarrow J_{\pi(1)}$.
2. ***Pour*** $i \leftarrow 2$ *jusqu'à* N ***Faire***
   2.1. *Résoudre le problème de deux jobs entre* J_{com} *et* $J_{\pi(i)}$ *en utilisant l'algorithme géométrique modifié.*
   2.2. *Construire le job composé* J' *de* J_{com} *et* $J_{\pi(i)}$ *suivant l'ordonnancement optimal.*
   2.3. $J_{com} \leftarrow J'$.
   FinPour,
3. *Extraire les séquences d'entrée* σ_r *des opérations dans chaque ressource* R_r *du job composé final* J_{com}.
4. *Calculer les dates de début au plutôt* S'_{jk} *de toutes les opérations en utilisant l'équation (6).*

Fin Algorithme.

7. Intégration avec la recherche tabou

La qualité de l'ordonnancement obtenu par l'algorithme 2.2 proposé dans la section 5 dépend fortement de la séquence des jobs choisie dans l'étape 1 de l'algorithme. Cette séquence permet à l'algorithme glouton d'examiner les jobs un après l'autre.

Dans cette section, nous nous focalisons sur la sélection des séquences de jobs permettant à l'algorithme glouton de fournir les meilleurs ordonnancements. Il est clair que, si le nombre de jobs est inférieur à sept, on peut se permettre de vérifier toutes les séquences (7! = 5040). Au-delà de cette limite, le nombre de possibilités augmente d'une manière exponentielle, et donc le besoin d'une approche pour explorer un grand nombre de séquences des jobs sans répétition est crucial.

Pour ce faire, nous utiliserons une approche de recherche tabou (Glover [1997]) pour explorer l'espace des solutions de la séquence des jobs. Cette heuristique est appliquée sur un problème artificiel. La motivation d'introduire ce problème artificiel est due au fait que l'évaluation d'un grand nombre de séquences des jobs en utilisant l'algorithme géométrique modifié nécessite un temps considérable.

7.1. Problème artificiel

Le problème que nous considérons est équivalent au problème d'une *machine avec temps de réglage* ou *single machine-scheduling problem with sequence-dependent setup times* (Pinedo [1995]), qui consiste à ordonnancer un ensemble de N tâches toutes disponibles à la date 0 sur une seule machine. Ainsi,

- Chaque tache i (i = 1, 2,..., N) correspond à un job de notre modèle, et nécessite zéro unité de temps.

- Le temps de changement d'une tache i à la tache suivante j est calculé de la manière suivante :

$$S_{ij} = \sum_{k=1}^{n_i} p_{ik} + \sum_{k=1}^{n_j} p_{jk} - C_{max_{ij}},$$

où $C_{max_{ij}}$ est le makespan obtenu par l'algorithme géométrique après la résolution du problème à deux jobs entre J_i et J_j.

Interprétation : La quantité S_{ij} peut être interprétée comme le profit de placer le job J_j juste après le job i.

Pour chaque séquence de jobs $\prod = \{\pi(1), \pi(2), ..., \pi(N)\}$ où $\pi(i)$ est la tache se trouvant à la position i, la fonction objectif artificielle consiste à maximiser le profit total donné par :

$$\text{Gain} = \sum_{i=1}^{N-1} S_{\pi(i)\pi(i+1)}.$$

Les paramètres de la recherche tabou sélectionnés dans notre application sont les suivants:

7.2. Paramètres de la recherche tabou

A. La solution initiale

La solution initiale est calculée en utilisant une heuristique d'insertion. Plus précisément, soit E_1, E_2 deux ensembles représentant les jobs déjà examinés et les jobs qui ne sont pas encore considérés, respectivement. Au début, l'ensemble E_1 contient les deux jobs donnant le makespan minimum sur toutes les paires possibles de jobs. Le job composé est alors calculé à la base de ces deux jobs. A chaque étape, un job $J_i \in E_2$ donnant le makespan minimum lorsqu'il est considéré pour l'ordonnancement avec J_{com} est sélectionné et inséré dans E_1. Le job composé est alors mis à jour. Cette démarche continue jusqu'à ce que l'ensemble E_2 devienne vide. L'ordre pour lequel les jobs sont insérés dans E_1 représente la séquence de jobs initiale.

B. Mouvement

Le mouvement de permutation qui consiste à échanger le job J_i se trouvant dans la position $\pi(i)$ et un autre job J_j se trouvant dans la position $\pi(j)$ est adopté. Il est à noter que l'évaluation de ces mouvements est effectuée en utilisant la fonction objectif artificielle.

C. Restriction tabou

Les jobs J_i et J_j sont les attributs du mouvement de permutation. Ils sont utilisés pour créer la condition tabou. Après l'exécution du mouvement, tout mouvement qui retourne le job J_i (resp. J_j) à la position $\pi(i)$ (resp. $\pi(j)$) est classé tabou. Cette restriction est renforcée en ajoutant ces informations dans la liste tabou.

D. Taille de la liste tabou

L'aspect dynamique de la liste tabou est adopté, et la taille de la liste est sélectionnée aléatoirement dans l'intervalle de référence [*N*/2, *N*], où *N* est le nombre de jobs. Cette taille est mise à jour chaque 2*N* itérations pour permettre de travailler avec d'autres tailles de la liste.

E. Critère d'aspiration

Dans cette application, nous avons utilisé le critère le plus connu et le plus utilisé dans la littérature qui est le critère d'aspiration globale. Ce critère permet à un mouvement tabou d'être candidat pour la sélection, s'il conduit à une nouvelle meilleure solution.

F. Critère d'arrêt

L'algorithme de recherche tabou est arrêté si un nombre d'opérations fixé à priori est atteint. Dans notre application, le nombre maximum est fixé à 5000 itérations.

7.3. Heuristique finale

En combinant l'algorithme glouton « algorithme 2.2 », l'étape d'amélioration de l'ordonnancement et l'heuristique de recherche tabou, l'heuristique finale que nous proposons pour résoudre le cas général de *JSMB* avec un nombre quelconque de jobs est décrite par l'algorithme 2.4 appelé *TSGA*.

Algorithme 2.4 : *Algorithme TSGA*

1. *Initialisation: Déterminer la séquence initiale du job en utilisant l'heuristique d'insertion. Soit* $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(N)}$ *cette séquence. Mettre le meilleur makespan* $C^*_{max} \leftarrow \infty$.
2. *Calculer l'ordonnancement* J_{com} *en utilisant l'algorithme glouton (algorithme 2.3) avec* π *comme séquence d'entrée. Posons* C_{max} *le makespan trouvé.*
3. ***Si*** $C_{max} < C^*_{max}$ ***Alors***
   - 3.1. *Mettre à jour le meilleur makespan* $C_{max}^* \leftarrow C_{max}$
   - 3.2. *Enregistrer l'ordonnancement* J_{com} *comme le meilleur ordonnancement,*

   FinSi.
4. *Critère d'arrêt:* ***Si*** *le nombre maximum d'itérations est atteint* ***Alors*** *Arrêter.*
5. *Trouver le meilleur mouvement non tabou en utilisant la "fonction objectif artificielle".*
6. *Réaliser le mouvement pour créer une nouvelle séquence de jobs* π
7. *Mettre à jour les structures de la recherche tabou, c'est-à-dire, liste tabou et la taille de la liste.*
8. *Aller à l'étape 2.*

Fin Algorithme.

---

8. Résultats numériques

Au contraire des problèmes classiques d'ordonnancement tel que le job shop classique, les benchmarks pour le cas avec blocage sont très limités. Nous présentons dans cette section les résultats obtenus sur 12 instances existantes.

8.1. Les instances

Les trois premières instances représentent une cellule manufacturière automatisée (Joshi et Ramaswamy [1996]), avec trois machines et un robot pour transporter les produits d'une machine à l'autre, charger/décharger les produits, qui sont au nombre de quatre. Dans le problème *RJ*1, les produits peuvent aller automatiquement d'une machine à une autre sans faire appel au robot. Dans le problème *RJ*2, le robot est utilisé pour transporter les produits. Enfin, dans le problème *RJ*3, il existe un stock de capacité 1 entre chaque paire de machines et le robot est utilisé pour manipuler les produits. Pour ces trois instances, les solutions optimales sont connues (Joshi et Ramaswamy [1996]). Une quatrième instance *MDX*1 avec multiples ressources est considérée. Cette instance est similaire à *RJ*2 avec la présence de deux robots au lieu d'un seul.

Les instances restantes sont générées aléatoirement par Damasceno [1999], et contiennent deux classes d'instances. La première classe comporte les instances avec des ressources unitaires, et contient cinq instances. Il y a neuf jobs à ordonnancer dans les deux premières instances et dix jobs dans les trois restantes. La deuxième classe comporte les instances avec des ressources multiples, et contient trois problèmes avec dix jobs à ordonnancer.

8.2. Résultats obtenus

Les solutions trouvées par notre algorithme sont comparées avec les résultats proposés dans Damasceno [1999], qui est l'une des rares approches qui traite les mêmes caractéristiques de notre modèle. Les résultats obtenus pour cinq exécutions indépendantes sont donnés dans le Tableau 2.1.

Tableau 2.1. Résultats de 5 exécutions indépendantes

Problème	algorithme TSGA			Damasceno, 1999	
	mauvais	meilleur	CPU secs	Makespan	CPU secs
RJ1	512	**512***	0.02	n.d	n.d
RJ2	560	**560***	0.02	568	31
RJ3	502	**502***	0.02	n.d	n.d
UDX1	90	**90**	1.39	98	n.d
UDX2	109	**109**	4.15	124	n.d
UDX3	103	**103**	8.41	115	n.d
UDX4	112	**112**	7.53	126	n.d
UDX5	104	**103**	0.09	122	n.d
MDX1	451	**451**	0.02	**451**	31
MDX2	33	**33**	0.37	43	n.d
MDX3	35	**35**	0.03	44	n.d
MDX4	47	**47**	5.16	56	n.d

*: solution optimale
n.d: n'est pas donné

Pour les trois premières instances, l'heuristique proposée permet de trouver les solutions optimales. Bien que la taille de ces instances soit modeste, ce résultat assure que l'approche ne donne pas des solutions trop éloignées des solutions optimales.

Pour le reste des instances, l'approche proposée améliore et apporte de nouveaux meilleurs résultats par rapport aux résultats donnés par Damasceno [1999] sauf pour l'instance *MDX*1 pour laquelle le makespan est le même. De plus, en observant la colonne du temps d'exécution, notre approche est plus rapide que l'approche de Damasceno [1999] et les temps d'exécution ne dépasse pas 10 *CPU secs* dans le pire des cas. Enfin, pour tester l'homogénéité de notre approche, c'est-à-dire, la variation de la solution d'une exécution à l'autre, nous l'avons lancé plusieurs fois. Les résultats donnés dans la colonne "*mauvais*" et "*meilleur*" indiquant respectivement le makespan de la mauvaise et la meilleure solution trouvée. Ces résultats indiquent que les deux valeurs sont les mêmes sauf dans le cas du problème *UDX*5 pour lequel la différence est très petite et est égale à 1 (valeur absolue). Ceci implique que notre algorithme est stable, et ne donne pas des solutions aberrantes d'une exécution à l'autre.

9. Extension de l'approche

Le critère de minimisation du makespan est très utilisé dans la théorie de l'ordonnancement, et trouve beaucoup d'applications dans la pratique. Cependant, dans d'autres applications pratiques, le souci d'un industriel est le respect des délais de livraison, la minimisation des retards, etc. Il est alors nécessaire de traiter d'autres types de critères.

L'algorithme polynomial proposé pour le cas de 2-*JSMB* est valable pour tout critère objectif régulier. Ce résultat et la méthodologie introduite donnent une souplesse supplémentaire à l'approche heuristique proposée. En effet, la même démarche utilisée peut être aussi appliquée pour résoudre le problème avec un critère objectif régulier différent du makespan. La programmation dynamique proposée reste valable du fait qu'elle ne prend pas en compte

de critère objectif dans son développement. La seule différence concerne la résolution du problème à deux jobs entre le job composé et le nouveau job. Dans ce dernier problème, le calcul de la valeur du critère est plus compliqué que dans le cas du makespan car l'exécution d'un job composant le job composé peut se terminer au milieu du job composé. Pour pouvoir prendre en compte les autres critères réguliers, il suffit de garder la date de fin de chaque job, chaque fois qu'une ligne verticale est croisée. Enfin, nous notons que même si nous supposons que le job composé est un job classique, et que la séquence d'entrée pour chaque opération composée est fixée, la résolution du problème à deux jobs entre le job composé et le nouveau job n'est pas optimale.

La résolution du problème avec des critères objectifs réguliers est un axe important de recherche, et constitue l'une de nos futures recherches.

10.Conclusion

Dans ce chapitre, nous avons considéré un modèle d'ordonnancement générique représentant le job shop multi-ressources avec blocage. Dans un premier temps, nous avons formulé le problème d'ordonnancement en donnant toutes ses caractéristiques. Ensuite, nous avons proposé un algorithme polynomial pour résoudre le cas de deux jobs. Cet algorithme est une généralisation de l'approche géométrique classique, et s'appuie sur une nouvelle caractérisation de blocage et de construction du réseau. Cette nouvelle extension pour le critère du makespan permet de montrer que le problème d'ordonnancement à deux jobs de notre modèle est polynomial pour tout critère objectif régulier.

En se basant sur cet algorithme, une heuristique gloutonne est proposée. Nous avons montré que dans le cas général, cette heuristique a des lacunes lorsque nous considérons le cas de ressources multiples. Pour remédier à ces problèmes, la nouvelle approche pour le cas à deux jobs est généralisée pour résoudre le cas à deux jobs entre le job composé et un nouveau job. Dans le but d'augmenter l'efficacité de l'approche, l'heuristique gloutonne est couplée avec une recherche tabou afin de trouver les séquences de traitement des jobs, et ainsi lui permettre d'être exécutée sur les bonnes séquences. L'heuristique finale combinant toutes les améliorations, est testée sur des instances issues de la littérature. Les résultats obtenus sont très satisfaisants, et l'approche proposée est très compétitive avec les méthodes existantes.

Chapitre 3

Job Shop Multi-Ressources avec Prise en Compte du Blocage : Cas des Ressources Unitaires

Dans ce chapitre, nous étudions un cas particulier du modèle job shop multi-ressources avec blocage introduit dans le chapitre précédent. Dans ce nouveau modèle, toutes les ressources sont disponibles en un seul exemplaire. Nous commençons d'abord par donner une nouvelle extension de la représentation classique par le graphe disjonctif, pour modéliser le problème d'ordonnancement. En se basant sur cette extension, une heuristique de recherche tabou est proposée pour explorer l'espace des solutions. Cette heuristique est basée sur une nouvelle structure du voisinage définie par deux mouvements de base : la permutation des arcs disjonctifs sur le chemin critique, et un mouvement de recouvrement du blocage si le premier échoue. Les résultats expérimentaux obtenus sur des instances de la littérature et sur de nouvelles instances générées aléatoirement, montrent que l'heuristique de recherche tabou donne rapidement des solutions satisfaisantes, et améliorent les meilleurs résultats connus.

Les résultats présentés dans ce chapitre ont été exposés lors d'une conférence internationale, MIC-01 (4th Metaheuristics International Conference)[2001], à Porto, Portugal. Ils ont fait l'objet d'un article publié dans Journal of Intelligent Manufacturing.

1. Introduction

Dans le chapitre précédent, nous avons introduit un modèle d'ordonnancement appelé *job shop multi-ressources avec blocage*, pour lequel une ressource peut être disponible en plusieurs exemplaires. Cette dernière hypothèse permet de modéliser par exemple, des stocks tampons de capacité limitée entre les machines, un atelier utilisant plusieurs *AGVs* identiques pour le transport des produits, etc. Il existe cependant d'autres problèmes pour lesquels les ressources sont disponibles en un seul exemplaire. C'est le cas par exemple pour la modélisation des problèmes d'ordonnancements du secteur ferroviaire.

Nous avons vu dans le chapitre 2 que la présence des ressources multiples complique considérablement le problème, en particulier au niveau de la détection du blocage. Dans ce chapitre nous nous intéressons à la résolution du problème d'ordonnancement du job shop multi-ressources avec blocage dans le cas des ressources unitaires. Ce problème est une généralisation du problème *job shop multiprocesseur* (Drozdowski [1996]), intégrant la contrainte *retenir et attendre*.

La suite de ce chapitre est organisée de la manière suivante. Dans la section 2, nous donnons une description du problème d'ordonnancement. Nous proposons ensuite, dans la section 3, une modélisation mathématique du problème d'ordonnancement. La section 4 présente le problème d'ordonnancement des trains et décrit la manière dont notre modèle s'y applique. Une nouvelle généralisation de la représentation classique par le graphe disjonctif est donnée dans la section 5, et les principales propriétés du graphe obtenu sont décrites. Nous présentons dans la section 6, une extension du graphe disjonctif pour modéliser le problème d'ordonnancement des trains. La section 7 est consacré aux approches utilisées pour résoudre les problèmes de détection et de recouvrement des situations de blocage. Dans la section 8, nous proposons un algorithme de recherche tabou utilisant les résultats précédents pour explorer l'espace des solutions sur le nouveau graphe disjonctif. Les résultats numériques sont présentés dans la section 9. Nous proposons, dans la section 10, un modèle de programmation linéaire en variables mixtes permettant d'exprimer mathématiquement les contraintes de blocage. Enfin, des extensions possibles de l'heuristique de recherche tabou et nos conclusions sont proposées aux sections 11 et 12.

2. Description du problème

Dans cette section, nous donnons les caractéristiques du modèle job shop multi-ressources avec blocage, pour lequel toutes les ressources sont disponibles en une seule unité, noté dans la suite *JSSB*. Ce modèle est un cas particulier du modèle traité dans le chapitre 2.

Le problème d'ordonnancement de *JSSB* est défini de la manière suivante. Un ensemble de ressources $\mathcal{R} = \{1, 2, \ldots, r, \ldots, m\}$ est disponible pour réaliser un ensemble de N jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_i, \ldots, J_N\}$. Chaque ressource r est disponible en un seul exemplaire, c'est-à-dire, que les ressources sont différentes les unes des autres. La réalisation d'un job J_i nécessite une séquence linéaire d'opérations $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$, où n_i est le nombre d'opérations du job J_i. La séquence d'opérations est aussi appelée gamme opératoire du job J_i. Chaque opération O_{ik} doit être réalisée sans interruption et nécessite un ensemble de

ressources $R_{ik} \subseteq \mathcal{R}$. La durée opératoire p_{ik} de l'opération O_{ik} est connue. Dans ce chapitre, nous adoptons la représentation suivante pour caractériser la gamme opératoire d'un job :

$$J_i = \{(R_{i1}, p_{i1}), (R_{i2}, p_{i2}), \ldots, (R_{in_i}, p_{in_i})\}.$$

Pour des raisons que nous expliquons dans la section 5, nous supposons que la dernière opération O_{in_i} de la gamme opératoire d'un job J_i est telle que $R_{in_i} = \varnothing$ et $p_{in_i} = 0$. Si cette hypothèse n'est pas vérifiée, nous pouvons toujours ajouter une opération fictive $(\varnothing, 0)$.

Enfin, la contrainte la plus importante du modèle *JSSB* concerne la libération des ressources à la fin d'une opération O_{ik}, appelée propriété *retenir et attendre*. Dans le *JSSB*, à la fin de l'opération O_{ik}, les ressources nécessaires R_{ik} sont retenues par le job J_i jusqu'à ce que les ressources nécessaires pour l'opération suivante sur la gamme $O_{i,k+1}$ soient toutes disponibles.

A cause de la contrainte *retenir et attendre*, et à la différence du modèle d'ordonnancement job shop multiprocesseur (Drozdowski [1996]), des situations de blocage peuvent se produire dans le système si les ressources ne sont pas bien coordonnées. Par conséquent, le problème d'ordonnancement consiste à trouver la meilleure allocation des ressources afin d'éviter les situations de blocage, et ce dans le but de minimiser un critère objectif fixé. Nous nous concentrons dans cette étude sur la minimisation du makespan C_{max}. En utilisant la notation présentée dans Brucker [1998] décrivant les problèmes d'ordonnancements en trois champs, notre problème *JSSB* est noté *JMPT* | *blocking* | C_{max}.

3. Formulation mathématique

Les variables de décision utilisées pour modéliser le problème d'ordonnancement *JSSB* doivent donner, d'une part, les dates de début des opérations notées S_{ik}, et d'autres part, la séquence d'entrée sur chaque ressource. Ces séquences d'entrée sont caractérisées par les indicateurs $y_{ik,i'k'}$ pour tout couple d'opérations $(O_{ik}, O_{i'k'})$ partageant au moins une ressource, c'est-à-dire, $R_{ik} \cap R_{i'k'} \neq \varnothing$. Les variables $y_{ik,i'k'}$ sont définies comme suit :

$$y_{ik,i'k'} = \begin{cases} 1 & \text{si l'opération } O_{ik} \text{ précède } O_{i'k'} \\ 0 & \text{sinon} \end{cases}$$

En utilisant ces indicateurs, le problème d'ordonnancement de *JSSB* peut être écrit de la manière suivante:

$$
(P)\quad \begin{cases}
Min\ C_{\max} & & \\
\text{sous les contraintes :} & & \\
S_{i,k+1} - S_{ik} \geq p_{ik} & \forall 1 \leq i \leq N, \forall 1 \leq k \leq n_i - 1 & (1)\\
S_{ik} - S_{i'k'+1} + M\, y_{ik,i'k'} \geq 0 & \forall (O_{ik}, O_{i'k'}) / \ i < i' \wedge R_{ik} \cap R_{i'k'} \neq \varnothing & (2)\\
S_{i'k'} - S_{i,k+1} + M\,(1 - y_{ik,i'k'}) \geq 0 & \forall (O_{ik}, O_{i'k'}) / \ i < i' \wedge R_{ik} \cap R_{i'k'} \neq \varnothing & (3)\\
C_{\max} \geq S_{in_i} & \forall 1 \leq i \leq N & (4)\\
\text{les séquences d'entrée } y_{ik,i'k'} \text{ ne conduisent pas à des situations du blocage} & & (5)\\
S_{ik} \geq 0 & \forall\ 1 \leq i \leq N, \forall 1 \leq k \leq n_i & \\
y_{ik,i'k'} \in \{0,1\} & \forall (O_{ik}, O_{i'k'}) / \ R_{ik} \cap R_{i'k'} \neq \varnothing &
\end{cases}
$$

où M est une constante suffisamment grande, par exemple $M \geq \sum_i \sum_k p_{ik}$.

Dans ce modèle, les contraintes (1) correspondent aux contraintes de précédence entre les opérations du même job. A la différence du modèle mathématique proposé dans le chapitre 2, pour lequel la contrainte de capacité des ressources n'est pas exprimée par des relations mathématiques, les contraintes (2) et (3) expriment le partage des ressources entre deux opérations O_{ik} et $O_{i'k'}$. En effet, si l'opération O_{ik} précède $O_{i'k'}$ ($y_{ik,i'k'} = 1$), alors la contrainte (3) implique $S_{i'k'} \geq S_{i,k+1}$ puisque les ressources communes sont retenues par le job J_i durant l'intervalle $[S_{ik}, S_{i,k+1})$. Dans le cas contraire, si $O_{i'k'}$ précède O_{ik}, alors la contrainte (2) implique $S_{ik} \geq S_{i',k'+1}$, ce qui assure le respect de la capacité des ressources entre O_{ik} et $O_{i'k'}$. Enfin, les contraintes (4) permettent de calculer la valeur du maksepan C_{max} puisque $O_{in_i} = (\varnothing, 0)$.

Comme nous l'avons mentionné dans le chapitre 2, la contrainte (5) exprimant l'absence de blocage, est difficilement modélisable mathématiquement. Nous verrons dans la section 10, que nous pouvons exprimer mathématiquement cette contrainte en utilisant une extension de la représentation classique avec le graphe disjonctif (Roy et Susmann [1964]), que nous proposons dans la section 5.

4. Problème d'ordonnancement des trains

Durant ces dix dernières années, le problème d'ordonnancement des trains a reçu une grande attention de la part des chercheurs (Cordeau et al. [1998]). En effet, la majorité des compagnies ferroviaires ont besoin des systèmes de contrôle fournissant des informations en temps réel sur la position et la vitesse des trains.

L'objectif principal est de diminuer la consommation d'énergie et de minimiser les retards des trains (Hallowell et Harker [1998]). Ces objectifs peuvent être atteints en développant des bons plannings hors ligne pour les trains. En revanche, les évènements incertains qui peuvent se produire tels que l'indisponibilité d'une ressource (conducteur, wagons, signaux, etc.), modifient l'ordonnancement. De plus, une stratégie en temps réel permettant de modifier l'ordonnancement hors ligne doit être développée, afin d'éliminer les

situations des collisions (blocage) qui peuvent se produire. Par conséquent, il est nécessaire de disposer d'un algorithme de réordonnancement (voir par exemple Adenso-Diaz et al. [1999]).

Un état de l'art sur les problèmes de routage et d'ordonnancement rencontrés par les compagnies ferroviaires est donné par Cordeau et al. [1998]. Vu la complexité de ces problèmes, seuls des cas particuliers avec uniquement une ou deux lignes, sont considérés (voir par exemple, Cai et al. [1998]).

4.1. Problématique

Un réseau ferroviaire est composé de stations, signaux et chemin « *track line* ». Les signaux permettent de contrôler le trafic dans le réseau, et ainsi évitent les collisions de trains. Ces signaux se trouvent généralement avant l'arrivée à une station, à la jonction de deux chemins, tout au long d'un chemin, etc. Un chemin entre deux signaux est appelé *block* (figure 3.1).

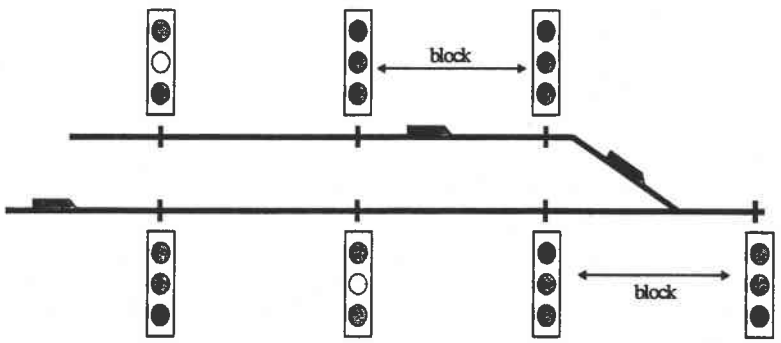


Figure 3.1 : Système de signalisation et blocks

Il est à noter que les stratégies utilisées pour manipuler les signaux, sont différentes d'un pays à l'autre. Nous décrivons ici, le système de signalisation italien décrit par Pacciarelli et Pranzo [2001], utilisé tout au long d'un chemin. Un signal peut être rouge, jaune ou vert. Un signal rouge signifie que le block suivant est soit indisponible, soit occupé par un autre train. Le signal jaune signifie que le block suivant est libre mais celui qui vient juste après est occupé. En revanche, le signal vert implique que les deux prochains blocks sont libres.

Ainsi, l'entrée dans un block dépend de la vitesse du train et la couleur du signal. Par exemple, les trains les moins rapides peuvent entrer dans un block seulement si le signal est vert ou jaune, alors que les trains rapides ne peuvent entrer dans un block que si la couleur du signal est verte, c'est-à-dire, les deux prochains blocks sont libres. En utilisant ce système de signalisation, nous avons :

Observation 1 *: un block ne peut contenir qu'un seul train à la fois.*

Pour parcourir un block entre deux signaux donnés, le train a besoin d'un temps connu à l'avance. Ceci est donné par l'observation suivante :

Observation 2 : le temps de parcours d'un block par un train est connu.

Il est clair que si un train moins rapide (resp. rapide) arrive à la fin d'un block et trouve le signal rouge (resp. rouge ou jaune), il doit attendre jusqu'à ce que le block (resp. les deux blocks) suivant (s) soit (soient) disponible (s). Cette contrainte correspond exactement à la propriété retenir et attendre. En d'autres termes,

Observation 3 : un block est retenu par le train jusqu'à ce que les blocks suivants soient disponibles.

4.2. Correspondance avec le modèle JSSB

Si on considère un block comme une ressource qui ne peut exécuter qu'une seule opération à la fois, le problème d'ordonnancement de train peut être explicitement modélisé par le modèle de *JSSB*, pour lequel une opération nécessite seulement une seule ressource (block). Plus précisément :

- *Un train correspond à un job dans le modèle JSSB.*
- *Un block correspond à une ressource dans le modèle JSSB. Il est évident què tous les blocks sont différents les un des autres.*
- *Le temps de parcours d'un block correspond au temps nécessaire pour exécuter une opération.*
- *Le routage utilisé (succession linéaire des blocks) par le train correspond à la gamme opératoire du job correspondant.*

Dans la section suivante, nous introduirons l'extension du graphe disjonctif. Ensuite, nous donnerons un exemple d'un petit réseau ferroviaire pour expliquer comment modéliser le problème d'ordonnancement.

5. Les modèles du graphe disjonctif

La représentation classique avec le graphe disjonctif a montré son efficacité pour représenter le problème d'ordonnancement de type job shop, ainsi que pour d'autres extensions de ce modèle (voir le chapitre 1). Dans cette section, nous donnons une nouvelle extension du graphe disjonctif pour représenter la propriété *retenir et attendre* du modèle *JSSB*. Dans la suite, nous utilisons l'exemple illustratif suivant à trois jobs et trois ressources pour illustrer les propriétés introduites :

$$J_1 = \{(M_1, M_2, 1), (M_3, 1), (\varnothing, 0)\}.$$

$$J_2 = \{(M_1, 2), (M_3, 1), (\varnothing, 0)\}.$$

$$J_3 = \{(M_1, 2), (M_2, 1), (\varnothing, 0)\}.$$

5.1. Graphe disjonctif sans blocage

Le graphe disjonctif pour un job shop multi-ressources sans blocage et avec ressources unitaires (noté *DGW*) est une extension directe du graphe disjonctif classique introduit par Roy et Susmann [1964]. Plus précisément, chaque opération est représentée par un sommet. Deux sommets fictifs 0 et * appelés respectivement la source et le puit sont ajoutés. Ces deux sommets correspondent respectivement, au début et à la fin des opérations. Il y a deux types d'arcs: les arcs conjonctifs et les arcs disjonctifs. L'ensemble des arcs conjonctifs comporte tous les arcs reliant deux opérations consécutives du même job. Ces arcs permettent de modéliser les contraintes de précédence entre les opérations du même job. De plus, le sommet fictif 0 est connecté via un arc conjonctif à la première opération de chaque job. De même, la dernière opération de chaque job est connectée au sommet fictif *.

Chaque arc disjonctif relie deux opérations partageant une ressource commune. Le poids de chaque arc (conjonctif ou disjonctif) est égal à la durée opératoire de l'opération située à l'extrémité initiale, exceptés les arcs conjonctifs partant de la source pour lesquels le poids est égal à zéro. Cette valeur signifie que tous les jobs sont disponibles à l'instant zéro.

Le graphe disjonctif pour l'exemple illustratif est donné par la figure 3.2.

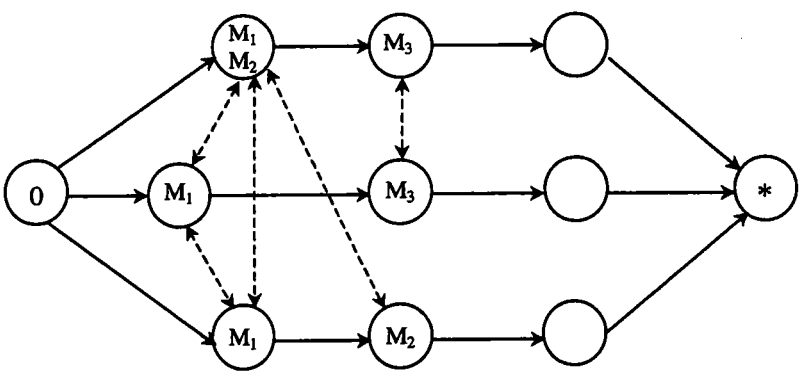


Figure 3.2 : Graphe disjonctif sans blocage

Comme dans le modèle du job shop classique, un ordonnancement réalisable est obtenu en choisissant une orientation pour chaque arc disjonctif, afin que le graphe obtenu soit sans circuits. Le makespan de la solution obtenue est donné par la longueur du plus long chemin du sommet source au sommet puit, appelé *chemin critique.*

La figure 3.3 donne le graphe disjonctif correspondant à l'ordonnancement suivant :

$$O_{11} \succ O_{21}, O_{11} \succ O_{31}, O_{11} \succ O_{32}, O_{12} \succ O_{22}, O_{21} \succ O_{31}$$

où "$a \succ b$" signifie "a précède b".

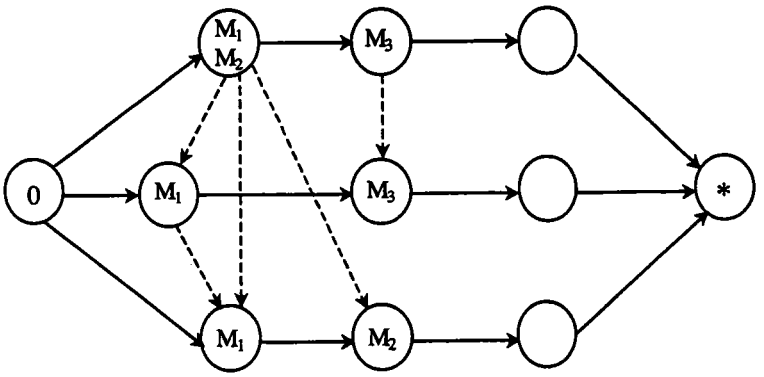


Figure 3.3 : Une sélection dans le graphe disjonctif sans blocage

5.2. Graphe disjonctif avec blocage

Le graphe disjonctif du modèle JSSB, appelé graphe disjonctif avec blocage (noté *DGB*) est obtenu à partir du graphe disjonctif sans blocage en modifiant la définition des arcs disjonctifs. Plus précisément, pour chaque arc disjonctif dans le graphe sans blocage entre les sommets O_{ik} et $O_{i'k'}$, c'est-à-dire, pour toute paire d'opérations qui partagent une ressource commune, nous introduisons un *couple disjonctif* de deux arcs défini de la manière suivante :

Définition 3.1 : Le couple disjonctif entre deux opérations O_{ik} et $O_{i'k'}$ dans le graphe disjonctif avec blocage, est défini de la manière suivante :

- Le premier arc démarre du sommet $O_{i,k+1}$ et finit au sommet $O_{i'k'}$,
- Le deuxième arc démarre du sommet $O_{i',k'+1}$ et finit au sommet O_{ik} ,
- Le poids de ces deux arcs est égal à zéro (figure 3.4).

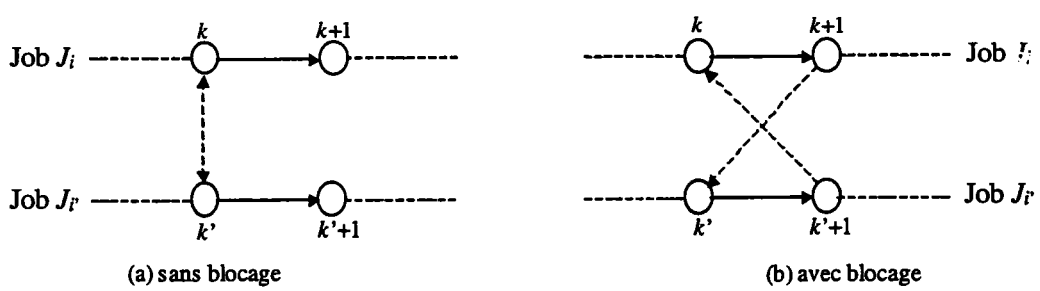


Figure 3.4 : Différence entre le graphe sans et avec blocage

La figure 3.5 présente le graphe disjonctif avec blocage pour l'exemple illustratif. On note par exemple, que l'arc disjonctif (O_{11}, O_{32}) dans la figure 3.2 est remplacé par le couple disjonctif {(O_{12}, O_{32}), (O_{33}, O_{11})} dans la figure 3.5.

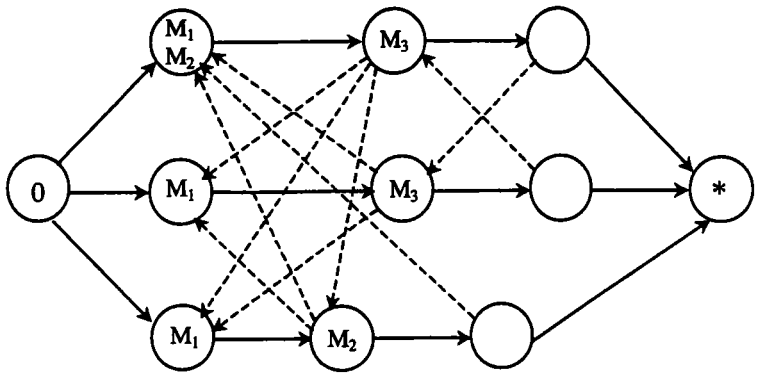


Figure 3.5 : Graphe disjonctif avec blocage

Les solutions réalisables sont obtenues en choisissant un arc parmi chaque couple disjonctif. Pour le couple disjonctif de O_{ik} et $O_{i'k'}$, l'arc $(O_{i,k+1}, O_{i'k'})$ est choisi si O_{ik} précède $O_{i'k'}$ et l'arc $(O_{i',k'+1}, O_{ik})$ est retenu dans le cas contraire. Ceci montre que les ressources retenues par une opération d'un job deviennent disponibles seulement après le passage à l'opération suivante du même job. En d'autres termes, la contrainte retenir et attendre est bien modélisée dans le graphe disjonctif avec blocage.

La figure 3.6, donne le graphe disjonctif avec blocage pour la même sélection choisie dans la figure 3.3.

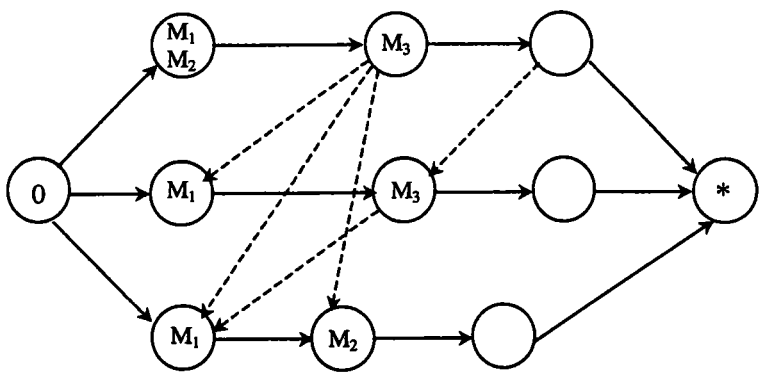


Figure 3.6 : Une sélection dans le graphe disjonctif avec blocage

Si l'orientation des couples disjonctifs est mal choisie, deux types de circuits peuvent se produire dans le graphe disjonctif avec blocage :

(i) *Les circuits contenant des arcs conjonctifs* : ces types de circuits peuvent aussi apparaître dans le graphe sans blocage, correspond à une sélection incompatible vis-à-vis des contraintes de précédence,

(ii) *Circuits engendrés seulement par des arcs disjonctifs* : ces circuits correspondent aux situations de blocage, et chaque circuit est un cycle de type "retenir et attendre".

Exemple 3.1 : Considérons un exemple à deux jobs et deux ressources. Les gammes opératoires sont les suivantes :

$$J_1 = \{(M_1, 1), (M_2, 1)\}$$

$$J_2 = \{(M_2, 1), (M_1, 1)\}$$

Si, nous considérons l'ordonnancement $\{S_{11} = 0, S_{12} = 1, S_{21} = 0, S_{22} = 1\}$ qui consiste à ordonnancer l'opération O_{11} avant O_{22} sur la ressource M_1, et ordonnancer O_{21} avant O_{12} sur la ressource M_2, la représentation du graphe disjonctif avec blocage pour cette sélection est donnée par la figure 3.7.

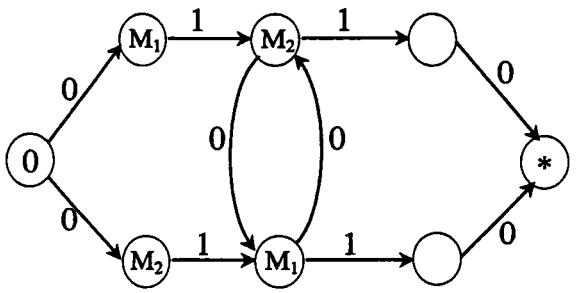


Figure 3.7 : Une situation de blocage dans le graphe disjonctif avec blocage

Le circuit O_{12} O_{22} O_{12} correspond à une situation de blocage, ce qui implique que l'ordonnancement proposé conduit lui-même à un blocage. Il est clair que cette situation se présente à l'instant $t = 1$ puisque à ce moment, le job J_1 a besoin de M_2 pour libérer M_1 et le job J_2 a besoin de M_1 pour libérer M_2.

5.3. Propriétés des graphes disjonctifs

L'objectif de cette section est d'établir les propriétés des deux graphes disjonctifs que nous utiliserons dans la suite pour chercher les ordonnancements optimaux. La méthode utilisée est similaire aux nombreuses heuristiques basées sur le graphe disjonctif (voir par exemple Van Laarhoven et al. [1992]). Ces heuristiques sont des méthodes itératives qui partent d'une solution initiale et essayent de l'améliorer en utilisant des perturbations locales. Ces perturbations sont basées sur la structure de voisinage pour une solution particulière. Ce voisinage est une fonction qui transforme une solution en une autre en appliquant des perturbations appelées mouvements. La définition de la structure du voisinage est importante pour l'efficacité des méthodes de recherche locale.

Pour le cas du job shop classique, la perturbation la plus populaire est basée sur le concept du chemin critique, et consiste à permuter/insérer les opérations utilisant la même machine, sur le chemin critique. Ce type de voisinage a les propriétés suivantes:

(i) Chaque solution voisine est réalisable,

(ii) A partir de toute solution initiale, la solution optimale peut être atteinte après un nombre fini de perturbations,

Ces deux propriétés sont très importantes pour le succès d'une méthode de recherche locale. La première propriété (i), garantissant que toute solution voisine est réalisable, permet de négliger le test de faisabilité, et ainsi d'accélérer la recherche. La deuxième propriété (ii) est relative à la convergence des méthodes et est connue sous le nom de *propriété de connectivité*. Pour une discussion détaillée sur les structures de voisinage appliquées au problème du job shop classique, nous renvoyons le lecteur à Jain [1998].

On peut facilement voir en utilisant des arguments similaires à ceux de Van Laarhoven et al. [1992] que les propriétés (i) et (ii) sont aussi vérifiées pour le graphe disjonctif sans blocage. C'est ce qui est décrit dans le théorème suivant :

Théorème 3.1

Pour toute solution réalisable initiale du graphe disjonctif sans blocage, la permutation de tout arc disjonctif sur le chemin critique conduit à une solution voisine réalisable, et la solution optimale peut être obtenue après un nombre fini de permutations. De plus, la permutation d'un arc disjonctif $(O_{ik}, O_{i'k'})$ n'augmente pas le makespan si et seulement si:

$$L^N(0, O_{ik}) + L^N(O_{ik}, *) \leq L^O(0, *) \quad (6)$$

$$\text{et } L^N(0, O_{i'k'}) + L^N(O_{i'k'}, *) \leq L^O(0, *) \quad (7)$$

où $L^O(x, y)$ (resp. $L^N(x, y)$) est la longueur du plus long chemin du sommet x au sommet y avant (resp. après) la permutation de l'arc disjonctif.

Les termes qui apparaissent dans le théorème 3.1 peuvent être déterminés de la manière suivante :

$$L^N(0, O_{i'k'}) = Max_{s \in In^O(O_{i'k'}) \wedge s \neq O_{ik}} \{L^O(0, s) + w(s, O_{i'k'})\}$$

$$L^N(0, O_{ik}) = Max\{L^O(0, O_{ik}), L^N(0, O_{i'k'}) + w(O_{i'k'}, O_{ik})\}$$

$$L^N(O_{ik}, *) = Max_{s \in Out^O(O_{ik}) \wedge s \neq O_{i'k'}} \{L^O(s, *) + w(O_{ik}, s)\}$$

$$L^N(O_{i'k'}, *) = Max\{L^O(O_{i'k'}, *), L^N(O_{ik}, *) + w(O_{i'k'}, O_{ik})\}$$

où :

$In^O(x)$ (resp. $Out^O(x)$) est l'ensemble de prédécesseurs (resp. successeurs) du sommet x dans le graphe disjonctif initial,

$w(x, y)$ est le poids de l'arc (x, y).

Preuve

Nous considérons une solution réalisable et une permutation d'un arc disjonctif $(O_{ik}, O_{i'k'})$ sur le chemin critique. Supposons que le graphe disjonctif obtenu après la permutation contienne un circuit. Alors il existe un chemin de O_{ik} à $O_{i'k'}$. Ce chemin existe aussi dans le graphe disjonctif initial. De plus, la longueur de ce chemin est strictement inférieure à p_{ik} puisque ce chemin contient au moins deux arcs disjonctifs et il n'existe pas d'opérations de durée opératoire nulle. Ceci est une contradiction car l'arc disjonctif $(O_{ik}, O_{i'k'})$ appartient au chemin critique dans le graphe initial, et par conséquent la portion entre O_{ik} et $O_{i'k'}$ est optimale.

Pour le problème de job shop classique, Van Laarhoven et al. [1992] définissent la structure de voisinage comme étant la permutation de deux arcs disjonctifs consécutifs sur le chemin critique, et montrent que le voisinage est connexe. En utilisant les mêmes

arguments, nous pouvons démontrer que la permutation de deux arcs consécutifs sur le chemin critique dans le graphe sans blocage, définit bien un voisinage connexe.

Si la permutation $(O_{ik}, O_{i'k'})$ n'augmente pas le makespan, alors la longueur du chemin de la source 0 au puit * dans le graphe disjonctif obtenu est inférieure ou égale à $L^O(0,*)$. C'est le cas pour les chemins utilisant les sommets O_{ik} et $O_{i'k'}$. En conséquence, les relations (6) et (7) sont satisfaites. D'autre part, si l'une des équations (6) ou (7) n'est pas satisfaite (par exemple (6)), alors dans le pire des cas, le chemin critique a la même longueur que le plus long chemin utilisant le sommet O_{ik}, c'est-à-dire, $L^N(0,O_{ik})+L^N(O_{ik},*)$ qui est plus grand que $L^O(0,*)$ et le makespan augmente. ■

Malheureusement, les propriétés (i) et (ii) ne sont pas satisfaites dans le cas du graphe disjonctif avec blocage. La différence principale vient du fait que la permutation entre deux opérations consécutives sur le chemin critique consiste à inverser l'arc disjonctif qui les relie dans le graphe sans blocage, alors qu'elle consiste à remplacer le premier arc disjonctif par le second dans le graphe disjonctif avec blocage. Par exemple, la permutation de O_{11} et O_{21} sur le chemin critique dans le graphe disjonctif avec blocage de la figure 3.6, remplace l'arc (O_{12}, O_{21}) par l'arc (O_{22}, O_{11}) et crée un circuit O_{11} O_{12} O_{13} O_{22} O_{11}. Par conséquent, la solution obtenue n'est pas admissible.

Néanmoins, si la permutation dans le graphe disjonctif avec blocage ne crée par des circuits, nous avons :

Théorème 3.2

Etant donnée une séquence d'entrée des opérations pour chaque ressource, si le graphe disjonctif avec blocage n'admet pas de circuits, alors le graphe disjonctif sans circuit n'en admet pas. De plus,

$$L^{DGB}(0,*) \geq L^{DGW}(0,*)$$

où $L^{DGB}(0,*)$ (resp. $L^{DGW}(0,*)$) est la longueur du chemin critique dans le graphe disjonctif avec blocage (resp. sans blocage). L'égalité se produit dans le cas où le chemin critique n'utilise pas deux arcs disjonctifs consécutifs.

En utilisant les théorèmes 3.1 et 3.2, une estimation du makespan dans le graphe disjonctif avec blocage après une permutation admissible peut être déterminée comme suit:

$$L^{DGB}(0,*) \geq L^{DGW}(0,*) \geq Max\left\{L^{DGW}(0,O_{ik})+L^{DGW}(O_{ik},*), L^{DGW}(0,O_{i'k'})+L^{DGW}(O_{i'k'},*)\right\}.$$

Preuve

Soit P un chemin du sommet x au sommet y dans le modèle *DGW*. Considérons un arc disjonctif $(O_{ik}, O_{i'k'})$ de poids p_{ik} sur ce chemin. Dans le pire des cas, cet arc disjonctif peut être remplacé par le chemin $O_{ik}\, O_{i,k+1}\, O_{i'k'}$ dans le graphe avec blocage *DGB*. Ainsi, la longueur du plus long chemin de O_{ik} à $O_{i'k'}$ dans le graphe *DGB*, est au moins égale à p_{ik}.

En remplaçant tous les arcs disjonctifs appartenant à P, nous construisons un nouveau chemin P' dans DGB tel que $L^{DGB}(x,y) \geq L^{DGW}(x,y)$. Si $x = 0$ et $y = *$ alors, nous avons $L^{DGB}(0,*) \geq L^{DGW}(0,*)$.

Si le chemin critique n'utilise pas deux arcs disjonctifs consécutifs, alors il consiste en une succession d'arcs conjonctifs et d'arcs disjonctifs. Chaque arc disjonctif $(O_{i,k+1}, O_{i'k'})$ est précédé par un arc conjonctif. La portion $O_{ik}\, O_{i,k+1}\, O_{i'k'}$ a la même longueur que l'arc disjonctif $(O_{ik}, O_{i'k'})$ dans le modèle DGW. En remplaçant toutes les portions $O_{ik}\, O_{i,k+1}\, O_{i'k'}$ du chemin critique par les arcs disjonctifs $(O_{ik}, O_{i'k'})$, nous créons un autre chemin critique dans le modèle DGW avec $L^{DGB}(0,*) = L^{DGW}(0,*)$. ■

Dans le cas général, l'égalité n'est pas toujours satisfaite comme nous pouvons le montrer par le contre-exemple suivant :

$$J_1 = \{(M_1, \alpha), (\varnothing, 0)\}$$
$$J_2 = \{(M_3, 1), (M_1, 1), (\varnothing, 0)\}$$
$$J_3 = \{(M_2, 1), (M_3, \alpha), (\varnothing, 0)\}$$

avec une valeur de $\alpha \geq 2$.

L'ordonnancement que nous considérons est $O_{11} \succ O_{22}, O_{21} \succ O_{32}$ correspondant au chemin critique 0 O_{11} O_{12} O_{22} O_{32} O_{33} * avec deux arcs disjonctifs consécutifs (O_{12}, O_{22}) et (O_{22}, O_{32}) dans le graphe disjonctif avec blocage (figure 3.8).

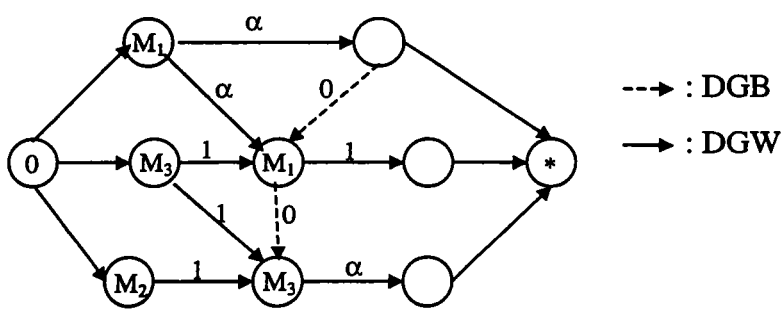


Figure 3.8 : Contre-exemple

Il est facile de vérifier que la longueur du chemin critique dans le graphe disjonctif sans blocage est égale à α+1, alors qu'elle est égale à 2α dans le graphe disjonctif avec blocage.

6. Modélisation d'un réseau ferroviaire

Dans cette section, nous allons valider l'extension du graphe disjonctif présentée dans la section 5 sur des problèmes d'ordonnancements rencontrés dans le secteur ferroviaire. Pour ce faire, nous nous basons sur l'exemple donné par la figure 3.9 représentant une station de train.

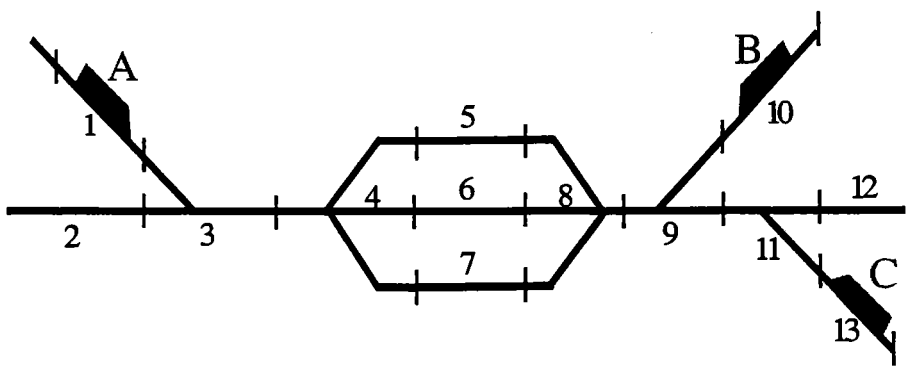


Figure 3.9 : Un réseau ferroviaire

Dans l'exemple de la figure 3.9, le train *A* est un train de marchandises qui doit passer sans arrêt (*no-wait*) du block 1 au block 12, en utilisant le quai 7. Le temps nécessaire pour le parcours est égal à α. Le train *B* doit passer du block 10 au block 2 en passant par le quai 5. Enfin, le train *C* doit aller du block 13 au block 2 en empruntant le quai 6.

Le modèle proposé doit permettre une résolution du problème de conflit des blocks pour le passage des trains (figure 3.10).

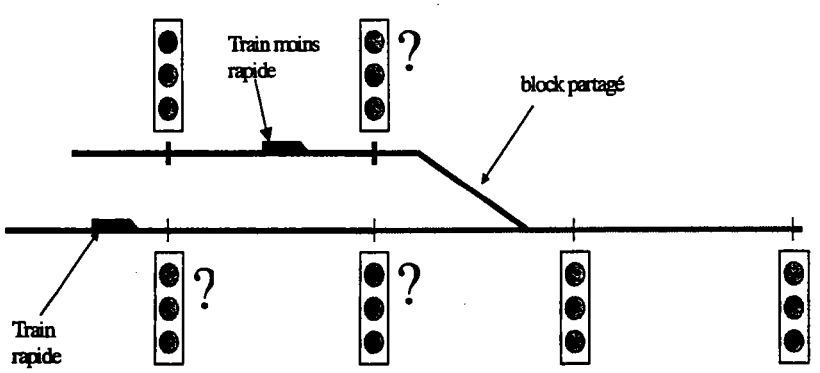


Figure 3.10 : Conflit des blocks

Pour résoudre le problème de conflit, nous modélisons chaque block par un sommet du graphe disjonctif. La figure 3.11 montre le graphe disjonctif modélisant la situation du conflit de la figure 3.10. Les deux sommets *i* et *j* correspondent au block partagé entre les deux trains.

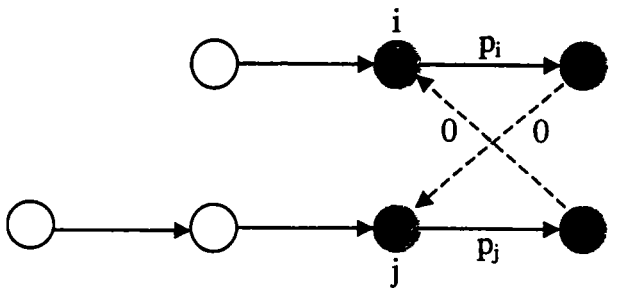


Figure 3.11 : Représentation par le graphe disjonctif

En utilisant le même principe pour décrire tous les conflits des blocks, nous obtenons le modèle du graphe disjonctif (figure 3.12) modélisant le réseau illustré par la figure 3.9. Notons que dans cette représentation, l'arc entre le sommet 12 et le sommet 1 ayant une longueur négative ($-\alpha$) est ajouté pour modéliser la contrainte *no-wait*.

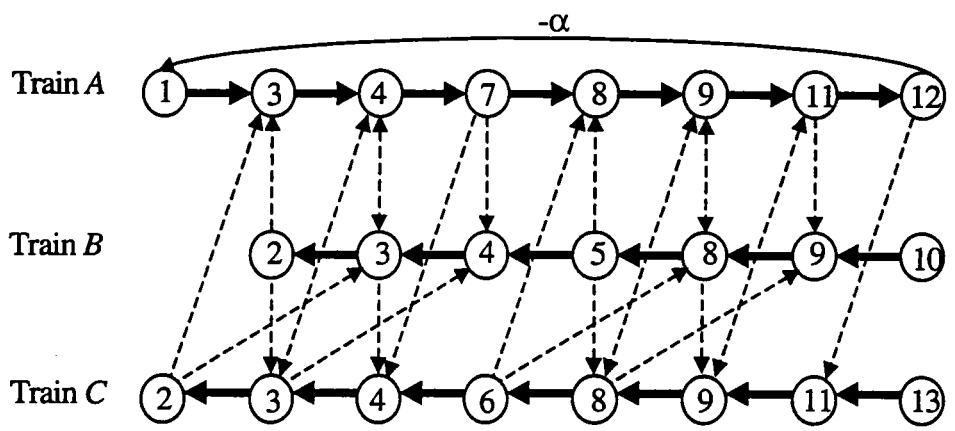


Figure 3.12 : Modélisation avec le graphe disjonctif

Remarque 3.1 : D'autres contraintes plus difficiles rencontrées en pratique, comme par exemple la date de départ au plus tard d'un train d'une station, peuvent être intégrées et modélisées par le nouveau modèle du graphe disjonctif.

7. Détection et recouvrement de blocage

A partir des discussions de la section précédente, deux problèmes sont rencontrés si nous appliquons une méthode de recherche locale au problème d'ordonnancement de *JSSB*. Le premier problème est la *vérification de la réalisabilité* de la solution obtenue après la permutation, c'est-à-dire, la détection des situations de blocage dans le nouvel ordonnancement. Le second problème appelé *recouvrement de blocage*, consiste à sortir d'une situation du blocage lorsque toutes les permutations du voisinage sont irréalisables.

7.1. Détection de blocage

Le problème de la détection du blocage dans notre cas, consiste à répondre à la question suivante :

Question 1 : *Soient un ordonnancement réalisable et une permutation* $(O_{ik}, O_{i'k'})$ *sur le chemin critique, l'ordonnancement obtenu est-il sans blocage?*

Plus précisément, est-ce que le graphe disjonctif avec blocage obtenu en enlevant le premier arc disjonctif $(O_{i,k+1}, O_{i'k'})$ et en ajoutant le deuxième arc disjonctif $(O_{i',k'+1}, O_{ik})$, est sans circuit (figure 3.13) ?.

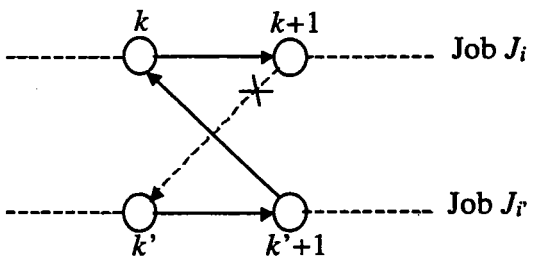


Figure 3.13 : Permutation d'un arc

Nous notons par DGB^O le graphe disjonctif avec blocage initial et par DGB' le graphe obtenu en supprimant l'arc disjonctif $(O_{i,k+1}, O_{i'k'})$ mais sans ajouter l'arc disjonctif $(O_{i',k'+1}, O_{ik})$. La question précédente concernant la détection du blocage peut être reformulée d'une manière équivalente comme suit:

Question 1' : *Existe-il un chemin de* O_{ik} *à* $O_{i',k'+1}$ *dans le graphe DGB' ?*

L'algorithme de détection du blocage que nous proposons consiste à répondre à la question 1'. Il explore en profondeur les sommets qui peuvent être atteints dans DGB' en partant du sommet O_{ik}. Il est clair qu'il existe un grand nombre de sommets à explorer. Pour remédier à cet inconvénient, nous proposons trois critères d'arrêt qui permettent de stériliser l'exploration d'un sommet qui ne conduit pas au sommet $O_{i',k'+1}$. Les critères d'arrêt utilisent la longueur du plus long chemin $L^O(x, y)$ dans le graphe DGB^O et le concept de niveau $l^O(x)$ associé à chaque sommet de DGB^O.

Puisque le graphe DGB^O est acyclique, il est possible de partitionner ses sommets en niveaux tels que tous les chemins vont d'un sommet de niveau l à un autre sommet de niveau l' avec $l < l'$. Les critères d'arrêt sont les suivants :

(i) $l^O(y) \geq l^O(O_{i',k'+1}) \Rightarrow \nexists$ de chemins de y à $O_{i',k'+1}$ dans DGB'.

(ii) $L^O(0, y) > L^O(0, O_{i',k'+1}) \Rightarrow \nexists$ de chemins de y à $O_{i',k'+1}$ dans DGB'.

(iii) $L'(O_{ik}, y) > L^O(O_{ik}, O_{i',k'+1}) \Rightarrow \nexists$ de chemins de y à $O_{i',k'+1}$ dans DGB'.

7.2. Recouvrement de blocage

Comme nous l'avons mentionné, contrairement au problème d'ordonnancement des jobs shops classiques pour lesquels la permutation conduit toujours à des solutions réalisables, des situations de blocage apparaissent fréquemment lorsque nous utilisons les méthodes de recherche locale pour résoudre le problème d'ordonnancement de *JSSB*. En plus, il existe des situations pour lesquelles toutes les permutations sur le chemin critique conduisent à des solutions irréalisables.

Pour traiter le cas des permutations (O_{ik}, $O_{i'k'}$) aboutissant à un blocage, un algorithme de recouvrement est nécessaire pour sortir de telles situations, en imposant bien sûr la permutation (O_{ik}, $O_{i'k'}$). L'algorithme que nous proposons est un algorithme de recouvrement par réordonnancement. Il consiste à ordonnancer le job J_i en imposant la permutation (O_{ik}, $O_{i'k'}$), c'est-à-dire, imposer que $O_{i'k'}$ précède O_{ik}.

7.2.1. Méthodologie suivie

L'idée principale de l'algorithme de recouvrement consiste à réordonnancer le job J_i sans altérer complètement la structure de la solution courante. En plus, la performance du système est maintenue en tenant compte des durées opératoires des opérations dans l'algorithme. Ceci est réalisé d'abord, en gardant la même séquence d'entrée dans les ressources, pour tous les jobs $\mathcal{J} \setminus J_i$, c'est-à-dire, en maintenant la même sélection des couples disjonctifs dans le graphe disjonctif avec blocage pour les jobs $\mathcal{J} \setminus J_i$. Ensuite, nous ordonnançons les opérations du job J_i en intégrant la contrainte exprimée par la permutation (O_{ik}, $O_{i'k'}$).

Sur la base des mêmes arcs disjonctifs pour les jobs $\mathcal{J} \setminus J_i$, nous pouvons construire un job composé J_{com} en regroupant dans une seule opération, les opérations des jobs $\mathcal{J} \setminus J_i$ pouvant s'exécuter en même temps. En conséquence de cela, le problème du recouvrement de blocage se ramène au problème d'ordonnancement de deux jobs J_{com} et J_i avec la contrainte de précédence imposée par la permutation, c'est-à-dire., $O_{com,u} \succ O_{ik}$ où $O_{com,u}$ correspond à la fin de l'opération $O_{i'k'}$.

7.2.2. Algorithme de réordonnancement

Le problème d'ordonnancement à deux jobs J_{com} et J_i peut être résolu polynomialement en utilisant une extension de l'algorithme géométrique proposée dans le chapitre 2, qui permet de prendre en compte la contrainte imposée par la permutation. Pour faciliter la description, nous expliquons cet algorithme sur l'exemple illustratif donné dans la section 5 (figure 3.6). Dans cette figure, la permutation des opérations O_{11} et O_{21} consiste à remplacer l'arc (O_{12}, O_{21}) par l'arc (O_{22}, O_{11}), et crée le circuit O_{11} O_{12} O_{13} O_{22} O_{11}. Dans ce cas, l'algorithme de recouvrement doit être utilisé avec $J_i = J_1$ et $\mathcal{J} \setminus J_i = \{J_2, J_3\}$.

L'algorithme de recouvrement est composé de 2 étapes principales :

A) Construction du job composé

Cette étape consiste à construire le job composé J_{com} des jobs $\mathcal{J} \setminus J_i$. Cette notion a été introduite dans le chapitre 2, et est résumée aux quatre étapes suivantes:

1) Obtention du graphe disjonctif avec blocage des jobs $\mathcal{J} \setminus J_i$ (figure 3.14),

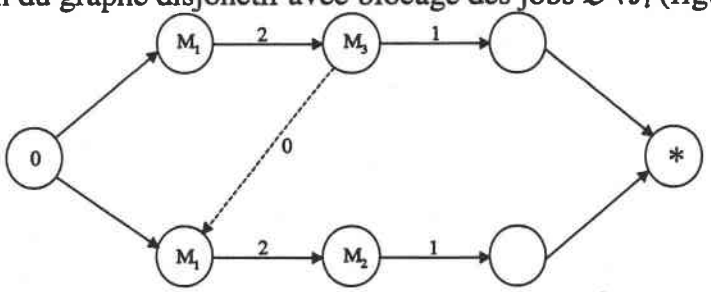


Figure 3.14 : Graphe disjonctif des jobs $\mathcal{J} \setminus J_i$

2) Détermination de l'ordonnancement correspondant en utilisant le chemin critique. Le diagramme de Gantt est donné par la figure 3.15,

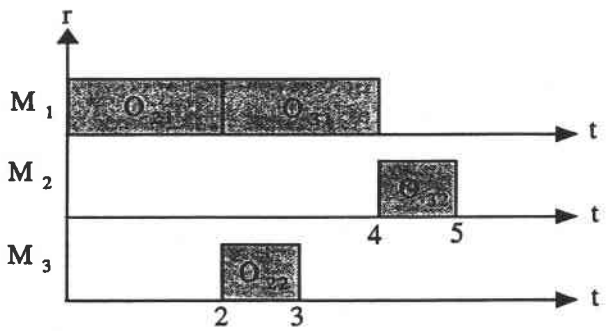


Figure 3.15 : Diagramme de Gantt des jobs $\mathcal{J} \setminus J_i$

3) Décomposition de l'axe de temps en des intervalles afin qu'aucune opération ne démarre/termine à l'intérieur d'un intervalle. Pour l'exemple illustratif, quatre intervalles sont obtenus : [0, 2], [2, 3], [3, 4] et [4, 5],

4) Association à chaque intervalle, d'une opération du job composé, avec les ressources occupées durant l'intervalle comme les ressources nécessaires pour l'opération. Pour l'exemple $J_{com} = \{(M_1, 2), (M_1, M_2, 1), (M_1, 1), (M_2, 1)\}$.

Le problème de réordonnancement consiste alors à ordonnancer les deux jobs J_{com} et $J_i = \{(M_1, M_2, 1), (M_3, 1), (\varnothing, 0)\}$ sous la contrainte de précédence $O_{com,1} \succ O_{11}$ imposée par la permutation. Ce problème peut être résolu en utilisant une extension de l'algorithme géométrique proposé dans le chapitre 2. Cet algorithme transforme le problème d'ordonnancement en la recherche du plus court chemin dans le plan. Cette recherche du plus court chemin est présentée dans le paragraphe suivant :

B) Extension de l'algorithme géométrique

1) Représentation des jobs dans le plan (figure 3.16). Chaque job est représenté sur un axe suivant sa gamme opératoire. J_{com} est représenté sur l'axe X et J_1 sur l'axe Y,

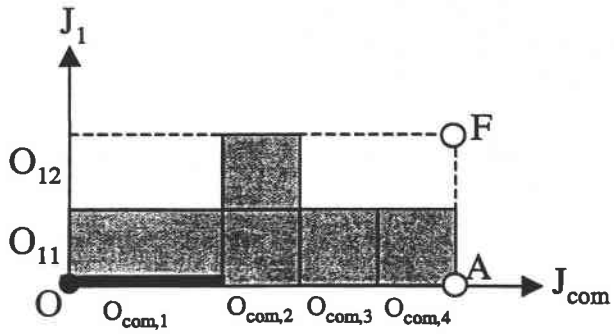


Figure 3.16 : Représentation géométrique du problème de réordonnancement

2) Création des obstacles pour chaque rectangle $(O_{ik}, O_{com,k'})$ tel que O_{ik} et $O_{com,k'}$ partagent une ressource commune. Les obstacles $(O_{11}, O_{com,1})$, $(O_{11}, O_{com,2})$, $(O_{11}, O_{com,3})$, $(O_{11}, O_{com,4})$, et $(O_{12}, O_{com,2})$ sont crées pour l'exemple illustratif,

3) Création des obstacles relatifs aux situations de blocage en utilisant l'algorithme de programmation dynamique modifié, qui a été présenté dans le chapitre 2. Aucun obstacle n'est crée pour l'exemple,

4) Imposition de la relation de précédence $O_{com,u} \succ O_{ik}$. Pour cela, nous créons un obstacle « *ligne horizontale* » sur la ligne de O_{ik} en démarrant de $X = 0^-$ et en terminant à la bordure droite de l'opération $O_{com,u}$. Pour l'exemple illustratif, le nouvel obstacle est représenté par une ligne en trait gras dans la figure 3.16.

Une fois que la représentation du problème d'ordonnancement dans le plan avec obstacles est déterminée, l'algorithme polynomial (*algorithme 3*) proposé dans le chapitre 2 est utilisé pour trouver le meilleur ordonnancement. Pour cet exemple, l'ordonnancement trouvé est représenté par la figure 3.17. Nous notons que dans ce cas, la solution est simple à obtenir puisqu'il existe un seul chemin O A F qui permet d'atteindre le point final F à partir de l'origine O.

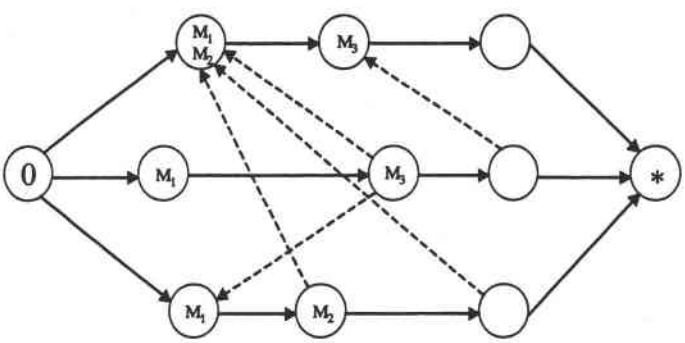


Figure 3.17 : Résultat du recouvrement de blocage

8. Algorithme de recherche tabou

Durant les dix dernières années, la recherche tabou (Glover [1997]) a été utilisée fréquemment pour résoudre les problèmes difficiles d'optimisation combinatoire rencontrés dans la pratique (Osman et Laporte [1996]). Le succès de cette méthode est dû à sa capacité d'explorer l'espace de solutions sans qu'elle soit piégée par des optima locaux, tout en empêchant le phénomène de cyclage. Le succès de cette méthode sur les problèmes d'ordonnancement du type job shop (Nowicki et Smutnicki [1996]), ainsi que d'autres extensions (Dauzère-Pérès et Paulli [1997]), a motivé le choix de cette heuristique pour résoudre le problème d'ordonnancement de JSSB étudié dans ce chapitre.

8.1. Paramètres utilisés dans la méthode

Les paramètres de la méthode sont : la solution initiale, la structure de voisinage, la restriction tabou, la taille de la liste, le critère d'aspiration et enfin l'évaluation des mouvements. Dans notre application, la recherche tabou est appliquée avec les paramètres suivants:

A. Solution initiale

Elle est déterminée d'une manière très simple, qui consiste à ordonnancer les jobs séquentiellement un après l'autre. En d'autres termes, toutes les opérations du job J_i précèdent les opérations du job J_{i+1} sur les ressources communes. Il est clair que cette solution est sans blocage.

B. Structure du voisinage

La création des solutions voisines est basée sur le concept du chemin critique. En effet, deux mouvements sont définis. Le premier mouvement appelé le *mouvement de permutation*, consiste à permuter deux opérations (O_{ik}, $O_{i'k'}$) sur le chemin critique. Plus précisément, la permutation (O_{ik}, $O_{i'k'}$) remplace le premier arc disjonctif $(O_{i,k+1}, O_{i'k'})$ par le second arc $(O_{i',k'+1}, O_{ik})$. Le second mouvement appelé *mouvement de recouvrement*, est associé à une permutation (O_{ik}, $O_{i'k'}$) sur le chemin critique conduisant à une situation de blocage. Le mouvement de recouvrement est réalisé par l'algorithme de réordonnancement proposé dans la section 7. Dans notre heuristique de recherche tabou, nous donnons la priorité aux mouvements de permutation, c'est-à-dire, tant que le mouvement de permutation est possible, aucun mouvement de recouvrement n'est réalisé.

C. Restriction tabou

Deux listes tabous sont utilisées, chacune correspondant à un type de mouvement. Chaque liste tabou contient les T derniers mouvements réalisés où T est la taille de la liste. Pour le mouvement de permutation (O_{ik}, $O_{i'k'}$), l'arc disjonctif $(O_{i,k+1}, O_{i'k'})$ est ajouté dans la liste tabou, ainsi il devient interdit durant les T prochaines permutations. Pour le mouvement de

recouvrement relatif à la permutation (O_{ik}, $O_{i'k'}$) imposant la relation de précédence $O_{i'k'} \succ O_{ik}$, le couple (i, i') est ajouté à la liste tabou. Par conséquent, il interdit tout mouvement de recouvrement ($O_{i'}$, O_{il}) qui impose la relation de précédence $O_{il} \succ O_{i'l'}$. Cette restriction qui est plus forte (Glover [1993]), est due au fait que le mouvement de recouvrement (O_{ik}, $O_{i'k'}$) peut changer plusieurs arcs disjonctifs en même temps.

D. Taille de la liste

L'un des paramètres importants de la recherche tabou est la taille de la liste tabou. Cette taille dépend fortement de la restriction tabou utilisée. Plusieurs aspects pour choisir la taille de la liste existent dans la littérature (Glover [1993]). Dans notre cas, l'aspect dynamique employée par Taillard [1991] pour résoudre le problème d'affectation quadratique, est adopté. L'intérêt de cet aspect est qu'il permet d'éliminer virtuellement la probabilité de cyclage en permettant de varier la taille de la liste durant la recherche dans un intervalle de référence. Pour la restriction tabou associée au mouvement de permutation, l'intervalle de référence est fixé expérimentalement à [$N/2$, N], alors qu'il est fixé à [$N/3$, $2N/3$] pour le mouvement de recouvrement. Ces tailles sont mises à jour chaque $2N$ itérations.

E. Critère d'aspiration

Le critère d'aspiration adopté correspond au critère le plus utilisé dans la littérature, connu sous le nom du critère d'aspiration globale. Cette aspiration permet d'accepter un mouvement tabou si le mouvement conduit à une nouvelle meilleure solution.

F. Evaluation du mouvement

Pour le mouvement de permutation, comme l'évaluation exacte demande de recalculer complètement le makespan, nous utilisons la borne inférieure donnée par le théorème 3.2 pour obtenir des estimations rapides. Le mouvement de recouvrement quant à lui, est évalué par l'algorithme de réordonnancement proposé dans la section 7.

G. Critère d'arrêt

La recherche tabou est arrêtée après un nombre d'itérations maximum fixé à l'avance. Dans cette application, le nombre maximum d'itération est fixé à 10000.

H. Stratégie d'intensification

En utilisant seulement les deux mouvements de permutation et de recouvrement, la recherche tabou peut être piégée dans des régions qui ne contiennent pas la solution optimale. Ceci est dû au fait que notre voisinage n'est pas connexe. Pour remédier à cet inconvénient, nous proposons une *stratégie d'intensification* de la recherche (Glover [1995]) pour rediriger la recherche tabou dans de nouvelles régions. Cette intensification est réalisée en sauvegardant les K meilleures solutions rencontrées durant la

recherche. Lorsque l'algorithme 1 est terminé, nous recommençons la recherche tabou à partir des K meilleures solutions enregistrées. Dans notre application, les listes tabous sont vidées et la taille des listes est modifiée pour permettre de trouver de nouvelles régions. Nous notons que cette approche est très similaire à celle proposée par Nowicki et Smutnicki [1996]. En revanche, dans cette dernière application, la liste tabou n'est pas vidée lorsqu'on recommence à partir d'une solution, et ce à cause de l'utilisation d'une taille de liste statique.

En utilisant les paramètres cités ci-dessus, la méthode de résolution du problème *JSSB* utilisant la recherche tabou peut être résumée comme suit :

Algorithme 3.1 : *RechTabou*

Données:

MAX_{iter} *: le nombre maximum d'itérations*

Résultat:

C^*_{max} *: le meilleur makespan trouvé,*

S^* *: la meilleure solution,*

1. *(Initialisation)*
   - *Déterminer une solution initiale réalisable,*
   - *Poser* $C_{max} \leftarrow \infty$, *Iter* $\leftarrow 0$, *où Iter est le compteur d'itérations*
2. *Déterminer un chemin critique dans le graphe disjonctif courant. Soit* C_{max} *la longueur de ce chemin,*
3. ***Si*** $C_{max} < C^*_{max}$ ***Alors***
   - $C_{max} \leftarrow C^*_{max}$
   - *Enregistrer la solution courante S comme la meilleure solution* S^*,

   FinSi,
4. *Déterminer la meilleure permutation non tabou* δ_1 *sur le chemin critique,*
   - ***Si*** δ_1 *existe* ***Alors*** *aller à l'étape 6,*
5. *Déterminer le meilleur mouvement de recouvrement* δ_2 *sur le chemin critique,*
6. ***Si*** δ_1 *ou* δ_2 *existe,* ***Alors*** *réaliser le mouvement* δ_1 *ou* δ_2
   - *Mettre à jour les listes tabous,*
   - *Iter* $\leftarrow$ *Iter + 1,*

   FinSi,
7. ***Si*** δ_1 *ou* δ_2 *existe et Iter* $< MAX_{iter}$ ***Alors*** *Aller à l'étape 2,* ***Sinon*** *Arrêter.*

Fin Algorithme.

---

9. Résultats numériques

Dans le but d'évaluer l'heuristique de recherche tabou proposée, une première expérimentation est effectuée sur une série de problèmes issus de la littérature. Ensuite, une seconde expérimentation est réalisée sur des problèmes générés aléatoirement.

9.1. Problèmes existants

Les problèmes de cette section contiennent les instances proposées par Ramaswamy et Joshi [1996] pour le cas d'une cellule de production robotisée, et celles générées par Damasceno [1999]. Les résultats obtenus sont illustrés par le tableau 3.1.

Tableau 3.1. Résultats de 5 exécutions indépendantes sur les instances existantes

Problème	*RechTabou*			Mati *et al.*, 2000a			Damasceno, 1999
	mauvais	meilleur	CPU secs	mauvais	meilleur	CPU secs	
RJ1	512	**512***	0.1	512	**512***	0.02	n.d
RJ2	560	**560***	0.1	560	**560***	0.02	568
DX1	91	90	3.4	90	90	1.39	98
DX2	109	109	2.1	109	109	4.15	124
DX3	104	103	5.4	103	103	8.41	115
DX4	113	112	6.2	112	112	7.53	126
DX5	103	103	3.6	103	103	0.09	122

* : solution optimale
n.d : n'est pas donné

Le tableau 3.1 montre que les résultats proposés par la méthode de recherche tabou sont toujours meilleurs que celles proposées par Damasceno [1999]. D'une part, la recherche tabou permet de trouver les solutions optimales pour les deux instances *RJ*1 et *RJ*2 dans un temps très court. D'autre part, nous remarquons que les résultats donnés par la recherche tabou et la méthode proposée dans le chapitre 2 sont très proches. En effet, les deux approches aboutissent toujours à la même solution avec un léger avantage pour la méthode proposée au chapitre 2. Enfin, les makespans donnés dans les colonnes "*mauvais*" et "*meilleur*" montre que l'heuristique de recherche tabou est stable.

9.2. Problèmes générés

Afin de réaliser une comparaison consistante entre la recherche tabou et la méthode proposée au chapitre 2, nous proposons un ensemble de 10 nouvelles instances avec des tailles relativement grandes. Ces instances sont généralement plus difficiles à résoudre que celles utilisées dans la première expérimentation. Cet ensemble est décomposé en deux classes chacune contenant 5 instances. Le nombre de jobs est égal à 20 dans la première classe et 30 dans la seconde. Les paramètres utilisés pour créer ces problèmes sont :

- Le nombre d'opérations par job.
- Le nombre de ressources disponibles.
- Les temps opératoires.
- Le nombre de ressources nécessaires pour chaque opération.

Pour chaque job, le nombre d'opérations est générée aléatoirement dans l'intervalle [5,10]. Le nombre de ressources disponibles est fixé à 10 dans toutes les instances. La durée opératoire est sélectionnée aléatoirement dans l'intervalle [1, 10]. Enfin, le nombre de ressources nécessaires pour chaque opération est sélectionné dans l'intervalle [1, 3], et les types de ressources sont choisis aléatoirement dans l'ensemble de ressources disponibles.

Pour ces nouvelles instances, les résultats obtenus par la recherche tabou sont comparés avec ceux donnés par l'heuristique gloutonne proposée dans chapitre 2. Ces résultats sont résumés dans le tableau 3.2.

Tableau 3.2. Résultats de 5 exécutions indépendantes sur les instances générées

Problème	taille	*RechTabou*			Mati et al., 2000c	
		mauvais	meilleur	CPU secs	mauvais	meilleur
MA1	20	346	341	14.3	356	**340**
MA2	20	395	**380**	9.4	399	386
MA3	20	341	**331**	8.4	343	336
MA4	20	388	**377**	21.5	391	385
MA5	20	391	**380**	15.6	397	392
MA6	30	576	**561**	12.8	596	588
MA7	30	630	**604**	17.9	646	634
MA8	30	563	**540**	23.5	574	559
MA9	30	566	**554**	16.4	585	570
MA10	30	650	**632**	8.5	656	638

Le tableau 2 montre que la recherche tabou améliore les solutions données par l'heuristique gloutonne. En effet, dans tous les cas sauf l'instance MA1, les makespans obtenus sont inférieurs à ceux donnés par l'heuristique gloutonne. Ceci s'explique par le fait que l'heuristique gloutonne n'arrive pas à obtenir de bonnes séquences des jobs qui permettent de trouver de bons makespans.

Ce dernier tableau confirme l'intérêt du graphe disjonctif employé et des paramètres de la recherche tabou utilisés. Il confirme aussi que, le graphe disjonctif reste l'une des meilleures représentations pour les problèmes d'ordonnancements de types job shops.

10. Modèle de programmation linéaire

Comme nous l'avons mentionné à la section 3, la modélisation mathématique du problème d'ordonnancement avec les contraintes de blocage est difficile à obtenir dans le cas général. Nous montrons dans la suite en s'appuyant sur l'extension proposée du graphe disjonctif que les contraintes de blocage pour le cas de *JSSB* peuvent être exprimées mathématiquement. Pour ce faire, nous gardons les mêmes définitions des variables de décision utilisées dans la section 3. Pour permettre d'exprimer les contraintes de blocage, les contraintes (2)-(3) du modèle de la section 3 sont remplacées par les contraintes (2')-(3'). La nouvelle formulation mathématique s'écrit sous la forme :

$$
(P^{\varepsilon})\ \begin{cases}
Min\ \ C_{\max} & & \\
\text{sous les contraintes :} & & \\
\quad S_{i,k+1} - S_{ik} \ \geq\ p_{ik} & \forall\, 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i - 1 & (1') \\
\quad S_{ik} - S_{i'k'+1} + M\, y_{ik,i'k'} \geq \varepsilon & \forall (O_{ik}, O_{i'k'}) / \ i < i' \wedge R_{ik} \cap R_{i'k'} \neq \varnothing & (2') \\
\quad S_{i'k'} - S_{i,k+1} + M\,(1 - y_{ik,i'k'}) \geq \varepsilon & \forall (O_{ik}, O_{i'k'}) / \ i < i' \wedge R_{ik} \cap R_{i'k'} \neq \varnothing & (3') \\
\quad C_{\max} \geq S_{in_i} & \forall\, 1 \leq i \leq N & (4') \\
\quad S_{ik} \geq 0 & \forall\ 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i & \\
\quad y_{ik,i'k'} \in \{0,1\} & \forall (O_{ik}, O_{i'k'}) / \ R_{ik} \cap R_{i'k'} \neq \varnothing &
\end{cases}
$$

où N est le nombre de jobs, $M > 0$ est un grand nombre et $\varepsilon > 0$ est une constante suffisamment petite. La constante $\varepsilon > 0$ est introduite pour modéliser l'opération intermédiaire de transfert et permet ainsi de régler le problème de blocage.

Dans cette formulation, les contraintes (1') sont les contraintes de précédence. Les contraintes de capacité des ressources sont satisfaites sous les contraintes (2')-(3'). La constante ε est introduite pour assurer l'absence de blocage. En effet, d'après (2') et (3'), nous avons $S_{ik} \geq S_{i',k'+1} + \varepsilon$ pour toutes les opérations $O_{i'k'}$ qui précèdent O_{ik} sur les ressources communes et $S_{i,k+1} + \varepsilon \leq S_{i'k'}$, pour toutes les opérations $O_{i'k'}$ qui succèdent O_{ik} sur les ressources communes. Puisque $S_{ik} + p_{ik} \leq S_{i,k+1}$, toutes les ressources R_{ik} sont disponibles à l'instant S_{ik}. Les contraintes (4') servent à définir la valeur du makespan.

Ainsi, toute solution de (P^{ε}) est une solution réalisable du problème (P) défini dans la section 3.

Théorème 3.3

Toute solution de (P^{ε}) est un ordonnancement réalisable. De plus, pour tout ordonnancement réalisable $\{S^*_{ik}, y^*_{ik,i'k'}\}$, il existe une solution $\{S_{ik}, y_{ik,i'k'}\}$ du problème (P) telle que :

$$S_{ik} \leq S^*_{ik} + \varepsilon \sum_{i=1}^{N} n_i \ \text{ et } \ y_{ik,i'k'} = y^*_{ik,i'k'}$$

Ce théorème implique que, pour tout $\varepsilon < \left(\sum_{i=1}^{N} n_i \right)^{-1}$, nous avons $C^*_{\max} \leq C^{\varepsilon}_{\max} < C^*_{\max} + 1$, où $C^{\varepsilon}_{\max}$ (resp. $C^*_{\max}$) est la valeur optimale de la fonction objectif de (P^{ε}) (resp. (P)). Par conséquent, l'ordonnancement optimal de (P) peut être obtenu à partir des séquences d'entrée $y_{ik,i'k'}$ de (P^{ε}) en utilisant le graphe disjonctif.

Preuve

Considérons le graphe disjonctif avec blocage en affectant à chaque couple disjonctif un poids égal à ε pour exprimer les équations (2') et (3') du modèle mathématique (P^{ε}). Nous montrons d'abord que toute solution de (P^{ε}) est un ordonnancement réalisable. Grâce aux contraintes (1') du modèle, toute solution de (P^{ε}) est réalisable en terme de précédence. En plus, la capacité des ressources est vérifiée puisque $S_{ik} \geq S_{i',k'+1} + \varepsilon$ si $O_{i'k'}$ précède O_{ik} et $S_{i'k'} \geq S_{i,k+1} + \varepsilon$ dans le cas contraire. Supposons que la solution soit irréalisable en terme de blocage, alors le graphe disjonctif avec blocage représentant l'ordonnancement donné par cette solution contient au moins un circuit $C = O_{ik},\ldots,O_{i'k'}, O_{ik}$. En plus, les opérations O_{ik} et $O_{i',k'-1}$ partagent des ressources communes à cause de l'existence du couple disjonctif $(O_{i'k'}, O_{ik})$ et l'opération $O_{i',k'-1}$ précède l'opération O_{ik}, c'est-à-dire, $y_{ik,i'k'-1} = 0$. De la relation (2') et (3') du modèle (P^{ε}), nous avons $S_{ik} - S_{i'k'} \geq \varepsilon$. De plus, le chemin de O_{ik} à $O_{i'k'}$ appartenant au circuit $C = O_{ik},\ldots,O_{i'k'}, O_{ik}$ conduit à $S_{i'k'} \geq S_{ik}$. Ceci est une contradiction. Donc, le graphe disjonctif avec blocage associé à tout ordonnancement obtenu par (P^{ε}) est sans circuit, et ainsi l'ordonnancement est réalisable.

Considérons maintenant un ordonnancement réalisable $\{S^*_{ik}, y^*_{ik,i'k'}\}$ et un sommet O_{ik} du graphe disjonctif avec blocage. Dans le pire des cas, le plus long chemin de la source O à O_{ik} utilise tous les sommets du graphe disjonctif avec blocage. Par conséquent, le plus long chemin de la source O à O_{ik} contient au plus $\sum_{i=1}^{N} n_i$ arcs disjonctifs, où chacun est pondéré par ε. Nous avons alors $S_{ik} \leq S^*_{ik} + \varepsilon \sum_{i=1}^{N} n_i$. Ceci termine la démonstration du théorème. ■

11. Extensions de l'approche

Les résultats présentés dans ce chapitre sont valables pour la minimisation du makespan. Pour tenir compte des autres critères objectifs réguliers tels que la somme des retards, la somme des dates de fin des jobs, etc., des modifications doivent être apportées à la recherche tabou.

La première modification concerne l'évaluation du mouvement après l'exécution d'une permutation sur le chemin critique. Une première solution à ce problème consiste à recalculer complètement le plus court chemin dans le nouveau graphe.

La deuxième modification concerne le mouvement de recouvrement du blocage. Pour pouvoir appliquer l'algorithme de réordonnancement proposé dans la section 7, l'approche géométrique doit être généralisée pour le problème d'ordonnancement entre le job composé et un job simple. Une manière d'étendre cette dernière approche a été décrite dans le chapitre 2. Les autres paramètres de la recherche tabou restent valables pour les autres critères réguliers.

12. Conclusion

Ce chapitre a été consacré à l'ordonnancement des systèmes de production de type job shop multi-ressources avec blocage, pour lesquels toutes les ressources sont disponibles en une seule unité. Dans un premier temps, nous avons donné une description du modèle étudié. Nous avons ensuite considéré un problème important d'ordonnancement des trains qui peut être modélisé par le modèle JSSB. Dans un second temps, nous avons généralisé la représentation classique avec le graphe disjonctif pour prendre en compte la contrainte *retenir et attendre*, et nous avons donné les propriétés du nouveau graphe. En se basant sur cette nouvelle généralisation, nous avons proposé une méthode intégrée utilisant l'heuristique de recherche tabou. Une des contributions de cette méthode est l'utilisation d'un mouvement original de recouvrement pour la définition de la structure de voisinage. La recherche tabou a été testée sur des instances issues de la littérature ainsi que sur de nouvelles instances générées aléatoirement. Les résultats obtenus sont très satisfaisants, et montre que l'heuristique proposée apporte des améliorations par rapport aux autres approches existantes sur les problèmes de grandes tailles. Enfin, un modèle de programmation linéaire à variables mixtes est proposé pour modéliser le problème d'ordonnancement de *JSSB*. Ce modèle est une conséquence directe du graphe disjonctif généralisé.

Les résultats proposés dans ce chapitre peuvent donner lieu à des recherches futures intéressantes. D'abord, le modèle mathématique peut être utilisé pour développer des méthodes exactes, comme par exemple la relaxation lagrangienne et les méthodes par séparation et évaluation. D'autre part, la recherche tabou et le modèle du graphe disjonctif proposé peuvent être utilisés pour contrôler en temps réel les différents problèmes représentés par le *JSSB*, et en particulier les problèmes d'ordonnancements des trains. En effet, en cas des retards des trains, l'ordonnancement établi hors ligne n'est plus valable et par conséquent il est souhaitable de le remettre en cause en réordonnançant les passages des trains.

Dans cette recherche, nous nous sommes limités à la minimisation du makespan. Il serait intéressant, de voir le comportement de cette approche sur d'autres critères objectifs en particulier sur la somme des retards. Un autre axe de recherche consisterait à construire et définir des voisinages possédant la propriété de connectivité, et à étudier les conséquences sur les résultats obtenus.

Deuxième Partie

Les Problèmes d'Ordonnancement avec Contraintes de Flexibilité

Dans cette partie, nous généralisons les modèles d'ordonnancement traités dans la première partie, en considérant en plus des contraintes de flexibilité sur l'exécution des opérations. Nous commençons par étudier le problème du job shop flexible avec des machines non-reliées, et nous proposons des nouveaux résultats de complexité pour le cas de la présence de seulement deux jobs. Ensuite, nous nous concentrons sur le modèle du job shop multi-ressources avec flexibilité des ressources, et nous proposons des méthodes heuristiques pour la résolution du problème. Enfin, nous étudions un modèle d'ordonnancement plus général appelé job shop multi-ressources avec prise en compte de blocage et de flexibilité des ressources. Pour ce modèle, nous proposons des nouveaux algorithmes polynomiaux pour la résolution de cas particuliers à deux jobs, et nous développons des heuristiques permettant de répondre au problème général comportant plusieurs jobs.

Chapitre 4

Complexité des Problèmes d'Ordonnancement de Job Shops avec Flexibilité des Machines

Dans ce chapitre, nous nous intéressons à la complexité des problèmes d'ordonnancement d'un job shop à deux jobs en présence de contraintes de flexibilité sur l'exécution des opérations. Dans ce type de problèmes, la machine nécessaire à l'exécution d'une opération n'est pas fixée, mais doit être choisie parmi un ensemble donné. Tout d'abord, nous montrons que le problème de la minimisation de toute fonction objectif régulière est NP-difficile au sens faible, et ce, même si la préemption des opérations est permise. Ensuite, nous proposons un algorithme pseudo-polynomial pour la résolution du problème. Pour le cas où la flexibilité d'un job est fixée, nous proposons dans un premier temps un algorithme polynomial pour résoudre le problème d'ordonnancement sans contrainte de préemption des opérations, et ceci pour n'importe quelle fonction objectif régulière. Dans un second temps, nous donnons une généralisation de l'algorithme pour le cas de la préemption des opérations.

Une partie des résultats de ce chapitre a été présentée lors de la conférence internationale ORP3 (Operational Research Peripatetic Post-graduate Programme) [2001], Paris, France. Ces résultats ont aussi fait l'objet d'un article soumis à European Journal of Operational Research.

1. Introduction

Le problème d'ordonnancement de type job shop tient un rôle important dans la théorie de l'ordonnancement et trouve beaucoup d'applications dans la pratique. Il consiste à ordonnancer un ensemble de jobs (travaux) sur un ensemble de machines dans le but de minimiser une fonction objectif donnée. La structure du modèle du job shop, notamment le fait que les gammes opératoires peuvent différer d'un job à l'autre, permet de modéliser de nombreuses applications pratiques. En dépit de ce fait, les nouvelles contraintes introduites dans les systèmes de production automatisés deviennent de plus en plus difficiles à intégrer, et rendent nécessaire l'introduction d'une flexibilité additionnelle. Une manière d'enrichir le modèle du job shop consiste ainsi à permettre à une opération d'être réalisée par un ensemble de machines.

Dans ce chapitre, nous nous intéressons au problème d'ordonnancement de type job shop avec contrainte de flexibilité sur l'exécution des opérations. Cette contrainte sera appelée dans ce qui suit *flexibilité des machines* et signifie que la machine nécessaire à l'exécution d'une opération n'est pas connue mais doit être choisie dans un ensemble donné a priori.

Etant une généralisation du problème d'ordonnancement de type job shop qui est NP-difficile (Garey et Johnson [1976]), le problème d'ordonnancement avec flexibilité des machines est aussi NP-difficile. Dans ce chapitre, nous nous concentrons sur les problèmes de *deux jobs avec flexibilité des machines*.

La suite de ce chapitre est organisée comme suit. Dans la section 2, nous donnons une description du problème d'ordonnancement à deux jobs avec flexibilité des machines. Nous étudions ensuite, dans la section 3, la complexité de ce problème dans le cas de la minimisation du makespan et de celle de la somme des dates de fin des deux jobs. Tout d'abord, nous montrons que le problème d'ordonnancement est NP-difficile au sens faible pour toute fonction objectif régulière, et nous déduisons un résultat de complexité pour le problème particulier du *flow shop hybride*. Ensuite, nous proposons un algorithme pseudo-polynomial permettant de résoudre le problème. Dans la section 4, nous considérons un cas particulier dans lequel la flexibilité d'un job est fixée. Pour ce cas, nous proposons un algorithme polynomial qui résout le problème d'ordonnancement lorsque la préemption des opérations n'est pas permise. L'algorithme donne des solutions optimales pour toute fonction objectif régulière. Enfin, nous donnons une généralisation de l'algorithme polynomial pour la prise en compte de la préemption des opérations.

2. Job shop à deux jobs avec flexibilité des machines

Le problème d'ordonnancement de type job shop à deux jobs avec flexibilité des machines se définit de la manière suivante. Deux jobs J_1 et J_2 doivent être réalisés sur un ensemble de m machines $M = \{M_1, \ldots, M_m\}$. Chaque job J_i est composé d'une séquence d'opérations $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$ appelée *gamme opératoire*, où n_i est le nombre d'opérations du job J_i. Une machine $M_k \in M$ peut réaliser au plus une opération à la fois et une opération ne peut être exécutée que par une machine à la fois. De plus, chaque opération O_{ik} peut être réalisée par n'importe quelle machine M_r dans un ensemble donné $\mu_{ik} \subseteq M$. La durée opératoire p_{ikr} de l'opération O_{ik} dépend alors de la machine M_r choisie. La préemption n'est pas permise, c'est-

à-dire une fois que l'exécution d'une opération est commencée, elle ne doit pas être interrompue.

Le problème d'ordonnancement consiste donc à affecter une machine à chaque opération et à déterminer la séquence des opérations sur les machines obtenues, et ceci afin de minimiser une fonction objectif donnée.

Dans le reste de ce chapitre, nous appellerons un job J_i ayant de la flexibilité sur l'exécution des opérations *job avec flexibilité des machines* ou encore *job flexible*. En adoptant la notation utilisée par Brucker [1998], le problème d'ordonnancement défini dans cette section s'écrit *JMPM, non-relié* | $n = 2$ | *f*. Le terme "non-relié" exprime l'hypothèse $p_{i,k,r_1} \neq p_{i,k,r_2}$ signifiant que les durées opératoires dépendent des machines sélectionnées. Ce problème d'ordonnancement à deux jobs généralise celui traité par Brucker et Schlie [1990]. Pour ce dernier problème, les auteurs supposent que les durées opératoires sont connues et ne dépendent pas des machines sélectionnées.

3. Etude de complexité

Une manière de montrer la NP-complétude d'un problème d'optimisation combinatoire consiste à trouver une transformation polynomiale d'un problème connu comme étant NP-difficile au problème considéré. Dans le domaine de la théorie de l'ordonnancement, le problème de *PARTITION* (Garey et Johnson [1976]), apparaît très souvent dans la littérature. C'est ce problème qui sera utilisé dans cette section pour démontrer la NP-complétude du problème d'ordonnancement traité.

Dans un but de faciliter la compréhension, nous rappelons, avant de présenter nos résultats de complexité, le problème de *PARTITION*.

3.1. Problème de partition

Le problème de PARTITION joue un rôle prépondérant dans les démonstrations de la NP-complétude de plusieurs problèmes d'ordonnancement. Un cas particulier du problème de PARTITION, appelé *even-odd PARTITION* (Garey et al. [1988]) est utilisé dans ce paragraphe pour démontrer la NP-complétude du problème d'ordonnancement du job shop à deux jobs avec flexibilité des machines.

Le problème *even-odd PARTITION* peut être défini de la manière suivante. Un ensemble de $2N$ entiers positifs $\{e_1, e_2, \ldots, e_{2N}\}$ est donné, tel que $e_i > e_{i+1}$ pour tout indice $1 \leq i < 2N$. Les $2N$ entiers vérifient la relation $\sum_{i=1}^{2N} e_i = 2E$. La question à laquelle il s'agit de répondre est la suivante :

Question : Existe-il une partition de l'ensemble $A = \{1, 2, \ldots, 2N\}$ en deux sous-ensembles A_1 et A_2 tels que :

(i) $\sum_{i \in A_1} e_i = \sum_{i \in A_2} e_i = E$.

(ii) $2i - 1 \in A_u \Leftrightarrow 2i \in A_{3-u}$, $\forall\, i \in \{1, 2, \ldots, 2N\}$ et $u \in \{1,2\}$.

Si la réponse à cette question est positive alors le problème de *even-odd PARTITION* a une solution.

3.2. Minimisation du makespan

Dans cette sous-section, nous nous intéressons à la minimisation du makespan ; une fonction objectif abondamment traitée dans la littérature. Dans le but de montrer la NP-complétude du problème *JMPM, non-relié* | $n = 2$ | C_{max}, nous proposons une réduction polynomiale du problème even-odd PARTITION au problème de décision suivant :

Existe-il un ordonnancement réalisable s^* pour le problème JMPM, non-relié / $n = 2$ / C_{max} tel que $C_{max} \leq y$ pour un entier donné y ?

Pour ce faire, nous considérons l'instance suivante notée P1, pour le problème *JMPM, non-relié / $n = 2$ / C_{max}*.

Instance *P1* :

Les gammes opératoires des jobs J_1 et J_2 sont identiques et définies comme suit :

$$J_1 = J_2 = \{[(M_1, p_1) \text{ ou } (M_2, p_2)], [(M_3, p_3) \text{ ou } (M_4, p_4)], \ldots, [(M_{2N-1}, p_{2N-1}) \text{ ou } (M_{2N}, p_{2N})]\}.$$

Les durées opératoires sont égales à $p_i = E + e_i$, $\forall\ i\ /\ 1 \leq i \leq 2N$. Nous fixons l'entier y à la valeur $NE + E$.

Nous montrons dans le théorème 4.1 suivant qu'un ordonnancement sans préemption tel que $C_{max} \leq y$ existe pour l'instance proposée si et seulement si le problème *even-odd PARTITION* a une solution.

Théorème 4.1

Le problème *JMPM, non-relié* | $n = 2$ | C_{max} est NP-difficile.

Preuve

Nous réduisons polynomialement, le problème de *even-odd PARTITION* au problème de décision suivant :

Existe-il un ordonnancement réalisable s^* pour le problème JMPM, non-relié / $n = 2$ / C_{max} tel que $C_{max}(s^*) \leq NE + E$?

Condition suffisante :

Si le problème *even-odd PARTITION* a une solution $A = A_1 \cup A_2$, $\sum_{i \in A_1} e_i = \sum_{i \in A_2} e_i = E$ et $2i - 1 \in A_u \Leftrightarrow 2i \in A_{3-u}$, alors nous pouvons construire l'ordonnancement suivant s^*. Les opérations du

job J_1 sont réalisées sur les machines M_i / $i \in A_1$ et les opérations du job J_2 sont réalisées sur les machines M_i / $i \in A_2$. Puisque $2k$-1 et $2k$ appartiennent à deux ensembles différents A_1 et A_2, chaque opération est affectée à une machine différente. Par conséquent, les jobs J_1 et J_2 sont réalisés sans retard. Par ailleurs, puisque $\sum_{i \in A_1} e_i = \sum_{i \in A_2} e_i = E$, le makespan associé à l'ordonnancement s^* proposé est égal à $C_{max} = C_1 = C_2 = \sum_{i \in A_1} (E + e_i) = NE + E$.

Nous avons donc construit un ordonnancement réalisable s^* et qui vérifie la contrainte $C_{max}(s^*) \leq NE + E$.

<u>*Condition nécessaire*</u> :

Supposons qu'il existe un ordonnancement réalisable s^* tel que $C_{max}(s^*) \leq NE + E$, et montrons que le problème de *even-odd PARTITION* admet une solution.

a) Les jobs J_1 et J_2 ne partagent pas des machines communes

Nous procédons par contradiction pour démontrer (*a*). Supposons sans perte de généralité que la machine $M_{2k\text{-}1}$ est utilisée par les deux jobs J_1 et J_2. D'après la définition de J_1 et J_2, la machine $M_{2k\text{-}1}$ est utilisée pour la réalisation des opérations O_{1k} et O_{2k}. Sans perte de généralité, nous supposons que l'opération O_{1k} précède l'opération O_{2k} sur la machine $M_{2k\text{-}1}$. La date de fin C_2 de la dernière opération du job J_2 est alors :

$$\begin{aligned} C_2 &\geq \sum_{i=1}^{k} p_{1i} + \sum_{i=k}^{N} p_{2i} \\ &> p_{11} + NE \qquad\qquad (car\ p_{ji} \geq E) \\ &\geq NE + \min\{p_1, p_2\} \\ &> NE + E \end{aligned}$$

où p_{ji} est la durée opératoire de l'opération O_{ji} qui dépend de l'affectation de la machine. L'inégalité $C_2 > NE + E$ contredit l'hypothèse $C_{max}(s^*) \leq NE + E$, et ceci démontre (a).

Le résultat (*a*) implique que les deux jobs sont réalisés sur des machines différentes sans temps mort. Par conséquent, nous avons :

$$C_1 + C_2 = \sum_{i=1}^{2N} p_i = 2NE + 2E \qquad (1)$$

Puisque $C_{max}(s^*) \leq NE + E$, alors :

$$\max\{C_1, C_2\} \leq NE + E \qquad (2)$$

En combinant (1) et (2), nous avons $C_1 = C_2 = NE + E$.

Ce dernier résultat associé à (*a*) conduit à la partition de l'ensemble $A = \{1, 2,..., 2N\text{-}1, 2N\}$ en deux sous-ensembles disjoints A_1 et A_2 relatifs respectivement aux machines utilisées par J_1 et J_2. Cette partition a les propriétés suivantes:

(iii) $\sum_{i \in A_1} e_i = \sum_{i \in A_2} e_i = E$,

(iv) $2k - 1 \in A_u \Leftrightarrow 2k \in A_{3-u}$, $\forall\ 1 \leq k \leq N$.

Ce qui implique que le problème *even-odd PARTITION* a une solution. ■

Remarque 4.1 : L'instance *P*1 proposée est un cas particulier du problème traité dans ce chapitre. Elle est plus connue sous le nom de *flow shop hybride*. Le théorème 4.1 implique donc que la minimisation du makespan dans un *flow shop hybride* à deux jobs est aussi NP-difficile. Ceci est donné par le théorème suivant :

Théorème 4.2

Le problème *FMPM, non-relié* | $n = 2$ | C_{max} est NP-difficile.

3.3. Minimisation de la somme des dates de fin

Nous nous intéressons dans cette sous-section à une autre fonction objectif régulière qui est la somme des dates de fin d'exécution C_1 et C_2 des jobs J_1 et J_2.

Théorème 4.3

Le problème *JMPM, non-relié* | $n = 2$ | $\sum C_i$ est NP-difficile.

Preuve

La preuve est basée sur la réduction polynomiale du problème *even-odd PARTITION* au problème *P*2 suivant :

Instance *P*2 :

Les gammes opératoires des jobs J_1 et J_2 sont définies comme suit :

$J_1 = \{[(M_1, p_1)$ ou $(M_2, p_2)], \ldots, [(M_{2N\text{-}1}, p_{2N\text{-}1})$ ou $(M_{2N}, p_{2N})], [(M', p_{2N+1})], [(M'', p_{2N+2})]\}$,
$J_2 = \{[(M_1, p_1)$ ou $(M_2, p_2)], \ldots, [(M_{2N\text{-}1}, p_{2N\text{-}1})$ ou $(M_{2N}, p_{2N})], [(M'', p_{2N+1})], [(M', p_{2N+2})]\}$,

où $p_i = E + e_i$, $\forall\ 1 \leq i \leq 2N$, $p_{2N+1} = p_{2N+2} = 2E$.

Question: Existe-il un ordonnancement réalisable tel que $C_1 + C_2 \leq 2NE + 10E$?

Condition suffisante :

La condition suffisante peut être démontrée en utilisant les mêmes arguments que la preuve du théorème 4.1, et par conséquent elle est omise.

Condition nécessaire :

Supposons qu'il existe un ordonnancement réalisable tel que $C_1 + C_2 \leq 2NE + 10E$. Soit C_{ik} la date de fin de l'opération O_{ik}.

Sans perte de généralité, nous supposons que $C_{1N} \leq C_{2N}$.

A) $I_i < 2E$, *où* I_i *est le temps mort total du job* J_i.

Du fait que :

$$\begin{aligned} C_1 + C_2 &= \sum_{k=1}^{N+2}(p_{1k} + p_{2k}) + I_1 + I_2 \\ &> \sum_{k=1}^{N}(E + E) + 8E + I_1 + I_2 \\ &= 2NE + 8E + I_1 + I_2, \end{aligned}$$

et d'après l'hypothèse $C_1 + C_2 \leq 2NE + 10E$, nous déduisons que $I_1 + I_2 < 2E$.

B) $C_{1N} > NE$ *du fait que* $C_{1N} \geq \sum_{k=1}^{N} p_{1k} > NE$.

C) $C_{2N} < NE+2E$.

Car sinon,

$$\begin{aligned} C_1 + C_2 &\geq C_{1N} + C_{2N} + 8E \\ &> (NE) + (NE + 2E) + 8E \\ &= 2NE + 10E \end{aligned}$$

et l'hypothèse $C_1 + C_2 \leq 2NE + 10E$ est remise en cause.

D) $O_{1,N+1} \prec O_{2,N+2}$ *et* $O_{2,N+1} \prec O_{1,N+2}$, où $O \prec O'$ signifie opération O précède O'.

Ceci peut être démontré par contradiction de la manière suivante :

Si $O_{2,N+2} \prec O_{1,N+1}$ alors $I_1 \geq C_{2,N+2} - C_{1,N} \geq 4E + C_{2N} - C_{1N} \geq 4E$, ce qui contredit (*A*).

Si $O_{1,N+2} \prec O_{2,N+1}$ alors $I_2 \geq C_{1,N+2} - C_{2N} \geq 4E + C_{1N} - C_{2N} > 4E + (NE) - (NE+2E) = 2E$, d'où une contradiction avec (*A*).

En utilisant le résultat donné par (*D*), nous en déduisons que :

$$C_{i,N+1} = C_{iN} + p_{i,N+1} = C_{iN} + 2E.$$

Et en combinant l'hypothèse $C_{1N} \leq C_{2N}$ et (D), nous obtenons :

$$C_1 = C_{1,N+2} = C_{2,N+1} + p_{1,N+2} = C_{2N} + 4E$$
$$C_2 = C_{2,N+2} = C_{2,N+1} + p_{2,N+2} = C_{2N} + 4E$$

Ce qui implique avec l'hypothèse $C_1 + C_2 \leq 2NE + 10E$ que $C_{2N} \leq NE + E$. De manière équivalente, nous pouvons montrer que $C_{1N} \leq NE + E$. Finalement, $\max\{C_{1N}, C_{2N}\} \leq NE + E$.

La preuve de la condition nécessaire du théorème 4.1 permet de conclure que le problème *even-odd PARTITION* a une solution. ■

3.4. Minimisation d'une fonction objectif régulière

Nous avons démontré que le problème d'ordonnancement à deux jobs était NP-difficile pour les deux fonctions objectifs C_{max} et $\sum C_i$. En utilisant la réduction entre les problèmes d'ordonnancement avec des fonctions objectifs régulières classiques (Lawler et al. [1989], page 9), nous en concluons que le problème d'ordonnancement de cette section est NP-difficile pour toute fonction objectif régulière. C'est ce qui est donné par le théorème suivant :

Théorème 4.4

Le problème *JMPM, non-relié* | $n = 2$ | f est NP-difficile pour tout critère régulier $f \in \{L_{\max}, \sum T_i, \sum w_i T_i, \sum w_i C_i, \sum U_i, \sum w_i U_i\}$.

3.5. Cas de la préemption

Pour les deux instances $P1$ et $P2$ proposées précédemment, tout ordonnancement avec préemption peut être transformé en un ordonnancement sans préemption sans altérer la valeur du critère considéré. C'est ce qui est énoncé par le théorème suivant :

Théorème 4.5

Le problème *JMPM, non-relié* | $n = 2$; *pmtn* | f est NP-difficile pour tout critère régulier $f \in \{L_{\max}, \sum T_i, \sum w_i T_i, \sum w_i C_i, \sum U_i, \sum w_i U_i\}$.

Preuve

En utilisant la réduction entre les problèmes d'ordonnancement avec des fonctions objectifs régulières, il est suffisant de démontrer la NP-compléture des deux critères $f = C_{max}$ et $f = \sum C_i$.

Nous utilisons d'abord l'instance $P1$ à deux jobs pour le cas $f = C_{max}$. La NP-complétude du problème *JMPM, non-relié*$| n = 2 ; pmtn | C_{max}$ découle du fait que pour tout ordonnancement avec préemption, la réaffectation des opérations interrompues aux machines non sélectionnées aboutit à un ordonnancement sans préemption avec des dates de fin plus petites, et ceci pour toutes les opérations. Plus précisément, considérons un ordonnancement avec préemption pour lequel l'opération O_{1k} est interrompue par l'opération O_{2k} sur la machine M_{2k-1}. La date de fin de O_{1k} est :

$$C_{1k} = S_{1k} + p_{2k-1} + p_{2k-1} > S_{1k} + 2E,$$

où p_{2k-1} est la durée opératoire des opérations O_{1k} et O_{2k} sur M_{2k-1}, et S_{1k} est la date du début de O_{1k}.

En réaffectant l'opération O_{1k} à la machine M_{2k}, la nouvelle date de fin de O_{1k} est :

$C'_{1k} = S_{1k} + p_{2k} = S_{1k} + E + e_{2k} < S_{1k} + 2E$, où p_{2k} est la durée opératoire de O_{1k} sur la machine M_{2k}.

En conséquence, $C'_{1k} < C_{1k}$ et les dates de fin de toutes les opérations restent inchangées.

L'instance $P2$ à deux jobs est utilisée pour prouver le résultat dans le cas $f = \sum C_i$. La condition suffisante est triviale, nous nous intéressons donc à la preuve de la condition nécessaire. Comme dans le cas de $f = C_{max}$, tout ordonnancement avec préemption des opérations O_{ik} avec $1 \le k \le N$ peut être transformé en un meilleur ordonnancement sans préemption des opérations O_{ik} avec $1 \le k \le N$.

Considérons maintenant un ordonnancement avec préemption des opérations $O_{i,N+1}$ et $O_{i,N+2}$. Sans perte de généralité, nous supposons que l'opération $O_{1,N+1}$ est interrompue par l'opération $O_{2,N+2}$ sur la machine M'. Il est clair que la préemption dure $2E$ unités de temps, conduisant à un temps mort d'au moins de $2E$ unités pour le job J_1, c'est-à-dire $I_1 \ge 2E$.

Nous avons donc la relation :

$$\begin{aligned} C_1 + C_2 &\ge \sum_{i=1}^{N+2}(p_{1i} + p_{2i}) + I_1 \\ &\ge p_{11} + p_{21} + \sum_{i=2}^{N+2}(p_{1i} + p_{2i}) + 2E \\ &\ge 2 \times \min\{p_1, p_2\} + 2(N-1)E + 8E + 2E \\ &> 2E + 2(N-1)E + 10E = 2NE + 10E \end{aligned}$$

Ceci contredit l'hypothèse nécessaire $C_1 + C_2 \le 2NE + 10E$. En conséquence, tout ordonnancement réalisable tel que $C_1 + C_2 \le 2NE + 10E$ est sans préemption et la preuve du théorème 4.3 complète la démonstration. ■

3.6. Algorithme pseudo-polynomial

Dans cette sous-section, nous proposons un algorithme pseudo-polynomial pour la résolution du problème d'ordonnancement à deux jobs. Nous considérons la minimisation de la durée totale (makespan) dans le cas de l'absence de préemption des opérations.

Soit :

$g(k_1, f_1, r_1, k_2, f_2, r_2)$: le temps minimum pour atteindre un état (k_1, k_2) pour lequel l'opération O_{1,k_1} (resp. O_{2,k_2}) est en cours avec la flexibilité $f_1 \in \mu_{1,k_1}$ (resp. $f_2 \in \mu_{2,k_2}$). La valeur r_1 (resp. r_2) représente la durée opératoire restante pour finir l'exécution de l'opération O_{1,k_1} (resp. O_{2,k_2}).

Il est clair que $0 \leq r_i \leq p_{i,k_i,f_i}$ et que l'opération O_{i,k_i} termine lorsque $r_i = 0$.

Dans ce qui suit nous développons un algorithme de programmation dynamique de complexité $O(n^4 K^4 T^4)$ pour calculer toutes les valeurs de $g(\bullet)$, où $n = \max\{n_1, n_2\}$, K est le nombre maximum des machines pour chaque opération, et T est la durée opératoire maximale.

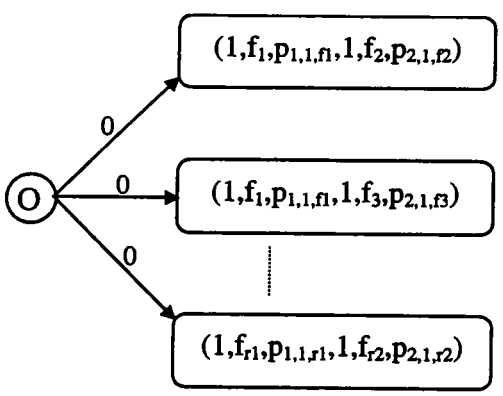
L'algorithme de programmation dynamique est basé sur la construction d'un réseau acyclique G défini comme suit :

Algorithme 4.1 : *ProgDyna*

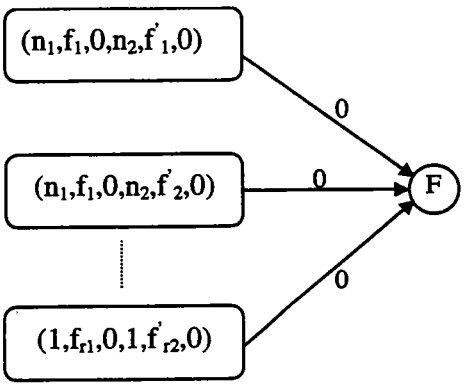
(A) Représenter chaque état $(k_1, f_1, r_1, k_2, f_2, r_2)$ par un sommet.

(B) Ajouter deux sommets fictifs O et F.

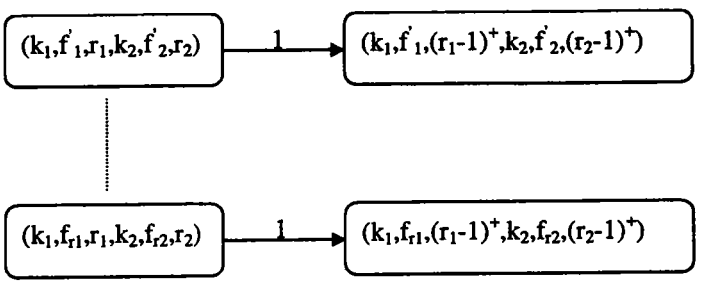
(C) Ajouter un arc de poids 0 de O à chaque état possible à l'instant 0^+.



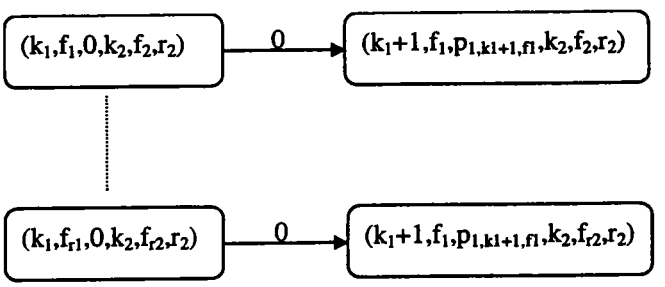
(D) Ajouter un arc de poids 0 de tout état $(n_1,f_1,0,n_2,f_2,0)$ à F.



(E) Ajouter un arc de poids 1 de chaque état $(k_1, f_1, r_1, k_2, f_2, r_2)$ tel que $r_1 + r_2 > 0$ à l'état $(k_1,f_1,(r_1-1)^+,k_2,f_2,(r_2-1)^+$.



(F) Ajouter un arc de poids 0 de tout état $(k_1, f_1, 0, k_2, f_2, r_2)$ à tout état $(k_1+1, f'_1, p_{1,k_1+1,f'_1}, k_2,f_2,r_2)$, où $f'_1 \in \mu_{1,k1+1}$.



(G) De même, ajouter des arcs de poids 0 depuis les états $(k_1,f_1,r_1,k_2,f_2,0)$.

(H) *Ajouter des arcs de poids 0 depuis les états $(k_1,f_1,0,k_2,f_2,0)$, comme dans (F) et (G).*

Notons que le réseau construit par l'algorithme *ProgDyna* est un réseau sans circuits, et chaque arc représente la progression du temps, ou la progression pour un job, de l'opération courante à l'opération suivante de sa gamme.

Théorème 4.6

Le problème de la minimisation du makespan est pseudo-polynomial et sa complexité est au plus $O(n^4 K^4 T^4)$.

Preuve

Le nombre de nœuds (resp. arcs) dans le réseau construit est au pire des cas égal à $O(n^2 K^2 T^2)$ (resp. $O(n^4 K^4 T^4)$). Puisque le réseau construit est acyclique, la recherche du plus court chemin pour déterminer le meilleur makespan prend un temps proportionnel à $O(|V|+|E|)$, où $|V| = O(n^2 K^2 T^2)$ et $|E| = O(n^4 K^4 T^4)$. Par conséquent, la complexité de l'algorithme *ProgDyna* est $O(n^4 K^4 T^4)$. ■

4. Cas d'un job flexible et d'un job de type job shop

Dans cette section, nous considérons un cas particulier du problème d'ordonnancement étudié dans la section précédente. Pour ce nouveau problème, le premier job est supposé flexible, c'est-à-dire que l'affectation des machines aux opérations n'est pas fixée. Le deuxième job est un job de type job shop, c'est-à-dire que les machines devant exécuter les opérations sont fixées à l'avance.

Dans un premier temps, nous proposons un algorithme polynomial pour la résolution du problème dans le cas où la préemption des opérations n'est pas permise. L'algorithme polynomial permet d'obtenir des ordonnancements optimaux pour toute fonction objectif régulière. Dans un second temps, nous proposons une extension de cet algorithme pour la prise en compte de la préemption des opérations.

4.1. Cas non préemptif

Dans cette section, nous proposons une extension de l'approche géométrique pour résoudre le problème d'ordonnancement à deux jobs, pour lequel un seul job est flexible. Sans perte de généralité, nous supposons que le job J_1 est le job flexible et le job J_2 est le job de type job shop.

La représentation géométrique dans le plan (Akers et Friedman [1955]) ne peut être directement appliquée. En effet, la définition même des obstacles dépend fortement de l'affectation des machines aux opérations du job flexible J_1. Il est cependant possible de définir des obstacles potentiels relatifs à deux opérations O_{1,k_1} et O_{2,k_2} si l'une des machines M_{1,r_1} de l'ensemble μ_{1,k_1} (ensemble des machines candidates à l'exécution de O_{1,k_1}) est utilisée par O_{2,k_2}.

Pour pouvoir construire le réseau $N = (V, E, d)$ dans lequel le makespan peut être déterminé, nous sommes amenés à définir :

(a) Les obstacles considérés.

(b) L'ensemble de sommets du réseau.

(c) Les successeurs immédiats d'un sommet.

(d) La distance entre deux sommets voisins.

4.1.1. *Nature des Obstacles*

Du fait que la machine exécutant une opération O_{1k} du job flexible n'est pas fixée, la nature du rectangle (O_{1i}, O_{2j}) dépend de la machine sélectionnée pour l'opération O_{1i}. Aussi, nous considérons un ensemble d'*obstacles potentiels* défini par l'ensemble de tous les obstacles possibles de notre représentation. Un obstacle potentiel est un rectangle $I_{1i} \times I_{2j}$ exprimant le fait que l'opération O_{2j} partage une des machines de l'ensemble μ_{1i} (machines candidates à l'exécution de l'opération O_{1i}).

4.1.2. *Ensemble des sommets*

L'ensemble des sommets du réseau N est composé des coins nord-ouest (*NW*) et sud-est (*SE*) de tous les obstacles potentiels. Dans ce qui suit, ces sommets sont notés par $NW(k_1, k_2)$ et $SE(k_1, k_2)$. Un sommet $NW(k_1, k_2)$ (resp. $SE(k_1, k_2)$) signifie que les opérations O_{1,k_1-1} et O_{2,k_2} (resp. O_{1,k_1} et O_{2,k_2-1}) sont terminées.

Il est clair que l'ensemble des sommets considéré contient tous les sommets définis dans l'approche géométrique classique, et ceci pour toute affectation donnée des machines aux opérations du job flexible.

4.1.3. *Successeurs d'un sommet*

Dans l'approche géométrique classique, chaque sommet v_i possède au plus deux successeurs immédiats : les coins *NW* et *SE* de l'obstacle rencontré en progressant diagonalement depuis v_i. En présence de flexibilité des machines, chaque sommet v_i peut avoir plus de deux successeurs. En effet, nous considérons comme successeurs directs de v_i, tous les coins *NW* et *SE* des obstacles potentiels qui peuvent être atteints en progressant diagonalement à partir de v_i.

L'algorithme *SuccSommet-1* suivant permet de déterminer tous les successeurs directs d'un sommet $v_i = (k_1, k_2)$ quelconque.

Algorithme 4.2 : *SuccSommet-1*

1. *Classer les flexibilités $M_i \in \mu_{1,k_1}$ par ordre croissant des durées opératoires de l'opération O_{1,k_1}, c'est-à-dire, $p_{1,k_1,M_1} \leq p_{1,k_1,M_2} \leq ... \leq p_{1,k_1,M_p}$, où p = card (μ_{1,k_1}).*
2. *Déterminer la première flexibilité M_q pour laquelle J_1 et J_2 peuvent être réalisés en parallèle jusqu'à la fin de l'opération O_{1,k_1}, en partant respectivement de O_{1,k_1} et O_{2,k_2}. Par convention, $q = p+1$ si la flexibilité M_q n'existe pas.*
3. *Décomposer l'ensemble des flexibilités μ_{1,k_1} en deux sous-ensembles S_L et S_U avec $S_L = \{1, 2, ..., q-1\}$ et $S_U = \{q+1, q+2, ..., p\}$. Les flexibilités M_i du sous-ensemble S_U sont négligées.*
4. *Pour chaque flexibilité M_i de O_{1,k_1} dans S_L, le passage diagonale en partant de v_i heurte un obstacle défini par les opérations O_{1,k_1} et $O_{2,k_2'}$. Les coins NW (k_1, $k_2'+1$) et SE (k_1+1, k_2') sont considérés comme deux successeurs directs du sommet v_i.*
5. ***Si** $q = p+1$ **Alors** Arrêter. **Sinon**, le passage diagonal, c'est-à-dire, l'exécution parallèle de J_1 et J_2 aboutit à la fin de la réalisation de l'opération O_{1,k_1} .aux opérations O_{1,k_1+1} et $O_{2,k_2'}$*
6. ***Si** $k_1+1 = n_1$ ou $k_2' = n_2$ **Alors** le sommet final F est considéré comme le dernier successeur direct du sommet v_i et l'exploration est arrêtée. **Sinon**, poser $k_1 \leftarrow k_1+1$, $k_2 \leftarrow k_2'$ et Aller à l'étape 1.*

Fin algorithme.

4.1.4. Distance entre deux sommets

La distance entre un sommet $v_i = (k_1, k_2)$ et un successeur direct $v_j = (k_3, k_4)$ est calculée au cours de l'exploration définie par l'algorithme *SuccSommet-1*. En effet, pour chaque arc $e = v_i v_j$ d'extrémité initiale v_i et terminale v_j, les machines qui doivent exécuter les opérations O_{1k} sont fixées par l'algorithme *SuccSommet-1*, et ceci, pour tout indice $k_1 \leq k \leq k_3$-1. Par conséquent,

- $d(v_i, v_j) = \sum_{k=k_2}^{k_4-1} p_{2k}$ si le sommet v_j est un coin nord-ouest (*NW*),
- $d(v_i, v_j) = \sum_{k=k_1}^{k_3-1} p_{1,k,M_k}$ si le sommet v_j est un coin sud-est (*SE*),

où M_k est la machine affectée (sur l'arc v_i v_j) à l'opération O_{1k}.

Lemme 4.1

Le sous-ensemble de flexibilités $S_U = \{q+1, q+2, ..., p\}$ défini par l'algorithme *SuccSommet-1* n'est pas nécessaire à l'obtention de l'ordonnancement optimal.

Preuve

Considérons les successeurs directs d'un sommet $v_i = (k_1', k_2')$. L'algorithme *SuccSommet-1* calcule une séquence de flexibilité $M_{q,k_1'}, M_{q,k_1'+1}, M_{q,k_1'+2}, ...$, et les sous-ensembles $S_{L,k_1'}, S_{U,k_1'}, S_{L,k_1'+1}, S_{U,k_1'+1}, S_{L,k_1'+2}, S_{U,k_1'+2}, ...$. Le passage diagonal, si les flexibilités sont choisies dans les sous-ensembles $S_{L,k_1'}$, $S_{L,k_1'+1}$, ..., est forcément arrêté par un obstacle. Dans ce cas, deux successeurs sont ajoutés comme cela est défini dans l'algorithme *SuccSommet-1*. Nous avons donc besoin de montrer que les machines M_t des sous-ensembles S_{Uk} pour tout $k \geq k'$ sont inutiles si les machines $M_{q,k_1'}, M_{q,k_1'+1}, ..., M_{q,k-1}$ sont choisies.

En partant du sommet v_i et en utilisant les machines $M_{q,k_1'}, M_{q,k_1'+1}, ..., M_{q,k-1}$, l'application de l'approche géométrique classique pour le cas du job shop implique qu'il existe un plus court chemin P^* de v_i au sommet final F obtenu en progressant diagonalement jusqu'à la fin de l'opération $O_{1,k-1}$. Soit x le point atteint et supposons que le chemin P^* obtenu correspond à une flexibilité $M_t \in S_{Uk}$ pour l'opération O_{1k}. Considérons alors un autre chemin P obtenu à partir de P^* en gardant les mêmes affectations sauf pour la machine M_{qk} appartenant à l'ensemble des flexibilités de l'opération O_{1k}. La figure 4.1 est la superposition de la représentation géométrique des deux chemins P et P^* avec un décalage à gauche de $\Delta = p_{1,k,M_t} - p_{1,k,M_q} \geq 0$ pour la représentation de P^*.

Dans ce qui suit, nous complétons la preuve en montrant que le chemin P^* peut être transformé en un autre chemin caractérisé par l'utilisation de la machine M_{qk} sans en augmenter la longueur.

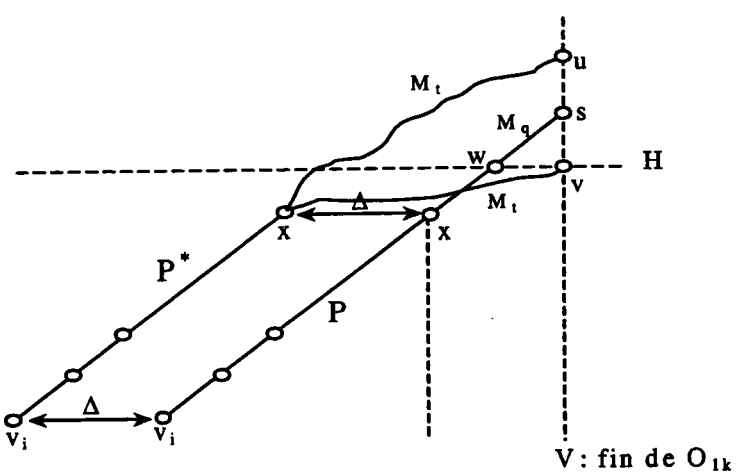


Figure 4.1 : Superposition de deux représentations géométriques

Il est clair que le chemin P progresse le long de la diagonale jusqu'à la fin de l'opération O_{1k}, c'est-à-dire, jusqu'à ce que la verticale V soit atteinte en un point s. Si le chemin P^* croise V en un point u situé au-dessus de s, nous pouvons remplacer la partie de P^* entre v_i et u par le chemin $v_i\ x\ s\ u$ (utilisation de la machine M_{qk}) sans augmenter la longueur du chemin. Si par contre, le chemin P^* croise V en un point v situé en dessous de s, nous considérons alors l'horizontale H passant par le point v. Soit w le point de croisement entre l'horizontale H et le chemin P. En remplaçant la partie de P^* entre v_i et v par le chemin $v_i\ x\ w\ v$ utilisant la machine M_{qk}, nous obtenons un nouveau chemin plus court que P^* de Δ. ■

Comme dans le cas de l'approche géométrique classique, le problème d'ordonnancement d'un job flexible et d'un job de type job shop est équivalent à la recherche du plus court chemin dans le plan. De manière équivalente, ce plus court chemin peut être déterminé dans le réseau construit $N = (V, E, d)$. L'ensemble des sommets V est composé des coins *NW* et *SE* obtenus par l'algorithme *SuccSommet-1* appliqué à chaque sommet.

Les théorèmes suivants montrent que le réseau construit est suffisant pour déterminer l'ordonnancement optimal, et donnent la complexité de l'algorithme polynomial proposé.

Théorème 4.7

L'ensemble des sommets construit en appliquant l'algorithme *SuccSommet-1* est suffisant pour déterminer un ordonnancement optimal.

Preuve

Pour chaque affectation des machines aux opérations du job J_1, une représentation géométrique dans le plan peut être définie. Le problème d'ordonnancement se ramène alors à la recherche du plus court chemin dans ce plan avec obstacles. Un réseau G est donc défini pour résoudre le problème du plus court chemin. D'après les résultats de Brucker [1988], l'ensemble des sommets du réseau G sont les coins *SE* et *NW* des obstacles, et chaque sommet v_i a deux successeurs : les coins *SE* et *NW* heurtés en progressant diagonalement depuis v_i.

Le réseau construit $N = (V, E, d)$ en appliquant l'algorithme *SuccSommet-1* contient tous les coins *NW* et *SE* des obstacles potentiels (k_1, k_2). Il est clair que l'ensemble des sommets V comporte tous les sommets du réseau G pour une affectation particulière. L'ensemble des successeurs directs d'un sommet $v_i = (k_1, k_2)$ est composé des coins *NW* et *SE* des obstacles potentiels heurtés en progressant diagonalement à partir de v_i. De plus, d'après le lemme 4.1, les coins *NW* et *SE* négligés par l'algorithme *SuccSommet-1* correspondent aux arcs qui ne peuvent pas appartenir au plus court chemin. Par conséquent, le réseau N est suffisant pour déterminer un ordonnancement optimal du problème. ■

Remarque 4.2 : Dans l'algorithme *SuccSommet-1*, au pire des cas, chaque progression diagonale impliquant l'utilisation d'une machine $M_{1k} \in \mu_{1k}$ par une opération O_{1k}, $k_1 \le k \le n_1$ heurte un obstacle potentiel. Ainsi, elle génère deux successeurs directs. Par conséquent, le nombre de successeurs immédiats d'un sommet $v_i = (k_1, k_2)$ est majoré par $\sum_{k=k_1}^{n_1} 2\,card(\mu_{1k})$ qui est à son tour borné par $2\times K\times n_1$, où $K = \underset{1\le k\le n_1}{Max}\{card(\mu_{1k})\}$.

4.1.5. Complexité de l'algorithme polynomial

La complexité de l'algorithme de résolution du problème d'ordonnancement à deux jobs en présence d'un seul job flexible dépend fortement du nombre des sommets (arcs) construits par l'algorithme *SuccSommet-1*. Plus précisément :

Théorème 4.8

Le problème noté *JMPM, non-relié / n = 2 ; J ; FJ / C_{max}*, où J signifie job de type job shop et FJ signifie job flexible, est polynomial et sa complexité est au plus $O\left(K \times n_1^2 \times n_2\right)$.

Preuve

Il est clair que le nombre maximum de sommets dans le réseau N est égal à $n_1 \times n_2$. De la remarque 4.2, nous déduisons que le nombre maximum des successeurs directs d'un sommet v_i est $2 \times K \times n_1$. Par conséquent, le nombre maximum d'arcs dans le réseau est $2 \times K \times n_1^2 \times n_2$. Puisque le réseau N est acyclique, la détermination du plus court chemin s'effectue en un temps proportionnel à $O(|V| + |E|)$. En conséquence, la complexité de l'algorithme est au plus $O\left(K \times n_1^2 \times n_2\right)$. ■

Le résultat donné par le théorème 4.8 implique que le problème étudié dans cette section est polynomial, mais seulement pour la minimisation du makespan. Dans ce qui suit, nous montrons que cet algorithme est applicable à n'importe quelle fonction objectif régulière.

Pour ce faire, nous considérons un critère régulier arbitraire $f(C_1, C_2)$. Les résultats donnés par le lemme 4.1 et le théorème 4.7 restent valables, et les machines ignorées par l'algorithme *SuccSommet-1* le sont encore. La construction du réseau est exactement la même que dans le cas du makespan. La seule différence concerne le calcul de la valeur du critère. Comme cela apparaît dans (Brucker [1998], page 184), nous avons besoin de considérer tous les arcs (x, y) du réseau N tels que le chemin P_{xy} déterminé par l'algorithme *SuccSommet-1* heurte l'obstacle final en un point u.

Soient :

- $d(O, x)$ la longueur du plus court chemin de O à x.
- $d(x, u)$, $d(u, y)$, $d(y, F)$ respectivement les longueurs des chemins de x à u, de u à y et de y à F.

La valeur du critère objectif correspondant au chemin P_{xy} est $f(C_1, C_2)$. Si le point u est sur la bordure droite de l'obstacle final alors :

$$C_1 = d(0, x)+d(x, u) \text{ et } C_2 = C_1 + d(u, y) + d(y, F).$$

Si par contre, u est sur la bordure supérieure alors

$$C_2 = d(0, x) = d(x, u) \text{ et } C_1 = C_2 + d(u, y) + d(y, F).$$

De plus, au pire des cas, il existe $2 \times K \times n_1^2 \times n_2$ arcs heurtant l'obstacle final. Puisque le calcul de $f(C_1, C_2)$ prend un temps proportionnel à $O(\max\{n_1, n_2\})$, le problème de minimisation d'un critère régulier reste polynomial et nous pouvons conclure que :

Théorème 4.9

Le problème *JMPM, non-relié* | $n = 2$; J ; FJ | f est polynomial pour tout critère régulier $f \in \left\{L_{\max}, \sum T_i, \sum w_i T_i, \sum w_i C_i, \sum U_i, \sum w_i U_i\right\}$.

4.2. Cas préemptif

Dans cette section, nous proposons une extension de l'algorithme géométrique proposé dans la section précédente pour résoudre le problème d'ordonnancement à deux jobs lorsque la préemption des opérations est permise.

Cette extension est similaire à l'approche proposée par Sotskov [1991] pour le problème d'ordonnancement du job shop à deux jobs avec préemption des opérations. Dans cette dernière, un chemin dans le plan allant de l'origine O au point final F peut progresser horizontalement ou verticalement à l'intérieur d'un obstacle. Afin de prendre en compte cette possibilité, nous renforçons l'ensemble des successeurs immédiats d'un sommet en définissant des sommets additionnels.

Plus précisément, dans l'étape 4 de l'algorithme *SuccSommet-1*, le passage diagonal partant d'un sommet v_i et utilisant une machine $f_i \in S_L$ heurte un obstacle $D = (k_1, k_2)$. Selon la manière de heurter l'obstacle D, les deux situations suivantes se présentent :

- Si la bordure inférieure est heurtée (figure 4.2(a)) alors le coin SE (k_1+1, k_2) et un nouveau point A obtenu en progressant verticalement à l'intérieur de l'obstacle D jusqu'à atteindre la bordure supérieure, sont deux successeurs directs de v_i.
- Si la bordure gauche est heurtée (figure 4.2(b)) alors le coin NW (k_1, k_2+1) et un nouveau point B obtenu en progressant horizontalement à l'intérieur de l'obstacle D jusqu'à atteindre la bordure droite, sont deux successeurs directs de v_i.

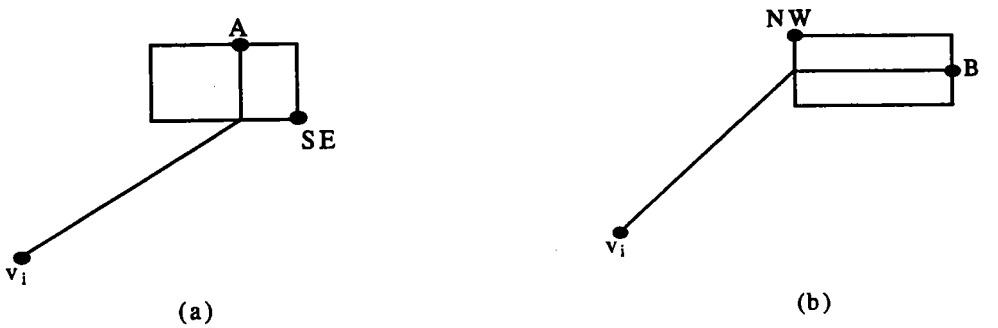


Figure 4.2 : Nouveaux successeurs du sommet v_i

Définition 4.1 : Nous appelons *coin singulier* tout point obtenu en progressant horizontalement ou verticalement à l'intérieur d'un obstacle. Les coins *SE* et *NW* classiques sont appelés *coins réguliers*.

Théorème 4.10

Le problème *JMPM, non-relié* | $n = 2$; *J* ; *FJ* ; *pmtn* | f est polynomial pour toute fonction objectif régulière $f \in \{L_{max}, \sum T_i, \sum w_i T_i, \sum w_i C_i, \sum U_i, \sum w_i U_i\}$ et sa complexité est $O(K^2 \times (n_1 + n_2) \times n_1^2 \times n_2)$.

Preuve

La complexité du nouvel algorithme dépend du nombre de sommets (arcs) du réseau construit. Il est clair que le nombre de coins réguliers est $n_1 \times n_2$. Pour déterminer le nombre de sommets du réseau, nous devons donc calculer une borne supérieure du nombre de coins singuliers obtenus à partir d'un coin v_i.

A chaque opération O_{1k} du job flexible, l'algorithme *SuccSommet-1* peut créer au plus $card(\mu_{1k})$ coins singuliers. Donc, avant l'arrivée au point final F, $K \times (n_1 + n_2)$ coins singuliers peuvent avoir été générés, où $K = \max_{1 \le k \le n_1} \{card(\mu_{1k})\}$. Par conséquent, le réseau N comporte au plus $K \times (n_1 + n_2) \times n_1 \times n_2$ sommets. En outre, chaque sommet v_i a au plus $2 \times K \times n_1$ successeurs directs. Par conséquent, le nombre d'arcs du réseau est borné par $2 \times K^2 \times (n_1 + n_2) \times n_1^2 \times n_2$. Puisque le réseau N est acyclique, la complexité du nouvel algorithme est $O(K^2 \times (n_1 + n_2) \times n_1^2 \times n_2)$. ■

5. Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'étude de la complexité des problèmes d'ordonnancement de type job shop à deux jobs avec flexibilité sur l'exécution des opérations. De nouveaux résultats de complexité ont été présentés et des algorithmes polynomiaux ont été proposés.

Dans un premier temps, nous avons démontré que le problème d'ordonnancement à deux jobs était NP-difficile au sens faible même si la préemption des opérations est permise. Les résultats ont été prouvés pour n'importe quel critère objectif régulier. La démarche utilisée pour les démonstrations a permis d'établir la complexité d'un problème particulier appelé *flow shop hybride* dans le cas de la minimisation du makespan. Nous avons ensuite proposé un algorithme pseudo-polynomial pour trouver des ordonnancements optimaux.

Dans un second temps, nous avons étudié un cas particulier du modèle précédent, cas où un seul job est flexible et l'autre est de type job shop. Pour ce modèle, un nouvel algorithme polynomial étendant l'approche géométrique classique a été développé pour la minimisation de toute fonction objectif régulière, dans le cas où la préemption des opérations n'est pas permise. En présence de la préemption, une extension de l'algorithme précédent a été proposée toujours pour la minimisation de n'importe quelle fonction objectif régulière.

Les différents résultats obtenus dans ce chapitre, en particulier les algorithmes polynomiaux proposés, ont été très encourageants. Ils ont en outre constitué un point de départ à des développements intéressants dans deux axes de recherche. Le premier axe est l'élaboration, par exemple en utilisant la méthodologie introduite dans le chapitre 2, de méthodes approchées permettant de résoudre le problème général à plusieurs jobs. C'est ce qui est décrit dans le chapitre suivant. Le second axe de recherche consiste à généraliser l'algorithme polynomial de manière à résoudre le problème d'ordonnancement d'un job flexible et d'un job de type job shop, avec la contrainte de flexibilité et la propriété *retenir et attendre*. Ce point sera traité dans le chapitre 6.

Chapitre 5

Job Shop Multi-Ressources avec Flexibilité des Ressources

Le modèle du job shop multi-ressources avec flexibilité des ressources fournit une représentation générique pour la résolution de plusieurs problèmes d'ordonnancement rencontrés dans les systèmes manufacturiers. Ce modèle est une généralisation du job shop flexible caractérisé par le fait qu'une opération nécessite plusieurs ressources pour sa réalisation, chacune de ces ressources pouvant être disponible en plusieurs unités. Le problème d'ordonnancement consiste alors à affecter un ensemble de ressources à chaque opération et à déterminer les séquences d'opérations sur les ressources choisies. Dans ce chapitre, nous généralisons l'algorithme polynomial présenté dans le chapitre précédent afin de résoudre le problème à deux jobs du modèle considéré ici. En utilisant la méthodologie introduite dans la première partie de la thèse, nous proposons ensuite une heuristique de résolution pour le cas général à plusieurs jobs. Afin d'en améliorer l'efficacité, cette heuristique est couplée à un algorithme génétique. Enfin, des résultats d'expérimentations, réalisées sur des instances issues de la littérature et sur de nouvelles instances générées aléatoirement, sont présentés.

Une partie des résultats proposés dans ce chapitre a été présentée dans les conférences internationales : IEEESMC [2001] (*IEEE Conference, Systems, Man, Cybernetics*), *à Tucson, Arizona et PMS* [2002] (*Project Management and Scheduling*), *à Valence, Espagne. Ces résultats font en outre l'objet d'un article qui sera prochainement soumis à la revue scientifique Annals of Operations Research.*

1. Introduction

Ces dix dernières années, un nombre important de travaux de recherche a été consacré aux problèmes d'ordonnancement de type flow shop, job shop, etc. Ces modèles classiques permettent d'aborder une partie des problèmes d'ordonnancement rencontrés dans la pratique. Cependant, certaines de leurs hypothèses, notamment la présence de machines en une seule unité ainsi que la possibilité pour ces machines de n'exécuter qu'une seule tâche à la fois, sont restrictives et constituent des limites à leur champ d'application. En effet, dans de nombreux ateliers existants tels que les systèmes flexibles manufacturiers (*FMSs*), la réalisation d'une opération peut nécessiter plusieurs ressources, et chaque ressource peut être disponible en plusieurs unités. Par ailleurs, pour faire face aux aléas de la production, les industriels sont de plus en plus amenés à introduire de la flexibilité sur l'exécution des opérations. Un moyen de créer une telle flexibilité est de permettre à une opération d'être exécutée par un ensemble de ressources candidates.

Dans la littérature, le problème d'ordonnancement de type job shop avec opérations multi-ressources et flexibilité, est appelé job shop multi-ressources avec flexibilité des ressources (Dauzère-Pérès et al. [1998]), ou job shop multi-mode (Brucker et Neyer [1998]). Un mode correspond à un sous-ensemble de ressources candidates à l'exécution d'une opération. Dans la suite de ce chapitre, nous adoptons pour notre problème le nom de job shop multi-ressources avec flexibilité des ressources.

Comme cela a été souligné au chapitre 1, toutes les approches existantes de la littérature pour résoudre les problèmes de job shop flexible et de job shop avec flexibilité des ressources traitent du cas où les machines (ressources) sont disponibles en une seule unité. Elles supposent de plus qu'une machine ne peut exécuter qu'une opération à la fois. Ces approches sont basées sur une représentation par le graphe disjonctif du problème d'ordonnancement. Cependant, si l'on considère qu'une machine peut être disponible en plusieurs unités, les approches existantes ne peuvent être appliquées directement. Elles doivent être transformées de manière à prendre en compte chacune des unités comme étant une machine (ressource) distincte. Dans Dauzère-Pérès et al. [1998], il a été constaté que cette transformation n'est pas toujours souhaitable du fait qu'elle augmente la taille du problème. Une autre approche est donc de ne pas différencier les ressources identiques. Dans ce chapitre, nous proposons une heuristique qui intègre les problèmes d'affectation et de séquencement, et ce, sans faire la distinction entre les ressources identiques.

La suite du chapitre est organisée de la manière suivante. Dans la section 2, nous donnons une description du problème d'ordonnancement de type job shop multi-ressources avec flexibilité des ressources. La complexité du problème particulier à deux jobs est abordée en section 3 avec la proposition d'un algorithme polynomial. La section 4 est dédiée à la méthode de résolution du problème général à plusieurs jobs. L'heuristique proposée est basée sur une nouvelle extension de l'algorithme polynomial du problème à deux jobs, sur la notion de job composé et sur des algorithmes génétiques. Les sections 5 et 6 sont respectivement consacrés aux résultats expérimentaux et aux conclusions.

2. Description du problème

Le modèle du job shop multi-ressources avec flexibilité noté *JSMF* est une généralisation directe du job shop flexible (cf. chapitre 1). Il intègre à la fois, les contraintes du modèle du *job shop multiprocesseur* noté *MPT* job shop (Drozdowski [1996]), dans lequel une opération peut nécessiter plusieurs ressources en même temps, et celles du modèle du job shop flexible.

Plus précisément, un ensemble de N jobs doit être réalisé sur un ensemble de ressources $\Re = \{1, 2, ..., r, ..., l\}$, où l est le nombre de types de ressources différentes. Le terme de ressources n'est pas restreint aux machines, mais désigne également des opérateurs humains, des ressources de transport, etc. Chaque ressource R_r est disponible en une quantité $Q_r \geq 1$. A chaque job J_i est associée une séquence linéaire d'opérations (appelée aussi *gamme opératoire*) $\{O_{i1}, O_{i2}, ..., O_{in_i}\}$, où n_i est le nombre d'opérations du job J_i. Les opérations doivent être réalisées suivant un ordre défini par le processus de fabrication. A chaque opération O_{ij} est associé une liste d'ensembles candidats de ressources $R_{ij} = \{R_{ij}^1, R_{ij}^2, ..., R_{ij}^{m_{ij}}\}$ nécessaires à sa réalisation, où $R_{ij}^k \subseteq \Re$ et m_{ij} est la cardinalité de l'ensemble R_{ij}. La durée opératoire de l'opération O_{ij} notée p_{ijk} est connue à l'avance, et dépend de l'ensemble des ressources R_{ij}^k sélectionné. La préemption des opérations n'est pas permise et nous supposons que les ressources utilisées par une opération sont toutes libérées en même temps, à la fin de son exécution.

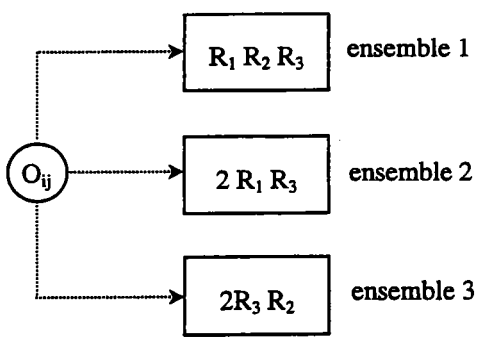


Figure 5.1 : Ensemble des ressources candidates

Dans la figure 5.1, la ressource R_2 est disponible en une seule unité contre deux pour les ressources R_1 et R_3. Trois choix de ressources sont possibles pour exécuter l'opération O_{ij} :

- Le premier ensemble contient les trois ressources R_1, R_2 et R_3.
- Avec le second ensemble, O_{ij} utilise deux types de ressources, mais avec deux unités de R_1.
- Le dernier ensemble candidat est composé de deux unités de R_3 et une unité de R_2.

Le problème d'ordonnancement du *JSMF* consiste à résoudre à la fois les problèmes d'affectation et de séquencement, et ce, dans le but de minimiser une fonction objectif donnée. Le problème d'affectation consiste à affecter à chaque opération un ensemble candidat de ressources, et le problème de séquencement consiste à déterminer la séquence de passage des opérations sur les ressources sélectionnées. Dans ce chapitre, nous considérons la minimisation du makespan.

3. Problème à deux jobs

Le problème d'ordonnancement du job shop multi-ressources avec flexibilité des ressources est une généralisation du *job shop flexible* ou *job shop with multi-purpose machines* de la littérature. Dans le chapitre précédent, nous avons montré que le problème d'ordonnancement à deux jobs du modèle *job shop flexible avec machines non-reliées* était NP-difficile. Ceci implique que le problème à deux jobs du modèle *JSMF* est aussi NP-difficile.

Dans cette section, nous considérons un cas particulier du problème *JSMF* à deux jobs noté 2-*JSMF* pour lequel le premier job est flexible, c'est-à-dire qu'un ensemble de ressources candidates est associé à chaque opération, et le deuxième job est un job shop multi-ressources, c'est-à-dire qu'une affectation des ensembles candidats aux opérations est déjà donnée. Nous généralisons l'algorithme polynomial proposé dans le chapitre 4 de manière à tenir compte des caractéristiques de *JSMF*. Nous rappelons que cet algorithme polynomial est basé sur la représentation géométrique (Akers et Friedman [1955]), et consiste à construire un réseau $N = (V, E, d)$ sur lequel l'ordonnancement optimal est obtenu. Le réseau est défini par l'ensemble de ses sommets, par le nombre de successeurs directs d'un sommet et par la distance entre deux sommets voisins.

3.1. Ensemble des sommets

L'ensemble des sommets V comporte tous les coins nord-ouest (*NW*) et sud-est (*SE*) des obstacles potentiels. Un obstacle potentiel (k_1, k_2) est défini si les deux conditions suivantes sont réunies :

(i) une ressource R_r utilisée par O_{2,k_2} apparaît dans un des ensembles candidats à l'exécution de O_{1,k_1},

(ii) la somme des quantités de R_r utilisées par O_{2,k_2} et O_{1,k_1} excède la capacité maximal Q_r de R_r.

L'origine O et le point final F sont considérés comme des sommets dégénérés. Il est clair que l'ensemble V contient tous les sommets apparaissant dans le cas du *job shop multi-ressources* (Brucker[1998]), et ceci pour toute affectation des ressources aux opérations du job flexible J_1. Notons qu'atteindre le sommet $v_j = (k_1, k_2)$ signifie que le job J_1 (resp. J_2) a déjà toutes ses opérations O_{1k}, avec $1 \leq k \leq k_1\text{-}1$ (resp. $1 \leq k \leq k_2\text{-}1$) exécutées.

3.2. Successeurs directs d'un sommet

Pour construire l'ensemble des arcs E du réseau, nous devons définir tous les successeurs directs d'un sommet v_i. Contrairement aux approches géométriques classiques pour lesquelles le nombre de successeurs est au plus égal à deux, le sommet $v_i = (k_1, k_2)$ peut avoir plusieurs successeurs directs. Ces successeurs sont obtenus en utilisant l'algorithme *SuccSommet-2* suivant :

Algorithme 5.1 : *SuccSommet-2*

1. *Classer les ensembles candidats* $R^t_{1,k1}$ *par ordre croissant des durées opératoires* $p_{1,k1,t}$, *c'est-à-dire,* $p_{1,k1,1} \leq p_{1,k1,2} \leq \ldots$.
2. *Déterminer le premier ensemble candidat* $R^q_{1,k1}$ *pour lequel* J_1 *et* J_2 *peuvent être réalisés simultanément jusqu'à la fin de l'opération* $O_{1,k1}$, *en partant respectivement de* $O_{1,k1}$ *et* $O_{2,k2}$. *Par convention,* $q = m_{1,k1}+1$ *si l'ensemble candidat* $R^q_{1,k1}$ *n'existe pas.*
3. *Décomposer l'ensemble des ensembles candidats en deux sous-ensembles* $S_L = \{1, 2, \ldots, q-1\}$ *et* $S_U = \{q+1, q+2, \ldots, m_{1,k1}\}$. *Les ensembles candidats dans* S_U *sont négligés.*
4. *Pour chaque ensemble candidat* $R^t_{1,k1}$ *avec* $t \in S_L$, *le passage diagonal en partant de* v_i *heurte un obstacle défini par* O_{1,k_1} *et* O_{2,k'_2}. *Les coins NW* (k_1, k'_2+1) *et SE* (k_1+1, k'_2) *sont considérés comme deux successeurs directs du sommet* v_i.
5. ***Si*** $q = m_{1,k1}+1$ ***Alors*** *Stop.*

   Sinon

   le passage diagonal, c'est-à-dire l'exécution parallèle de J_1 *et* J_2, *conduit aux opérations* O_{1,k_1+1} *et* O_{2,k'_2} *à la fin de la réalisation de l'opération* $O_{1,k1}$.

   Fin Sinon.
6. ***Si*** $k_1+1 = n_1$ *ou* $k'_2 = n_2$ ***Alors***

   le sommet final F est considéré comme le dernier successeur direct du sommet v_i *et l'exploration est arrêtée.*

   Fin Si.

   Sinon

   - *Poser* $k_1 \leftarrow k_1+1$, $k_2 \leftarrow k'_2$.
   - *Aller à l'étape 1.*

   Fin Sinon.

Fin algorithme.

Notons que cet algorithme est une généralisation de l'algorithme proposé dans le chapitre 4 pour lequel seulement une machine était nécessaire à la réalisation d'une opération. Les ensembles candidats de l'ensemble S_U négligés à l'étape 3 de l'algorithme *SuccSommet-2* ne conduisent pas à des solutions optimales. C'est ce qui est justifié par le lemme 5.1 suivant :

Lemme 5.1
Les ensembles candidats $R^t_{1,k1}$ avec $t > q$ ne sont pas nécessaires à l'obtention de l'ordonnancement optimal.

Preuve
Nous prouvons que les ensembles candidats $R^t_{1,k1}$ $t > q$, peuvent être négligés sans que la valeur du makespan n'augmente. Pour ce faire, soit P un plus court chemin du sommet v_i au sommet final F utilisant l'ensemble candidat $R^t_{1,k1}$ pour l'opération O_{1,k_1}. Soit r le point de croisement entre P et la verticale V_1 représentant la fin de l'opération $O_{1,k_1\text{-}1}$.

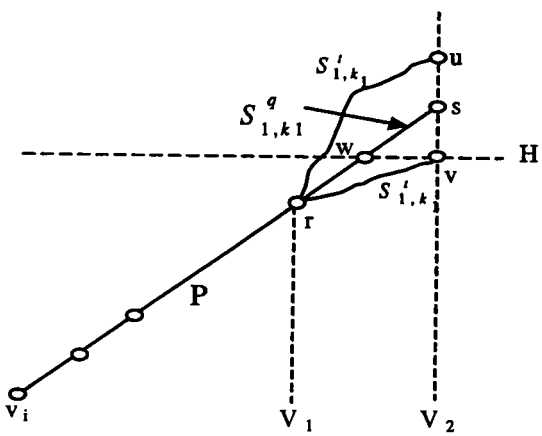


Figure 5.2 : Plus court chemin

Nous considérons un chemin P^* obtenu à partir de P en gardant les mêmes ensembles candidats, sauf $R^t_{1,k1}$ pour l'opération O_{1,k_1}, qui est remplacé par $R^q_{1,k1}$. Il est clair que P^* progresse diagonalement jusqu'à atteindre la verticale V_2 représentant la fin de l'opération O_{1,k_1} (figure 5.2). Si le chemin P croise V_2 à un point "u" situé au-dessus de "s", alors nous pouvons remplacer la partie du chemin P entre v_i et "u" par "v_i r s u" sans augmenter la valeur du makespan, puisque le job J_2 n'est jamais interrompu. Si par contre P croise V_2 à un point "v" situé en dessous de s, nous considérons l'horizontale H croisant "v". Soit w le point d'intersection entre H et la diagonale de P^*. En remplaçant la partie de P entre v_i et "v" par le chemin "v_i r w v", nous obtenons un nouveau chemin plus court, du fait que $p_{1,k_1,q} \leq p_{1,k_1,t}$ et que le job flexible J_1 n'est jamais interrompu entre r et v. ■

3.3. Distance entre deux sommets

La distance entre le sommet $v_i = (k_1, k_2)$ et chaque successeur direct $v_j = (k_3, k_4)$ peut être calculée au cours de l'exploration. En effet, les ensembles candidats pour les opérations du job flexible J_1 situées dans l'intervalle $k_1 \leq k \leq k_3\text{-}1$ sont déjà fixés par l'algorithme *SuccSommet-2*.

Dans le cas où v_j est un sommet *SE* (resp. *NW*), la distance entre v_i et v_j est la projection sur l'axe X (resp. Y). Plus précisément,

- $d(v_i, v_j) = \sum_{k=k_2}^{k_4-1} p_{2k}$ si le sommet v_j est un coin *NW*.
- $d(v_i, v_j) = \sum_{k=k_1}^{k_3-1} p_{1k}$ si le sommet v_j est un coin *SE*.

où p_{1k} est la durée de O_{1k} en utilisant l'ensemble candidat $R'_{1,k1}$ choisi par l'algorithme *SuccSommet-2* (sur l'arc v_i v_j).

Théorème 5.1

L'ensemble des sommets construits par l'algorithme *SuccSommet-2* est suffisant pour obtenir le plus court chemin dans le plan et donc, la solution optimale pour le problème d'ordonnancement.

Preuve

Dans le réseau $N = (V, E, d)$ défini par l'algorithme *SuccSommet-2*, l'ensemble des successeurs directs d'un sommet $v_i = (k_1, k_2)$ contient tous les coins *SE* et *NW* des obstacles potentiels rencontrés lors d'une progression diagonale partant de v_i et utilisant les ensembles candidats choisis. Par conséquent, l'ensemble V contient tous les sommets du réseau pour le problème du job shop multi-ressources (Brucker [1998]) ; c'est-à-dire, pour une affectation donnée. Par ailleurs, les coins *NW* et *SE* négligés par l'algorithme *SuccSommet-2* correspondent aux arcs qui ne conduisent pas à un plus court chemin (lemme 5.1). Par conséquent, le réseau construit N est suffisant pour l'obtention d'un ordonnancement optimal. ■

3.4. Complexité du problème 2-JSMF

Le réseau N étant sans circuits, la détermination du plus court chemin prend un temps propositionnel à $O(|V| + |E|)$. L'ensemble des sommets (arcs) dépend du nombre de successeurs directs d'un sommet v_i générés par l'algorithme *SuccSommet-2*. Le pire des cas se produit lorsque le passage diagonal, en utilisant tous les ensembles candidats associés à une opération sauf le dernier, heurte un obstacle. Dans ce cas, deux successeurs sont ajoutés pour chaque ensemble candidat. Par conséquent, un sommet $v_i = (k_1, k_2)$ peut avoir au maximum $2 \times \sum_{j=k_1}^{n_1} m_{1j}$ successeurs directs, qui est borné par $2 \times K \times n_1$, où $K = \max_{1 \le j \le n_1} \{m_{1j}\}$.

Théorème 5.2

Le problème d'ordonnancement *JSMF* | $n = 2$ | C_{max} est polynomial et sa complexité est au plus $O(K \times n_1^2 \times n_2)$.

Preuve

La complexité de l'algorithme est une conséquence directe de l'absence de circuits dans le réseau N. En effet, puisque l'ensemble des sommets V est composé des coins SE et NW des obstacles potentiels, sa cardinalité est bornée par $n_1 \times n_2$. De plus, le nombre maximum de successeurs directs d'un sommet v_i est $2 \times K \times n_1$. Par conséquent, le nombre maximum d'arcs du réseau est $2 \times K \times n_1^2 \times n_2$, et la complexité de l'algorithme est au plus $O(K \times n_1^2 \times n_2)$. ■

Remarque 5.1 : L'algorithme polynomial peut être étendu pour résoudre le problème 2-*JSMF* avec n'importe quel critère objectif régulier. Pour ce faire, il suffit de modifier le calcul de la valeur du critère à chaque fois que l'obstacle final défini par l'origine O et le point final F est heurté. Puisque la valeur du critère peut être calculée en un temps polynomial ($\max\{n_1, n_2\}$), nous avons :

Théorème 5.3

Le problème d'ordonnancement *JSMF* $\mid n = 2 \mid f$ est polynomial pour n'importe quel critère régulier f, et sa complexité est au plus $O(K \times n_1^2 \times n_2)$.

3.5. Exemple

Nous considérons dans cette section un exemple permettant d'illustrer le fonctionnement de l'algorithme. Soient deux jobs J_1 et J_2 à ordonnancer sur un ensemble de cinq ressources R_i, $i = 1,\ldots,5$. Le job J_1 est le job flexible et J_2 est le job de type job shop multi-ressources. Les deux jobs comprennent quatre opérations, avec les gammes opératoires suivantes :

$$\text{Job 1} = \{[R_1\, R_2\,(3),\ 2R_1\,(4),\ R_3\, R_4\,(5)];[R_2\,(4), R_3\,(5)];[R_5\,(1)];[R_1\,(2),\ R_2\,(2)]\}$$
$$\text{Job 2} = \{R_3\, R_4\,(1)\,;R_2\,(4)\,;\ R_3\,(2)\,;\ R_2\, R_3\,(1)\}$$

La ressource R_5 est disponible en une seule unité, contre deux pour les ressources R_1, R_2, R_3 et R_4. Les ensembles candidats pour toutes les opérations sont déjà classés dans l'ordre croissant des durées opératoires (étape 1 de l'algorithme).

Considérons d'abord un sommet pour illustrer le cas où un des ensembles candidats permet d'exécuter simultanément les deux jobs J_1 et J_2. A partir du sommet origine $O = (0, 0)$, l'étape 2 de l'algorithme *SuccSommet-2* vérifie l'exécution simultanée des deux jobs J_1 et J_2 en utilisant le premier ensemble candidat $R_1\ R_2$ (3). Puisque ce dernier partage la ressource R_2 avec l'opération O_{22} du job J_2, le passage diagonal heurte l'obstacle (1,2) (figure 5.3). Ainsi, les coins $SE = (2,2)$ et $NW = (1,3)$ sont des successeurs directs du sommet (0, 0). Ensuite, l'algorithme teste le deuxième ensemble candidat $2R_1$ (4). Dans ce cas, les jobs J_1 et J_2 peuvent être réalisés simultanément et l'étape 2 se termine avec $q = 2$ représentant le rang de l'ensemble candidat $2R_1$ (4). L'étape 3 de l'algorithme néglige le troisième ensemble candidat $R_3\ R_4$ (5).

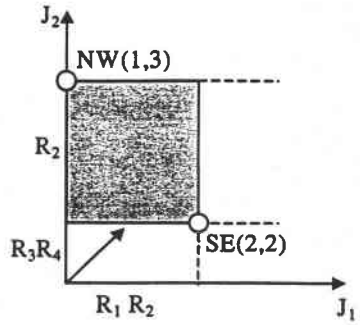


Figure 5.3 : Représentation géométrique avec $R_{11}^1 = R_1\ R_2(3)$

Considérons maintenant le sommet (2, 2) pour illustrer le cas où aucun ensemble candidat ne permet d'exécuter simultanément les deux jobs J_1 et J_2. Au sommet SE (2, 2) les opérations O_{12} du job J_2 et O_{22} du job J_2 sont les prochaines à être exécutées. Le passage diagonal utilisant le premier (resp. second) ensemble candidat R_2 (4) (resp. R_3 (5)) heurte l'obstacle défini par le rectangle (2, 2) (resp. (2, 3)). Donc, dans les deux cas, les deux jobs ne peuvent être réalisés en parallèle. L'exploration du sommet (2,2) est alors terminée (étape 5).

Le réseau $N = (V, E, d)$ construit par l'algorithme *SuccSommet-2* est donné par la figure 5.4.

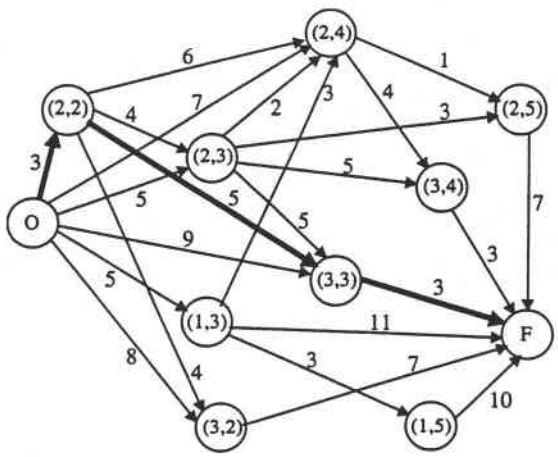


Figure 5.4 : Réseau construit

Le plus court chemin dans le réseau obtenu N est donné par le chemin **O→(2,2)→(3,3)→F**. Il correspond à l'affectation suivante des ensembles candidats aux opérations du job flexible :

- Opérations O_{11}, O_{13} et O_{14} : premier ensemble candidat.
- Opération O_{12} : second ensemble candidat.

Les dates du début des opérations de l'ordonnancement optimal sont :

$$S_{11} = 0, S_{12} = 3, S_{13} = 8, S_{14} = 9, S_{21} = 0, S_{22} = 3, S_{23} = 8, S_{24} = 10,$$

où S_{jk} est la date du début de l'opération k du job J_j. La valeur du makespan est égale à 11.

4. Approche de résolution

Etant une extension du problème d'ordonnancement du *job shop multiprocesseur* qui est NP-difficile, le problème d'ordonnancement étudié dans ce chapitre est aussi NP-difficile. Ceci implique qu'il n'est pas possible de trouver rapidement des solutions optimales au moyen de méthodes exactes. Aussi avons-nous opté pour l'utilisation d'une heuristique capable de fournir des solutions satisfaisantes en un temps d'exécution raisonnable. Cette heuristique est basée sur l'algorithme de résolution du problème à deux jobs, sur la notion de job composé, et utilise la méthodologie introduite dans le chapitre 2. Par souci de clarté, nous rappelons, avant la description de l'heuristique, la notion de job composé.

4.1. Job composé

L'idée d'utiliser un job composé pour développer une heuristique gloutonne a été introduite dans le chapitre 2. Pour un ordonnancement donné, le but est de regrouper les opérations des jobs ordonnancés de manière à construire un *job global*. Pour le problème étudié dans ce chapitre, le job global est caractérisé par le fait que l'affectation des ensembles candidats à chacune des opérations ainsi que les séquences de passage des opérations sur les ressources sont fixes.

La construction du job composé est donnée par l'algorithme *JobCompose* suivant :

Algorithme 5.2 : *JobCompose*

1. *Construire le diagramme de Gantt de l'ordonnancement considéré.*
2. *Déterminer l'ensemble de points* $\{t_0, t_1, \ldots, t_L\}$ *dans l'intervalle* $[0, C_{max}]$, *correspondant aux débuts et fins des opérations. Par convention,* $t_0 = 0$, $t_{i+1} > t_i$, *et* $t_L = C_{max}$. *En procédant ainsi l'intervalle* $[0, C_{max}]$ *est décomposé en L sous-intervalles.*
3. *Associer à chacun des sous-intervalles* $[t_{k-1}, t_k]$ *une opération* $O_{com,k}$ *du job composé tels que l'ensemble des ressources nécessaires à* $O_{com,k}$ *contient les ressources utilisées par tous les jobs dans l'intervalle* $[t_{k-1}, t_k]$. *La durée opératoire de* $O_{com,k}$ *est* $p_{com,k} = t_k - t_{k-1}$.

Fin Algorithme.

4.2. Ordonnancement d'un job composé et d'un job flexible

Dans cette section, nous apportons une modification à l'algorithme de la section précédente. En effet, l'application directe de cet algorithme, en présence d'un job composé, peut générer des ordonnancements préemptifs.

Considérons l'instance à deux jobs introduite dans la section 5.3 et commençons par construire le job composé associé à l'ordonnancement optimal trouvé.

La première phase consiste en la représentation de la solution par le diagramme de Gantt (figure 5.5).

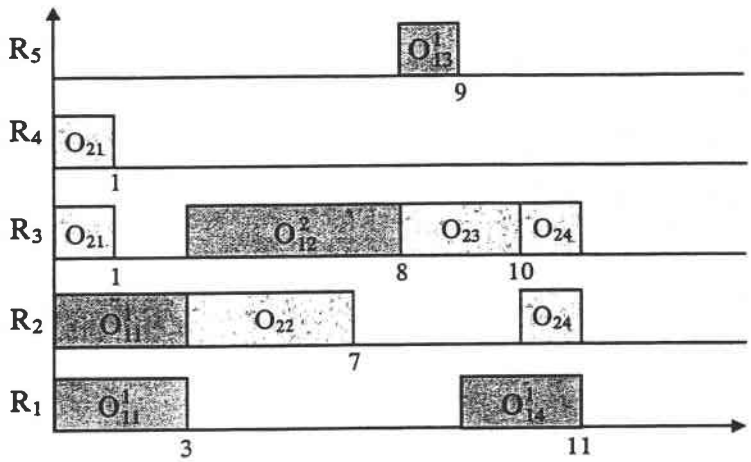


Figure 5.5 : Diagramme de Gantt

L'intervalle [0, C_{max}] est décomposé en 7 sous-intervalles [0, 1], [1, 3], [3, 7], [7, 8], [8, 9], [9, 10] et [10, 11]. Le nombre d'opérations du job composé est n_{com} = 7. Les ressources nécessaires à chaque opération sont les ressources occupées sur l'intervalle correspondant.

La gamme opératoire du job composé est donc donnée par :

$$J_{com} = \{R_1\ R_2\ R_3\ R_4(1), R_1\ R_2(2), R_2\ R_3(4), R_3(1), R_3\ R_5(1), R_1\ R_3\ (1), R_1\ R_2\ R_3(1)\}.$$

Considérons à présent, l'ordonnancement du job J_{com} et d'un nouveau job J_3 dont la gamme opératoire est la suivante :

$$J_3 = \{[R_5(2),\ R_1\ (4)]; [R_2\ (2)]; [R_4\ (1)]; [R_1\ R_3(3)]\}.$$

Examinons l'affectation du premier ensemble candidat $R_5(2)$ à l'opération O_{31}. En partant de l'origine O, le passage diagonal heurte l'obstacle défini par le rectangle (2, 2) puisque $O_{com,2}$ et O_{32} partagent la ressource R_2. Si J_{com} était un job ordinaire, l'algorithme *SuccSommet-2* ajouterait les coins SE = (3, 2) et NW = (2, 3) comme successeurs directs de O. Le chemin résultant qui va de O au coin SE est montré par la figure 5.6. Malheureusement, ce chemin n'est pas possible dans le cas du job composé car :

i) L'opération O_{11} utilisant les ressources R_1 et R_2 appartient à $O_{com,1}$ et $O_{com,2}$.

ii) Les opérations O_{11} et O_{32} partagent la ressource R_2.

iii) Le passage horizontal H nécessite la préemption de O_{11} à cause de i) et ii).

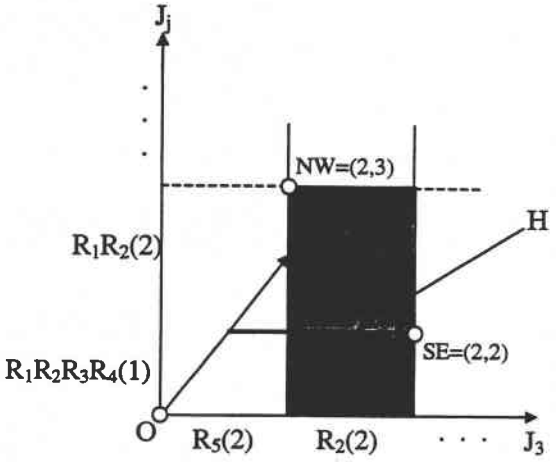


Figure 5.6 : Cas classique

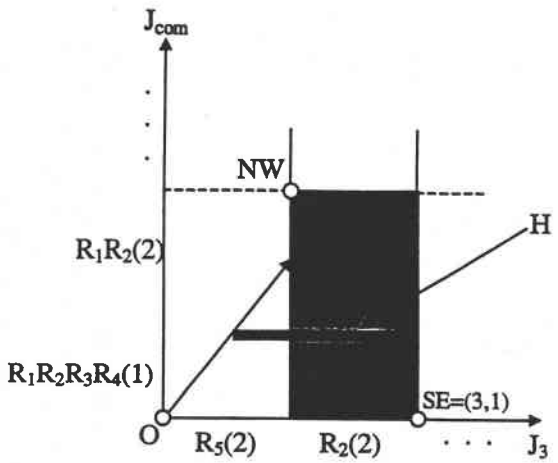


Figure 5.7 : Cas du job composé

Pour éviter d'obtenir des solutions irréalisables vis-à-vis de la contrainte de préemption, l'algorithme *SuccSommet-2* est modifié de la manière suivante. Lors d'un passage diagonal en partant d'un sommet v_i, chaque fois qu'une ligne horizontale H exprimant la fin d'une opération du job composé est croisée, nous continuons le long de l'horizontale H jusqu'à ce que l'une des conditions suivantes soit satisfaite :

i) Le passage horizontal est arrêté par une opération du job composé.

ii) Un coin SE est atteint. Dans ce cas, ce coin est choisi comme successeur direct du sommet v_i.

L'ajout des voisins obtenus en progressant horizontalement permet d'explorer des coins qui peuvent conduire à des plus courts chemins dans le plan. Pour l'exemple considéré, le coin $SE = (3, 1)$ est ajouté dans l'ensemble des successeurs directs de O (figure 5.7).

En résumé, pour le problème d'ordonnancement d'un nouveau job avec le job composé, l'algorithme *SuccSommet-2* est modifié en :

(i) Interdisant des passages horizontaux pour satisfaire la contrainte de non-préemption des opérations.

(ii) Ajoutant des sommets (arcs) additionnels dans l'ensemble des successeurs directs d'un sommet v_i. Ces nouveaux voisins sont obtenus en progressant horizontalement chaque fois qu'une horizontale est croisé.

Remarque 5.2 : L'ensemble S_U des ensembles candidats défini dans l'algorithme *SuccSommet-2* peut être encore négligé. La preuve est similaire à celle du théorème 5.1.

Remarque 5.3 : Le nombre de successeurs directs d'un sommet v_i avec le nouvel algorithme *SuccSommet-2* dépend des durées opératoires. Pour un sommet v_i, une borne supérieure de ce nombre est $n_j + 2 \times K \times n_{com}$. Par conséquent, la complexité de l'algorithme modifié pour la résolution du problème d'ordonnancement entre le job composé et un nouveau job est égale à

$O\left((n_j \times n_{com})(n_j + K \times n_{com})\right)$. Notons que n_{com} est au plus égal à $\sum_{i=1}^{l} n_i$, où l est le nombre des jobs déjà examinés.

4.3. *Algorithme glouton*

L'heuristique que nous proposons pour résoudre le problème d'ordonnancement de *JSMF* est un algorithme glouton qui ordonnance les jobs l'un après l'autre suivant une séquence donnée. Cette méthodologie présentée dans le chapitre 2 a été introduite par (Mati et al. [2001c]). Au départ, seuls les deux premiers jobs J_1 et J_2 de la séquence sont considérés. Les ensembles candidats pour le job J_1 sont affectés aux opérations du job en utilisant des règles de priorité. Le problème d'ordonnancement de J_1 et J_2 devient ainsi un problème de type 2-*JSMF* qui peut être résolu par l'algorithme polynomial proposé dans la section 3. Avec la solution optimale obtenue, le job composé J_{com}, comprenant les opérations des deux jobs ordonnancés, est construit. Une fois le job composé obtenu, le problème d'ordonnancement de ce job composé et d'un nouveau job de la séquence est résolu. Ainsi, un nouveau job composé est construit. Ces étapes sont répétées jusqu'à l'ordonnancement de tous les jobs.

L'algorithme glouton pour le problème *JSMF* est donné par l'algorithme *HeurGlout* suivant :

Algorithme 5.3 : *HeurGlout*

1. *Sélectionner une séquence d'entrée initiale des jobs. Sans perte de généralité, soit J_1, J_2, ..., J_N cette séquence.*
2. *Affecter les ensembles candidats pour le job J_1 en utilisant des règles de priorité.*
3. *Poser $J_{com} = J_1$.*
4. ***Pour** $i \leftarrow 2$ jusqu'à N **Faire***
   - *4.1. Résoudre le problème 2-JSMF entre J_{com} et J_i.*
   - *4.2. Décaler à gauche les opérations des jobs déjà examinées.*
   - *4.3. Construire le nouveau job composé J' des jobs J_1, J_2, ..., J_i en utilisant l'algorithme JobCompose.*
   - *4.4. Poser $J_{com} \leftarrow J'$.*

   Fin Pour.
5. *L'ordonnancement final et le makespan sont déterminés à partir du job composé J_{com}.*

Fin algorithme.

A l'étape 1, une séquence d'entrée fournit à l'algorithme glouton l'ordre de traitement des jobs. Il est clair que la performance de l'algorithme dépend fortement de la séquence choisie. Afin d'optimiser cette séquence, un algorithme génétique est utilisé. Cet algorithme génétique est décrit dans la section suivante.

A l'étape 2 de l'algorithme, deux règles de priorité sont testées pour déterminer l'affectation des ensembles candidats. Dans la première règle, l'ensemble candidat affecté à une opération est celui pour lequel la durée opératoire est minimale. En cas de durée opératoire égale, un choix arbitraire est réalisé. La seconde règle est basée sur la charge des ressources, et affecte à l'opération l'ensemble candidat ayant la plus petite charge.

A l'étape 4.2, une procédure est activée pour décaler à gauche les opérations du job composé de manière à éliminer les temps morts sur les ressources. Cette procédure ne modifie pas l'ordre du passage des opérations sur les ressources.

4.4. Algorithme génétique

L'heuristique finale permettant de résoudre le problème *JSMF* est composée de l'heuristique gloutonne précédente et d'un algorithme génétique. Le choix d'un algorithme génétique plutôt qu'une recherche tabou ou un recuit simulé, est motivé par le fait qu'il est difficile de déterminer rapidement le meilleur voisin d'une séquence donnée.

Les paramètres de l'algorithme génétique que nous proposons sont les suivants. Les chromosomes sont constitués de permutations de jobs représentant l'ordre de traitement des jobs (séquence d'entrée pour l'algorithme *HeurGlout*).

Exemple 5.1 : Chromosomes dans le cas de cinq jobs :

Chromosome Ch_1	1	3	4	5	2
Chromosome Ch_2	4	5	2	3	1

Le mérite (fitness) de chaque chromosome est le makespan calculé par l'algorithme glouton qui utilise le chromosome comme séquence d'entrée des jobs. Cette valeur du makespan est alors utilisée pour sélectionner les deux solutions parentes (chromosomes) sur lesquelles l'opérateur de croisement est appliqué. La population initiale est générée aléatoirement et sa taille est fixée à $2\times N$, où N est le nombre de jobs.

L'opérateur de croisement adopté est celui donné par Oliver et al. [1987] pour résoudre le problème du voyageur de commerce. Le croisement est basé sur l'identification des jobs qui occupent les mêmes positions dans les deux solutions parentes.

Considérons les deux chromosomes de l'exemple 5.1 pour expliquer le principe d'obtention des enfants. Nous commençons par sélectionner pour chaque position un job de Ch_1 ou de Ch_2. Si le job de la position j est choisi dans Ch_1 (resp. Ch_2), il est interdit de sélectionner le même job dans Ch_2 (resp. Ch_1) pour les positions restantes. Un *cycle label* est donc introduit pour déterminer les jobs qui doivent être choisis dans un des parents et ceux qui doivent être pris dans l'autre.

Le cycle label pour les chromosomes Ch_1 et Ch_2 est le suivant :

Parent 1	1	3	4	5	2
Parent 2	4	5	2	3	1
Cycle label	1	2	1	2	1

Le premier (resp. second) enfant réalisable est obtenu en sélectionnant les jobs des positions {1, 3, 5} du premier (resp. second) parent et les jobs des positions {2, 4} du second (resp. premier) parent. Pour plus de détails, le lecteur est renvoyé à Oliver et al. [1987].

Enfant 1	1	5	4	3	2
Enfant 2	4	3	2	5	1

L'opérateur de croisement est appliqué avec une probabilité égale à 0.9.

L'opérateur de mutation utilisé consiste à permuter aléatoirement les positions de deux jobs des enfants obtenus. Cet opérateur de mutation est appliqué avec une probabilité égale à 0.05.

Enfin, l'algorithme génétique est arrêté après un nombre de générations fixé à 1000.

5. Résultats expérimentaux

Dans cette section, nous proposons trois types d'expérimentation pour montrer l'efficacité de l'approche proposée. Dans la sous-section 5.1, nous présentons les résultats obtenus sur les benchmarks du job shop flexible de la littérature. Dans la sous-section 5.2, les tests sont réalisés sur des benchmarks du job shop multi-ressources avec flexibilité des ressources. Dans ces derniers benchmarks, toutes les ressources sont disponibles en une seule unité. Finalement, nous étudions dans la sous-section 5.3 le modèle générique pour lequel les ressources peuvent être disponibles en plusieurs unités. Pour ce dernier cas, des benchmarks ont été générées de façon aléatoire.

5.1. Job shop flexible

Bien que notre programme soit proposé pour l'ordonnancement des jobs shops multi-ressources avec flexibilité, nous l'avons testé sur le cas particulier du job shop flexible, et ce, afin de confronter notre méthode à des approches spécifiques à ce problème. Le premier ensemble de benchmarks considéré contient les instances générées par Hurink et al. [1994], pour lesquelles la durée opératoire est identique pour toutes les machines capables de réaliser une opération. Le second ensemble contient les instances générées par Brandimarte [1993], pour lesquelles les durées opératoires sont différentes.

Pour le premier type d'instances, nos résultats sont comparés individuellement à ceux obtenus par Hurink et al. [1994], Dauzère-Pérès et Paulli [1994], Dauzère-Pérès et Paulli [1997] et Gambardella et Mastrollili [2000], notées respectivement *HJT*, *DP*1, *DP*2 et *MG*. Notre approche est notée *JYM*. Les tableaux 5.1 et 5.2 donnent les résultats de cette comparaison.

Tableau 5.1. Comparaison avec les approches à deux phases sur les instances de Hurink et al. [1994]

instance	*edata*			*rdata*			*vdata*		
	HJT	*DP*1	*JYM*	*HJT*	*DP*1	*JYM*	*HJT*	*DP*1	*JYM*
la01	611	609	**609***	574	576	574	573	578	572
la02	655	660	**655***	535	545	535	531	548	**529***
la03	573	575	563	481	489	**457***	482	490	479
la04	578	573	**568***	509	515	509	504	504	**502***
la05	503	503	**503***	460	465	**457***	464	472	460
la06	833	833	**833***	801	803	**799***	802	803	802
la07	765	778	765	752	762	**750***	751	754	750
la08	845	845	**845***	767	770	769	766	770	766
la09	878	884	**878***	859	856	853	854	859	**853***
la10	866	866	**866***	806	806	**804***	805	806	807
la11	1106	1113	1103	1073	1077	1073	1073	1072	**1071***
la12	960	960	**960***	937	973	**936***	940	938	**936***
la13	1053	1053	**1053***	1039	1041	**1038***	1040	1041	**1038***
la14	1151	1123	**1123***	1071	1072	**1070***	1071	1071	**1070***
la15	1111	1116	**1111***	1093	1099	**1090***	1091	1092	**1089***
la16	924	930	**892***	717	721	**717***	717	71	**717***
la17	757	723	**707***	646	655	**646***	646	646	**646***
la18	864	861	850	674	704	674	663	663	**663***
la19	850	805	**796***	725	717	715	617	682	**617***
la20	919	901	865	756	761	**756***	756	756	**756***
la21	1066	1046	1046	861	887	856	826	815	836
la22	919	912	918	790	793	784	745	762	744
la23	980	972	972	884	878	853	826	829	826
la24	952	940	918	825	843	825	796	796	784
la25	970	967	970	823	834	823	770	783	757
la26	1169	1116	1156	1086	1081	1081	1058	1058	1058
la27	1230	1220	1230	1109	1106	1106	1088	1090	1087
la28	1204	1165	1169	1097	1097	1097	1073	1077	1073
la29	1210	1178	1178	1016	1010	1008	995	1002	995
la30	1253	1266	1225	1105	1112	1089	1071	1074	1071

* : solution optimale

Tableau 5.2. Comparaison avec les approches intégrées sur les instances de Hurink et al. [1994]

instance	*edata*			*rdata*			*vdata*		
	DP2	*MG*	*JYM*	*DP2*	*MG*	*JYM*	*DP2*	*MG*	*JYM*
la01	609	609	**609***	574	571	574	572	570	572
la02	655	655	**655***	532	530	535	529	529	**529***
la03	554	550	563	479	478	**457***	479	477	479
la04	568	568	**568***	504	502	509	503	502	**502***
la05	503	503	**503***	458	457	**457***	460	457	460
la06	833	833	**833***	800	799	**799***	800	799	802
la07	765	762	765	750	750	**750***	750	749	750
la08	845	845	**845***	767	765	769	466	765	766
la09	878	878	**878***	854	853	853	853	853	**853***
la10	866	866	**866***	805	804	**804***	805	804	807
la11	1103	1103	1103	1072	1071	1073	1071	1071	**1071***
la12	960	960	**960***	936	936	**936***	936	936	**936***
la13	1053	1053	**1053***	1038	1038	**1038***	1038	1038	**1038***
la14	1123	1123	**1123***	1070	1070	**1070***	1070	1070	**1070***
la15	1111	1111	**1111***	1090	1090	**1090***	1089	1089	**1089***
la16	915	892	**892***	717	717	**717***	717	717	**717***
la17	707	707	**707***	646	646	**646***	646	646	**646***
la18	843	842	850	669	666	674	663	663	**663***
la19	796	796	**796***	703	700	715	617	617	**617***
la20	864	857	865	756	756	**756***	756	756	**756***
la21	1046	1017	1046	846	835	856	814	806	836
la22	890	882	918	772	760	784	744	739	744
la23	953	950	972	853	842	853	818	815	826
la24	918	909	918	820	808	825	784	777	784
la25	955	941	970	802	791	823	757	756	757
la26	1138	1125	1156	1070	1061	1081	1056	1054	1058
la27	1215	1186	1230	1100	1091	1106	1087	1085	1087
la28	1169	1149	1169	1085	1080	1097	1072	1070	1073
la29	1157	1118	1178	1004	998	1008	997	994	995
la30	1225	1204	1225	1089	1078	1089	1071	1069	1071

* : solution optimale

Pour le second type d'instances, notre approche est comparée à celle développée par Brandimarte [1993] noté *BR* ainsi que celle proposée par Gambardella et Mastrolilli [2000]. Les résultats sont résumés dans le tableau 5.3 suivant.

Tableau 5.3. Comparaison sur les instances de Brandimarte [1993]

instance	Borne inférieure	*BR*		*MG*		*JYM*	
		C_{max}	CPU secs	C_{max}	CPU secs	C_{max}	CPU secs
mk01	36	42	n.d	**40**	0.01	**40**	3.53
mk02	24	32	n.d	**26**	0.73	**26**	2.76
mk03	204	211	n.d	**204***	0.01	**204***	0.27
mk04	48	81	n.d	**60**	0.08	62	8.32
mk05	168	186	n.d	**173**	0.96	**173**	12.89
mk06	33	86	n.d	**58**	3.26	64	10.41
mk07	133	157	n.d	144	8.91	**143**	51.65
mk08	523*	**523***	n.d	**523***	0.02	**523***	0.34
mk09	299	369	n.d	**307**	0.15	**307**	7.28
mk10	165	296	n.d	**198**	7.69	**198**	43.30

n.d : non donné
* : solution optimale

5.2. Job shop multi-ressources avec flexibilité des ressources

Les benchmarks utilisés dans cette section ont été générés aléatoirement par Dauzère-Pérès et al. [1998]. Le nombre de jobs (resp. ressources) est le même pour toutes les instances. Il est égal à 10 (resp. 20). Deux groupes d'instances sont proposés. Dans le premier groupe, le nombre d'opérations de chaque job est égal à 10, alors qu'il est fixé à 15 dans le second. Le tableau 5.4 dresse la comparaison des résultats de notre méthode avec ceux de l'heuristique proposée par Dauzère-Pérès et al. [1998] noté *DRL*, sur 50 instances sélectionnées.

5.3. Cas des ressources multiples

Dans cette partie, 10 instances proches des applications pratiques sont considérées, dans le ces où certaines ressources peuvent être disponibles en plusieurs unités. Les instances ont été générées aléatoirement de la manière suivante. Le nombre de jobs est fixé à 10 pour les cinq premières instances et à 20 pour les cinq dernières. Le nombre de ressources disponibles est fixé à 10 pour toutes les instances Le nombre d'opérations de chaque job est généré dans l'intervalle [5, 10]. Le nombre de ressources pour chaque opération est sélectionné dans l'intervalle [1, 3], et les types de ressources nécessaires sont tirés aléatoirement parmi les ressources disponibles. La durée opératoire de chaque opération est sélectionnée dans l'intervalle [5, 20]. Le nombre d'ensembles candidats est choisi dans l'intervalle [1, 5]. Finalement, la quantité de ressources disponibles est tirée aléatoirement dans l'intervalle [1, 4]. Le tableau 5.5 récapitule les données des instances considérées.

Tableau 5.4. Comparaison sur les problèmes mjs00-mjs49 de Dauzère-Pérès et al. [1998]

instance	*DRL*		*JYM*			instance	*DRL*		*JYM*		
	C_{max}	CPU secs	$C_{début}$	C_{max}	CPU secs		C_{max}	CPU secs	$C_{début}$	C_{max}	CPU secs
mjs00	**387**	34.8	490	**387**	235.90	mjs25	1513	149.1	1615	**1460**	456.92
mjs01	**361**	15.2	444	374	24.52	mjs26	1481	77.3	1559	**1318**	378.55
mjs02	**384**	15.1	456	**384**	288.05	mjs27	1566	135.7	1627	**1493**	234.67
mjs03	**378**	26.6	448	**378**	166.77	mjs28	**1395**	363.0	1592	1407	764.88
mjs04	**394**	41.4	492	400	13.01	mjs29	1336	241.7	1480	**1317**	1327.5
mjs05	**643**	179.8	806	663	95.50	mjs30	**218**	37.4	259	224	945.56
mjs06	585	32.7	695	**583**	358.58	mjs31	**218**	28.9	257	227	228.05
mjs07	**644**	58.2	729	658	173.01	mjs32	**219**	33.4	286	**219**	487.23
mjs08	**575**	78.6	719	**575**	387.23	mjs33	**224**	30.6	285	243	187.54
mjs09	**568**	49.1	702	584	143.27	mjs34	**213**	39.5	295	220	235.39
mjs10	928	88.3	1045	**920**	332.88	mjs35	**265**	14.5	306	**265**	473.95
mjs11	1057	91.7	1254	**996**	276.81	mjs36	**225**	50.5	288	240	320.87
mjs12	859	84.9	937	**846**	387.36	mjs37	**207**	61.8	256	218	222.80
mjs13	**827**	180.5	1073	845	192.98	mjs38	**241**	30.9	283	**241**	457.32
mjs14	**946**	94.3	1119	955	154.23	mjs39	**210**	29.2	243	219	654.52
mjs15	1469	99.9	1596	**1401**	479.76	mjs40	**241**	19.8	262	**241**	150.19
mjs16	1312	196.9	1478	**1311**	653.43	mjs41	**218**	59.1	265	231	525.03
mjs17	1572	503.5	1656	**1488**	879.65	mjs42	**250**	16.7	300	**250**	87.93
mjs18	1544	150.6	1669	**1475**	653.06	mjs43	**219**	20.1	281	**219**	150.46
mjs19	1572	180.5	1645	**1463**	756.64	mjs44	**258**	23.3	326	**258**	146.42
mjs20	1033	89.6	1170	**1004**	564.23	mjs45	**296**	61.9	360	305	254.45
mjs21	916	241.9	1068	**902**	582.10	mjs46	**300**	87.2	396	324	85.76
mjs22	**924**	51.0	1048	948	237.54	mjs47	**333**	119.7	415	357	230.56
mjs23	957	135.5	1092	**939**	1203.2	mjs48	**327**	83.6	409	341	177.25
mjs24	918	88.9	1032	**905**	654.70	mjs49	**356**	30.2	435	**356**	194.14

Tableau 5.5. Données des problèmes générés

instance	nombre de jobs	nombre d'opération par job	nombre de ressources	unités de chaque ressource	nombre de flexibilité	Nombre de ressources pour chaque opération	Durée opératoire
My01	10	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My02	10	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My03	10	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My04	10	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My05	10	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My06	20	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My07	20	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My08	20	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My09	20	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]
My10	20	[5, 10]	10	[1, 4]	[1, 5]	[1, 3]	[5, 20]

Tableau 5.6. Résultats sur les instances générées

instance	*JYM*			
	C_{deb}	C_{mauv}	C_{meil}	CPU secs
My01	120	102	101	37.34
My02	149	132	126	54.65
My03	146	133	130	59.07
My04	160	136	131	44.89
My05	137	110	108	66.23
My06	213	188	185	98.20
My07	242	200	196	124.09
My08	206	187	183	89.66
My09	211	171	170	135.32
My10	199	159	154	103.52

5.4. Discussions

Nous avons testé notre algorithme général sur trois types de problèmes :

- Le job shop flexible.
- Le job shop multi-ressources avec flexibilité dans le cas unitaire.
- Le job shop multi-ressources avec flexibilité dans le cas général.

A partir des résultats donnés par le tableau 5.1, nous pouvons conclure que notre méthode est plus performante que les deux approches hiérarchiques proposées par Hurink et al. [1994] et Dauzère-Pérès et Paulli [1994]. En effet, l'approche de Hurink et al. [1994] donne de meilleurs makespans que notre heuristique pour seulement 3 instances sur 120. Pour ces trois instances, l'écart maximal entre les deux heuristiques est de 1%. D'autre part, l'approche de Dauzère-Pérès et Paulli [1994] est meilleure que la notre pour 4 instances, la déviation maximale étant de 2%.

D'après le tableau 5.3, les résultats donnés par notre méthode sont toujours meilleurs que ceux donnés par Brandimarte [1993]. Ces résultats sont de plus comparables à ceux donnés par Mastrollili et Gambardella [2000]. Sur les dix instances considérées, notre heuristique trouve le même makespan pour sept d'entre elles, améliore la valeur du makespan pour l'instance Mk07, et est très proche de la meilleure solution pour les autres.

Le fait que les résultats donnés par des approches qui séparent les problèmes d'affectation et de séquencement (Hurink et al. [1994], Dauzère-Pérès et Paulli [1994] et Brandimarte [1993]) restent inférieures aux nôtres suggère qu'il est préférable d'utiliser une approche intégrée qui résout simultanément les deux problèmes.

D'autre part, la comparaison avec les meilleures approches connues pour le job shop flexible donnée dans le tableau 5.2, montre qu'il est possible de résoudre efficacement le problème sans avoir recours à la représentation par le graphe disjonctif (qui est la base de toutes les approches proposées dans la littérature). En effet, la méthode que nous proposons permet d'obtenir la solution optimale pour 40 instances, et génère des bons ordonnancements pour le reste des instances. Pour les instances *edata* ayant une petite flexibilité, l'écart de notre makespan par rapport à la meilleure valeur connue est inférieure à 2% en moyenne avec un écart maximal de 5%. Pour les instances *rdata*, l'écart moyen (resp. maximal) est de 1.4% (resp. 4%). Pour les instances *vdata* ayant en revanche une flexibilité plus grande, la qualité des résultats n'est pas détériorée, et l'écart moyen (resp. maximal) est égal à 0.5% (resp. 2%).

Les bonnes performances de l'heuristique proposée sont dues d'une part, à l'optimalité de l'algorithme qui résout le problème à deux jobs, et d'autre part, à la résolution simultanée des problèmes d'affectation et de séquencement. Les résultats proposés par notre heuristique sont comparables à ceux donnés par Dauzère-Pérès et Paulli [1997] ; Gambardella et Mastrollili [2000], qui restent les meilleures approches connues pour le job shop flexible.

Pour le second type de tests sur les instances du job shop multi-ressources avec flexibilité des ressources, notre heuristique est très compétitive par rapport à l'approche proposée par Dauzère-Pérès et al. [1998]. En effet, le tableau 5.4 montre que, sur 50 instances, notre heuristique fournit de meilleures solutions dans 17 de cas et le même makespan dans 12 de cas. Dans le cas où les makespans donnés par notre heuristique sont supérieurs, l'écart moyen est égal à 3 %. Cette comparaison conduit donc à la même conclusion que pour le job shop flexible et confirme l'efficacité de l'approche proposée pour ce problème d'ordonnancement.

Pour la dernière classe d'instances, notre heuristique est lancée 10 fois à partir de populations initiales choisies de façon aléatoire, ceci afin d'évaluer la robustesse des résultats en fonction de la population de départ. Le tableau 5.6, donnant la valeur de départ, mauvais et le meilleur makespan, ainsi que le temps du calcul nécessaire pour obtenir le meilleur résultat, montre que notre heuristique est stable. En effet, notre heuristique ne donne pas de solutions

aberrantes d'une exécution à l'autre, l'écart maximal entre la meilleure et la plus mauvaise valeur du makespan ne dépassant pas 4%.

6. Extensions de l'approche

La première extension envisagée est l'optimisation d'autres critères objectifs. L'algorithme polynomial permettant de résoudre le problème à deux jobs classiques reste applicable. La seule modification à apporter concerne la résolution du problème d'ordonnancement d'un job composé et d'un job classique.

La deuxième extension possible est d'intégrer une nouvelle contrainte sur la libération des ressources. En pratique, les ressources d'un ensemble candidat à l'exécution d'une opération peuvent être libérées à des dates différentes. Par exemple, un opérateur humain reste quelques minutes au début de l'exécution d'une opération pour vérifier le bon fonctionnement de la machine. Le reste de l'opération peut être exécuté par la machine de façon autonome (dans ce cas $p_{humain} << p_{machine}$). L'opérateur est donc libre dès l'instant p_{humain} et peut effectuer d'autres opérations. Aussi est-il crucial de prendre en compte cette contrainte si l'opérateur humain est une ressource critique (Dauzère-Pérès et Pavageau [1999]).

7. Conclusion

Dans ce chapitre, nous avons présenté une approche intégrée pour résoudre le problème d'ordonnancement du job shop multi-ressources avec flexibilité des ressources. Nous avons dans un premier temps, proposé un algorithme polynomial pour la résolution d'un problème particulier à deux jobs. Dans un second temps, nous avons développé une heuristique gloutonne afin de résoudre le problème général à un nombre quelconque de jobs. Cette heuristique est basée sur la notion de job composé et sur une nouvelle extension de l'algorithme polynomial. Elle est en outre guidée par un algorithme génétique permettant d'en améliorer les performances.

L'heuristique proposée a d'abord été testée sur plusieurs benchmarks issus de la littérature puis sur de nouveaux benchmarks générés de façon aléatoire. Les résultats obtenus ont montré que notre méthode était capable de fournir plusieurs solutions optimales pour les instances du job shop flexible. Dans le cas où les solutions optimales n'ont pas été atteintes, les résultats sont comparables aux meilleurs résultats connus. Pour les benchmarks du job shop multi-ressources avec flexibilité des ressources, notre heuristique a fourni des nouvelles solutions en moyenne meilleures que celles des approches existantes. Enfin, les résultats obtenus sur les benchmarks générés ont montré que l'heuristique proposée n'est pas sensible au choix de la population initiale.

Les différentes comparaisons réalisées dans ce chapitre confirment l'efficacité de la méthodologie utilisée, à savoir, l'ordonnancement itératif des jobs en utilisant des algorithmes polynomiaux pour résoudre les problèmes à deux jobs. Cette méthodologie peut être adoptée pour résoudre d'autres types de problèmes d'ordonnancement. Un de ces problèmes, que nous traitons dans le chapitre suivant, consiste à intégrer au modèle étudié dans ce chapitre la propriété de "retenir et attendre" relative à la libération des ressources à la fin de l'exécution des opérations.

Chapitre 6

Job Shop Multi-Ressources avec Prise en Compte de Blocage et de Flexibilité des Ressources

Dans ce chapitre, nous généralisons le modèle d'ordonnancement traité au chapitre précédent en y ajoutant la propriété *retenir et attendre*. Dans le nouveau problème, chaque opération nécessite donc la présence simultanée de plusieurs ressources et l'ensemble de ces ressources est à choisir parmi plusieurs. La nouveauté est que les ressources affectées à une opération ne sont libérées que si les ressources nécessaires à l'opération suivante, du job considéré, sont disponibles. Dans un premier temps, nous proposons un algorithme polynomial original pour le problème à deux jobs, dans le cas où les durées des opérations ne dépendent pas des flexibilités choisies. Dans le cas où les durées opératoires dépendent de la flexibilité, un nouvel algorithme polynomial est proposé pour un cas particulier. En utilisant les deux algorithmes développés ainsi que le concept du job composé, nous proposons alors des heuristiques pour résoudre le problème général avec un nombre quelconque de jobs. Afin d'améliorer les performances, les heuristiques sont couplées à une recherche tabou et à un algorithme génétique. Enfin, des résultats expérimentaux sont présentés pour attester de l'efficacité de l'approche.

Les résultats de ce chapitre font l'objet de deux articles à soumettre au journaux IIE Transactions, et International Journal of Production Research.

1. Introduction

Dans le chapitre précédent, nous avons traité le problème d'ordonnancement de type *job shop multi-ressources avec flexibilité des ressources*. Ce modèle intégrant plusieurs contraintes propres aux systèmes flexibles de production permet d'aborder de nombreux problèmes rencontrés dans la pratique. Une contrainte qui n'a pas été considérée dans le modèle, est la capacité du stock tampon entre les machines. C'est pourtant une contrainte importante dans la mesure où les nouveaux systèmes de production sont de plus en plus composés de lignes de fabrication avec des stocks intermédiaires très limités.

Dans ce chapitre, nous introduisons un nouveau modèle d'ordonnancement permettant d'intégrer cette contrainte. Ce nouveau modèle s'appuie sur le modèle du job shop multi-ressources avec flexibilité des ressources étudié dans le chapitre 5, ainsi que celui traité dans le chapitre 2. Trois propriétés caractérisent donc le nouveau modèle : les opérations multi-ressources, la flexibilité des ressources et la propriété retenir et attendre. La considération de ces trois hypothèses conduit à un modèle d'ordonnancement plus réaliste permettant d'aborder une grande variété de problèmes rencontrés dans la pratique.

La suite de ce chapitre est organisée de la manière suivante. Dans la section 2, nous donnons une description du nouveau problème d'ordonnancement. Les deux sections suivantes sont consacrées à la résolution du problème à deux jobs. Dans la section 3, nous étudions le cas où la durée d'une opération ne dépend pas de la flexibilité choisie. Nous proposons un nouvel algorithme polynomial pour la minimisation de tout critère objectif régulier. La section 4 est dédiée au cas où la durée opératoire dépend de la flexibilité sélectionnée. Nous y proposons un deuxième algorithme polynomial pour résoudre le problème particulier où un seul job a la flexibilité des ressources. Dans la dernière partie du chapitre, le problème d'ordonnancement général, c'est-à-dire avec un nombre quelconque de jobs, est étudié. La section 5 décrit la méthodologie utilisée et présente les heuristiques développées pour la résolution du problème. La section 6 présente quelques résultats expérimentaux pour attester de l'efficacité des approches proposées.

2. Description du problème

Le problème d'ordonnancement étudié dans ce chapitre est une extension des problèmes d'ordonnancement étudiés dans les chapitres 2 et 5. Il est appelé *Job Shop Multi-Ressources avec Blocage et Flexibilité des ressources*, noté par la suite par *JSMBF*.

Le problème d'ordonnancement de type *JSMBF* se définit de la manière suivante. Un ensemble de ressources $\mathcal{R} = \{R_1, R_2, \ldots, R_s\}$ est disponible pour la réalisation d'un ensemble de jobs $\mathcal{J} = \{J_1, \ldots, J_i, \ldots J_N\}$, où s est le nombre de ressources et N étant le nombre de jobs. Chaque ressource est disponible en un seul exemplaire, c'est-à-dire que toutes les ressources sont différentes. Chaque job J_i est composé d'une séquence d'opérations $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$, où n_i est le nombre d'opérations du job J_i, appelée *gamme opératoire*. De plus, chaque opération O_{ij} nécessite la présence simultanée d'un ensemble de ressources $R_{ij} \subseteq \mathcal{R}$. Cet ensemble n'est pas connu à priori mais doit être choisi dans une liste de combinaisons de ressources possibles (ensembles candidats) μ_{ij}, associée à chaque opération. Cette dernière hypothèse est connue dans la littérature sous le nom de *flexibilité des ressources* (Dauzère-Pérès et al. [1998]). La préemption des opérations n'est pas permise. Nous supposons que les

durées opératoires pour tous les ensembles candidats associés à une opération, sont entières et connues à l'avance. Les temps de préparation (*setup time*) sont soit négligeables, soit inclus dans les durées opératoires.

La dernière caractéristique importante de notre modèle concerne la libération des ressources à la fin de l'exécution des opérations. Dans les modèles classiques d'ordonnancement, les ressources utilisées par une opération sont libérées dès que son exécution est achevée. Ici, nous supposons que les ressources utilisées par une opération O_{ij} sont retenues jusqu'à ce que les ressources nécessaires à l'exécution de l'opération suivante $O_{i,j+1}$ de la gamme soient toutes disponibles. C'est la propriété de *retenir et attendre*.

Comme nous l'avons mentionné dans le chapitre 2, la considération de la propriété retenir et attendre peut générer des situations de blocage dans le système de production, si les ressources ne sont pas bien coordonnées (Mati et al. [2001b, 2001c]).

Le problème d'ordonnancement consiste à résoudre simultanément les problèmes liés à l'affectation des ensembles candidats à chaque opération ainsi que le problème de séquencement. Le but est la détermination d'une solution optimale sans blocage minimisant une fonction objectif donnée. Nous nous focalisons dans ce chapitre sur la minimisation de la durée total de réalisation des jobs ou *makespan*.

3. Problème à deux jobs : durées opératoires indépendantes de la flexibilité

Dans cette section, nous nous focalisons sur le problème d'ordonnancement de type *JSMBF* avec seulement deux jobs. Nous supposons que la durée d'une opération O_{ij} ne dépend pas de la flexibilité $f \in \mu_{ij}$ sélectionnée parmi les ensembles candidats. Cette durée opératoire est notée p_{ij}. Ainsi, le modèle d'ordonnancement considéré généralise celui étudié par (Brucker et Schlie [1990]) à travers l'introduction d'opérations multi-ressources et la propriété de *retenir et attendre*.

Dans le cas où les ressources nécessaires à l'utilisation d'une opération sont connues, le problème d'ordonnancement à deux jobs peut être modélisé par une représentation géométrique dans le plan (Akers et Friedman [1955]). Dans le cas où les ressources ne sont pas encore affectées, il est possible de définir des obstacles potentiels relatifs à deux opérations O_{1,k_1} et O_{2,k_2} s'il existe au moins deux ensembles candidats $f_1 \in \mu_{1,k_1}$ et $f_2 \in \mu_{2,k_2}$ partageant des ressources communes.

Un plus court chemin réalisable dans le plan avec obstacles, partant de l'origine O et arrivant au point final F correspond à une solution optimale du problème d'ordonnancement (voir chapitre 1). De plus, déterminer le plus court chemin dans le plan est équivalent à la recherche d'un plus court chemin sans contraintes dans un réseau $N = (V, E, d)$ défini en se basant sur une représentation géométrique.

Pour pouvoir construire le réseau N qui permet de déterminer le makespan, nous sommes amenés à définir :

(a) Les obstacles considérés.

(b) L'ensemble des sommets du réseau.

(c) Les successeurs immédiats d'un sommet.

(d) La distance entre deux sommets voisins.

3.1. Définition des obstacles

Afin de pallier les situations de blocage qui peuvent se produire dans le système, des obstacles autres que ceux du partage des ressources doivent être ajoutés. Aussi, nous proposons un algorithme de programmation dynamique permettant de déterminer simultanément les obstacles relatifs aux partages des ressources et aux situations de blocage.

L'algorithme utilise les indicateurs booléens définis comme suit :

$$I(k_1,k_2,f_1,f_2)=\begin{cases}1 & \text{si l'opération } O_{1,k1} \text{ avec l'ensemble candidat } f_1 \text{ et } O_{2,k2} \\ & \text{avec l'ensemble candidat } f_2 \text{ sont en conflit de ressource,} \\ & \text{ou si le blocage ne peut être évitée depuis l'état } (k_1,k_2,f_1,f_2), \\ 0 & \text{sin on.}\end{cases}$$

où l'état (k_1, k_2, f_1, f_2) signifie que les deux opérations O_{1,k_1} et O_{2,k_2} sont exécutées en même temps en utilisant respectivement les ensembles candidats f_1 et f_2.

Notons que les indicateurs $I(k_1, k_2, f_1, f_2)$ définissent non seulement les blocages immédiats mais aussi les blocages futurs « *impending deadlock* ».

Grâce à ces variables booléennes, tous les obstacles peuvent être définis par l'équation récursive suivante :

$$I(k_1,k_2,f_1,f_2)=\begin{cases}1 & \text{si l'opération } O_{1,k1} \text{ avec l'ensemble candidat } f_1 \text{ et } O_{2,k2} \\ & \text{avec l'ensemble candidat } f_2 \text{ sont en conflit de ressource,} \\ \underset{t\in\mu_{1,k1+1}}{\wedge}\left[I(k_1+1,k_2,t,f_2)\right] \wedge \left[\underset{t\in\mu_{2,k2+1}}{\wedge}\left[I(k_1,k_2+1,f_1,t)\right]\right] & \text{sin on}\end{cases} \quad (1)$$

Par convention :

$$I(k_1, n_2+1, f_1, f_2) = 0, \forall\, k_1 = 1, 2,\ldots, n_1+1,$$
$$I(n_1+1, k_2, f_1, f_2) = 0, \forall\, k_2 = 1, 2,\ldots, n_2+1, \forall\, f_1, f_2.$$

L'algorithme de programmation dynamique que nous proposons est basé sur l'équation récursive (1), et calcule tous les indicateurs $I(k_1, k_2, f_1, f_2)$ par récurrence sur $k = k_1 + k_2$.

Algorithme 6.1 : (*Détection de blocages et de conflits de ressources*)

1. *Initialisation* : $I(k_1, n_2+1, f, t) = 0$, $\forall k_1=1, 2,..., n_1+1$ et $I(n_1+1, k_2, f, t) = 0$, $\forall k_2=1, 2,..., n_2+1$, $\forall f, t$.
2. ***Pour*** $k = n_1 + n_2$ *jusqu'à* 2 ***Faire***

   Pour *tout état* (k_1, k_2) *tel que* $k_1 + k_2 = k$, $k_1 \leq n_1$, $k_2 \leq n_2$ ***Faire***

   Pour *toute flexibilité* $f_1 \in \mu_{1,k1}$ ***Faire***

   Pour *toute flexibilité* $f_2 \in \mu_{2,k2}$ ***Faire***

   Déterminer $I(k_1, k_2, f_1, f_2)$ *en appliquant l'équation récursive* (1).

   Fin Pour.

   Fin Pour.

   Fin Pour.

   Fin Pour.

Fin Algorithme.

Le théorème suivant justifie l'utilisation de la programmation dynamique pour la recherche des obstacles, et donne la complexité de cette recherche.

Théorème 6.1

La programmation dynamique proposée interdit seulement les états relatifs au partage des ressources et des situations de blocage. De plus, tous les obstacles peuvent être trouvés en un temps proportionnel à $O(m_1 \times m_2)$, où $m_i = K_i \times n_i$, $K_i = \max\{\text{card}(\mu_{ij})\}$, i = 1, 2, $1 \leq j \leq n_i$.

Preuve

Nous devons montrer que la programmation dynamique ajoute seulement les obstacles relatifs aux situations de blocage. La preuve est donnée par récurrence sur $k_1 + k_2 = n_1 + n_2, \ldots, 2$. La propriété est vraie pour $k_1 + k_2 = n_1+1 + n_2+1$. Supposons que la relation est vraie pour tout $k_1 + k_2 \geq k$, c'est-à-dire que l'état (k_1, k_2, f_1, f_2) est un état de blocage ou de partage des ressources si et seulement si $I(k_1, k_2, f_1, f_2) = 1, f_1 \in \mu_{1,k_1}$, et $f_2 \in \mu_{2,k_2}$.

Soit (k_1, k_2, f_1, f_2) un état tel que $k_1 + k_2 = k$-1. Si la contrainte de capacité des ressources entre les opérations O_{1,k_1} en sélectionnant l'ensemble candidat f_1 et O_{2,k_2} en sélectionnant l'ensemble candidat f_2 n'est pas satisfaite, alors la propriété est vraie et nous avons $I(k_1, k_2, f_1, f_2) = 1$. Sinon, la valeur de $I(k_1, k_2, f_1, f_2)$ est calculée à partir des valeurs de $I(k_1+1, k_2, t_1, f_2)$, avec $t_1 \in \mu_{1,k_1+1}$ et les valeurs de $I(k_1, k_2+1, f_1, t_2)$, avec $t_2 \in \mu_{2,k_2+1}$. Puisqu'il

existe seulement deux jobs, soit J_1 passe en premier à son opération suivante, soit c'est J_2 qui progresse en premier. L'état suivant est donc (k_1+1, k_2, t_1, f_2) ou (k_1, k_2+1, f_1, t_2). Par conséquent, l'état (k_1, k_2, f_1, f_2) est un état de blocage, si et seulement si, les indices $I(k_1+1, k_2, t_1, f_2)$ $\forall\, t_1 \in \mu_{1,k_1+1}$, et $I(k_1, k_2+1, f_1, t_2)$, $\forall\, t_2 \in \mu_{2,k_2+1}$ sont des états de blocage ou de partage des ressources. D'après l'hypothèse de récurrence, l'état (k_1, k_2, f_1, f_2) est un état de blocage, si et seulement si, $I(k_1+1, k_2, t_1, f_2) = 1$, $\forall\, t_1 \in \mu_{1,k_1+1}$, et $I(k_1, k_2+1, f_1, t_2) = 1$, $\forall\, t_2 \in \mu_{2,k_2+1}$, c'est-à-dire, si et seulement si $I(k_1, k_2, f_1, f_2) = 1$.

La propriété est donc vraie pour tout état (k_1, k_2, f_1, f_2) tel que $k_1 + k_2 = k - 1$ et le théorème est prouvé. Par ailleurs, le nombre d'opérations élémentaires effectuées par l'algorithme 6.1 (étape 2) est proportionnel à $O(m_1 \times m_2)$. ■

3.2. Ensemble des sommets

Dans la plupart des extensions de l'approche géométrique proposées pour résoudre des problèmes à deux jobs, l'ensemble des sommets du réseau est caractérisé par les coins nord-ouest (*NW*) et sud-est (*SE*) des obstacles rencontrés ou par les coins singuliers se trouvant sur les bordures des obstacles (cas de la préemption). Dans les travaux de Brucker et Jurisch [1993], l'introduction d'une nouvelle contrainte (date de disponibilité) a conduit à l'ajout d'un troisième axe représentant le temps.

Dans notre étude, la propriété retenir et attendre ainsi que la flexibilité des ressources nous obligent à renfoncer les sommets par une information additionnelle. Cette information est relative aux derniers ensembles candidats utilisés pour atteindre le sommet, et permet de progresser horizontalement ainsi que verticalement à partir du sommet.

(A) Caractérisation d'un coin SE

Si le coin sud-est d'un obstacle potentiel (k_1, k_2) est rencontré partant d'un sommet en allant d'abord diagonalement et puis horizontalement (figure 6.1 (a)), cela signifie que l'opération O_{1,k_1} (resp. O_{2,k_2-1}) du job J_1 (resp. J_2) est déjà terminée, et ceci, avec un ensemble candidat f_1 (resp. f_2). L'ensemble f_1 (resp. f_2) est ensuite utilisé pour la progression verticale (resp. horizontale) à partir du sommet *SE*.

A cause de la contrainte *retenir et attendre*, la progression verticale à partir du sommet *SE* est interdite, puisqu'elle longe la bordure droite de l'obstacle. De ce fait, il n'est pas nécessaire de conserver l'information relative à la flexibilité f_1 de O_{1,k_1}. En revanche, la progression horizontale à partir du sommet *SE* peut être possible. Cette progression doit être réalisée pendant que le job J_2 retient toujours l'ensemble candidat f_2 de O_{2,k_2-1}. Par conséquent, il est nécessaire de mémoriser cette flexibilité.

Ainsi, les coins *SE* des obstacles rencontrés sont caractérisés par des quadruplets (k_1, k_2, ϕ, f_2) (figure 6.1 (a)).

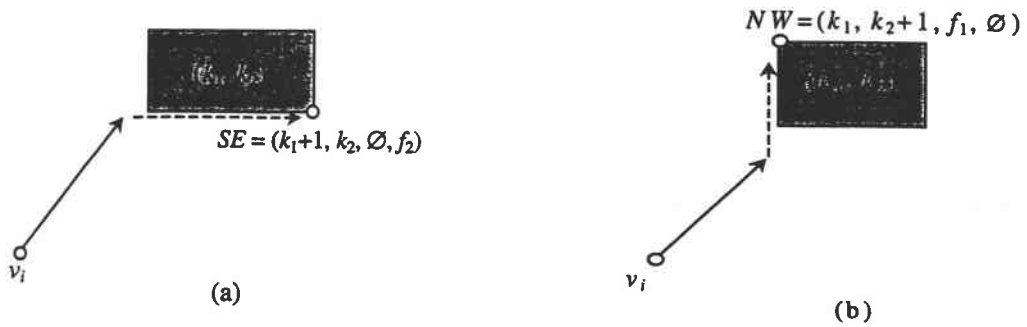


Figure 6.1 : Caractérisation des sommets *SE* et *NW*

(B) Caractérisation d'un coin NW

Le coin *NW* d'un obstacle potentiel entre les opérations O_{1,k_1} et O_{2,k_2}, signifie la fin d'exécution des opérations O_{1,k_1-1} et O_{2,k_2} des jobs J_1 et J_2 respectivement. La progression horizontale, qui longe la bordure supérieure de l'obstacle, est interdite. Par contre, la progression verticale doit être réalisée pendant que le job J_1 retient toujours l'ensemble candidat f_1 de O_{1,k_1-1}.

Ainsi, les coins *NW* du réseau sont caractérisés par des quadruplets $(k_1, k_2, f_1, \varnothing)$ (figure 6.1 (b)), où $\varnothing$ signifie que l'ensemble candidat f_2 n'est pas nécessaire.

3.3. Successeurs directs d'un sommet

Dans l'approche géométrique classique, chaque sommet v_i a au plus deux successeurs immédiats : les coins *NW* et *SE* de l'obstacle rencontré en partant diagonalement de v_i. Avec la présence de la flexibilité des ressources et de la propriété de *retenir et attendre*, chaque sommet v_i peut avoir plus de deux successeurs. En effet, nous considérons comme successeurs directs de v_i, tous les coins $SE = (k_1, k_2, \varnothing, f_2)$ (resp. $NW=(k_1, k_2, f_1, \varnothing)$) des obstacles potentiels obtenus en partant de v_i et en progressant diagonalement, puis horizontalement (resp. verticalement).

L'algorithme *SuccSommet-3* suivant permet de déterminer tous les successeurs directs d'un sommet sud-est $v_i = (k_1, k_2, \varnothing, f_2)$.

Algorithme 6.2 : *SuccSommet-3* (*Successeurs directs d'un sommet* $SE=(k_1, k_2, \varnothing, f_2)$)

1. *Initialisation* : $h = k_2$, $v = k_1$, $P_h = \{f_2\}$, $P_v = \varnothing$, $E_1 = \mu_{1v}$, $E_2 = \mu_{2h}$, $p_1 = p_{1v}$, $p_2 = p_{2h}$.

2. ***Si*** $P_h \neq \varnothing$ ***Alors*** *Appeler la procédure Pass-horizontale avec l'horizontale h et l'ensemble des flexibilités* P_h.

   Si $P_v \neq \varnothing$ ***Alors*** *Appeler la procédure Pass-verticale avec la verticale v et l'ensemble des flexibilités* P_v.

3. $P_h = \varnothing$, $P_v = \varnothing$, $R = \varnothing$.

   Si $p_1 < p_2$ ***Alors*** *Crois = verticale*, $h = h + 1$.

   Si $p_1 > p_2$ ***Alors*** *Crois = horizontale*, $v = v + 1$.

   Si $p_1 = p_2$ ***Alors*** *Crois = diagonale*, $h = h + 1$, $v = v + 1$.

4. ***Pour*** *tout ensemble candidat* $t_1 \in E_1$ ***Faire***

   Pour *tout ensemble candidat* $t_2 \in E_2$ ***Faire***

   Si *le rectangle* (k_1, k_2, t_1, t_2) *n'est pas un obstacle* ***Alors***

   Si *Crois = horizontale* ***Alors*** $P_h = P_h \cup \{t_2\}$, $R = R \cup \{t_1\}$.

   Si *Crois = verticale* ***Alors*** $P_v = P_v \cup \{t_1\}$, $R = R \cup \{t_2\}$.

   Si *Crois = diagonale* ***Alors*** $P_h = P_h \cup \{t_2\}$, $P_v = P_v \cup \{t_1\}$.

   Fin Si.

   Fin Pour.

   Fin Pour.

5. ***Si*** *tous les passages diagonaux heurtent un obstacle* ***Alors*** *Arrêter.*

6. ***Si*** $(h = n_1 + 1)$ ou $(v = n_2 + 1)$ ***Alors***

   - *Ajouter le point final F comme successeur direct.*
   - *Arrêter.*

   Fin Si.

7. ***Si*** *Crois = horizontale* ***Alors*** $E_1 = R$, $E_2 = \mu_{2h}$.

   Si *Crois = verticale* ***Alors*** $E_1 = \mu_{1v}$, $E_2 = R$.

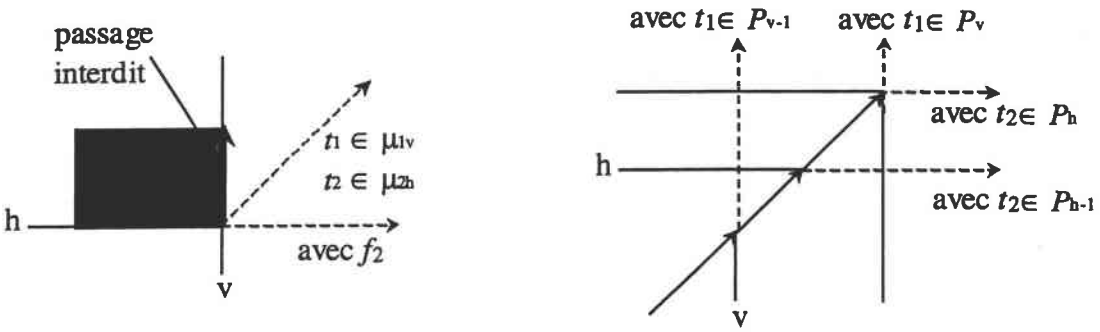
   Si *Crois = diagonale* ***Alors*** $E_1 = \mu_{1v}$, $E_2 = \mu_{2h}$.

8. *Aller à l'étape* 2.

Fin Algorithme.

L'ensemble P_h (resp. P_v) est utilisé pour enregistrer les flexibilités des opérations du job J_2 (resp. J_1) qui ont permis de progresser diagonalement. L'ensemble E_1 (resp. E_2) est utilisé pour connaître les flexibilités du job J_1 (resp. J_2) à utiliser pour progresser diagonalement, à l'étape suivante.

A l'étape d'initialisation, les affectations $P_h = \{f_2\}$, $P_v = \varnothing$ signifient que seul le passage horizontal utilisant la flexibilité f_2 est possible (figure 6.2 (a)). La valeur p_1 (resp. p_2) représente la durée restante pour achever la réalisation d'une opération du job J_1 (resp. J_2).



(a) première progression (b) progressions suivantes

Figure 6.2 : Progressions depuis un sommet *SE*

L'étape 4 de l'algorithme 6.2 permet de déterminer toutes les combinaisons d'ensembles candidats permettant de progresser diagonalement jusqu'à la fin de O_{1,k_1} ou O_{2,k_2} (figure 6.2 (b)). Notons que les sommets *SE* et *NW* des obstacles heurtés à cette étape ne sont pas ajoutés dans les successeurs directs du sommet v_i. En effet, ces sommets ont déjà été ajoutés lors de l'exploration de l'horizontale (resp. verticale) correspondant à *SE* (resp. *NW*), c'est-à-dire à l'étape précédente.

L'exploration du sommet v_i est arrêtée dans les deux cas suivants :

i) Soit aucune combinaison d'ensemble candidats ne permet de progresser diagonalement.

ii) Soit l'obstacle final défini par l'origine *O* et le point final *F* est heurté. Dans ce cas, le point final *F* est ajouté comme dernier successeur possible du sommet v_i.

La procédure *Pass-horizontale* (resp. *Pass-verticale*) utilisée par l'algorithme *SuccSommet-3* précédent, permet de générer les sommets voisins rencontrés en progressant horizontalement (resp. verticalement) et est définie de la manière suivante :

Procédure Pass-horizontale

Données : sommet de départ v, horizontale h, ensemble des flexibilités E_h.

Résultat : sommets voisins horizontaux.

Pour** tout ensemble candidat $f \in E_h$ **Faire

1. *Déterminer le coin SE du premier obstacle potentiel atteignable en allant horizontalement.*
2. ***Si** le passage est possible **Alors***
   - *Soit k_1 l'abscisse du coin SE obtenu.*
   - ***Si** $k_1 < n_1 + 1$ **Alors***

     Ajouter le coin SE (k_1, h, ∅, f) comme successeur direct de v.

     Fin Si.

     ***Sinon** le passage horizontal n'aboutit à aucun successeur.*

   Fin Si.

Fin Pour.

Fin Procédure.

Dans la procédure *Pass-horizontale*, la progression sur l'horizontal h n'est pas possible s'il existe une opération O_{1k} telle que $I(k, h\text{-}1, f, t) = 1$, $\forall f \in \mu_{1k}$, c'est-à-dire si le rectangle défini par les opérations O_{1k} et $O_{2,h\text{-}1}$ est un obstacle quelle que soit l'affectation des ensembles candidats à l'opération O_{1k}. Dans ce cas, le passage horizontal sur la bordure supérieure de l'obstacle défini par O_{1k} et $O_{2,h\text{-}1}$ n'est pas permis à cause de la propriété retenir et attendre.

Procédure Pass-verticale

Données : sommet de départ v, verticale h, ensemble de flexibilité E_v.

Résultat : sommets voisins verticaux.

Pour** tout ensemble candidat $f \in E_v$ **Faire

1. *Déterminer le coin NW du premier obstacle potentiel atteignable en allant verticalement.*
2. ***Si** le passage est possible **Alors***
   - *Soit k_2 l'ordonnée du coin NW obtenu.*
   - ***Si** $k_2 < n_2 + 1$ **Alors***

     Ajouter le coin NW (h, k_2, f, ∅) comme successeur direct de v.

     Fin Si.

     ***Sinon** le passage vertical n'aboutit à aucun successeur.*

   Fin Si.

Fin Pour.

Fin Procédure.

Remarque 6.1 : Dans le cas où le sommet de départ $v_i = (k_1, k_2, f_1, \varnothing)$ est un coin *NW*, la seule modification à apporter à l'algorithme *SuccSommet-3* concerne l'étape d'initialisation. Cette étape devient : $P_h = \varnothing$, $P_v = \{f_1\}$.

Remarque 6.2 : Dans l'algorithme Pass-horizontale (resp. Pass-verticale), le coin $SE = (k_1, h, \varnothing, t)$ (resp. $NW=(h, k_2, t, \varnothing)$) du premier obstacle potentiel atteignable, a les propriétés suivantes :

i) Le rectangle (k_1, h) (resp. (h, k_2)) est un obstacle potentiel : $I(k_1, h, f_1, f_2) = 1$ (resp. $I(h, k_2, f_1, f_2) = 1$), pour au moins un ensemble candidat $f_1 \in \mu_{1,k_1}$ (resp. $f_1 \in \mu_{1h}$) et $f_2 \in \mu_{2h}$ (resp. $f_2 \in \mu_{2,k_2}$).

ii) Le passage est permis vis-à-vis de la propriété de *retenir et attendre* : la progression horizontale (resp. verticale) ne longe pas la bordure supérieure (resp. droite) d'un obstacle potentiel.

3.4. Distance entre deux sommets

Lors du passage du sommet $v_1=(k_1, k_2, f_1, f_2)$ à un sommet voisin $v_2 =(k_3, k_4, t_1, t_2)$, il est clair que tous les ensembles candidats pour les opérations O_{1k}, $k_1 \leq k \leq k_3$-1 et O_{2k}, $k_2 \leq k \leq k_4$-1 ont déjà été choisis.

Si v_2 est un sommet *NW*, la distance entre v_1 et v_2 est égale à la projection sur l'axe Y :

$$d(v_1,v_2) = \sum_{k=k_2}^{k_4-1} p_{2k} \ .$$

Si par contre v_2 est un sommet *SE*, la distance est égale à la projection sur l'axe X :

$$d(v_1,v_2) = \sum_{k=k_1}^{k_3-1} p_{1k} \ .$$

Avant de donner la complexité de l'algorithme, nous montrons que le réseau construit, c'est-à-dire l'ensemble des sommets du réseau et la définition des successeurs, sont suffisants pour obtenir le plus court chemin dans le plan, ce qui correspond au makespan optimal du problème d'ordonnancement à deux jobs.

Théorème 6.2

Le plus court chemin de O à F dans le réseau construit correspond à un plus court chemin dans le plan avec les obstacles définis et donc au makespan optimal pour le problème à deux jobs étudié.

Preuve

Pour chaque affectation des ensembles candidats aux opérations des jobs J_1 et J_2, le problème se ramène à un problème de type *job shop multi-ressources avec blocage*. Celui-ci a été étudié dans le chapitre 2. Une représentation géométrique dans le plan peut être utilisée et le problème d'ordonnancement équivaut alors à la recherche du plus court chemin dans ce plan avec obstacles. Nous définissons donc un réseau G pour déterminer ce plus court chemin. D'après les résultats donnés par (Mati et al. [2001c]), l'ensemble des sommets de G sont les coins *SE* (resp. *NW*) des obstacles obtenus en progressant d'abord diagonalement puis horizontalement (resp. verticalement).

Dans le réseau $N = (V, E, d)$ construit par l'algorithme *SuccSommet-3*, il est clair que V contient tous les sommets du réseau G pour une affectation particulière. En effet, l'ensemble des successeurs directs d'un sommet $v_i = (k_1, k_2, f_1, f_2)$ comprend tous les coins *SE* (resp. *NW*) des obstacles potentiels heurtés en progressant diagonalement puis horizontalement (resp. verticalement). Par conséquent, le réseau N est suffisant pour obtenir un ordonnancement optimal du problème étudié. ■

3.5. Complexité de l'algorithme

La complexité de la recherche du makespan optimal pour le problème à deux jobs dépend de l'algorithme de programmation dynamique, de la construction du réseau et du calcul du plus court chemin.

Théorème 6.3

Le problème *JMPTF* | $n = 2$, *blocking* | C_{max} est polynomial et sa complexité est au plus $O(m_1 \times m_2 \times (m_1 + m_2))$, où $m_i = K_i \times n_i$, $K_i = \max\{\text{card}\,(\mu_{ij})\}$, i = 1, 2, $1 \leq j \leq n_i$.

Preuve

La détermination de tous les obstacles dans le plan en utilisant la programmation dynamique prend un temps $O\,(m_1 \times m_2)$. Le nombre de sommets du réseau est au maximum égal à $m_1 \times m_2$. Le nombre de sommets *SE* (resp. *NW*) obtenus en progressant horizontalement (resp. verticalement) est au maximum égal à $card\,(\mu_{2,k_2})$ (resp. $card\,(\mu_{1,k_1})$). En effet, chaque horizontale (resp. verticale), représentant la fin d'une opération O_{2,k_2} (resp. O_{1,k_1}), est explorée avec éventuellement tous les ensembles candidats de μ_{2,k_2} (resp. μ_{1,k_1}). Ainsi, le nombre de successeurs directs d'un sommet est au maximum égal à $m_1 + m_2$. Par conséquent, le nombre maximum d'arcs dans le réseau est borné par $m_1 \times m_2 \times (m_1 + m_2)$. Puisque le réseau construit est sans circuit, la recherche du plus court chemin prend un temps proportionnel à $O(|E| + |V|)$. La complexité finale est donc $O(m_1 \times m_2) + O(|E| + |V|) = O(m_1 \times m_2 \times (m_1 + m_2))$. ■

3.6. Fonction objectif régulière

L'algorithme de recherche du plus court chemin dans le réseau construit peut être modifié pour prendre en compte des fonctions objectifs régulières. Il suffit de calculer la valeur de la fonction objectif chaque fois que l'obstacle final, délimité par le point d'origine O et le point final F, est rencontré. Le nombre de chemins heurtant l'obstacle final est au pire des cas égal au nombre de sommets $m_1 \times m_2$. Puisque l'évaluation de la fonction objectif prend un temps proportionnel à n_1 (resp. n_2) dans le cas où la bordure supérieure (resp. droite) de l'obstacle final est heurtée, nous pouvons en déduire que :

Théorème 6.4

Le problème *JMPTF* | $n = 2$, *blocking* | f est polynomial pour toute fonction objectif régulière, et sa complexité est au plus $O(m_1 \times m_2 \times (m_1 + m_2))$.

4. Problème à deux jobs : durées opératoires dépendantes de la flexibilité

Dans (Mati et Xie [2001g]), nous avons montré que le problème d'ordonnancement du *job shop flexible avec machines non-reliées* (*job shop with multi-purpose unrelated machines*) était NP-difficile si les durées opératoires dépendent des flexibilités associées aux opérations. De ce fait, le problème d'ordonnancement de type job shop multi-ressources avec blocage et flexibilité des ressources considéré dans ce chapitre est aussi NP-difficile. Dans cette section, la durée d'une opération O_{1,k_1} est notée $p_{1,k_1,f}$, où $f \in \mu_{1,k_1}$.

Dans cette section, nous nous focalisons sur un cas particulier du problème à deux jobs pour lequel seul le job J_1 est flexible. Nous proposons dans ce qui suit une nouvelle extension de l'approche géométrique pour résoudre le problème d'ordonnancement. Cette extension est basée sur une nouvelle manière de déterminer les sommets du réseau et les successeurs directs d'un sommet.

4.1. Définition des obstacles

Comme pour le cas où les durées opératoires sont indépendantes de la flexibilité, les obstacles dans le plan caractérisent les partages de ressources et les situations de blocage.

Pour déterminer ces obstacles, nous proposons un algorithme de programmation dynamique par chaînage arrière. Cet algorithme est une adaptation de celui proposé dans la section précédente et utilise des indicateurs booléens définis de la manière suivante :

$$I(k_1,k_2,f)=\begin{cases}1 & \text{si l'opération } O_{1,k1} \text{ avec l'ensemble candidat } f_1 \text{ et } O_{2,k2} \\ & \text{sont en conflit de ressource,} \\ & \text{ou le blocage est inévitable partant de l'état } (k_1,k_2,f) \\ 0 & \text{sin on}\end{cases}$$

où l'état (k_1, k_2, f) signifie que les deux opérations O_{1,k_1} et O_{2,k_2} sont en cours d'exécution, et l'ensemble candidat f est utilisé pour l'opération O_{1,k_1}.

Tous les obstacles peuvent alors être définis par l'équation récursive suivante :

$$I(k_1,k_2,f)=\begin{cases}1 & \text{si l'opération } O_{1,k1} \text{ avec l'ensemble candidat } f_1 \text{ et } O_{2,k2} \\ & \text{sont en conflit de ressource,} \\ \bigwedge\limits_{t\in\mu_{1,k1+1}} [I(k_1+1,k_2,t)] \wedge [I(k_1,k_2+1,f)] & \text{sin on}\end{cases} \quad (2)$$

Par convention :

$$I(k_1, n_2+1, f) = 0, \forall\ k_1 = 1, 2, ..., n_1+1,$$
$$I(n_1+1, k_2, f) = 0, \forall\ k_2 = 1, 2, ..., n_2+1, \text{ pour tout ensemble candidat } f.$$

L'algorithme de programmation dynamique que nous proposons est basé sur l'équation récursive (2), et calcule tous les indicateurs $I(k_1, k_2, f)$ par récurrence sur $k = k_1 + k_2$.

Algorithme 6.3 : (*Détection de blocages et de conflits de ressources*)

1. *Initialisation* : $I(k_1, n_2+1, t) = 0$, $\forall k_1=1, 2,..., n_1+1$ *et* $I(n_1+1, k_2, t) = 0$, $\forall k_2=1, 2,..., n_2+1$, $\forall t$.

2. ***Pour*** $k = n_1 + n_2$ *jusqu'à 2* ***Faire***

   Pour *tout état* (k_1, k_2) *tel que* $k_1 + k_2 = k$, $k_1 \leq n_1$, $k_2 \leq n_2$ ***Faire***

   Pour *toute flexibilité* $f \in \mu_{1,k1}$ ***Faire***

   Déterminer $I(k_1, k_2, f)$ *en appliquant l'équation récursive (2).*

   Fin Pour.

   Fin Pour.

   Fin Pour.

Fin Algorithme.

Théorème 6.5

La programmation dynamique proposée interdit seulement les obstacles relatifs au partage des ressources et les situations de blocage. De plus, tous les obstacles peuvent être définis en un temps proportionnel à $O(m_1 \times n_2)$.

Preuve

La preuve de ce théorème est similaire à celle du théorème 6.1 et est donc omise. ■

4.2. Ensemble des sommets

Comme nous l'avons mentionné dans la section 3, la présence simultanée de la propriété de *retenir et attendre* et de la flexibilité des ressources nous oblige à ajouter sur chaque sommet une information relative à la flexibilité. Contrairement au cas de durées opératoires indépendantes de la flexibilité, l'information mémorisée pour les coins *SE* est différente de celle pour les coins *NW*.

(A) Caractérisation d'un coin SE

L'information enregistrée pour définir le coin *SE* d'un obstacle (k_1, k_2) est une conséquence des trois propriétés suivants :

i) *Propriété retenir et attendre* : La progression verticale à partir du coin *SE* n'est pas permise car elle longe la bordure droite de l'obstacle.

ii) *J_2 est un job de type job shop multi-ressources sans flexibilité* : La progression horizontale est réalisée pendant que J_2 retient toujours l'ensemble de ressource de l'opération O_{2,k_2-1}.

iii) *Règle de progression* :

- Que ce soit pour le passage horizontale ou pour le passage diagonale, le job J_1 doit progresser en premier en passant à l'opération O_{1,k_1+1}.
- Le choix de l'ensemble candidat f' pour O_{1,k_1+1} ne dépend pas de la flexibilité f choisie pour l'opération O_{1,k_1}. Il dépend seulement de l'admissibilité de l'état (k_1+1, k_2-1).
- Le calcul des indicateurs $I(k_1+1, k_2-1, f')$ ne dépend pas des opérations précédentes (cf. section 4.1).

D'après (i)-(iii), il n'est pas nécessaire de mémoriser la flexibilité de l'opération O_{1,k_1}.

Par conséquent, les coins *SE* de notre réseau peuvent être caractérisés par $SE = (k_1, k_2, *)$.

(B) Caractérisation d'un coin NW

L'information nécessaire pour définir un coin *NW* n'est pas la même que celle d'un coin *SE*. Les sommets *NW* sont caractérisés par le triplet (k_1, k_2, f) signifiant que la progression verticale à partir de ce sommet est réalisée avec l'ensemble candidat f utilisé par l'opération O_{1,k_1-1}. Le passage horizontale, par contre, est interdit à cause de la propriété retenir et attendre.

4.3. Successeurs directs d'un sommet

Pour le problème du *job shop multi-ressources avec blocage*, étudié dans le chapitre 2, les successeurs directs d'un sommet v_i sont les coins *SE* (resp. *NW*) obtenus en progressant diagonalement, puis horizontalement (resp. verticalement). Avec la présence de la flexibilité des ressources, nous considérons comme successeurs directs de v_i tous les coins *SE*=(k_1, k_2, *) (resp. *NW*=(k_1, k_2, f)) des obstacles potentiels pouvant être atteints à partir de v_i en progressant diagonalement puis horizontalement (resp. verticalement).

L'algorithme *SuccSommet-4* suivant permet de déterminer tous les successeurs directs d'un sommet $v_i = (k_1, k_2, f)$. Cet algorithme fait progresser les opérations du job flexible l'une après l'autre et ajoute, à chaque fois, les voisins suffisants pour déterminer un plus court chemin dans le plan.

Algorithme 6.4 : *SuccSommet-4* (*Successeurs directs du sommet* $v_i=(k_1, k_2, f)$)

1. *Initialisation* : $h = k_1$, $v = k_2$, ***Si*** v_i *est un sommet SE* ***Alors*** $E_v = \varnothing$, ***Sinon*** $E_v = \{f\}$.

2. ***Si*** $E_v \neq \varnothing$ ***Alors***

   Appeler la procédure Pass-verticale avec l'ensemble des flexibilités dans E_v.

   Fin Si.

3. *Classer les ensembles candidats* $f_i \in \mu_{1h}$ *par ordre croissant des durées opératoires, c'est-à-dire,* $p_{1,h,f_1} \leq p_{1,h,f_2} \leq \ldots \leq p_{1,h,f_p}$, *où* $p = card(\mu_{1h})$.

4. ***Pour*** *toute flexibilité* $t \in \mu_{1h}$ ***Faire***

   4.1. *Progresser diagonalement jusqu'à la fin de l'opération* O_{1h} *en utilisant la flexibilité t.*

   4.2. ***Si*** *le passage diagonal heurte un obstacle* ***Alors***

   - *Ajouter les coins SE et NW avec la flexibilité choisie pour* $O_{1,h-1}$.
   - *Soit* k_2' *la coordonnée sur l'axe Y du coin SE.*
   - *Appeler la procédure Pass-horizontale avec la colonne de départ v et la colonne d'arrivée* k_2'.

   Fin Si.

   Sinon

   - *Soit* k_2' *l'indice de l'opération où la diagonale croise la verticale représentant la fin de l'opération* O_{1h}.
   - *Appeler la procédure Pass-horizontale avec la colonne de départ v et la colonne d'arrivée* k_2'.

   Fin Sinon.

   Fin Pour.

5. ***Si*** *avec tous les ensembles candidats un obstacle est heurté* ***Alors*** *Arrêter.*

6. ***Si*** $(k_1 = n_1)$ ou $(\forall\, k_2', k_2' = n_2 + 1)$ ***Alors***

   - *Ajouter le point final F comme successeur direct du sommet* v_i.
   - *Arrêter.*

   Fin Si.

   Sinon

   - *Soit* E_p *l'ensemble des flexibilités permettant de progresser diagonalement sans heurter un obstacle.*
   - $E_v \leftarrow E_p$, $h \leftarrow h + 1$, $v \leftarrow k_2'$.
   - *Aller à l'étape* 2.

   Fin Sinon.

Fin Algorithme.

La procédure *Pass-horizontale* (resp. *Pass-verticale*) définie dans l'algorithme précédent, permet de générer les sommets voisins rencontrés en progressant horizontalement (resp. verticalement) :

Procédure Pass-horizontale

Données : colonne de départ k_2, colonne d'arrivée k_2'.

Résultat : sommets voisins horizontaux.

Pour *$j \leftarrow k_2$ jusqu'à k_2'* ***Faire***

Si *l'horizontale h_j n'est pas encore explorée* ***Alors***

1. *Chercher le premier obstacle potentiel rencontré en progressant horizontalement.*
2. ***Si*** *cet obstacle existe* ***alors***
   - *Soit k l'opération du job J_1 correspondante.*
   - *Ajouter le coin SE (k, j, *).*

Fin Si.

Sinon *le passage horizontal n'aboutit à aucun successeur.*

Fin Si.

Fin Pour.

Fin Procédure.

Deux cas ne permettent pas de trouver un obstacle en progressant sur l'horizontale h_j :

i) Soit un obstacle se trouvant sur l'horizontale h_{j-1} nous empêche de passer sur sa bordure supérieure.

ii) Soit l'obstacle final définissant la fin du job flexible J_1 est atteint.

Remarque 6.3 : A partir d'un sommet de départ $v_i = (k_1, k_2, f)$, chaque horizontale est explorée une seule fois par l'algorithme *SuccSomm*-2.

Procédure Pass-verticale

Données : verticale k_1, un ensemble de flexibilité E_v.

Résultat : sommets voisins verticaux.

1. *Initialisation : nbr ← card (E_v).*
2. *Classer les flexibilités de E_v dans l' ordre croissant des durées opératoires.*
3. ***Pour** toute flexibilité $f_i \in E_v$ **Faire***
   - *Déterminer l'indice de la dernière opération qui peut être réalisée en progressant verticalement.*
   - *Soit ind_i l'indice obtenu.*

   Fin Pour.
4. ***Si** le coin NW=(k_1, k, f_1) obtenu avec la première flexibilité $f_1 \in E_v$ existe **Alors***

   Ajouter le sommet NW dans les successeurs directs.

   Fin Si.
5. ***Pour** i← 2 jusqu'à (nbr - 1) **Faire***

   Si** la flexibilité f_i n'est pas négligée **Alors

   Pour** j← i+1 jusqu'à nbr **Faire

   Si** la flexibilité f_j n'est pas négligée **Alors

   Si** $ind_i \geq ind_j$ **Alors

   Le coin NW=(k_1, k, f_j) obtenu en utilisant la flexibilité f_j n'est pas nécessaire (f_j est négligée).

   Fin Si.

   Sinon

   Ajouter le coin NW=(k_1, k_2, f_j) comme successeur direct.

   Fin Sinon.

   Fin Si.

   Fin Pour.

   Fin Si.

   Fin Pour.

Fin Procédure.

Lemme 6.1

Un coin *NW* obtenu en progressant sur une verticale *V* avec une flexibilité *f* ignorée par l'algorithme *Pass-verticale* n'est pas nécessaire pour obtenir un plus court chemin dans le plan.

Preuve

Montrons que tout ordonnancement obtenu en utilisant une flexibilité *f* ignorée par l'algorithme *Pass-verticale*, peut aussi être obtenu avec une flexibilité *t* sans que la valeur du makespan ne soit dégradée.

Soit *P* le plus court chemin du sommet v_i au sommet final *F* utilisant la flexibilité *f*, et notons par p_1 le point de croisement de *P* avec la verticale *V*. Considérons une flexibilité *t* telle que $ind_t \geq ind_f$. La progression diagonale utilisant *t* croise la verticale *V* en un point p_2. La coordonnée de p_2 est inférieure ou égale à celle de p_1 puisque les flexibilités sont classées par ordre croissant des durées opératoires. Deux cas sont alors possibles :

(a) S'il n'existe aucun obstacle potentiel entre les points p_1 et p_2 (figure 6.2 (a)) alors en remplaçant la partie de *P* entre v_i et p_2 par $v_i\ p_1\ p_2$, nous obtenons un nouveau chemin réalisable *P'* ayant la même longueur que *P* puisque le job J_2 n'est jamais interrompu. De plus, tout état atteint par une progression diagonale avec la flexibilité *f* peut l'être avec *t*, le calcul des indicateurs d'états ne dépendant des flexibilités précédentes (cf. section 4.1).

(b) S'il existe un obstacle D_1 entre p_1 et p_2 (figure 6.2 (b)), alors d'après l'étape 4 de l'algorithme 6.4, le coin NW_1 (avec la flexibilité *t*) de D_1 est ajouté comme successeur direct de v_i. A partir de NW_1, la première progression possible est verticale et utilise la flexibilité *t*. Si ce passage vertical heurte un obstacle potentiel avant d'atteindre p_2 alors, nous répétons ce processus jusqu'à obtenir un sommet NW_j à partir duquel le point p_2 est atteint. Dans ce dernier cas, nous pouvons remplacer le chemin de P entre v_i et p_2 par le nouveau chemin réalisable $P = v_i\ p_1\ NW_1\ NW_2 \ldots NW_j\ p_2$, et ceci sans altérer la longueur du chemin.

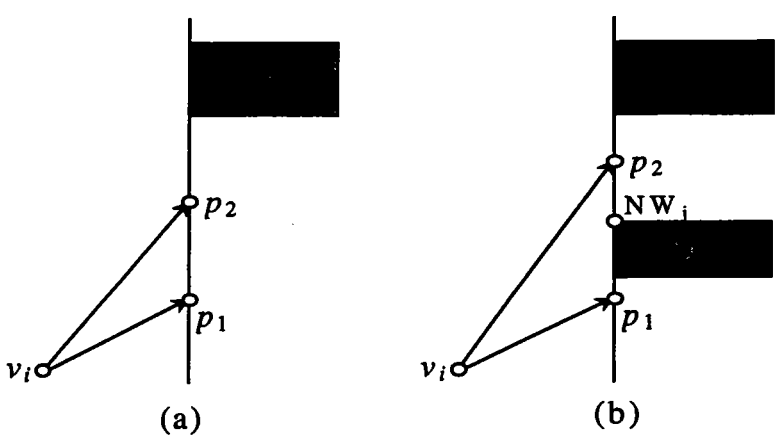


Figure 6.2 : Points de croisement.

En résumé, nous pouvons modifier dans tous les cas le chemin P en remplaçant la flexibilité f négligée par une flexibilité considérée t sans altérer la valeur du plus court chemin. ■

4.4. Distance entre deux sommets

Lors du passage d'un sommet $v_1 = (k_1, k_2, f)$ à un sommet voisin $v_2 = (k_3, k_4, t)$, il est clair que toutes les flexibilités des opérations O_{1k}, $k_1 \leq k \leq k_3$-1 sont déjà affectées. La distance entre ces deux sommets peut être calculée lors de l'exploration de l'algorithme *SuccSommet-4*. Plus précisément :

- $d(v_1,v_2) = \sum_{k=k_2}^{k_4-1} p_{2k}$, si le sommet v_2 est un sommet *NW*.
- $d(v_1,v_2) = \sum_{k=k_1}^{k_3-1} p_{1,k,f_k}$,si le sommet v_2 est un sommet *SE*.

où f_k est la flexibilité choisie pour l'opération O_{1k}.

En utilisant la représentation géométrique, un chemin dans le plan correspond à une solution réalisable du problème d'ordonnancement du job flexible et du job sans la flexibilité des ressources. Le théorème suivant montre que la valeur du plus court chemin dans le plan correspond au makespan optimal du problème d'ordonnancement.

Théorème 6.6

Le plus court chemin partant de l'origine O et arrivant au point final F dans le réseau construit correspond à un plus court chemin dans le plan avec obstacles définis et donc à la solution optimale du problème à deux jobs considéré.

Preuve

Le problème d'ordonnancement étudié est une extension du modèle *job shop multi-ressources avec blocage* étudié dans le chapitre 2. Nous avons montré que le réseau G défini par les coins *SE* (resp. *NW*) des obstacles obtenus en progressant diagonalement puis horizontalement (resp. verticalement) était suffisant pour trouver l'ordonnancement optimal.

L'ensemble des successeurs directs d'un sommet v_i contient tous les coins *SE* (resp. *NW*) des obstacles potentiels heurtés en progressant diagonalement puis horizontalement (resp. verticalement) à partir de v_i. Il est clair que V contient tous les sommets du réseau G pour une affectation particulière. D'après le lemme 6.1, les coins *NW* négligés par l'algorithme *SuccSommet-4* correspondent aux arcs qui ne conduisent pas à un plus court chemin. Par conséquent, le réseau N est suffisant pour déterminer un ordonnancement optimal du problème étudié. ■

4.5. Complexité de l'approche

La complexité de l'algorithme de recherche du makespan optimal pour le problème à deux jobs dépend de l'algorithme de programmation dynamique, de la construction du réseau et du calcul du plus court chemin.

Théorème 6.7

Le problème *JMPTF* | $n = 2$, *J*, *FJ*, *blocking* | C_{max} est polynomial et sa complexité est au plus $O(m_1 \times n_2 \times (m_1 + n_2))$, où *J* désigne le job de type job shop multi-ressources, et *FJ* le job flexible.

Preuve

La détermination de tous les obstacles dans le plan en utilisant la programmation dynamique prend un temps proportionnel à $O(m_1 \times n_2)$. Le nombre de sommets du réseau est au maximum égal à $(m_1+1) \times n_2$. Par ailleurs, le nombre de successeurs directs d'un sommet est au plus égal à $(n_2 + m_1)$, chaque horizontale étant explorée une seule fois. Par conséquent, le nombre maximal d'arcs dans le réseau est borné par $(m_1+1) \times n_2 \times (n_2 + m_1)$. Puisque le réseau construit est sans circuits, la recherche du plus court chemin prend un temps proportionnel à $O(|E| + |V|)$. La complexité finale est donc égale à $O((m_1+1) \times n_2) + O(|E| + |V|) = O(m_1 \times n_2 \times (m_1 + n_2))$. ■

4.6. Fonction objectif régulière

L'algorithme de recherche du plus court chemin dans le réseau construit peut être modifié pour prendre en compte d'autres fonctions objectifs régulières. Il suffit pour cela de calculer la valeur de la fonction objectif à chaque fois que l'obstacle final est rencontré. Le nombre de chemins heurtant l'obstacle final est au pire des cas égal à $m_1 \times n_2$. Puisque le calcul de la fonction objectif prend un temps proportionnel à n_1 (resp. n_2) dans le cas où la bordure supérieure (resp. droite) de l'obstacle final est heurtée, nous pouvons en déduire que :

Théorème 6.8

Le problème *JMPTF* | $n = 2$, *J*, *FJ*, *blocking* | f est polynomial pour toute fonction objectif régulière, et sa complexité est au plus $O(m_1 \times n_2 \times (m_1 + n_2))$.

5. Heuristique pour la résolution du problème général

L'heuristique permettant de résoudre le problème général avec un nombre quelconque de jobs est basée sur la méthodologie décrite dans le chapitre 2. Il s'agit d'une heuristique gloutonne qui ordonnance les jobs l'un après l'autre suivant une séquence de traitement fixée à priori. Les deux premiers jobs de la séquence sont ordonnancés de façon optimale par les algorithmes polynomiaux proposés dans les sections 3 et 4. Une fois le makespan obtenu, un

job composé est construit puis ordonnancé avec le job suivant de la séquence. Ce procédé est réitéré jusqu'au traitement de tous les jobs.

5.1. Construction du job composé

Deux jobs étant ordonnancés, la première étape de la construction du job composé est la représentation de la solution optimale par le diagramme de Gantt. La deuxième étape consiste à décomposer l'intervalle [0, C_{max}] en sous-intervalles selon les dates de fin des opérations. Chaque intervalle va correspondre à une opération du job composé. Par conséquent, le nombre d'opérations du job composé est égal au nombre d'intervalles crées. Les durées opératoires sont égales aux longueurs des intervalles associés et les ressources nécessaires sont les ressources occupées pendant ces intervalles.

Remarque 6.4 : Une fois le job composé construit, il est considéré comme un job multi-ressources avec blocage et sans flexibilité des ressources. Ce qui signifie que l'ordre d'entrée des jobs sur les ressources n'est pas remis en cause.

5.2. Heuristique gloutonne

Dans cette sous-section, nous proposons deux heuristiques appelées *Heur-même* et *Heur-diff* pour résoudre le problème d'ordonnancement *JSMBF* avec plusieurs jobs. La première heuristique est dédiée au cas où la durée d'une opération ne dépend pas de l'ensemble candidat sélectionné. La deuxième heuristique permet de résoudre le problème où les durées opératoires dépendent des flexibilités. Les deux heuristiques proposées sont décrites par les algorithmes 6.5 et 6.6.

Algorithme* 6.5 : *Heur-même

1. *Sélectionner une séquence d'entrée des jobs. Sans perte de généralité, soit $J_1, J_2, ..., J_N$ cette séquence.*
2. $J_{com} \leftarrow J_1$.
3. ***Pour** $i \leftarrow 2$ jusqu'à N **Faire***
   - *3.1. Résoudre le problème à deux jobs entre J_{com} et J_i en appliquant l'algorithme de la section 3.*
   - *3.2. Construire le job composé J' grâce à l'ordonnancement obtenu.*
   - *3.3. $J_{com} \leftarrow J'$.*

   Fin Pour.
4. *Déterminer les séquences d'entrée σ_r des opérations sur chaque ressource R_r à partir du job composé J_{com}.*

Fin Algorithme.

Pour le cas de durées opératoires dépendantes de la flexibilité, la seule différente concerne l'étape 2 d'initialisation. En effet, si les jobs J_1 et J_2 sont tous les deux flexibles, le problème d'ordonnancement ne peut pas être résolu efficacement. Afin de pouvoir utiliser l'algorithme polynomial de la section 4, nous fixons grâce à une règle de priorité les flexibilités du premier job de la séquence. La règle de priorité choisie est basée sur la charge de travail sur les ressources.

Algorithme* 6.6 : *Heur-diff

1. *Sélectionner une séquence d'entrée des jobs. Sans perte de généralité, soit $J_1, J_2, \ldots, J_N$ cette séquence,*
2. *$J_{com} \leftarrow J_1$, et affecter les ensembles candidats du job J_{com}.*
3. ***Pour** $i \leftarrow 2$ jusqu'à N **Faire***
   - *3.1. Résoudre le problème à deux jobs entre J_{com} et J_i en appliquant l'algorithme de la section 4.*
   - *3.2. Construire le job composé J' grâce à l'ordonnancement obtenu.*
   - *3.3. $J_{com} \leftarrow J'$.*

   Fin Pour.
4. *Déterminer les séquences d'entrée σ_r des opérations sur chaque ressource R_r à partir du job composé J_{com}.*

Fin Algorithme.

5.3. Amélioration de l'ordonnancement

La construction d'un job composé permet de regrouper tous les jobs déjà examinés. Ainsi, le problème d'ordonnancement du *JSMBF* se ramène à un ensemble de problèmes à deux jobs, ce qui réduit la complexité de la résolution.

Cependant, les algorithmes 6.5 et 6.6 appliqués au problème entre un job composé J_{com} et un job J_i, peuvent générer des ordonnancements avec délai. Ceci est dû à la présence d'opérations composées (cf. chapitre 2).

Pour obtenir un ordonnancement sans délai, nous utilisons la procédure proposée dans la section 6 du chapitre 2. Nous commençons par extraire les séquences d'entrée σ_r des opérations sur chaque ressource R_r. Ensuite, nous ordonnançons pour tous les jobs J_j, les opérations O_{jk} une après l'autre selon la séquence d'entrée σ_r. Ainsi, la date de début au plus tôt de l'opération O_{jk} respectant la séquence d'entrée σ_r peut être calculée comme suit :

$$S'_{jk} = \max\left\{S'_{j,k-1} + p_{j,k-1,f}, \max_{r \in R_{jk}}\{Disp(r)\}\right\} \qquad (3)$$

où f est la flexibilité choisie pour l'opération $O_{j,k-1}$, et $Disp(r)$ la date de disponibilité de la ressource r.

La performance des algorithmes 6.5 et 6.6 dépend fortement de la séquence d'entrée des jobs choisie à l'étape 1. Aussi, ces algorithmes sont couplées respectivement à un algorithme de recherche tabou et à un algorithme génétique. Ces deux méthodes ont seulement pour but l'optimisation de la séquence d'entrée des jobs.

5.4. Optimisation de la séquence d'entrée des jobs

Dans cette sous-section, nous nous focalisons sur la détermination de bonnes séquences d'entrée des jobs nécessaires à l'application des algorithmes gloutons 6.5 et 6.6. Une première possibilité est d'explorer toutes les combinaisons possibles. Mais ceci n'est envisageable que dans le cas où le nombre de jobs est assez faible. Nous pouvons également faire démarrer les algorithmes gloutons à partir de séquences générées aléatoirement. Le risque est alors de ne pas considérer certaines séquences pouvant donner de bonnes solutions.

Dans le but d'explorer intelligemment l'espace de toutes les séquences possibles, nous proposons deux métaheuristiques qui ont montré leur efficacité à résoudre les problèmes d'optimisation combinatoire : la recherche tabou et les algorithmes génétiques. La recherche tabou est utilisée pour optimiser la séquence d'entrée de l'algorithme 6.5, c'est-à-dire pour le cas de durées opératoires égales. L'algorithme génétique est utilisé pour l'optimisation de la séquence d'entrée de l'algorithme 6.6, c'est-à-dire pour le cas de durées opératoires dépendantes de la flexibilité.

5.4.1. Recherche tabou

La méthodologie utilisée dans cette sous-section est similaire à celle utilisée dans le chapitre 2 pour l'optimisation de la séquence des jobs. La recherche tabou est appliquée à un problème artificiel équivalent au problème *single machine-scheduling with sequence-dependent setup times* (Pinedo [1995]), qui consiste à ordonnancer un ensemble de N tâches toutes disponibles à la date 0 sur une seule machine. La détermination du problème artificiel est possible du fait que le problème d'ordonnancement à deux jobs flexibles est polynomial. Pour plus de détails, nous renvoyons le lecteur à la section 7 du chapitre 2.

5.4.2. Algorithme génétique

Le problème d'ordonnancement à deux jobs flexibles avec des durées opératoires dépendantes de la flexibilité est NP-difficile, ce qui implique que le makespan optimal ne peut être déterminé de manière efficace. Par conséquent, le problème artificiel défini précédemment ne peut être obtenu rapidement.

De ce fait, plutôt que d'utiliser une heuristique de recherche locale, nous avons opté pour une heuristique de population, en l'occurrence, les algorithmes génétiques.

Pour les paramètres utilisés ainsi que la manière de les générer, nous renvoyons le lecteur à la section 4 du chapitre 5.

5.5. Heuristiques finales

En utilisant les algorithmes gloutons, la procédure d'amélioration de l'ordonnancement et la phase d'optimisation des séquences d'entrée des jobs, nous proposons des heuristiques pour résoudre le problème d'ordonnancement de *JSMB*. Dans le cas de durées opératoires égales (resp. différentes) l'heuristique finale notée *JSMBF-même* (resp. *JSMBF-diff*) est donnée par l'algorithme 6.7 (resp. 6.8).

Algorithme 6.7 : *JSMBF-même*

1. *Initialisation: Déterminer la séquence d'entrée en utilisant l'heuristique d'insertion. Soit* $J_{\pi(1)}, J_{\pi(2)}, \ldots, J_{\pi(N)}$ *cette séquence.* $C^*_{max} \leftarrow \infty$.
2. *Calculer l'ordonnancement* J_{com} *en utilisant l'algorithme 6.5 avec* π *comme séquence d'entrée.*
3. *Déterminer les nouvelles dates de début en utilisant l'équation (3).*
4. *Soit* C_{max} *le makespan trouvé.*
5. ***Si*** $C_{max} < C^*_{max}$ ***Alors***
   - 5.1. ***Mettre à jour le meilleur makespan*** $C^*_{max} \leftarrow C_{max}$
   - 5.2. ***Enregistrer*** *l'ordonnancement* J_{com} ***comme étant le meilleur*** *ordonnancement;*

   Fin Si.
6. *Critère d'arrêt:* ***Si*** *le nombre maximum d'itérations est atteint* ***Alors*** *stop.*
7. *Trouver le meilleur mouvement non tabou en utilisant la "fonction objectif artificielle".*
8. *Réaliser le mouvement pour créer une nouvelle séquence de jobs* π
9. *Mettre à jours les structures de la recherche tabou.*
10. *Aller à l'étape 2.*

Fin Algorithme.

Pour l'algorithme 6.7 ci-dessus, la méthode d'insertion utilisée à l'étape d'initialisation est similaire à celle de l'algorithme 2.3 du chapitre 2.

Algorithme 6.8 : *JSMBF-diff*

1. *Initialisation :*
   Générer aléatoirement une population initiale de pop solutions. $C^*_{max} = \infty$.
2. ***Pour*** $j \leftarrow 1$ *jusqu'à pop* ***Faire***
   - *Appliquer l'algorithme 2.6 en démarrant de la séquence d'entrée j.*
   - *Déterminer les nouvelles dates de début en utilisant l'équation (3).*
   - *Affecter le makespan trouvé* C_{max} *au fitness de la séquence.*
   - ***Si*** $C_{max} < C^*_{max}$ ***Alors*** $C^*_{max} \leftarrow C_{max}$

   Fin Pour.
3. ***Pour*** $j \leftarrow 1$ *jusqu'à pop/2* ***Faire***

   3.1. Sélectionner deux séquences d'entrée parmi la population.

   3.2. ***Si*** $p_c < 0.9$ ***Alors***
   - *Appliquer l'opérateur de croisement.*
   - *Les deux séquences obtenues remplacent les anciennes.*
   - ***Si*** $p_m < 0.05$ ***Alors***

     Appliquer l'opérateur de mutation.

     Fin Si.

   Fin Si.

   Sinon

   Garder les deux séquences d'entrée dans la population.

   Fin Sinon.

   Fin Pour.
4. *Critère d'arrêt :*

   Si *le nombre limite de générations est atteint* ***Alors***
   - *Renvoyer la meilleure solution ainsi que son makespan.*
   - *Arrêter.*

   Fin Si.

   Sinon *Aller à l'étape 2.*

Fin Algorithme.

6. Résultats expérimentaux

Nous présentons dans cette section les résultats obtenus par les heuristiques *JSMBF-même* et *JSMBF-diff* sur une série d'instances. Ces instances sont basées sur celles du job shop flexible, auxquelles nous ajoutons la propriété de *retenir et attendre*.

Pour le cas des durées opératoires égales, nous effectuons les tests sur les instances générées par Hurink et al. [1994]. Pour le cas des durées opératoires différentes, nous utilisons les instances de Brandimarte [1993]. Ces choix sont motivés par l'absence de jeu de données pour le problème d'ordonnancement avec prise en compte simultanée du blocage et de la flexibilité des ressources. Les résultats obtenus sont donnés par les tableaux 6.1 et 6.2.

Tableau 6.1. Résultats sur les extensions des problèmes *vdata* dans Hurink et al. [1994]

instance	*JSMBF-même*			instance	*JSMBF-même*		
	C_{deb}	C_{meil}	CPU secs		C_{deb}	C_{meil}	CPU secs
la01	1145	763	47.53	la16	1372	893	374.77
la02	1123	704	72.97	la17	1281	816	312.98
la03	918	662	38.54	la18	1263	937	141.01
la04	1042	688	34.21	la19	1331	991	271.14
la05	1098	641	68.95	la20	1493	947	167.61
la06	1523	1104	51.53	la21	1891	1291	417.67
la07	1596	1063	120.08	la22	2280	1155	327.23
la08	1497	1099	46.40	la23	1775	1216	276.32
la09	1517	1178	57.12	la24	2077	1307	334.18
la10	1694	1085	92.56	la25	1858	1210	475.45
la11	1765	1548	182.98	la26	3464	1869	312.94
la12	1756	1323	168.33	la27	2788	1877	386.44
la13	1796	1532	146.31	la28	2905	1902	190.79
la14	2166	1484	108.66	la29	2485	1763	82.15
la15	1805	1540	45.56	la30	3123	1978	124.31

Tableau 6.2. Résultats sur les extensions des instances de Brandimarte [1993]

instance	*JSMBF-diff*			instance	*JSMBF-diff*		
	C_{deb}	C_{meil}	CPU secs		C_{deb}	C_{meil}	CPU secs
mk01	68	54	21.47	mk06	123	98	192.60
mk02	51	39	11.87	mk07	279	219	70.82
mk03	328	254	128.37	mk08	914	763	300.44
mk04	124	96	167.84	mk09	717	538	378.88
mk05	326	263	33.82	mk10	349	284	170.50

7. Discussions

La première observation que nous pouvons tirer des tableaux 6.1 et 6.2 est que la qualité de la solution initiale fournie par la procédure d'insertion (resp. génération aléatoire) dans le cas de durées opératoire égales (resp. différentes) n'est pas toujours bonne. La seconde observation est que les heuristiques proposées améliorent les solutions initiales en un temps d'exécution raisonnable. De plus, des expériences ne figurant pas dans les tableaux que nous avons réalisées en partant de différentes séquences montrent que la solution initiale n'influe pas fortement sur la qualité de la solution finale. Ceci signifie que les heuristiques proposées sont robustes ; elles ne donnent pas des solutions aberrantes d'une exécution à une autre.

Finalement, nous avons réalisé des expérimentations sur un temps d'exécution assez grand. Nous avons alors observé que les deux heuristiques continuaient à trouver de nouvelles meilleures solutions. Ceci prouve que les heuristiques gloutonnes utilisées à l'intérieur des algorithmes *JSMBF-même* et *JSMBF-diff* sont performantes si la séquence d'entrée des jobs est optimale.

8. Conclusion

Dans ce chapitre, nous nous sommes intéressés à un modèle d'ordonnancement avec des contraintes de flexibilité des ressources et la présence de la propriété de *retenir et attendre*. Ce nouveau modèle d'ordonnancement généralise ceux traités dans les chapitres 2 et 5. Nous avons proposé deux problèmes d'ordonnancement, l'un avec des durées opératoires égales et l'autre avec des durées opératoires dépendantes de la flexibilité.

Dans un premier temps, nous avons proposé des algorithmes polynomiaux originaux pour résoudre le problème d'ordonnancement à deux jobs. Le premier algorithme est dédié au cas où la durée d'une opération ne dépend pas de la flexibilité choisie, et ceci avec deux jobs flexibles. Dans le cas où les durées opératoires dépendent des flexibilités, l'algorithme polynomial a été proposé pour un cas particulier avec seulement un job flexible. Dans un second temps, nous avons utilisé ces algorithmes polynomiaux pour développer des heuristiques gloutonnes permettant de résoudre le problème d'ordonnancement à plusieurs jobs. Les heuristiques ont en outre été couplées à une recherche tabou et à un algorithme

génétique de sorte que leurs performances soient augmentées. Enfin, les heuristiques ont été testées sur des instances généralisant celles de la littérature.

Les approches présentées dans ce chapitre peuvent être utilisées en temps réel pour l'ordonnancement et le contrôle des systèmes de production automatisés. Par ailleurs, les algorithmes polynomiaux restent applicables dans le cas où les ressources sont disponibles en plusieurs exemplaires. En revanche, les heuristiques gloutonnes doivent être modifiées pour prendre en compte cette nouvelle hypothèse. Ces modifications, que nous sommes en train d'étudier, concernent la résolution du problème d'ordonnancement entre un job composé et un job flexible.

Finalement, les bons résultats de la méthodologie proposée nous incitent à l'appliquer sur des modèles d'ordonnancement plus généraux. Une des généralisations intéressantes du modèle est l'extension de la notion de flexibilité des ressources à celle de flexibilité des routages. En d'autres termes, nous envisageons d'étudier le cas où les gammes opératoires des jobs sont non linéaires.

Conclusion Générale

Dans cette thèse, nous nous sommes intéressés aux problèmes d'ordonnancement des systèmes de production automatisés. Dans ce type de systèmes, la fabrication d'un produit nécessite la présence simultanée de plusieurs moyens de production automatisés tels que les moyens de transport, les machines numériques, etc. Nous avons en outre pris en compte la capacité des stocks tampons entre les machines, souvent supposée infinie dans les travaux de recherches en ordonnancement.

Les travaux présentés sont consacrés à la résolution de plusieurs problèmes d'ordonnancement rencontrés dans des systèmes de production automatisés, et proposent des nouveaux modèles permettant d'intégrer les principales caractéristiques de ces systèmes. Dans le premier chapitre, nous avons mené une étude bibliographique portant sur les principaux modèles d'ordonnancement existants dans les systèmes de production, en particulier, nous nous sommes intéressés à ceux qui prennent en compte les principales caractéristiques et contraintes considérées par notre recherche. Cette étude a montré que la littérature concernant l'ordonnancement des systèmes de production avec la prise en compte des différents moyens de production n'est pas très riche comparée à celle dédiée aux problèmes classiques d'ordonnancement.

La première partie de la thèse est consacrée aux problèmes d'ordonnancement en présence d'opérations multi-ressources et de la propriété retenir et attendre ; une propriété très importante pour une modélisation réaliste des systèmes de production, et dont la plupart des travaux de recherche existants a fait abstraction. Pour ce modèle, nous avons proposé deux méthodes novatrices pour le contrôle et l'ordonnancement. La première méthode est basée sur une nouvelle méthodologie et utilise des algorithmes polynomiaux originaux. La deuxième méthode est basée sur une nouvelle représentation du problème d'ordonnancement par le graphe disjonctif et utilise une heuristique de recherche locale fondée sur une définition originale de la structure du voisinage.

Dans la deuxième partie de la thèse, nous nous sommes intéressés aux problèmes d'ordonnancement en présence de flexibilité sur l'exécution des opérations. Dans un premier temps, une étude de complexité a été réalisée, permettant de situer la complexité des problèmes d'ordonnancement étudiés par rapport à celle des modèles existants dans la littérature. Des résultats originaux de complexité ont donc été présentés pour des fonctions objectifs régulières, et des nouveaux algorithmes polynomiaux ont été développés. Dans un second temps, nous avons généralisé ces algorithmes polynomiaux et proposé des méthodes heuristiques pour le problème d'ordonnancement en présence simultanée d'opérations multi-ressources et de flexibilité des ressources. Enfin, nous avons étudié un nouveau modèle d'ordonnancement combinant ceux étudiés précédemment. Ce modèle est caractérisé par la

présence d'opérations multi-ressources, de flexibilité des ressources et de la propriété retenir et attendre. Des nouveaux algorithmes polynomiaux ont été présentés et des méthodes heuristiques développées.

Les résultats numériques présentés tout au long de cette thèse montrent que la méthodologie proposée, consistant à ordonnancer les jobs un à la suite de l'autre est très efficace. En effet, la comparaison de nos résultats avec ceux de méthodes plus spécifiques de la littérature, montrent que nos algorithmes sont très compétitifs ; ils permettent de générer plusieurs solutions optimales et proposent des nouvelles meilleures solutions. De plus, dans le cas où les meilleures solutions ne sont pas atteintes, l'écart par rapport à celles-ci est très faible.

Au delà de l'objectif global qui consistait à proposer des méthodes efficaces pour le contrôle et l'ordonnancement dans les systèmes de production automatisés, nous espérons que cette thèse pourra constituer un premier pas vers de nouveaux travaux de recherche sur les problèmes d'ordonnancement intégrant des contraintes industrielles rencontrées dans les systèmes de production automatisés.

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] Abadi, I.N.K., Hall, N.G., and Sriskandarajah, C., (2000). Minimizing Cycle Time in a Blocking Flow Shop. *Operations Research*, 48, 1, 177-180.

[3] Adams, J. Balas, E., and Zawack, D., (1988). The Shifting Bottleneck Procedure for the Job-shop Scheduling. *Management Science*, 34, 3, 391-401.

[4] Adanso-Diaz, B., Oliva Gonzalez, M., and Gonzalez-Torre, P., (1999). On-line Timetable Re-scheduling in Regional Train Services, *Transportation Research Part B*, 33, 387-398.

[5] Akers, S. B., Friedman, J., (1955). A Non-numerical Approach to Production Scheduling Problems. *Operations Research*, 3, 429-442.

[6] Applegate, D., Cook, W. (1991). A Computational Study of the Job-shop Scheduling Problem. *ORSA Journal of Computing*, 3, 149-156.

[7] Baker, K.R. (1974). *Introduction to Sequencing and Scheduling*. Wiley, New York.

[8] 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.

[9] Balas, E., and Vazacopoulos, A., (1998). Guided Local Search with Shifting Bottleneck for Job-shop Scheduling. *Management Science*, 44, 2, 262-275.

[10] Banaszak, Z.A., B.H. Krogh (1990). Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows. *IEEE Transactions on Robotics and Automatation*, 6, 724-734.

[11] Barnes, J.W., and Chambers, J.B., (1996). Flexible Job Shop Scheduling by Tabu Search. Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, *Technical Report Series ORP96-09*, http://www.cs.utexas.edu/users/jbc/.

[12] Bilge, Ü., and G. Ulusoy (1995). A Time Windows Approach to Simultaneous Scheduling of Machines and Material Handling System in an FMS. *Operations Research*, Vol. 43, No. 6, pp. 1058-1070.

[13] Blazewicz, J., Eiselt, H., Finke, G., and Woeginger, G.J., (1991). Scheduling Tasks and Vehicles in a Flexible Manufacturing Systems. *International Journal of Flexible Manufacturing Systems*, 4, 5-16.

[14] Blazewicz, J. and G. Finke (1994). Scheduling with Resource Management in Manufacturing Systems. *European Journal of Operational Research*, 76, 1-14.

[15] Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., and Weglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer, Berlin.

[16] 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.

[17] Bozer, Y. A., (1989). Guided Vehicle Systems : Information/Control System implication of Alternative Design and Operations Strategies. In *Advance Information Technologies for Industrial Material Flow Systems*, eds S.Y. Nof, and C.L. Moodie, NATO ASI Series, 417-436. Springer-Verlag, Berlin, 1989.

[18] Brandimarte, P., (1993). Routing and scheduling in a flexible job shop by tabu search, *Annals of Operations Research*, 41, 157-183.

[19] Brucker, P., (1988). An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40, 353-359.

[20] Brucker, P., (1998). *Scheduling Algorithms*. Springer-Verlag, Berlin Heidelberg.

[21] Brucker, P., Schlie, R., (1990). Job-shop scheduling with multi-purpose machines, *Computing*, 45, 369-375.

[22] Brucker, P., Jurisch, B., (1993). A new lower bound for job-shop scheduling problem, *European Journal of Operational Research*, 64, 156-167.

[23] 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.

[24] Brucker, P., Jurisch, B., and Krämer, A., (1994). The Job-shop Problem and Immediate Selection. *Annals of Operations Research*, 50, 73-114.

[25] Brucker, P., and J. Neyer, (1998). Tabu search for the multi-mode job-shop problem, *Operations Research Spektrum*, 20, 21-28.

[26] Brucker, P. Drexl, A., Möhring, R., Neumann, K., and Pesch, E., (1999). Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112, 3-41.

[27] Cai, X., Goh, C.J., and Mees, A.I., (1998). Greedy Heuristics for Rapid Scheduling of Trains on a Single Track. *IIE Transactions*, 30, 481-493.

[28] Carlier, J., and Chretienne, P. (1988). *Les Problèmes d'ordonnancement*. Masson, Paris, France.

[29] Carlier, J., and Pinson, E. (1989). An Algorithm for Solving the Job Shop Problem. *Management Science*, 35, 164-176.

[30] 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.

[31] 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.

[32] Chen, M., (1996). A Mathematical Programming Model for AGVs Planning and Control in Manufacturing Systems. *Computers and Industrial Ingineering*, 30, 4, 647-658.

[33] 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.

[34] Cho, H., Kumaran, T.K, and Wysk, R.A., (1995). Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems. *IEEE Transactions on Robotics and Automation*, 11, 3, 413-421.

[35] Chu, F., and X.L. Xie (1997). Deadlock analysis of Petri nets Using Siphons and

Mathematical Programming. *IEEE Transaction on Robotics and Automation*, 13, 793-804.

[36] Coffman, E. G., Elphiek, M. and Shoshani, A., (1971). Systems Deadlocks. *Computing Surveys*, 3, 2, 67-78.

[37] 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.

[38] Conway, R.W., Maxwell, W.L., Miller, L.W. (1967). *Theory of Scheduling*. Addison Wesley, Reading, Mass., USA.

[39] Cordeau, J.F., Toth, P., and Vigo, D., (1998). A Survey of Optimization Models for Train Routing and Scheduling. *Transportation Science*, 32, 4, 380-420.

[40] Crama, Y., Van de Klundert, J., (1997). Cyclic Scheduling of Identical Parts in Robotic Cell. *Operations Research*, 45, 6, 952-965.

[41] Damasceno, B.C. and X.L. Xie (1998). Scheduling and Deadlock Avoidance of a Flexible Manufacturing System. *Proceeding IEEE Conference on Systems, Man and Cybernetic*, 564-569, San Diego, USA.

[42] Damasceno, B.C. and X.L. Xie (1999). Petri Nets and Deadlock-free Scheduling of Multiple-Resource Operations. *Proceeding. IEEE Conference on Systems, Man and Cybernetic*, Tokyo, Japan.

[43] Damasceno, B.C. (1999). Ordonnancement des Systemes de Production Multi-ressources avec la Prise en Compte de Blocage, *P.h.D Thesis*, Université de Metz, France.

[44] 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.

[45] Dauzère-Pérès, S., (1994). A Connected Neighborhood Structure for the Multiprocessor Job-shop Scheduling Problem. *Management Report Series No.180*, Rotterdam School of Management, Erasmus Universiteit Rotterdam, The Netherlands.

[46] Dauzère-Pérès, S., and J. Paulli, (1994). Solving the general multi-processor job-shop scheduling problem. *Management Report Series No.182*, Rotterdam School of Management, Erasmus Universiteit Rotterdam, The Netherlands.

[47] Dauzère-Pérès, S., and J. Paulli (1997). An Integrated Approach for Modeling and Solving the Multiprocessor Job-Shop Scheduling Problem Using Tabu Search. *Annals of Operations Research*, 70, 281-306.

[48] Dauzère-Pérès, S., W. Roux and J. B. Lasserre. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107, 289-305.

[49] Dauzère-Pérès, S., and C. Pavageau., (1999). Extensions of an Integrated Approach for Multi-resource Shop Scheduling. *Research Report*, Ecole des Mines de Nantes.

[50] David, R., and Alla, H., (1992). *Petri Nets and Grafcet*. London, U. K.: Prentice-Hall.

[51] Dessouky, M., Dessouky, M., and Verma, S., (1998). Flowshop Scheduling with Identical Jobs and Uniform Parallel Machines. *European Journal of Operational Research*, 109, 620-631.

[52] Drozdowski, M, (1996). Scheduling Multiprocessor Tasks – An Overview. *European Journal of Operational Research*, 94, 215-230.

[53] Esquirol, P., and Lopez, P. (1999). L'ordonnancement. Economica, Paris, France.

[54] Ezpeleta, J., J.M. Colom, and J. Martinez (1995). A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. *IEEE Transactions on Robotics and Automation*, 11, 173-184.

[55] Fanti, M.P., B. Maione, and Turchiano, B., (1996). Digraph-theoretic Approach for Deadlock Detection and Recovery in Flexible Procudtion Systems. *Studies in Informatics and Control*, 5, 4, 373-383.

[56] Fanti, M.P., B. Maione, S. Mascolo, and B. Turchiano (1997). Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems. *IEEE Transactions on Robotics and Automation*, 13, 347-363.

[57] Fanti, M.P., B. Maione, and Turchiano, B., (1998). Deadlock Avoidance in Flexible Production Systems with Multiple Capacity Resources. *Studies in Informatics and Control*, 7, 4, 343-364.

[58] Ferland, J.A., and Fortin, L., (1989). Vehicles Scheduling with Sliding Time Windows. *European Journal of Operational Research*, 38, 213-226.

[59] 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.

[60] French, S., (1982). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Horwood, Chichester.

[61] Garey, M.R., Johnson, M.R., (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman.

[62] 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.

[63] Gendreau, M., Laporte, G., and Potvin, J.Y., (1997). Vehicle Routing: Modern Heuristics. In: *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra editors, John-Wiley publishers.

[64] 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.

[65] 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.

[66] Glover, F., (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8, 1, 156-166.

[67] Glover, F., (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13, 533-549.

[68] 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.

[69] Glover, F., (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1, 3, 190-206.

[70] Glover, F., (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2, 1, 4-32.

[71] Glover, F. (1995). Tabu Search Fundamentals and Uses. *Revised and Expanded, Technical Report*, Graduate School of Business, University of Colorado, Bolder, CO.

[72] Glover, F., E. Taillard and D. de Werra. (1993). A User's Guide to Tabu Search. *Annals of Operations Research*, 41, 3-28.

[73] Glover, F., and Laguna, M., (1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

[74] Goldberg, D.E., (1989). *Genetic Algorithms in Search : Optimisation and Machine Learning*. Addison-Wesley, Reading, Massachusetts.

[75] 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.

[76] Greenwood, N.R., (1988). *Implementing Flexible Manufacturing Systems*. Macmillan, London.

[77] Gupta, J.N.D., (1988). Two-Stage Hybrid Flowshop Scheduling Problems. *Journal of Operational Research Society*, 39, 4, 359-364.

[78] Hall, N.G. and C. Sriskandarajah (1996). A Survey of Machine Scheduling Problems with Blocking and No-Wait In Process. *Operations Research*, 44, 510-525.

[79] Hall, N.G., Kamoun, H., and Sriskandarajah, C., (1997). Scheduling in Robotic Cells: Classification, Two and Three Machine Cells. *Operations Research*, 45, 3, 421-439.

[80] Hallowell, S.F., and Harker, P.T., (1998). Predecting On-time Performance in Scheduled Railroad Operations : Methodology and Application to Train Scheduling. *Transportation Research Part A*, 32, 4, 279-295.

[81] Han, M., and Mcginnis, L. F., (1989). Control of Material Handling Transporter in Automated Manufacturing. *IIE Transactions*, 21, 184-190.

[82] Hansen, P. (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Presented at the *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

[83] Hardgrave, W.H., Nemhauser, G.L., (1963). A geometric model and a graphical algorithm for a sequencing problem. *Operations Research*, 11 889-900.

[84] Holland, J., (1975). *Adaptive in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[85] Hsieh, F.S., and Cheng, S.C. (1994). Dispatching-driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems. *IEEE Transactions on Robotic and Automation*, 10, 2, 196-209.

[86] Hurink, J., B. Jurisch, and Thole, M., (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines, *Operations Research Spektrum*, 15, 205-215.

[87] Hurink, J., and Knust, S., (2001). Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics*, 112, 199-216.

[88] Hurink, J., and Knust, S., (1999). A Tabu Search Algorithm for Scheduling a Single Robot in a Job-Shop Environment. *Research Repport*, University of Osnabrück.

[89] Jackson, J.R., (1956). An Extension of Johnson's Results on Job Lot Scheduling. *Naval Research Logistic Quarterly*, 1, 61-68.

[90] Jain, A. S, (1998). A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem. *P.h.D Thesis*, University of Dundee, Scotland.

[91] Jain, A. S., and S. Meeran, (1999). Deterministic job-shop scheduling: Past, present and future, *European Journal of Operational Research*, 113, 2, 390-434.

[92] Jeng, M.D., S.C. Chen and C.S. Lin (1996). A Search Approach Based on the Petri Nets Theory for FMS Scheduling. *Proc. 13th IFAC World Congress*, San Francisco, vol. B, 55-60.

[93] Jonhson, S.M. (1954). Optimal two- and three-stage Production Scheduling with Setup Times Included. *Naval Research Logistic Quarterly*, 1, 61-68.

[94] Jurisch, B. (1992). Scheduling Jobs in Shops with Multi-purpose Machines. *P.h.D Thesis*, University of Osnabrück, Germany.

[95] Jurisch, B., (1995). Lower bounds for the job-shop scheduling problem on multi-purpose machines. *Discrete Applied Mathematics*, 58, 145-156.

[96] Kamoun, H., and Sriskandarajah, C., (1993). The Complexity of Scheduling Jobs in Repetitive Manufacturing Systems. *European Journal of Operational Research*, 70, 350-364.

[97] Kamoun, H., Hall, N.G., and Sriskandarajah, C., (1999). Scheduling in Robotic Cells : Heuristics and Cell Design. *Operation Research*, 47, 6, 821-835.

[98] Kim, C.W., and Tanchoco, J.M.A., (1991). Conflict-Free Shortest-Time Bidirectional AGV Routing. *International Journal of Production Research*, 29, 12, 2377-2391.

[99] King, R.E., Hodgson, T., and Chafee, F.W., (1993). Robot Task Scheduling in a Flexible Manufacturing Cell. *IIE Transactions*, 25, 80-87.

[100] Kirkpatrick, S., Gellatt, C.D., and Vecchi, M.P., (1983). Optimization by Simulated Annealing. *Science*, 220, 671-680.

[101] Knust, S., (1999). Shop-Scheduling Problems with Transportation. *P.h.D. Thesis*, University of Osnabrück. Germany.

[102] Kouvelis, P., (1992). Design and Planning Problems in Flexible Manufacturing Systems : A Critical Review. *Journal of Intelligent Manufacturing*, 3, 75-99.

[103] Krishnamurthy, N.N., Batta, R., and Karwan, M. H., (1993). Developing Conflict-Free Routes for Automated Guided Vehicles. *Operations Research*, 41, 6, 1177-1190.

[104] Kusiak, A., (1990). *Intelligent Manufacturing Systems*. Prentice Hall, Englewood Cliffs, New Jersey.

[105] Langston, M.A., (1987). Interstage Transportation Planning in the Deterministic Flow-shop Environment. *Operations Research*, 35, 556-564.

[106] Langevin, A., Lauzon, D., and Riopel, D., (1996). Dispatching, Routing, and Scheduling of Two Automated Guided Vehicles in a Flexible Manufacturing System. *The International Journal of Flexible Manufacturing Systems*, 8, 247-262.

[107] 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.

[108] Lawrence, S., (1984). Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques, GSIA, Carnegie-Mellon University, Pittsburgh.

[109] Lee, D.Y., and F. DiCesare, (1994). Scheduling Flexible Manufacturing Systems Using

Petri Nets and Heuristic Search. *IEEE Transactions on Robotic and Automation*, 10, 123-132.

[110] Linn, R., and Zhang, W., (1999). Hybrid Flow Shop Scheduling: A Survey. *Computers and Industrial Engineering*, 37,1-2, 57-61.

[111] Logendran, R., and Sriskandarjah, C., (1996). Sequencing of Robot Activities and Parts in Two Machines Robotic Cells. *International Journal of Production Research*, 34, 12, 3447-3463.

[112] Lopez, P., Roubellat, F., (2001). *Ordonnancement de la Production*. Hermes Science, France.

[113] Mastrolilli; M., and L.M. Gambardella, (2000). Effective Neighborhood Functions for the Flexible Job Shop Problem. *Journal of Scheduling*, 3, 1, 3-20.

[114] Mati, Y., N. Rezg and X.L. Xie, (2000a). A Shortest Path Approach for Deadlock-free Scheduling of Automated Manufacturing Systems. *Proceeding of MCPL*, Grenoble, France.

[115] Mati, Y., Rezg, N., Xie, X.L., (2000b). Geometric approach and taboo search for scheduling flexible manufacturing systems. *Rapport de recherche* RR-4137, INRIA, France.

[116] Mati, Y., (2001a). On the Complexity of the two-job shop Problems with Multi-purpose Unrelated Machines. *Operational Research Peripatetic Post-Graduate Programmme, ORP3*, Paris, France.

[117] Mati, Y., Rezg, N., Xie, X.L., (2001b). 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.

[118] Mati, Y., Rezg, N., Xie, X.L., (2001c). Geometric approach and taboo search for scheduling flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 17, 6, 805-818.

[119] Mati, Y., Rezg, N., Xie, X.L., (2001d). A Heuristic Approach for Scheduling FMS with Automated Guided Vehicles. *International Conference on Industrial Engineering and Production Management* IEPM, Quebec, Canada.

[120] Mati, Y., Rezg, N., Xie, X.L., (2001e). An integrated Greedy Heuristic for a Flexible Job Shop Scheduling Problem. *IEEE International Conference on Systems, Man, and Cybernetics*, Tucson, Arizona, USA.

[121] Mati, Y., Rezg, N., Xie, X.L., (2001f). Job Shop Scheduling Problem with Blocking : A Taboo Search Approach. *4th Metaheuristics International Conference*, Porto, Portugal.

[122] Mati, Y., and X. Xie, (2001g). The complexity of the two-job shop scheduling problems with multi-purpose unrelated machines. Soumis à *European Journal of Operational Research.*

[123] Mati, Y., Rezg, N., Xie, X.L., (2002a). Greedy Heuristic and Genetic Algorithms for the Multi-resource Shop Scheduling with Resource Flexibility. *Project Management and Scheduling*, Valence, Spain.

[124] Mati, Y., Rezg, N., Xie, X.L., (2002b). An Integrated Greedy Heuristic Guided by Genetic Algorithms for the Multi-resource Shop Scheduling with resource Flexibility. *En préparation.*

[125] Mati, Y., Rezg, N., Xie, X.L., (2002c). Scheduling Flexible Manufacturing System with Blocking and Resource Flexibility. *En préparation.*

[126] Mati, Y., Rezg, N., Xie, X.L., (2002d). A Heuristic Algorithm Based on the Geometric Approach for Scheduling and Deadlock Avoidance in FMSs. *En préparation.*

[127] McMaron, G.B., and Florian, M., (1975). On Scheduling with Ready Time and Due Dates to Minimize Maximum Lateness. *Operations Research*, 23, 3, 475-482.

[128] Moursli, O., and Pochet, Y., (2000). A Branch-and-bound Algorithm for the Hybrid Flowshop. *International Journal of Production Economics*, 64, 1-3, 113-125.

[129] Murata, T., (1989). Petri Nets : Properties, Analysis and Applications. *Proceedings of IEEE*, 77, 4, 541-580.

[130] Negenman, E.G., (2001). Local Search Algorithms for the Multiprocessor Flow Shop Scheduling Problem. *European Journal of Operational Research*, 128, 147-158.

[131] 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.

[132] Nowicki, E., and Smutnicki, C., (1996). A Fast Taboo Search Algorithm for the Job-Shop Problem. *Management Science*, 42, 6, 797-813.

[133] Nowicki, E., and Smutnicki, C., (1998). The Flow Shop with Parallel Machines: A Tabu Search Approach. *European Journal of Operational Research*, 106, 226-253.

[134] Nowicki, E., (1999). The Permutation Flow Shop with Buffers: A Tabu Search Approach. *European Journal of Operational Research*, 116, 205-219.

[135] 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.

[136] Osman, I. H. and G. Laporte (1996). Metaheuristics: A Bibliography. *Annals of Operations Research*, 63, 513-623.

[137] Pacciarelli, D., and Pranzo, M., (2001). A Tabu Search Algorithm for the Railway Scheduling Problem. *Proceeding of MIC'2001 Conference*, 159-163.

[138] Panwalker, S.S., and Iskander, W., (1977). A Survey of Scheduling Rules. *Operations Research*, 25, 1, 45-61.

[139] 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.

[140] Paulli, J., (1995). A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research*, 86, 32-42.

[141] Pearl, J., (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[142] Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, Etats-Unis.

[143] Proth, J.M. and X.L. Xie (1996). *Petri Nets: A Tool for Design and Management of Manufacturing Systems*. John Wiley & Sons.

[144] Raman, N., Talbot, F.B., and Rachamadgu, R.V., (1986). Simultaneous Scheduling of

Machines and Material Handling Devices in Automated Manufacturing. In *Proceeding of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems*, K.E. Stecke and R. Suri, eds. Elsevier, Amsterdam, 455-466.

[145] Ramaswamy, S.E. and S.B. Joshi (1996). Deadlock-free Schedules for Automated Manufacturing Workstations. *IEEE Transactions on Robotics and Automation*, 12, 391-400.

[146] Reveliotis, S., Lawley, M., and Ferreira, P., (1997). Polynomial Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems. *IEEE Transactions on Automatic Control*, 42, 1344-1357.

[147] Rinnooy Kan, A.H.G., (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Haque.

[148] Rochat, Y., and Taillard, E.D., (1995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1, 147-167.

[149] Roy, B., and Susmann, B., (1964). Les Problèmes d'ordonnancement avec Contraintes Disjonctives. *Note DS n° 9 bis*, SEMA, Montrouge.

[150] Sabuncuoglo, I., and Hommertzheim, D.L., (1992a). Experimental Investigation of FMS Machine and AGV Scheduling of FMS Machine and AGV Scheduling Rules Against the Mean Flow-time Criterion. *International Journal of Production Research*, 30, 1617-1635.

[151] Sabuncuoglo, I., and Hommertzheim, D.L., (1992b). Dynamic Dispatching Algorithm for Scheduling Machines and Automated Guided Vehicles in a Flexible Manufacturing System. *International Journal of Production Research*, 30, 1059-1079.

[152] Sethi, S.P., C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak (1992). Sequencing of Parts and Robot Moves in a Robotic Cell. *International Journal of Flexible Manufacturing Systems*, 4, 331-358.

[153] Sotskov, Y.N., (1985). Optimal servicing two jobs with a regular criterion, In: *Automation of Designing Processes*, Minsk, 86-95 (in Russian).

[154] Sotskov, Y.N., (1991). The complexity of shop-scheduling problems with two or three jobs, *European Journal of Operational Research*, 53, 326-336.

[155] Stecke, K.E., (1985). Design, Planning, Scheduling, and Control Problems of Flexible Manufacturing Systems. *Annals of Operations Research*, 3, 3-12.

[156] Sun, T-H., C-W. Cheng, and L-C. Fu (1994). A Petri Net Based Approach to Modeling and Scheduling for an FMS and a Case Study. *IEEE Transactions on Industrial Electronics*, Vol. 41, No. 6, pp. 593-601.

[157] Szwarc, W., (1960), Solution of the Akers-Friedman scheduling problem. *Operations Research*, 8, 6, 782-788.

[158] Taillard, E. D., (1990). Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, 47, 1, 65-74.

[159] Taillard, E. D., (1990). Robust Taboo Search for the Quadratic Assignment problem. *Parallel Computing*, 17, 443-455.

[160] 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.

[161] Ulusoy, G., F. Sivrikaya-Serifuglu, and Ü. Bilge (1997). A Genetic Algorithm Approach to Simultaneous Scheduling of Machines and Automated Guided Vehicles. *Computers & Operations Research*, Vol. 24, No. 4, pp. 335-351.

[162] 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.

[163] Viswanadham, N., Narahari, Y., and Johnson, T.L., (1990). Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models. *IEEE Transactions on Automatic and Control*, 6, 6, 713-723.

[164] Wysk, R.A., Yang, N.S., and Joshi, S. (1991). Detection of Deadlocks in Flexible Manufacturing Cells. *IEEE Transactions on Automatic and Control*, 7, 6, 853-859.

[165] Xing, K.Y., Hu, B.S., and Chen, H.X., (1996). Deadlock Avoidance Policy for Petri Net Modeling of Flexible Manufacturing Systems with Shared Resources. *IEEE Transactions on Automatic and Control*, 41, 2, 289-295.

[166] Xiong, H., and Zhou M.C., (1998). Scheduling of Semiconductor Test Facility Perti Nets and Hybrid Heuristic Search. *IEEE Transactions on Semiconductor Manufacturing*, 11, 3, 384-393.

UFR MIM
Mathématique Informatique Mécanique Automatique

DOCTORAT
DE L'UNIVERSITE DE METZ

Thèse soutenue le 12 juin 2002
par Monsieur Yazid MATI

Signatures des membres du jury :

- M. CARLIER Jacques
- M. CHAUVET Fabrice
- M. COMPAGNE Jean-Pierre
- M. DAUZERE-PERES Stéphane
- M. FREIN Yannick
- M. GENTINA Jean-Claude
- M. PROUST Christian
- M. REZG Nidhal
- M. XIE Xiaolan

RAPPORT APRES SOUTENANCE

sur la thèse ayant pour sujet :

« Les problèmes d'ordonnancement dans les systèmes de production automatisés : modèles, complexités et approches de résolution. »

MENTION OBTENUE :

Mention très honorable avec félicitations.

UNIVERSITE DE METZ

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Nom et Prénom de l'auteur : M. Yazid MATI

Titre de la thèse : « Les problèmes d'ordonnancement dans les systèmes de production automatisés : modèles, complexités et approches de résolution. »

Date de soutenance : le 12 juin 2002

Président du jury : J. C GENTINA

Membres du jury : M.M. CARLIER, CHAUVET, COMPAGNE, DAUZERE-PERES , FREIN, GENTINA, PROUST, REZG et XIE

Reproduction de la thèse soutenue :

☒ **thèse pouvant être reproduite en l'état**

☐ **thèse pouvant être reproduite après corrections suggérées au cours de la soutenance**

☐ **thèse interdite de reproduction**

Le Directeur de thèse mandaté par le Président du Jury atteste que les corrections ont bien été effectuées
le : 12/06/2002
Signature :

Le Président du Jury :

UFR MIM Mathématique Informatique Mécanique Automatique

RAPPORT APRES-SOUTENANCE (suite)

Rapport sur la soutenance de thèse de Yazid MATI

Le Jury a très vivement apprécié la présentation très claire ainsi que la synthèse aboutie d'un travail d'une très grande densité. M. Mati, en allant à l'essentiel, a fait preuve de très grandes qualités pédagogiques.

Au cours de la discussion qui a suivi la présentation, M. Mati a pu témoigner de sa grande maîtrise du sujet ainsi que d'une parfaite connaissance de l'état de l'art, se référant de manière appropriée et honnête aux travaux les plus importants dans son domaine de compétence. Les réponses aux multiples questions du jury ont été claires et toujours pertinentes. Sachant prendre du recul sur son travail, il a su mettre en exergue les apports essentiels de son travail de thèse.

Le jury tient unanimement à signaler la forte originalité et l'intérêt du travail de M. Yazid Mati notamment sa contribution à une approche intégrée de l'ordonnancement des systèmes de productions flexibles tant sur le plan théorique que pratique. Il est à signaler que trois revues internationales de très bon rang confirment l'originalité et la qualité de la recherche de M. Mati.

Pour toutes ces raisons, le jury accorde à l'unanimité à M. Yazid Mati le grade de Docteur en Automatique de l'Université de Metz, et lui attribue la mention très honorable avec félicitations.

J.C GENTINA J. CARLIER

N. REZG

X. XIE

U. FILEIN

C. PROUST

S.P. CAMPAGNE

T. CHAUVET

S. DAUZERE-PERES

Résumé : Les Problèmes d'Ordonnancement dans les systèmes de production Automatisé : Modèles, Complexité et Approches de Résolution.

Les travaux de recherche proposés dans cette thèse portent essentiellement sur les problèmes d'ordonnancement rencontrés dans les systèmes de production automatisés et tiennent compte de nouvelles contraintes, notamment la considération des moyens de transport, des stocks tampons de capacité limitée, de la flexibilité de ressources, etc. Trois objectifs principaux sont fixés, à savoir la proposition de nouveaux modèles d'ordonnancement, l'étude de la complexité de ces problèmes et enfin le développement d'approches intégrées pour l'ordonnancement et le contrôle des systèmes étudiés. Les deux premiers objectifs sont considérés dans une optique théorique alors que le troisième a un but pratique. Plus précisément, nous développons des modèles d'ordonnancement permettant d'incorporer les différentes caractéristiques des systèmes de production automatisés. De nouveaux résultats de complexité sont présentés et des algorithmes polynomiaux sont proposés dans plusieurs cas particuliers. En outre, des algorithmes originaux pour l'ordonnancement et le contrôle des systèmes de production étudiés sont développés. Les résultats d'expérimentations réalisées sur des instances issues de la littérature montrent l'efficacité des différents algorithmes proposés.

Mots clefs : Ordonnancement, systèmes de production automatisés, stockage limité, ressources multiples, blocage, flexibilité, analyse de complexité, algorithme polynomial.

Abstract : Scheduling Problems in Automated Manufacturing Systems : Models, Complexity and Solution Approaches.

The research presented in this thesis focuses on the scheduling problem encountered in automated manufacturing systems, and takes into account the new constraints such as transportation resources, automated storage with finite capacity, resource flexibility, etc. Three main objectives are fixed, namely the proposition of new scheduling models, the study of complexity analysis and finally the development of integrated approaches for scheduling and deadlock avoidance of studied systems. The two first objectives are considered from the theoretical point of view, whereas the third has a practical goal. More precisely, we propose several scheduling models that incorporate the various characteristics of automated manufacturing systems. New results of complexity are presented and polynomial algorithms are proposed, for particular cases of our models. Also, integrated approaches for scheduling and deadlock avoidance are developed. Computational results realized on problems from the literature show the efficiency of the proposed approaches.

Keywords : Scheduling, automated manufacturing systems, limited storage, multiple resources, deadlock, flexibility, complexity analysis, polynomial algorithm.

Thèse préparée à l'INRIA Lorraine dans le projet MACSI

Imprimé à l'INRIA