Résolution exacte de problèmes d'ordonnancement de type flowshop de permutation en présence de contraintes d'écarts temporels entre opérations

Julien Fondrevelle

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Résolution exacte de problèmes d'ordonnancement de type *flowshop* de permutation en présence de contraintes d'écarts temporels entre opérations

THÈSE

présentée et soutenue publiquement le 10 novembre 2005

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Julien Fondrevelle

Composition du jury

Rapporteurs :	Gerd Finke	Professeur à l'Université Joseph Fourier de Grenoble
	Alain Guinet	Professeur à l'INSA de Lyon
Examinateurs :	Federico Della Croce	Professeur à l'Ecole Polytechnique de Turin
	Jean-Yves Marion	Professeur à l'INPL
Directrice de Thèse :	Marie-Claude Portmann	Professeur à l'INPL
Co-directeur de Thèse :	Ammar Oulamara	Maître de Conférences à l'INPL

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Avant toute chose, je tiens à avertir le lecteur sur le point de s'aventurer à lire cette page, que le contenu de celle-ci peut s'apparenter à un long discours de lauréat, après une remise de prix. Elle risque donc de se révéler totalement ennuyeuse pour celui qui ne se sentirait pas directement concerné par ce travail. Bien qu'ils puissent être considérés comme un simple étalage de bons sentiments, tous les remerciements qui suivent n'en restent pas moins absolument sincères.

Je souhaiterais associer au travail présenté dans ce mémoire un certain nombre de personnes, qui, chacune à son niveau, ont contribué à la réussite de celui-ci. Je pense en tout premier lieu à Marie-Claude Portmann, sans qui cette thèse n'aurait pu voir le jour. C'est elle qui m'a fait découvrir le monde passionnant de la recherche opérationnelle et de l'ordonnancement en particulier. Malgré un emploi du temps plus que chargé, elle a toujours su se rendre disponible, que ce soit pour apporter un regard critique sur notre travail, pour proposer de nouvelles pistes de recherche intéressantes, ou pour m'orienter dans mes choix professionnels. Je voudrais aussi saluer ici sa bonne humeur permanente et l'excellente ambiance qu'elle a réussi à créer au sein de l'équipe MACSI Nancy. Je remercie également Ammar Oulamara, pour sa présence quotidienne, ses conseils et sa décontraction. Son association avec Marie-Claude pour l'encadrement de cette thèse s'est révélée parfaite.

Je tiens à exprimer ma reconnaissance et mon profond respect à Monsieur Gerd Finke, Professeur à l'Université Joseph Fourier de Grenoble, et à Monsieur Alain Guinet, Professeur à l'INSA de Lyon, qui ont accepté, en tant que rapporteurs, d'évaluer notre travail. J'espère qu'ils auront eu autant de plaisir à lire ce mémoire que j'en ai eu à mener cette étude. Je remercie également Monsieur Federico Della Croce, Professeur à l'Ecole Polytechnique de Turin, qui après avoir bien voulu jouer les interprètes lors d'une conférence à Sienne, a accepté de participer à ce jury. Enfin, je remercie Monsieur Jean-Yves Marion, Professeur à l'Ecole des Mines de Nancy, de s'être intéressé à ce travail, même s'il ne s'inscrit pas exactement dans son domaine de recherche.

J'ai été amené lors de ces trois années de thèse à cotoyer un grand nombre de personnes, que ce soit pour mes travaux de recherche, dans le cadre d'enseignements que j'ai menés, ou tout simplement pour régler des détails administratifs. Je voudrais ainsi remercier l'ensemble de mes "co-équipiers" au sein de MACSI, notamment Mohammed et Asma pour leur gentillesse, Freddy, dont le travail et l'expérience m'ont beaucoup apporté, ainsi que Wahiba, Zerouk et Aimé. D'une manière générale, je souhaiterais saluer la communauté "ordonnancement" française, qui fait preuve d'un grand dynamisme (notamment à travers les groupes GOThA et Bermudes), au sein de laquelle j'ai toujours pris plaisir à évoluer, notamment lors des conférences auxquelles j'ai assisté. Je remercie en particulier Ameur et les Tourangeaux pour leur bonne humeur, qu'ils m'ont fait partager à Marrakhech, ainsi qu'Anthony Caumond, pour les discussions que nous avons pu avoir au sujet des time lags.

En ce qui concerne les cours auxquels j'ai participé, je voudrais exprimer ma reconnaissance à Henri Amet, qui m'a permis de passer de l'autre côté du miroir, en me permettant de diriger des TD de recherche opérationnelle aux Mines, quelques années après les avoir suivis en tant qu'élève. Merci à Martine Cadot de m'avoir fait confiance pour de nombreux enseignements à l'Université Henri Poincaré, et à Christian Zanne et Barthélémy Zoz, pour m'avoir confié des TD et TP à l'ENSGSI, ce qui m'a permis de me confronter à une autre pédagogie. J'ai pris énormément de plaisir à participer à l'ensemble de ces cours.

Je ne voudrais pas oublier d'associer à ces remerciements le personnel administratif des différents établissements au sein desquels j'ai travaillé (Loria-INRIA, CNRS, INPL, UHP), en particulier Françoise Laurent à l'Ecole des Mines, Christel Wiemert à l'INRIA, Nadine Beurné au Loria et Isabelle Prévot au CNRS, pour l'aide qu'elles m'ont apportée tout au long de ces trois ans. Merci également au service de documentation du Loria que j'ai souvent sollicité, pour me procurer une grande partie des références qui figurent dans ce mémoire. J'aimerais aussi saluer l'ensemble de mes nouveaux collègues à l'UTBM, avec qui je démarre une nouvelle aventure, notamment Alexandre Caminada, qui m'a permis de terminer cette thèse dans de très bonnes conditions. Enfin, je voudrais terminer cette série de remerciements en exprimant toute ma gratitude à Madame Hélène Kirchner, directrice du Loria et de l'INRIA Lorraine, ainsi que Monsieur Louis Schuffenecker, actuellement Président de l'INPL, qui, lorsqu'il était encore Directeur des Etudes à l'Ecole des Mines, m'a encouragé à m'engager dans la voie de la recherche, en me permettant de suivre le DEA GSI parallèlement à ma troisième année aux Mines.

Durant ces trois années de thèse, j'ai toujours pu m'appuyer sur le soutien de ma famille et de mes proches. Je tenais à leur témoigner toute mon affection. Je voudrais tout d'abord remercier mes parents de m'avoir accompagné dans tous mes choix. De chacun d'eux, je pense avoir hérité d'une qualité importante : la curiosité scientifique de mon père, et l'amour de l'enseignement de ma mère. Je remercie également ma soeur, mes beaux-parents et mon beau-frère, pour leur bonne humeur et leur profonde gentillesse. J'ai bien entendu gardé le meilleur pour la fin : je ne saurais exprimer suffisamment de reconnaissance pour celle qui est devenue depuis peu mon épouse, Sandrine, et qui a toujours été là pour moi, que ce soit pour me réconforter et me changer les idées, pour guider mes choix, mais également pour jouer le chairman ou le rapporteur afin de corriger mes exposés ou mes articles. Merci pour tout.

Table des matières

Introduction générale

Dans un contexte économique hautement concurrentiel, où il faut produire au moindre coût tout en respectant les contraintes techniques permettant d'assurer la qualité attendue des produits, et tout en livrant les clients dans les délais, les problèmes d'ordonnancement prennent une importance de plus en plus grande. On les retrouve notamment dans les outils d'aide à la décision incorporés dans les progiciels de gestion de production vendus souvent conjointement avec les ERP (*Enterprise Resource Planning*). En conséquence, le nombre de thèses en ordonnancement, dont beaucoup sont financées en France dans le cadre de bourses CIFRE (Convention Industrielle de Formation par la REcherche) ou de contrats industriels, s'est accru de manière importante ces dernières années. Ces travaux de recherche intègrent des contraintes liées à des applications industrielles de plus en plus variées.

Le présent mémoire s'inscrit dans ce cadre. Il est le résultat des travaux de recherche que j'ai menés lors de ma thèse, sous la direction de Marie-Claude Portmann et d'Ammar Oulamara, au sein de l'équipe MACSI (Modélisation, Analyse et Conduite des Systèmes Industriels) du Loria - INRIA Lorraine. Cette thèse, qui s'est déroulée dans le cadre d'une Bourse de Docteur Ingénieur du CNRS, co-financée par la Région Lorraine, a fait suite au projet de recherche de DEA, que j'avais également effectué dans l'équipe MACSI, et qui portait sur l'ordonnancement de produits périssables.

Un des principaux thèmes de recherche étudiés par l'équipe MACSI concerne l'optimisation et l'aide à la décision pour la gestion des systèmes de production, et plus particulièrement l'ordonnancement de la production, domaine dans lequel plusieurs thèses ont été soutenues récemment. Les activités menées dans le cadre d'une de ces thèses, débutée en 2000 sous la forme d'un contrat CIFRE dans une entreprise d'édition de progiciels pour la gestion de production, ont fait émerger l'importance des contraintes d'écarts temporels dans plusieurs secteurs industriels : pharmaceutique, agro-alimentaire ou encore sidérurgique. Or, ce type de contraintes, qui régissent les durées séparant l'exécution de différentes opérations, était jusque là passé sous silence par la plupart des travaux dans la littérature. Comme nous l'expliquons dans le premier chapitre, nous utilisons pour désigner ces écarts temporels à respecter le terme *time lags*, bien qu'il s'agisse d'un mot anglais, étant donné que c'est le terme que l'on retrouve le plus dans la littérature, y compris francophone.

L'équipe MACSI a donc orienté une partie de ses activités de recherche vers l'étude des problèmes d'ordonnancement avec prise en compte des contraintes de time lags, dans un premier temps par l'intermédiaire de la thèse CIFRE menée en entreprise [Deppner,04]. Celle-ci a conduit à la mise au point d'une approche de résolution adaptée aux problèmes industriels rencontrés dans la pratique, que ce soit au niveau de la taille des données à traiter ou au niveau du nombre et de la variété de contraintes incorporées dans le modèle. Ces problèmes étant par nature NP-

difficiles et extrêmement complexes, seules des méthodes approchées peuvent permettre de les traiter dans un temps de calcul raisonnable. Afin d'approfondir l'analyse des problèmes d'ordonnancement avec contraintes de time lags, l'équipe MACSI a décidé de développer parallèlement aux travaux précédents une autre approche, complémentaire, qui se consacre à la résolution exacte de problèmes, quitte à ce que les modèles considérés soient plus simples que ceux traités dans la thèse CIFRE. C'est dans ce contexte que j'ai commencé à travailler sur les problèmes d'ordonnancement, lors de mon stage de DEA dans un premier temps, puis lors de ma thèse.

Le premier chapitre de ce mémoire décrit le problème étudié. Après avoir rappelé brièvement les définitions et concepts de base de l'ordonnancement d'atelier, nous expliquons en quoi il est intéressant d'étudier les problèmes impliquant des time lags. Nous présentons le modèle que nous avons utilisé et citons plusieurs applications industrielles, issues de domaines très variés.

Le second chapitre concerne la complexité des problèmes d'ordonnancement de type flowshop, en présence de time lags. Nous rappelons les principaux résultats tirés de la littérature. Nous présentons également plusieurs résultats que nous avons obtenus et qui permettent de mieux caractériser ces problèmes : des algorithmes polynomiaux pour un certain nombre de problèmes de flowshop de permutation avec durées d'exécution identiques et time lags minimaux uniquement, un résultat de NP-complétude au sens fort pour le problème de flowshop à deux machines avec time lags maximaux identiques, des propriétés de dominance, ainsi qu'une analyse de performance d'heuristiques. De plus, nous introduisons un critère particulier, très peu abordé dans la littérature, qui traduit l'utilisation des machines. Nous déduisons de résultats classiques des preuves pour la complexité des problèmes de flowshop avec cette fonction objectif. Enfin, nous terminons le deuxième chapitre avec l'étude d'un problème intégrant une contrainte additionnelle : la présence d'une machine à traitement par fournées, de type parallèle. Après avoir rappelé ce qu'implique cette nouvelle contrainte, nous présentons quelques propriétés et résultats de complexité, pour des problèmes de flowshop à deux machines combinant traitement par fournées et time lags minimaux.

Dans le troisième chapitre, nous présentons un état de l'art sur les approches de résolution proposées dans la littérature pour traiter des problèmes d'ordonnancement avec contraintes de time lags. Selon la structure du problème considéré, les références sont plus ou moins nombreuses et les techniques de résolution plus ou moins sophistiquées. Nous nous intéressons plus particulièrement aux problèmes d'ordonnancement d'atelier, pour lesquels nous tentons d'être le plus complet possible, mais nous considérons également l'ordonnancement de projet. Pour des raisons de clarté, nous avons choisi de séparer cet état de l'art de nos propres méthodes de résolution, qui font l'objet du quatrième chapitre, alors qu'au niveau de la complexité, nous avons regroupé les résultats issus de la littérature avec ceux que nous avons démontrés.

L'approche de résolution exacte que nous avons mise en place pour le flowshop de permutation est exposée dans le quatrième chapitre. Nous présentons dans un premier temps les éléments génériques susceptibles d'être appliqués à de nombreux problèmes de ce type, en présence de time lags. Nous analysons notamment le problème restreint qui consiste à déterminer le meilleur ordonnancement respectant une séquence donnée. L'existence d'un algorithme polynomial permettant de traiter ce problème nous conduit à adopter un schéma de séparation classique pour le flowshop de permutation, proposé par Ignall et Schrage [Ignall et Schrage,65]. Celui-ci consiste à construire progressivement la permutation des travaux par ajout en fin de séquence partielle. Nous considérons ensuite successivement plusieurs problèmes. Nous nous penchons tout parti-

culièrement sur la minimisation du makespan dans un flowshop de permutation à m machines avec time lags minimaux et maximaux. Nous examinons également la minimisation de la somme pondérée des dates de fin sur les machines pour le même type de problème, la minimisation du plus grand retard sur deux machines avec traitement sans attente et temps de montage et de démontage séparés, et enfin la minimisation du plus grand retard pour un flowshop de permutation avec time lags exacts. Pour chaque problème, nous décrivons les techniques spécifiques que nous avons adoptées pour sa résolution, en particulier les bornes inférieures et les règles de dominance éventuelles. Nous expliquons également comment nous avons généré des jeux d'essais pour ces problèmes qui n'ont pas encore été traités de manière exacte dans la littérature, en nous demandant quelles peuvent être les principales caractéristiques des données numériques pour des applications industrielles, tout en cherchant à couvrir le plus de familles d'instances possibles. A partir des résultats obtenus sur ces jeux d'essais, nous comparons nos méthodes exactes avec des heuristiques que nous avons aussi développées, en nous limitant à des méthodes itératives simples, moins sophistiquées que des métaheuristiques, mais plus élaborées que des règles de priorité.

Le mémoire se termine par un chapitre de conclusion, dans lequel nous proposons un bilan de notre étude et quelques perspectives, dont nous avons commencé à analyser la faisabilité, pour la poursuite de ces travaux.

Table des figures

Chapitre 1

Le problème et ses applications industrielles

Résumé

Ce premier chapitre présente la problématique située au coeur de notre travail. Nous rappelons dans un premier temps quelques concepts fondamentaux en ordonnancement d'atelier. Dans un second temps, nous introduisons le concept de contraintes d'écarts temporels et nous mettons en évidence l'intérêt d'un tel concept dans le cadre des problèmes d'ordonnancement. Puis nous décrivons les modèles considérés dans cette thèse et présentons de nombreuses applications industrielles.

Sommaire

1.1 Introduction

Nous souhaitons que le présent mémoire puisse être parfaitement compris par tout spécialiste d'optimisation combinatoire ou de recherche opérationnelle sans qu'il soit nécessairement spécialiste en ordonnancement. C'est pourquoi nous avons estimé nécessaire de présenter rapidement dans ce chapitre les définitions et les concepts principaux liés à l'environnement de notre travail. Par ailleurs, comme il existe de nombreux ouvrages (récents) en ordonnancement, soit très pédagogiques (par exemple [Lopez et Roubellat,01], [GOThA,04]), soit destinés aux chercheurs de haut niveau (comme [Tanaev et al.,94], [Blazewicz et al.,01], [Pinedo,02]), nous n'avons pas jugé utile d'être exhaustif (une liste régulièrement mise à jour et référençant les nombreux ouvrages sur le sujet est disponible sur la page du Groupe de recherche en Ordonnancement Théorique et Appliqué à l'adresse [GOThA]). Nous avons donc décidé de faire ici une très brève introduction, dans laquelle nous rappelons simplement les notions de base utiles à la compréhension de ce mémoire, en commençant par la présentation des problèmes d'ordonnancement d'atelier.

1.2 Généralités sur les problèmes d'ordonnancement d'atelier

1.2.1 Les différentes structures d'atelier

Un problème d'ordonnancement consiste à "programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution" (Carlier et Chrétienne [Carlier et Chretienne,88]). Derrière cette définition générale se cache une multitude de problèmes, dont les modèles, les problématiques et les approches de résolution peuvent varier considérablement. Dans le cadre de notre étude, nous nous intéressons plus particulièrement aux problèmes d'ordonnancement d'atelier. Les tâches sont alors généralement désignées sous le nom d'opérations et regroupées en travaux (ou *jobs*) et les ressources sont appelées machines. Selon la structure de l'atelier, on distingue généralement les situations suivantes :

- les problèmes à une machine : dans ce cas, chaque travail est constitué d'une seule opération qui doit être effectuée sur la machine considérée
- les problèmes à machines parallèles : chaque travail est également constitué d'une seule opération mais celle-ci peut être effectuée par plusieurs machines. En plus du problème de séquencement des opérations, la question de l'affectation des opérations aux machines se pose donc. De plus, les machines peuvent être identiques ou non. Dans ce dernier cas, l'ensemble des opérations qu'elles peuvent traiter, ainsi que les durées de traitement, peuvent varier d'une machine à l'autre
- les problèmes de *flowshop* : la gamme de fabrication, qui correspond à l'ordre d'exécution des opérations au sein des travaux, est linéaire, fixée à l'avance et identique pour tous les travaux, ce qui permet de numéroter les machines dans l'ordre d'exécution des opérations à traiter
- les problèmes de *jobshop* : la gamme de fabrication est linéaire et fixée à l'avance, mais peut varier d'un travail à l'autre
- les problèmes d'*open shop* : la gamme de fabrication n'est pas fixée à l'avance

Bien que cette thèse soit rédigée en français, nous employons les termes anglais de *flowshop*, *jobshop* et *open shop* pour désigner les structures d'atelier que nous venons de décrire, étant donné qu'il n'existe pas d'équivalents français.

Ces modèles sont les plus classiquement rencontrés dans la littérature, mais de nombreux problèmes ne rentrent pas dans ce cadre (problèmes multi-machines, problèmes avec des serveurs...). MacCarthy et Liu [MacCarthy et Liu,93] ont proposé une hiérarchie pour les problèmes d'ordonnancement d'atelier de base (figure 1.1).

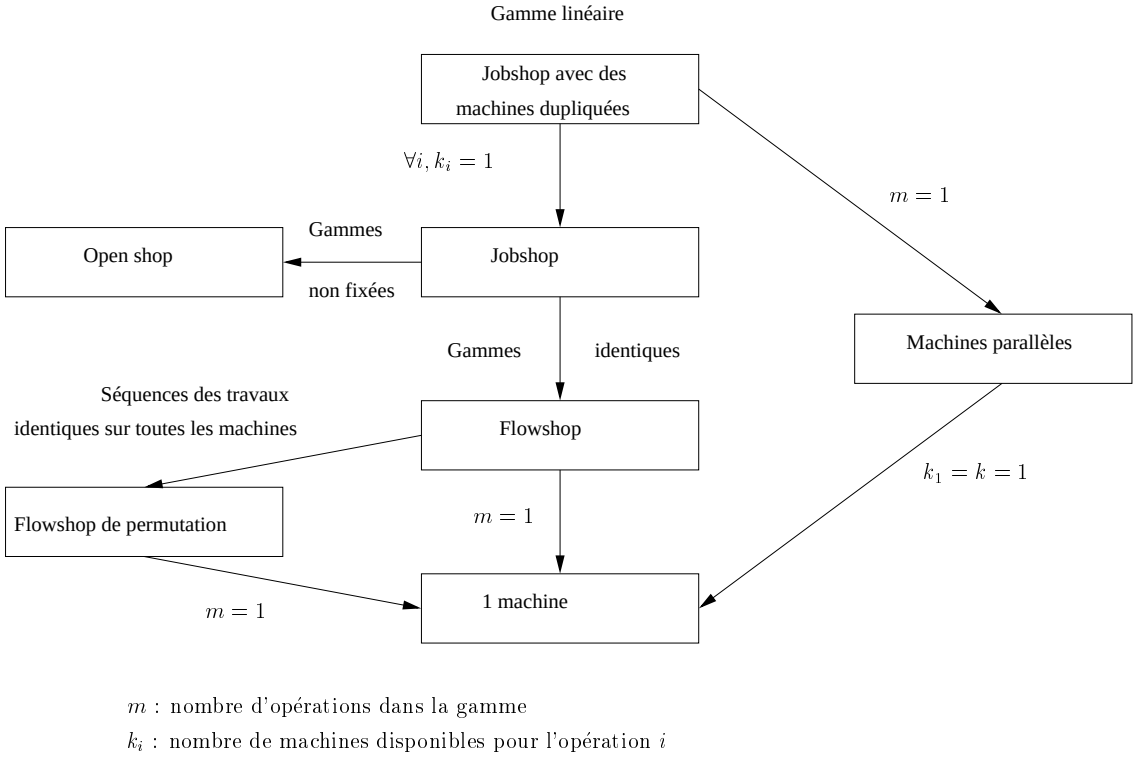

FIG. 1.1 – Hiérarchie des problèmes d'ordonnancement [MacCarthy et Liu,93]

1.2.2 Critères considérés

Outre la structure de l'atelier, les problèmes d'ordonnancement sont également caractérisés par le ou les critères que l'on cherche à optimiser. Ces critères s'expriment généralement en termes de durée, de retard, ou de coût. Les plus classiquement étudiés sont des fonctions des dates de fin des travaux C_j, que l'on doit minimiser. On est alors ramené à des problèmes d'optimisation de type min-max ou min-somme. On distingue notamment :

- la date de fin de l'ordonnancement (makespan) définie par $C_{max} = max\{C_j / 1 \leq j \leq n\}$
- la somme des dates de fin des travaux $\sum C_j = \sum_{1 \leq j \leq n} C_j$
- le plus grand retard $L_{max} = max\{C_j - d_j / 1 \leq j \leq n\}$, où d_j désigne la date de fin souhaitée pour le travail j
- la somme des retards $\sum T_j = \sum_{1 \leq j \leq n} max(0, C_j - d_j)$
- le nombre de travaux en retard $\sum U_j = \sum_{1 \leq j \leq n} U_j$ avec $U_j = 1$ si $C_j > d_j$ (travail j en retard) et $U_j = 0$ sinon

Pour les critères sous forme de somme, il est possible d'attribuer à chaque travail un poids noté w_j, ce qui permet de représenter l'importance relative des travaux les uns par rapport aux autres. L'objectif est alors de minimiser la somme pondérée.

Tous les critères que nous venons de mentionner s'expriment comme des fonctions $f(C_1, C_2, \ldots, C_n)$ des dates de fin des travaux C_j, croissantes selon chaque composante, les autres restant fixes. On dit qu'ils sont réguliers. Cette propriété joue un rôle central puisqu'elle permet de limiter la recherche d'une solution optimale aux ordonnancements ayant une structure particulière. Nous revenons sur cette notion, ainsi que sur les définitions d'ordonnancement sans délai, semi-actif et actif, dans la section 4.3.1.

1.2.3 Contraintes prises en compte

Le dernier ensemble de caractéristiques permettant de décrire un problème d'ordonnancement est constitué par les contraintes à respecter. Celles-ci peuvent être de nature très différentes et portent aussi bien sur les précédences entre travaux que sur les indisponibilités éventuelles des machines ou encore sur l'existence de temps de réglage des machines avant le traitement d'un travail. Parmi ces contraintes, on retrouve assez fréquemment des dates de disponibilité pour les travaux, qui imposent à chaque travail j de ne pas débuter avant une certaine date r_j, et des durées de latence q_j qui viennent s'ajouter à la date de fin sur la dernière machine. Dans cette thèse, nous nous sommes plus particulièrement intéressés à des contraintes concernant la succession temporelle des opérations, que nous présentons en détail dans les sections suivantes.

Ainsi, nous avons vu comment définir un problème d'ordonnancement, notamment dans le cas des problèmes d'ordonnancement d'atelier qui nous intéressent plus spécialement. Une notation a été introduite pour permettre de caractériser rapidement tout problème d'ordonnancement [Graham et al.,79]. Celle-ci repose sur trois champs $\alpha|\beta|\gamma$ correspondant respectivement à la structure de l'atelier, aux contraintes prises en compte et au(x) critère(s) à optimiser. Le glossaire, placé à la fin de ce mémoire, présente quelques exemples pour ces trois champs. Avant de nous pencher sur les spécificités des problèmes que nous étudions, il convient de donner quelques précisions concernant les approches généralement adoptées pour traiter les problèmes d'ordonnancement.

1.2.4 Approches de résolution

Comme nous l'avons mentionné auparavant, un problème d'ordonnancement consiste à déterminer une solution respectant les contraintes fixées telle qu'un critère soit minimisé ou maximisé. Il s'agit donc d'un problème d'optimisation discrète, puisque l'on suppose habituellement que toutes les variables (en particulier les dates d'exécution) sont des entiers. Face à de tels problèmes, la première question que l'on est amené à se poser concerne la complexité. Ce point est abordé dans l'introduction du chapitre 2 et nous n'insistons pas ici sur cet aspect. Nous supposons désormais que le problème d'ordonnancement à traiter est NP-difficile. Dans ce cas, il convient de s'interroger sur le type de méthodes que l'on va employer pour sa résolution : une méthode exacte, exponentielle et utilisable seulement pour des instances de taille limitée, ou une méthode approchée, applicable à toute taille d'instance, mais pour laquelle on perd la garantie d'obtenir toujours en fin d'exécution une solution optimale. On trouve dans la littérature plusieurs typologies des méthodes de résolution ([GOThA,93], [Portmann,97]), qui distinguent généralement :

- les méthodes de programmation linéaire, pour lesquelles le problème est formulé de telle sorte que le critère et les contraintes s'expriment comme des fonctions linéaires des variables de décision. En ordonnancement, les variables sont souvent entières, voire binaires, ce qui rend les problèmes beaucoup plus difficiles à résoudre et limite les résolutions exactes à des instances de petite taille
- les méthodes par construction, qui consistent à construire de manière itérative la solution. Parmi ces méthodes, on retrouve les algorithmes de listes, pour lesquels les travaux sont successivement ajoutés à la solution partielle selon une liste triée, qui constituent une des techniques les plus simples et les plus intuitives. On parle de méthodes gloutonnes lorsque la décision prise lors d'une itération n'est plus remise en cause par la suite
- les méthodes par décomposition, qui consistent à diviser le problème initial en plusieurs sous-problèmes de plus petite taille, plus faciles à résoudre, puis à "fusionner" les solutions de chacun d'eux pour obtenir une solution globale. Selon le type de décomposition, on parle de décomposition hiérarchique, structurelle, spatiale, temporelle (voir [Portmann,87], [Portmann,88])
- les méthodes par voisinage, encore appelées méthodes de recherche locale. A partir d'une solution initiale, on se déplace dans l'ensemble des solutions en passant à chaque étape d'une solution à un de ses "voisins", jusqu'à ce qu'une condition d'arrêt soit vérifiée. Les modalités du déplacement, notamment le choix de la nouvelle solution courante parmi les voisins, peuvent être plus ou moins sophistiquées, et conduisent à de nombreux algorithmes tels que la méthode de plus forte pente, la recherche tabou [Glover,90], le recuit simulé [Kirkpatrick et al.,83], ...
- les méthodes évolutionnistes, à base de populations. Contrairement aux précédentes, ces méthodes considèrent un ensemble de solutions, appelé population, qui évolue au cours du processus, et dans laquelle les individus les plus forts, c'est-à-dire les meilleures solutions, sont favorisés. Dans cette catégorie, on retrouve en particulier les algorithmes génétiques [Holland,75], [Goldberg,89], pour lesquels l'évolution se fait par l'intermédiaire de croisement entre solutions (combinaison des caractéristiques de deux solutions) et de mutation (modification d'une solution). On peut également citer les colonies de fourmi [Colorni et al,91], qui reproduisent le comportement social des insectes.
- les méthodes issues de l'intelligence artificielle, notamment les techniques de propagation de contraintes [Erschler,76]. Celles-ci utilisent une exploration arborescente et cherchent à transformer le problème en un problème équivalent en retirant du domaine des variables les valeurs qui n'appartiennent à aucune solution et en simplifiant l'expression des contraintes. Le raisonnement énergétique [Lopez,91], par exemple, permet de limiter la localisation d'une opération sur une ressource dans un intervalle de temps. Dans cette catégorie, on retrouve aussi les algorithmes à réseaux de neurones qui font appel à des techniques d'apprentissage.

Chacune de ces méthodes possède bien entendu ses propres avantages et inconvénients, en termes de simplicité d'implémentation, de complexité temporelle, c'est-à-dire de temps d'exécution, et de qualité des solutions fournies. Une tendance actuelle en optimisation combinatoire consiste à concevoir des approches de résolution ayant recours à plusieurs méthodes parmi celles que nous venons de citer, afin de parvenir à une plus grande performance. On parle alors d'hybridation, comme dans le cas des algorithmes mémétiques qui combinent algorithmes génétiques et amélioration par recherche locale. Certaines techniques arborescentes utilisant à la fois la programmation linéaire et la propagation de contraintes ont aussi montré leur efficacité pour la résolution de problèmes d'optimisation.

De nombreux exemples illustrant le fonctionnement d'une partie de ces méthodes figurent dans le chapitre 3, qui présente un état de l'art sur les approches de résolution proposées dans la littérature pour les problèmes d'ordonnancement en présence de contraintes de time lags. En outre, les algorithmes que nous avons développés sont décrits dans le chapitre 4. Il y est notamment question de Procédures par Séparation et Evaluation (PSE). Celles-ci, dont la présentation détaillée fait l'objet de la section 4.2, font partie des méthodes par décomposition et peuvent, dans certains cas, intégrer de façon sous-jacente, au sein d'une approche arborescente, des méthodes par construction.

1.3 Définition du concept de contraintes d'écarts temporels

Le modèle classique, en ordonnancement d'atelier, ne considère que des relations de précédence simples entre opérations. C'est notamment le cas pour les opérations consécutives au sein des travaux, dans le flowshop et le jobshop. Si l'opération i précède l'opération j, alors l'exécution de l'opération j ne peut débuter qu'après la fin de traitement de l'opération i. Par contre, elle peut débuter à n'importe quel moment à partir de cette date. Ce modèle peut se révéler trop restrictif lorsque l'on cherche à représenter certaines situations rencontrées dans le monde industriel, comme nous le verrons dans la section 1.5. En effet, certaines contraintes peuvent imposer à la seconde opération de ne débuter qu'après un laps de temps déterminé depuis la fin de la première opération. Dans ce cas, l'attente entre les deux opérations est minorée. A l'inverse, d'autres contraintes peuvent obliger l'intervalle de temps entre les deux opérations à ne pas dépasser une valeur donnée. L'attente est alors majorée. Ces contraintes, qui portent sur l'écart de temps entre deux opérations, permettent de généraliser les contraintes de précédence simples. Dès 1959, dans le cadre de l'ordonnancement de projet, Roy [Roy,59] a proposé un modèle intégrant ce type de contraintes appelées contraintes de potentiel. Cependant, elles ont fait l'objet d'assez peu d'études, notamment en ordonnancement d'atelier, où les contraintes d'attentes maximales sont très rarement étudiées. De plus, on peut les rencontrer dans la littérature sous de nombreuses dénominations : contraintes de potentiel, contraintes d'attentes, contraintes de précédence généralisées, contraintes de délais ou encore contraintes d'écarts temporels (*time lags* en anglais). Dans ce mémoire, nous les désignons par le terme *time lags*, qui reste malgré tout le plus utilisé dans la littérature. La section suivante présente les principales notations employées ainsi que le modèle considéré.

1.4 Modélisation mathématique

Nous nous limitons à l'étude de problèmes de type flowshop en présence de time lags au sein des travaux, bien qu'il soit possible de considérer de telles contraintes entre des opérations quelconques, appartenant à des travaux différents. De nombreuses situations industrielles correspondent en effet à ce cas [Deppner,04]. Nous supposons dans un premier temps que les time lags ne sont définis qu'entre les opérations successives de chaque travail. Nous expliquons dans la section 4.8.2 comment généraliser une partie des résultats obtenus en présence de time lags entre opérations quelconques au sein des travaux. Le problème général peut alors être formulé de la manière suivante.

On considère n travaux $1, 2, \ldots, n$ à traiter sur m machines. Chaque travail j est composé de m opérations élémentaires telles que la k-ième opération doit être exécutée sur la machine k

pendant $p_{j,k}$ unités de temps. Il existe entre tout couple d'opérations successives $(k, k+1)$ de j des contraintes de précédence généralisées, représentées par un time lag minimal $\theta_{j,k}^{min}$ et un time lag maximal $\theta_{j,k}^{max} \geq \theta_{j,k}^{min}$ tels que :

$t_{j,k+1} \geq C_{j,k} + \theta_{j,k}^{min}$ et $t_{j,k+1} \leq C_{j,k} + \theta_{j,k}^{max}$ où $t_{j,k+1}$ désigne la date de début du travail j sur la machine $k+1$ et $C_{j,k}$ la date de fin de j sur la machine k.

La préemption n'est pas autorisée de telle sorte qu'une fois commencée, une opération ne peut pas être arrêtée et va jusqu'à son terme. Les dates de début et de fin de chaque opération sont donc liées par la relation :

$\forall j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\}, C_{j,k} = t_{j,k} + p_{j,k}$.

En outre, chaque machine ne peut traiter qu'une opération à la fois. Supposons que $\pi_k = (\pi_k(1), \pi_k(2), \ldots, \pi_k(n))$ désigne la séquence de passage des travaux sur la machine k, on a l'ensemble de contraintes suivantes :

$\forall j \in \{1, 2, \ldots, n-1\}, t_{\pi_k(j+1),k} \geq C_{\pi_k(j),k}$.

L'objectif est alors de déterminer les séquences de traitement des travaux sur les machines π_k et les dates de fin associées $C_{j,k}$ de manière à minimiser le critère considéré.

Un modèle mathématique rigoureux peut par exemple être donné par le programme linéaire suivant, en supposant que le critère est le makespan C_{max} et que M désigne un entier suffisamment grand par rapport aux autres données du problèmes.

- Variables :
  - $\forall i \neq j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\}, X_{i,j,k} = 1$ si i est traité avant j sur la machine k, 0 sinon
  - $\forall i \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\}, C_{i,k}$ est la date de fin du travail i sur la machine k

- Objectif : $Min\ C_{max}$

- Contraintes :
  - $X_{i,j,k} + X_{j,i,k} = 1$ $(\forall i \neq j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\})$
  - $C_{i,1} \geq p_{i,1}$ $(\forall i \in \{1, 2, \ldots, n\})$
  - $C_{i,k+1} \geq C_{i,k} + p_{i,k+1} + \theta_{i,k}^{min}$ $(\forall i \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m-1\})$
  - $C_{i,k+1} \leq C_{i,k} + p_{i,k+1} + \theta_{i,k}^{max}$ $(\forall i \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m-1\})$
  - $C_{i,k} \geq C_{j,k} + p_{i,k} - M.X_{i,j,k}$ $(\forall i \neq j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\})$
  - $C_{max} \geq C_{i,m}$ $(\forall i \in \{1, 2, \ldots, n\})$
  - $X_{i,j,k} \in \{0, 1\}$ $(\forall i \neq j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\})$
  - $C_{i,k} \geq 0$ $(\forall i \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m\})$

La première famille de contraintes stipule que pour tout couple de travaux (i, j), l'un est nécessairement traité avant l'autre. Les autres contraintes sont assez naturelles. L'utilisation du grand entier M permet de distinguer les situations où i précède j et où j précède i. Dans le premier cas, la contrainte n'a pas d'influence sur $C_{i,k}$ puisque le membre droit est négatif, tandis

que dans le second cas, elle traduit le fait que i ne peut pas débuter son exécution avant la fin de j, sur la machine k. Les variables sont de deux types : binaires pour les $X_{i,j,k}$ et réelles pour les $C_{i,k}$. Ces dernières prendront en fait des valeurs entières si toutes les données du problème (durées opératoires, time lags) sont entières, ce que nous supposons. Il s'agit donc d'un programme linéaire en variables mixtes, sur lequel nous n'insistons pas. Pour une comparaison de plusieurs modèles possibles pour le flowshop classique, nous renvoyons le lecteur à [Stafford et al.,05].

On constate facilement que le problème classique du flowshop (sans time lags) est un cas particulier de notre problème, pour lequel on a $\forall j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m-1\}, \theta_{j,k}^{min} = 0$ (contraintes de précédence simple, une fois le travail terminé sur la machine k, il est immédiatement disponible pour le traitement sur la machine $k+1$), et $\forall j \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, m-1\}, \theta_{j,k}^{max} = +\infty$ (une fois le travail terminé sur la machine k, on peut attendre aussi longtemps qu'on le souhaite avant de l'exécuter sur la machine $k+1$).

Il convient de noter également que nous avons choisi de définir les time lags entre la fin d'une opération et le début de l'opération suivante (*stop-start time lags*). Certains auteurs considèrent les time lags de début à début (*start-start*) ou de fin à fin (*stop-stop*). Mais on peut facilement montrer que, du moment que les durées opératoires sont fixes et connues à l'avance, les problèmes qui en découlent sont équivalents (voir par exemple [De Reyck et Herroelen,98], [Bartusch et al.,88], [Szwarc,86]). Nous proposons de désigner les problèmes d'ordonnancement avec time lags minimaux et maximaux à l'aide de $\theta_{j,k}^{min}$ et $\theta_{j,k}^{max}$ dans le second champ (β) de la notation présentée en section 1.2. Les indices j et k font référence aux travaux et aux machines, respectivement, et nous les omettons quand les valeurs des time lags sont identiques, pour tous les travaux ou pour toutes les machines. Par exemple, dans le cas d'un flowshop à deux machines, à chaque travail j est associé un seul time lag minimal θ_j^{min} et un seul time lag maximal θ_j^{max}.

Un cas particulier du flowshop est souvent considéré dans la littérature : le flowshop de permutation, pour lequel la séquence de passage des travaux sur les machines est la même pour toutes les machines. Formellement, on a alors : $\forall k \in \{1, 2, \ldots, m\}, \pi_k = \pi$, où π désigne la séquence de traitement des travaux. En ce qui concerne le programme linéaire, les variables traduisant la précédence ne dépendent plus de la machine k. Le nombre de séquences acceptables est majoré par $(n!)^m$ dans le cas général, alors qu'il est réduit à $n!$ si l'on se limite aux ordonnancements de permutation. D'autre part, cette restriction peut aussi correspondre à une réalité industrielle. En effet, on peut rencontrer des situations dans lesquelles les produits, une fois la production lancée, ne peuvent plus se dépasser entre deux machines consécutives, en raison de contraintes technologiques (par exemple, absence ou faible capacité de stockage entre les machines, utilisation d'un convoyeur de largeur limitée pour mettre les produits en file d'attente). La majeure partie de nos travaux concernent le flowshop de permutation, que nous désignons par Fm_π dans le premier champ de la notation présentée en section 1.2, où m désigne le nombre de machines. Dans ce cas, nous montrons que, même en présence de time lags, toute solution intéressante du problème peut être obtenue à partir de la permutation (voir la section 4.3.1). Nous analysons également dans quelle mesure les approches développées pour le flowshop de permutation peuvent être étendues au flowshop général, dans la section 4.8.3.

1.5 Applications industrielles

L'un des principaux intérêts des time lags est de pouvoir représenter, avec un même modèle, des situations réelles très variées. Dès que deux opérations au sein des gammes sont reliées par une contrainte temporelle, des time lags peuvent être utilisés. Pourtant, la littérature en ordonnancement sur le sujet reste très limitée. Les time lags sont étudiés principalement dans le cadre de l'ordonnancement de projet. En ordonnancement d'atelier, les recherches ont surtout porté sur les time lags minimaux et il existe très peu de références concernant les time lags maximaux. Cependant, en y regardant de plus près, les time lags sont omniprésents dans le monde industriel. Comment alors expliquer le manque d'intérêt qui leur est porté ? On peut supposer que, comme beaucoup d'autres contraintes (telles que les contraintes de calendriers, l'indisponibilité des machines), les time lags ne sont pas pris en compte pour simplifier le modèle et permettre une résolution plus efficace du problème. S'il est vrai que dans certains cas, ils ne représentent pas une contrainte forte du problème et peuvent être violés, ils constituent à l'inverse une caractéristique fondamentale de nombreux exemples et doivent être absolument respectés, sous peine de conduire à la perte pure et simple des produits.

a) Quelques exemples

Dans sa thèse, Deppner [Deppner,04] présente plusieurs cas d'études rencontrés, qui peuvent conduire à des problèmes d'ordonnancement avec time lags. Ils concernent en particulier :

- une entreprise de fabrication de papier thermique dont la production fait appel à un traitement dans des bains de produits chimiques. Ceux-ci ne doivent pas être disponibles trop tôt avant utilisation sous peine de perdre leurs propriétés chimiques. On a donc dans ce cas une contrainte d'attente maximale ;
- les services de type "vidéo à la demande", notamment via Internet. Les séquences vidéo sont envoyées au client, en respectant à la fois des contraintes d'écart minimal afin d'éviter de surcharger le buffer et des contraintes d'écart maximal correspondant aux durées des séquences de sorte que celles-ci s'enchaînent sans interruption ;
- une société de conception et fabrication d'appareils à engrenage, et une société de production de presses utilisées pour l'impression de journaux. Toutes les deux ont recours à des contrôles réguliers pendant des opérations de longue durée. Avant de réaliser un contrôle, il est nécessaire d'attendre qu'un état stable soit atteint, de manière à ce que les mesures effectuées soient significatives. A l'inverse, le contrôle ne doit pas être réalisé trop tard sans quoi la quantité de produits non conformes risque d'être importante en cas de défaut ;
- un centre de formation, pour lequel l'enchaînement des modules enseignés doit répondre à certaines contraintes temporelles ;
- le service informatique d'une grande administration. Les procédures de sauvegarde de données font intervenir une succession d'opérations avec accès à des têtes d'écriture. L'existence d'un nombre limité de ces têtes d'écriture conduit à mettre en attente certaines opérations, dont l'exécution doit pourtant respecter des délais précis ;
- une entreprise pharmaco-chimique. La réalisation des différentes opérations du process, de la préparation du matériel et des matières premières jusqu'au conditionnement, obéit à des contraintes temporelles fortes, notamment d'écart maximal. Deppner insiste particulièrement sur le fait que certaines d'entre ellles sont dues aux propriétés chimiques des produits, tandis que d'autres proviennent de normes réglementaires. D'autre part, l'ensemble du process est extrêmement complexe et met en jeu d'autres contraintes (calendriers, blocage, ...)

que nous ne détaillons pas.

b) Intérêt des time lags maximaux

Comme l'illustrent les exemples précédents, la prise en compte de time lags maximaux pour un problème d'ordonnancement est la plupart du temps liée à des contraintes spécifiques au type d'applications :

- dans le domaine de l'agriculture [Foulds et Wilson,05], avec les contraintes portant sur l'enchaînement des différents travaux lors des récoltes
- dans l'industrie agro-alimentaire, avec l'existence de produits périssables et de normes sanitaires. Par exemple, Hodson et al. [Hodson et al.,85] s'intéressent à la production de plats surgelés. La principale contrainte du process concerne le délai maximal de 30 minutes entre la cuisson des produits et leur surgélation
- dans les industries chimiques et pharmaceutiques, comme nous venons de le voir
- dans le domaine du génie civil, avec des projets de construction (voir [Bartusch et al.,88] ou [Heilmann,03])

c) Intérêt des time lags minimaux

D'une manière générale, on peut avoir recours à des time lags minimaux dans des cas très différents. Bien entendu, ils peuvent être associés à des contraintes d'écart minimal particulières, comme par exemple des temps de séchage, de refroidissement. Mais ils peuvent également être utilisés pour représenter des opérations qui ne nécessitent pas de machines, et qui sont effectuées sur des ressources non critiques. Les problèmes considérés par Gupta [Gupta,96] et Riezebos et al. [Riezebos et al.,95], qui concernent l'ordonnancement de travaux composés de deux opérations séparées par des time lags minimaux, proviennent d'une application à un système de production automatisé de type FMS (Flexible Manufacturing System : système de production flexible). Chaque machine correspond en fait à une station de travail sur laquelle plusieurs opérations sont effectuées : chargement, traitement, déchargement. Entre celles-ci, d'autres activités, ne nécessitant pas l'utilisation de la station, interviennent, comme par exemple un transport manuel. Les time lags minimaux correspondent alors à la durée de ces activités réalisées "off line".

Finta et Liu [Finta et Liu,94] considèrent un problème d'ordonnancement sur un système informatique multiprocesseur. Ils se ramènent à un problème à une machine, en considérant les communications sur le bus comme des opérations et les processus exécutés sur les processeurs en parallèle comme des time lags minimaux. Plus généralement, on pourrait aussi envisager d'assimiler les délais de communication entre processeurs à des time lags minimaux. Cependant, en informatique, on suppose généralement que de tels délais n'interviennent qu'entre des processeurs différents. Ils dépendent donc de l'affectation des tâches aux ressources, et le modèle qui en résulte n'est donc pas exactement identique à celui que nous considérons avec les time lags. On pourrait considérer une hypothèse similaire pour la technologie de groupe, en introduisant des temps de déplacement entre les machines appartenant à des cellules de travail différentes.

Il est aussi possible de recourir à des time lags minimaux dans le cas d'une production par lots, lorsque la taille du lot de transfert est inférieure à la taille du lot de traitement. Les produits unitaires passent ainsi d'une machine à la suivante selon la quantité définie par le lot de transfert. Par conséquent, il est possible de débuter l'exécution d'un travail sur une machine avant la fin de

son exécution sur la machine précédente, ce qui se traduit par un recouvrement. Dans ce cas, les time lags minimaux de type start-start représentent le temps de traitement d'un lot de transfert sur la première machine, tandis que ceux de type stop-stop représentent le temps de traitement d'un lot de transfert sur la seconde machine [Baker,90]. Comme nous l'avons vu, il est toujours possible de se ramener à un seul type de time lags. Si l'on utilise des time lags de type stop-start, ceux-ci sont dans ce cas négatifs, ce qui correspond à la possibilité de recouvrement partiel. On retrouve cet emploi des time lags minimaux dans [Khurana et Bagga,84] dans le cadre de la production d'essence de térébenthine, et dans [Kim et al.,96] pour la fabrication de circuits imprimés.

De manière analogue, les time lags minimaux peuvent être utilisés pour représenter des temps de montage et démontage, dans le cas où ceux-ci sont anticipatoires (c'est-à-dire ne nécessitant pas la présence du travail sur la machine) et indépendants de la séquence de traitement, comme nous le montrons dans la section 4.6. Dans ce cas, les contraintes sont parfois désignées par NSDST et NSDRT (*Non Sequence-Dependent Setup/Removal Times*). Nous attirons l'attention du lecteur sur le fait que cette manière de procéder n'est pas nouvelle (voir [Szwarc,83], [Baker,90], [Botta et Guinet,96], [Espinouse et al.,00]).

d) Problèmes d'ordonnancement particuliers

Nous terminons cette section par la description d'applications pratiques conduisant à des problèmes d'ordonnancement très particuliers, dans lesquels on retrouve des contraintes qui se rapprochent des time lags. Le problème du *coupled-task*, étudié dans [Shapiro,81] et dans [Orman et Potts,97], est issu d'une application aux systèmes radars. Chaque travail est composé de deux opérations de durées fixées, correspondant respectivement à la transmission et à la réception d'une onde électro-magnétique. Entre ces deux opérations, il existe un écart de temps déterminé, qui dépend en particulier de la distance à la cible visée. Le radar ne peut exécuter qu'une opération à la fois, et l'on cherche en général à minimiser les temps morts. Dans la pratique, l'ordonnancement est effectué en temps réel, de manière dynamique. Etant donné que les ondes se propagent à la vitesse de la lumière, les durées mises en jeu sont de l'ordre de la micro-seconde. Par conséquent, le problème obéit à des contraintes très fortes. L'ordonnancement de type *coupled-task* peut être considéré comme un problème avec time lags, dans la mesure où l'écart séparant les deux opérations de chaque travail correspond à un time lag minimal et un time lag maximal de même valeur.

Chu et Proth [Chu et Proth,96] étudient le cas d'un laboratoire d'analyses médicales automatisé. Les durées des réactions chimiques appartiennent à un intervalle de temps, tandis que les autres opérations telles que l'introduction d'éléments dans les tubes à essai ou la lecture des résultats des analyses ont des durées fixes. Le modèle proposé représente ces dernières comme des opérations "classiques", alors que des time lags minimaux et maximaux sont associés aux réactions chimiques. Le même type de modèle peut être employé en sidérurgie, lorsque certains traitements des métaux requièrent une température précise. Dans ce cas, les time lags correspondent au réchauffement ou au refroidissement entre les opérations [Caumond et al.,04b]. Ces deux problèmes sont assez proches du *hoist scheduling*, qui fait l'objet du paragraphe suivant.

Le problème de *hoist scheduling* comporte à la fois des contraintes temporelles mais également des contraintes fortes en termes de partage de ressources. Il concerne surtout les systèmes de traitement de surface, pour lesquels les produits à traiter doivent visiter des cuves contenant des solutions chimiques dans un certain ordre. La durée passée dans chaque cuve est comprise

dans un intervalle de temps afin d'assurer les propriétés recherchées pour les produits. De plus, le transport des produits d'une cuve à une autre est réalisé par des robots, qui sont disponibles en quantité limitée. Le problème consiste alors à ordonnancer non seulement les opérations de traitements des produits dans les cuves, mais également les mouvements des robots qui assurent le transport. En plus des contraintes déjà citées, la plupart des travaux dans ce domaine considèrent que la capacité des cuves est limitée. Etant donné que celles-ci sont utilisées comme espace de stockage avant l'arrivée du robot, on doit donc faire face à des contraintes additionnelles de blocage de ressource. Enfin, ce type de problèmes est généralement étudié dans le cadre d'une production cyclique, pour laquelle le système se retrouve périodiquement dans le même état. Les hypothèses sont donc assez différentes de celles que nous considérons dans cette étude. Par conséquent, nous n'insistons pas sur ce type de problèmes. Pour un état de l'art sur ce sujet, le lecteur peut consulter [Manier et Bloch,03].

1.6 Conclusion du chapitre

Ce chapitre nous a permis de mettre en évidence l'importance des time lags dans les problèmes d'ordonnancement. Ces contraintes, qui généralisent la relation de précédence classique entre opérations, peuvent être rencontrées dans de nombreuses applications industrielles. Par ailleurs, elles permettent aussi de modéliser des opérations réalisées sur des ressources non critiques, pour lesquelles on ne se préoccupe pas de l'ordonnancement, ainsi que des temps de montage et démontage, sous certaines conditions.

Le modèle que nous proposons considère des time lags de type stop-start, définis entre la fin d'une opération et le début de la suivante. En outre, nous nous limitons au cas où les time lags sont définis uniquement entre opérations consécutives au sein des travaux.

Les chapitres suivants sont consacrés à l'analyse et à la résolution des problèmes relatifs à ce modèle. Dans le chapitre 2, nous nous intéressons à la complexité de ces problèmes. Puis nous décrivons, dans le chapitre 3, les approches de résolution proposées dans la littérature pour traiter ces problèmes, quelle que soit leur structure (machine unique, atelier de type flowshop, projet, ...). Dans le chapitre 4, nous présentons les méthodes que nous avons développées dans le cadre du flowshop de permutation. Nous présentons des perspectives pour la poursuite de ce travail dans le chapitre 5.

Chapitre 2

Complexité des problèmes de flowshop en présence de contraintes d'écarts temporels

Résumé

Dans ce chapitre, nous nous intéressons à la complexité des problèmes d'ordonnancement de type flowshop, en présence de time lags minimaux et maximaux. Nous introduisons tout d'abord un critère particulier, la somme (pondérée) des dates de fin sur les machines, qui, à notre connaissance, n'a fait l'objet que de très peu d'études dans la littérature. Après avoir montré en quoi cet objectif peut être intéressant d'un point de vue pratique, nous déduisons des résultats de complexité pour les problèmes de flowshop sans time lags, avec ce critère. Nous considérons ensuite les problèmes avec time lags minimaux uniquement. Les principaux résultats tirés de la littérature sont rappelés, et nous présentons également des algorithmes polynomiaux pour un certain nombre de problèmes de flowshop de permutation avec durées d'exécution identiques. En ce qui concerne les problèmes avec time lags maximaux, nous distingons le cas sans attente, pour lequel il existe déjà de nombreux résultats, et le cas général avec time lags maximaux quelconques. Nous fournissons de nouvelles propriétés de dominance, et nous montrons que le problème de flowshop à deux machines avec time lags maximaux identiques est NP-difficile au sens fort. En outre, nous analysons la performance d'heuristiques dans le pire cas. La dernière partie du chapitre est consacrée aux problèmes impliquant une machine à traitement par fournées, de type parallèle, en présence de time lags minimaux. Nous exposons les particularités de ces problèmes et nous présentons quelques propriétés et résultats de complexité. Le chapitre se conclut par une synthèse des résultats pour l'ensemble des problèmes considérés.

Sommaire

2.1 Introduction

D'une manière générale, les problèmes d'ordonnancement constituent un domaine de l'optimisation combinatoire. Il est donc très important de connaître la complexité des problèmes qui conditionne les approches de résolutions adoptées. En règle générale, les problèmes d'ordonnancement appartiennent à l'une des catégories suivantes (pour une présentation plus complète des concepts et de la théorie de la complexité, nous renvoyons le lecteur vers l'ouvrage de Garey et Johnson [Garey et Johnson,79]) :

- les problèmes polynomiaux, pour lesquels on peut montrer qu'il existe un algorithme dont le nombre d'opérations élémentaires est polynomial en la taille des données d'entrée. On parle alors d'algorithme polynomial
- les problèmes NP-difficiles, pour lesquels l'existence d'un algorithme polynomial de résolution exacte est impossible si l'on accepte la conjecture P≠NP

On peut facilement vérifier que pour tout problème d'optimisation que nous considérons dans cette thèse, le problème de décision correspondant appartient à la classe NP, dans la mesure où il est toujours possible de vérifier en un temps polynomial si un ordonnancement donné est solution du problème. Nous ne le mentionnons donc pas systématiquement dans les démonstrations.

D'autre part, la question de la dominance des ordonnancements de permutation, pour lesquelles l'ordre de traitement des travaux est le même sur toutes les machines, joue aussi un rôle capital.

Dans ce chapitre, nous ne considérons que des time lags définis entre des couples d'opérations successives au sein des travaux. Comme nous l'avons mentionné dans le chapitre précédent (voir section 1.4), l'absence de contraintes de ce type entre deux opérations peut néanmoins être représentée par des time lags (0 pour les time lags minimaux, $+\infty$ pour les time lags maximaux). D'autre part, la prise en compte de time lags généralisés (c'est-à-dire définis entre des opérations quelconques au sein des travaux), conduit à des problèmes plus généraux, donc au moins aussi difficiles, au sens de la complexité. Afin d'éviter de considérer des situations particulières pour lesquelles un travail pourrait débuter sur la machine $k+1$ avant de débuter son exécution sur la machine k, nous supposons dans ce chapitre que les time lags minimaux, ne sont pas trop grands en valeur absolue, s'ils sont négatifs. Plus précisément, on considère que $\forall j,k, \theta_{j,k}^{min} \geq -p_{j,k}$, ce qui correspond à des time lags minimaux de début à début positifs ou nuls.

Nous précisons que les résultats concernant le nouveau critère $\sum w_k MC_k$ figurent dans [Fondrevelle et al.,05b]. L'étude des problèmes de flowshop de permutation avec durées identiques et time lags minimaux est également menée dans [Fondrevelle et al.,04a], tandis que les nouveaux résultats théoriques sur le flowshop avec time lags maximaux sont repris dans [Fondrevelle,03b] et [Fondrevelle et al.].

2.2 Définition d'un critère lié à l'utilisation des machines

Outre les critères classiques présentés dans la section 1.2, nous nous intéressons à une fonction objectif qui est très peu traitée dans la littérature, mais dont la minimisation peut refléter une réalité industrielle : la somme des dates de fin sur les machines, que nous notons $\sum MC_k$ pour éviter la confusion avec la somme des dates de fin des travaux $\sum C_j$. En suivant cette notation, nous désignons par MC_k la date de fin sur la machine k, c'est-à-dire la date de fin de dernier

travail traité par cette machine : $MC_k = max_j\{C_{j,k}\}$. Il est bien entendu possible de considérer la version pondérée de ce critère, $\sum w_k MC_k$, pour laquelle chaque machine k possède un poids w_k qui traduit son importance relative vis-à-vis des autres machines. Ce critère est étroitement lié à l'utilisation des ressources : étant donné que les durées d'exécution sont fixées, minimiser la somme des dates de fin sur les machines revient à minimiser les temps morts, si l'on considère que les machines sont toutes activées à partir de la date 0 mais qu'elles sont stoppées dès que leur dernière opération est terminée. La caractérisation des temps morts est représentée sur la figure 2.1. Dans la littérature, la définition du temps mort total peut varier d'une référence à l'autre, selon que les auteurs considèrent uniquement les temps morts intermédiaires (zone 2), les temps morts initiaux et les temps morts intermédiaires (zones 1 et 2) ou encore les temps morts initiaux, intermédiaires et finaux (zones 1, 2 et 3). Le critère de la somme des dates de fin sur les machines est lié au temps mort total défini par les zones 1 et 2.

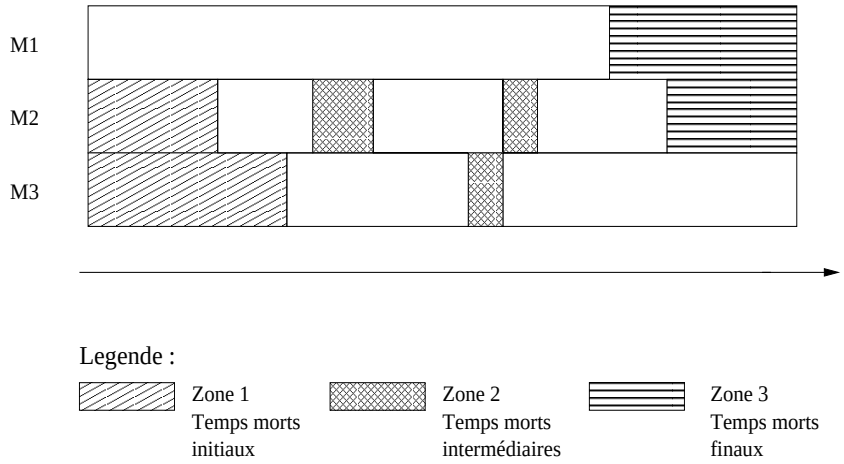

FIG. 2.1 – Caractérisation des temps morts

Plusieurs auteurs définissent également le temps mort total de cette manière et cherchent à minimiser cette fonction (voir par exemple [Framinan et al.,03], [Sridhar et Rajendran,96]). Ho et Gupta [Ho et Gupta,95] s'intéressent au cas des machines "dominantes". A notre connaissance, aucune étude n'est consacrée à la version pondérée de ce critère. Il convient de distinguer les problèmes où l'objectif est de minimiser le temps mort total de ceux où une contrainte impose aux machines de traiter les opérations sans temps mort (ce qui revient à imposer des temps morts intermédiaires nuls). Par exemple, Saadani et al. [Saadani et al.,03] s'intéressent à la minimisation du makespan sur un flowshop de permutation à trois machines, où les temps morts intermédiaires sont nuls. Ce type de problème est issu du domaine de la sidérurgie, où le process de coulée continue impose une succession des opérations sur les machines sans temps morts. Dans ce cas, on ne prend bien évidemment en compte que les temps morts intermédiaires.

La minimisation de la somme des dates de fin sur les machines a aussi une signification d'un point de vue pratique. Ho et Gupta [Ho et Gupta,95] évoquent la situation où les machines requises pour le traitement des travaux sont louées et où le coût de location est proportionnel à la durée d'utilisation de la machine (en supposant, comme nous l'avons déjà mentionné, que les machines démarrent toutes à l'instant 0 mais qu'elles sont arrêtées lorsqu'elles ont achevé l'exécution de leur dernière opération). Du fait de leurs caractéristiques techniques et des opérations

qu'elles effectuent, les machines n'ont pas nécessairement le même coût de location, ce qui justifie l'intérêt porté à la version pondérée. On retrouve la même problématique en ce qui concerne les coûts liés à la main d'oeuvre, et plus particulièrement le paiement des heures supplémentaires. Si la production est effectuée pendant une période où les employés sont payés en heures supplémentaires, l'objectif peut être de réaliser cette production de manière à les libérer le plus tôt possible. En fonction des compétences requises pour chaque opérateur, le coût unitaire associé à chaque poste de travail n'est pas forcément le même. Il serait possible d'affiner un tel modèle en définissant des coûts unitaires dépendant non seulement des machines, mais également de la période.

Cette nouvelle fonction objectif possède plusieurs propriétés générales qui peuvent nous aider à déterminer la complexité de nombreux problèmes qui lui sont relatifs. Considérons par exemple un problème de flowshop à deux machines $F2|\beta|$ où β désigne un ensemble de contraintes additionnelles.

Propriété. 1 *Si, pour tout ordonnancement réalisable du problème $F2|\beta|$, il est possible de caler à gauche les opérations sur la première machine de manière à ce que la date de fin sur cette machine soit $MC_1 = \sum_{1 \leq j \leq n} p_{j,1}$ et que l'ordonnancement reste inchangé sur la seconde machine, alors le problème de minimisation de la somme (éventuellement pondérée) des dates de fin sur les machines est équivalent au problème de minimisation du makespan (c'est-à-dire $F2|\beta| \sum MC_k$, $F2|\beta| \sum w_k MC_k$ et $F2|\beta|C_{max}$ sont équivalents).*

Preuve. Considérons la version pondérée du critère. Dans le cas d'un tel problème à deux machines, la fonction objectif $\sum w_k MC_k$ est la somme de deux termes $w_1 MC_1$ et $w_2 MC_2$. Puisque, par hypothèse, le premier terme peut être fixé à une valeur constante minimale pour chaque solution, le problème consiste à minimiser le second terme $w_2 MC_2 = w_2 C_{max}$, ce qui revient à minimiser le makespan. □

Nous allons voir par la suite que de nombreux problèmes vérifient la propriété précédente.

Par ailleurs, on constate facilement que le problème de la minimisation du makespan est un cas particulier du problème de minimisation de la somme pondérée des dates de fin sur les machines, dans lequel tous les poids sont nuls à l'exception de celui de la dernière machine : $\forall k, 1 \leq k \leq m-1, w_k = 0$ and $w_m = 1$. Ceci nous conduit au résultat suivant :

Propriété. 2 *$Fm|\beta|C_{max}$ se réduit polynomialement à $Fm|\beta| \sum w_k MC_k$.*

Bien entendu, $Fm|\beta| \sum MC_k$ est aussi un cas particulier de $Fm|\beta| \sum w_k MC_k$.

Les propriétés de réduction élémentaire entre les fonctions objectifs classiques, auxquelles nous avons ajouté les deux nouvelles fonctions, sont représentés sur la figure 2.2.

2.3 Problèmes de flowshop sans time lags ($\theta_{j,k}^{min} = 0$ et $\theta_{j,k}^{max} = +\infty$)

Dans cette section, nous rappelons les principaux résultats de complexité concernant les problèmes de flowshop classique, c'est-à-dire en l'absence de time lags. Ces problèmes ont fait l'objet de nombreuses études depuis plus d'un demi-siècle, que ce soit du point de vue de la complexité, ou de la résolution quand les problèmes sont connus pour être NP-difficiles. Le but n'étant pas de faire un état de l'art exhaustif sur le sujet, nous nous contentons de rappeler les

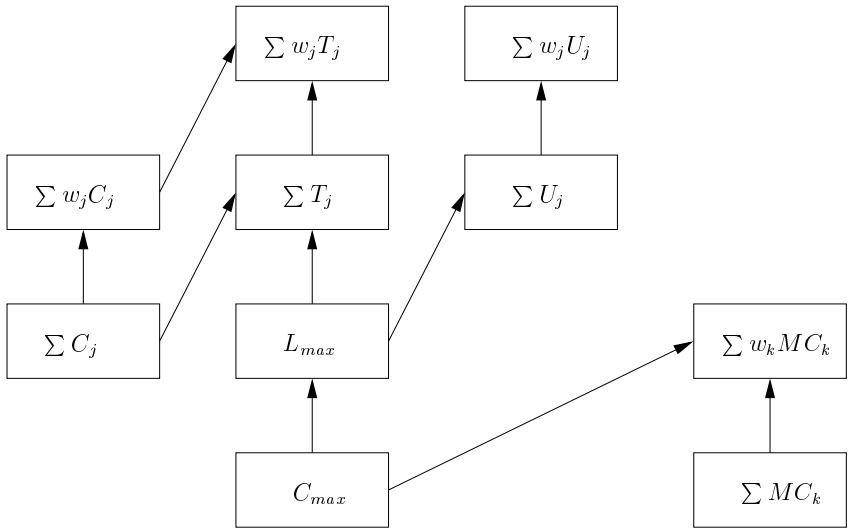

FIG. 2.2 – Réductions élémentaires des principales fonctions objectifs

résultats fondamentaux pour les comparer avec ceux qui concernent les cas impliquant des time lags et ainsi analyser l'influence des time lags sur la complexité des problèmes. Un récapitulatif des résultats de complexité pour de nombreux problèmes d'ordonnancement d'atelier est répertorié sur le site [Brucker et Knust].

2.3.1 Minimisation du makespan

L'un des résultats les plus connus en ordonnancement est l'optimalité de la règle de Johnson [Johnson,54] pour le problème de la minimisation du makespan dans un flowshop à deux machines ($F2||C_{max}$), qui conduit à un algorithme polynomial exact. Cette règle a servi de base à de nombreuses méthodes de résolution, qu'ils s'agissent d'algorithmes exacts pour des problèmes polynomiaux, d'heuristiques ou de bornes inférieures pour des problèmes NP-difficiles. Etant donné son importance, nous rappelons ici brièvement son fonctionnement.

Propriété. 3 *Règle de Johnson [Johnson,54]. Pour le problème $F2||C_{max}$, il existe un ordonnancement optimal dans lequel le travail i précède le travail j sur les deux machines si $min(p_{i,1}, p_{j,2}) \leq min(p_{j,1}, p_{i,2})$.*

Il existe plusieurs manières d'implémenter cette règle afin d'obtenir un algorithme polynomial. L'algorithme proposé par Johnson [Johnson,54], de complexité $O(n^2)$ est le suivant :

Algorithme. Johnson 1.
- $N = \{1, 2, \ldots, n\}$ (ensemble des indices des travaux non placés), $q = 1$, $r = n$.
- Tant que $N \neq \emptyset$ faire :
  - Soient (j, k) tels que $p_{j,k} = min\{p_{i,h}/i \in N, h \in \{1, 2\}\}$.
  - Si $k = 1$, alors placer j en position q dans la séquence. $q = q + 1$.
  - Sinon, placer j en position r dans la séquence. $r = r - 1$.
  - $N = N \setminus \{j\}$
- La séquence complète est optimale.

Un algorithme plus efficace, de complexité $O(nlogn)$ consiste à procéder de la manière suivante [Carlier et Chretienne,88] :

Algorithme. Johnson 2.
- Soient les ensembles $U = \{j/p_{j,1} < p_{j,2}\}$ et $V = \{j/p_{j,1} \geq p_{j,2}\}$.
- Soient σ_U la séquence des éléments de U triés dans l'ordre croissant des $p_{j,1}$ et σ_V la séquence des éléments de V triés dans l'ordre décroissant des $p_{j,2}$.
- La séquence optimale est obtenue en concaténant σ_U et σ_V.

On retrouve un algorithme similaire dans [Bellmann et al.,82], à ceci près que le premier ensemble est défini par $\{j/p_{j,1} \leq p_{j,2}\}$ et le second par $\{j/p_{j,1} > p_{j,2}\}$. La séquence obtenue peut donc être différente en présence de travaux dont les durées sur la première et la deuxième machines sont identiques.

On peut remarquer que l'ordonnancement obtenu grâce à la règle précédente est un ordonnancement de permutation. Les ordonnancements de permutation sont donc dominants pour la minimisation du makespan dans un flowshop à deux machines. Cette propriété reste valable pour $m = 3$, mais pas pour un nombre supérieur de machines. Plus précisément, on peut établir le résultat suivant (voir, par exemple, [Tanaev et al.,94]).

Propriété. 4 *Pour le critère du makespan, dans un flowshop classique à m machines, les ordonnancements pour lesquels les séquences de traitement des travaux sont identiques sur les deux premières machines d'une part, et sur les deux dernières machines d'autre part, sont dominants.*

Cependant, un tel résultat n'est pas suffisant pour nous permettre d'obtenir facilement une solution optimale au problème. En effet, celui-ci se révèle être NP-difficile au sens fort quand le nombre de machines est supérieur ou égal à trois.

Théorème. 1 *[Garey et al.,76]. Le problème $Fm||C_{max}$ est NP-difficile au sens fort dès que $m \geq 3$.*

Etant donné la dominance des ordonnancements de permutation dans le cas de trois machines, le résultat de complexité précédent est également valable si l'on se restreint au flowshop de permutation $Fm_{\pi}||C_{max}$. Néanmoins, il existe des cas particuliers pour lesquels le problème devient polynomial. Nous présentons ici les plus connus.

- Cas particulier 1 : pour $m = 3$, si $min_i\{p_{i,1}\} \geq max_i\{p_{i,2}\}$ alors on peut utiliser la règle de Johnson sur un problème à deux machines fictives où les durées du travail j sont données par $p'_{j,1} = p_{j,1} + p_{j,2}$ et $p'_{j,2} = p_{j,2} + p_{j,3}$. Cette situation correspond au cas où la seconde machine est dominée par la première, c'est-à-dire que la charge sur la seconde machine est toujours plus faible que celle sur la première. On peut alors considérer que la seconde machine n'est pas goulet d'étranglement. Johnson [Johnson,54] mentionne ce cas particulier sous la forme $\forall i, j, p_{i,1} \geq p_{j,2}$.

- Cas particulier 2 : pour $m = 3$, si $min_i\{p_{i,3}\} \geq max_i\{p_{i,2}\}$ alors on retrouve une situation similaire à la précédente, à ceci près que la seconde machine est cette fois dominée par la troisième. La méthode de résolution précédente reste valable.

- Cas particulier 3 : pour $m = 3$, si $\forall j, p_{j,2} \leq min\{p_{j,1}, p_{j,3}\}$ alors la méthode précédente reste valable [Burns et Rooker,78], [Szwarc,77].

- Cas particulier 4 : pour un nombre de machines m quelconque, telles que pour tout indice $k, 1 \leq k \leq m-1$, la machine $k+1$ domine la machine k (série croissante de machines dominantes), un ordonnancement optimal est obtenu à partir d'une séquence (a, σ) où le travail a est tel que $\forall j, \sum_{1 \leq k \leq m-1} p_{j,k} \geq \sum_{1 \leq k \leq m-1} p_{a,k}$ et où σ désigne une séquence quelconque des $n-1$ travaux restants [Ho et Gupta,95].

- Cas particulier 5 : de manière symétrique, pour une série décroissante de m machines dominantes, un ordonnancement optimal est obtenu à partir d'une séquence (σ', b) où le travail b est tel que $\forall j, \sum_{2 \leq k \leq m} p_{j,k} \geq \sum_{2 \leq k \leq m} p_{b,k}$ et où σ' désigne une séquence quelconque des $n-1$ travaux restants [Ho et Gupta,95].

On peut trouver encore d'autres cas particuliers polynomiaux dans [Burns et Rooker,75], [Burns et Rooker,76], [Szwarc,74], [Achugbue et Chin,82].

2.3.2 Minimisation de la somme pondérée des dates de fin sur les machines

En reprenant les idées de la propriété 1, on démontre facilement les théorèmes suivants :

Théorème. 2 *L'algorithme de Johnson fournit un ordonnancement optimal pour le problème* $F2||\sum w_k MC_k$.

Preuve. La preuve découle directement de l'application de la propriété 1.

Théorème. 3 $F3||\sum MC_k$ *est NP-difficile au sens fort.*

Eléments de preuve. Dans ce cas, on ne peut pas appliquer la propriété 1. La propriété 2 nous donne le résultat de complexité pour le cas pondéré, mais pas pour le cas non pondéré. Néanmoins, il est possible d'utiliser une preuve similaire à celle qui est employée pour la minimisation du makespan (voir par exemple [Tanaev et al.,94]). Celle-ci fait appel à une réduction du problème classique 3-PARTITION, et construit un ordonnancement pour lequel les dates de fin sur les machines 1 et 2 sont respectivement égales à leurs bornes inférieures $\sum_{1 \leq j \leq n} p_{j,1}$ et à $min\{p_{i,1}/1 \leq i \leq n\} + \sum_{1 \leq j \leq n} p_{j,2}$. La réduction est donc également valable pour le critère $\sum MC_k$.

2.4 Problèmes de flowshop avec contraintes d'attentes minimales ($\theta_{j,k}^{max} = +\infty$)

Dans cette section, nous nous limitons à la prise en compte de time lags minimaux, c'est-à-dire que nous supposons que tous les time lags maximaux sont infinis ($\forall j, k, \theta_{j,k}^{max} = +\infty$). Un nombre assez important d'articles dans la littérature sont consacrés à ce type de problèmes, qui peuvent être assimilés à des problèmes de flowshop avec des temps de transport entre les machines [Brucker et al.,04a].

Avant toute chose, il convient de remarquer que les ordonnancements de permutation ne sont plus dominants, même dans le cas de deux machines, et de durées unitaires. Il suffit pour cela de considérer l'exemple très simple qui suit :

Exemple. Soient $n = 2$ travaux de durées unitaires sur les $m = 2$ machines, dont les time lags minimaux sont respectivement $\theta_1^{min} = 2$ et $\theta_2^{min} = 0$. L'ordonnancement S correspondant aux séquences $(1, 2)$ sur la première machine et $(2, 1)$ sur la seconde a une durée totale de 4 (voir figure 2.3a). Cette valeur est égale à la borne inférieure $max\{p_{j,1} + \theta_{j,1}^{min} + p_{j,2}/1 \leq j \leq n\}$ donc S est optimal. Or, les deux ordonnancements de permutation possibles, S_1 et S_2, associés respectivement aux permutations $(1, 2)$ et $(2, 1)$ ont des makespans égaux à 5 (figure 2.3b,c). L'ensemble des ordonnancements de permutation n'est donc pas dominant pour cet exemple.

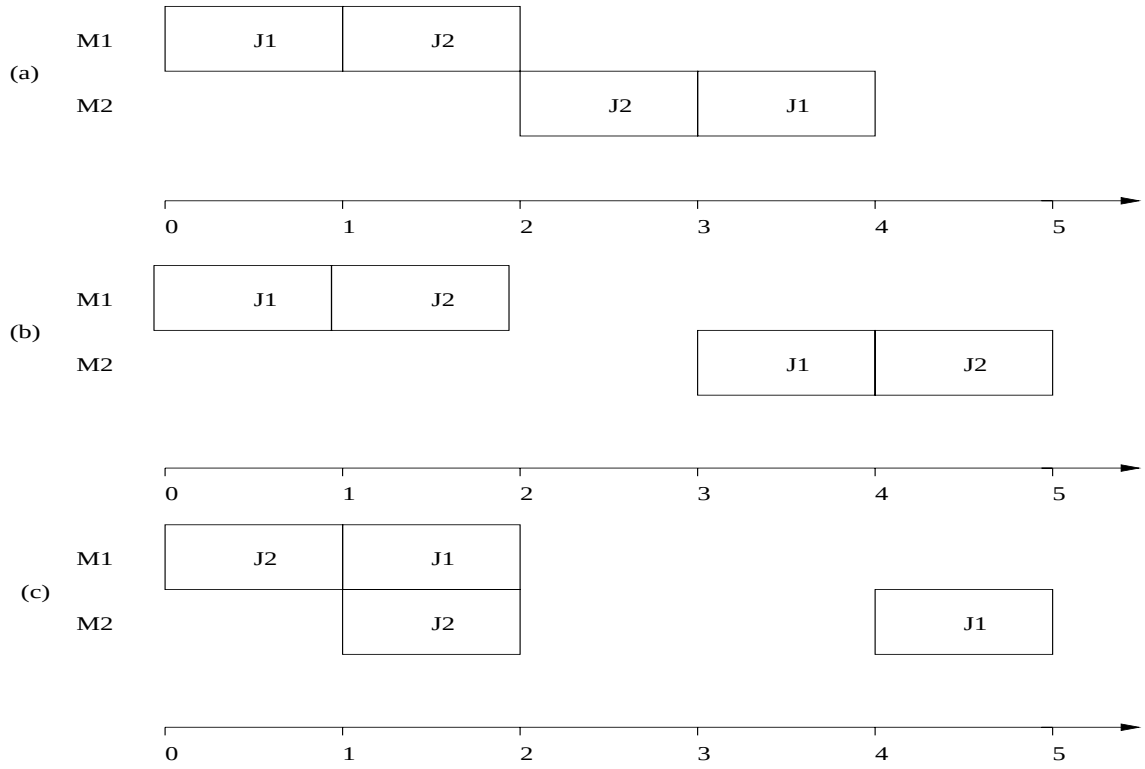

FIG. 2.3 – Non dominance des ordonnancements de permutation

Nous sommes donc amenés à distinguer les problèmes de flowshops de permutation et les problèmes de flowshops généraux, dès $m = 2$ machines. Les résultats de la section suivante vont d'ailleurs montrer que la complexité de ces problèmes n'est pas nécessairement la même.

2.4.1 Minimisation du makespan dans le cas de deux machines

Un premier cas particulier à considérer est celui pour lequel tous les time lags minimaux sont égaux ($\forall j, \theta_j^{min} = \theta$). On peut alors facilement démontrer que ce problème est équivalent au problème classique sans time lags en décalant toutes les opérations sur la seconde machine de θ [Brucker et al.,04a]. L'algorithme de Johnson [Johnson,54] est par conséquent optimal pour ce problème, que l'on se restreigne aux solutions de permutation ou pas.

Supposons maintenant que les time lags minimaux ne sont pas tous égaux. Le problème restreint aux ordonnancements de permutation, $F2_\pi|\theta_j^{min}|C_{max}$ a été étudié dès la fin des années

50 par Mitten [Mitten,59]. Celui-ci a montré que ce problème peut être résolu par une extension de l'algorithme de Johnson [Johnson,54], que nous désignons comme étant l'algorithme de Mitten-Johnson. Cet algorithme consiste à appliquer la règle de Johnson à un problème à deux machines et n travaux dont les durées opératoires sont données par : $\forall j, p'_{j,1} = p_{j,1} + \theta_j^{min}$ et $p'_{j,2} = p_{j,2} + \theta_j^{min}$ (les durées artificielles sont égales aux durées réelles auxquelles on ajoute le time lag).

Si au contraire on considère le problème général $F2|\theta_j^{min}|C_{max}$, le résultat précédent n'est plus valable, et le problème devient NP-difficile au sens fort [Lawler et al.,93], [Dell'Amico,96]. Yu et al. [Yu et al.,04] précisent ce résultat en montrant que le problème est toujours NP-difficile au sens fort si toutes les durées opératoires sont unitaires ($F2|p_{j,k} = 1, \theta_j^{min}|C_{max}$). Ce dernier problème, qui paraît pourtant extrêmement simple (une instance est uniquement caractérisée par les valeurs des time lags minimaux) illustre l'influence des contraintes d'écarts temporels sur la complexité des problèmes. D'autre part, si l'on considère cette fois que les durées opératoires ne dépendent que des travaux mais pas des machines, et que les time lags ne peuvent prendre que deux valeurs, le problème $F2|p_{j,1} = p_{j,2}, \theta_j^{min} \in \{\theta_1, \theta_2\}|C_{max}$ reste également NP-difficile au sens fort [Yu,96].

Néanmoins, il existe des cas particuliers pour lequels les ordonnancements de permutation sont dominants, et qui conduisent donc à des problèmes polynomiaux [Dell'Amico,96] :

- Cas 1 : $max\{\theta_j^{min}/1 \leq j \leq n\} \leq min\{p_{j,1} + \theta_j^{min}/1 \leq j \leq n\}$
- Cas 2 : $max\{\theta_j^{min}/1 \leq j \leq n\} \leq min\{\theta_j^{min} + p_{j,2}/1 \leq j \leq n\}$

Dans sa thèse, Yu [Yu,96] fournit une condition plus générale qui assure la dominance des ordonnancements de permutation : $\forall i, j, \theta_i^{min} \leq \theta_j^{min} + max\{p_{j,1}, p_{j,2}\}$. Sous cette condition, le problème peut donc être résolu en $O(nlog(n))$ grâce à l'algorithme de Mitten-Johnson décrit précédemment.

2.4.2 Autres problèmes avec contraintes d'attentes minimales

La minimisation du makespan dans un flowshop général avec time lags minimaux étant NP-difficile au sens fort, même avec deux machines et des durées unitaires, nous nous intéressons dans cette partie au flowshop de permutation. Notons tout d'abord que la propriété mentionnée précédemment dans le cas à deux machines et concernant les problèmes avec time lags minimaux de même valeur se généralise pour un nombre quelconque de machines : si les time lags ne dépendent que des machines et pas des travaux ($\forall j, j', k, \theta_{j,k}^{min} = \theta_{j',k}^{min}$), alors on est ramené à un problème sans time lags.

Par ailleurs, le problème $F3_\pi|\theta_{j,k}^{min}|C_{max}$ est NP-difficile au sens fort, puisque le problème équivalent sans time lags l'est déjà [Garey et al.,76]. De la même manière, $F2_\pi|\theta_j^{min}|\sum C_j$, $F2_\pi|\theta_{j,k}^{min}, r_j|C_{max}$ et $F2_\pi|\theta_{j,k}^{min}|L_{max}$ sont également NP-difficiles au sens fort [Garey et al.,76], [Lenstra et al.,77]. En fait, si l'on regarde de plus près la complexité des problèmes de flowshop [Brucker et Knust], on remarque que seule la minimisation du makespan dans le cas de deux machines peut être résolue avec un algorithme polynomial, à moins de se restreindre à des durées de traitement égales, voire unitaires. C'est pour cette raison que nous allons nous focaliser sur de

tels cas particuliers. Bien que cette restriction puisse sembler excessive et très peu représentative de la réalité, l'étude de problèmes d'ordonnancement avec durées opératoires identiques peut s'avérer utile. D'une part, si les durées réelles sont très homogènes, c'est-à-dire de valeurs très proches, une approximation consistant à les considérer comme égales peut donner une idée du résultat ou de certains choix. D'autre part, on peut éventuellement rencontrer de tels problèmes après relaxation de certaines contraintes, lors de l'établissement de bornes inférieures par exemple.

a) Propriété des flowshops de permutation à durées identiques et time lags minimaux

Considérons dans un premier temps un problème de flowshop de permutation à deux machines, avec durées identiques et time lags minimaux et sans autre contrainte ($F2_\pi|p_{j,k} = p, \theta_j^{min}|\gamma$). Soit π une permutation des travaux : $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. Le lemme suivant fournit une formule pour le calcul de la date de fin de chaque travail, lorsque la séquence des travaux sur les deux machines est donnée par π.

Lemme. 1 *Si π désigne la séquence des travaux pour un problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\gamma$, alors la date de fin sur la deuxième machine du travail placé en position i s'exprime de la manière suivante :*
$C_{\pi(i),2} = max_{1 \leq u \leq i}\{\theta_{\pi(u)}^{min}\} + (i+1)p$

Preuve. Nous utilisons une preuve par récurrence sur l'indice i pour démontrer ce résultat. Puisque les ordonnancements calés à gauche sont dominants et que les seules contraintes considérées sont les contraintes de time lags minimaux, il est possible d'ordonnancer les travaux sur la première machine entre les dates 0 et n, sans temps mort. Ainsi, $\forall i, C_{\pi(i),1} = i.p$.

Le premier travail $\pi(1)$ se termine à la date p sur la première machine, il est donc disponible à la date $p + \theta_{\pi(1)}^{min}$ sur la deuxième machine. Puisqu'il s'agit du premier travail à traiter sur la deuxième machine, sa date de fin sur cette machine est $C_{\pi(1),2} = \theta_{\pi(1)}^{min} + 2p$. Donc la formule est vérifiée au rang 1.

Supposons qu'elle le soit également au rang i, $1 \leq i \leq n-1$. De même que précédemment, pour le travail en position $i+1$ la contrainte de time lag impose que $t_{\pi(i+1),2} \geq (i+1)p + \theta_{\pi(i+1)}^{min}$. D'autre part, la contrainte de précédence avec $\pi(i)$ conduit à $t_{\pi(i+1),2} \geq C_{\pi(i),2}$. En utilisant l'hypothèse de récurrence on trouve que la date de fin sur la deuxième machine du travail en position $i+1$ est $C_{\pi(i+1),2} = max\{(i+1)p + \theta_{\pi(i+1)}^{min}, max_{1 \leq u \leq i}\{\theta_{\pi(u)}^{min}\} + (i+1)p\} + p = max_{1 \leq u \leq i+1}\{\theta_{\pi(u)}^{min}\} + (i+2)p$. La formule est donc vraie au rang $i+1$. □

De manière analogue, on pourrait montrer que :

- pour les problèmes de type $F3_\pi|p_{j,k} = 1, \theta_{j,k}^{min}|\gamma$, la date de fin sur la machine 3 du travail placé en position i est donnée par $C_{\pi(i),3} = max_{1 \leq u \leq v \leq i}\{\theta_{\pi(u),1}^{min} + \theta_{\pi(v),2}^{min}\} + (i+2)p$ (la date de fin sur la machine 2 étant la même que précédemment)

- pour les problèmes de type $F2_\pi|p_{j,k} = 1, r_j, \theta_j^{min}|\gamma$, la date de fin sur la machine 2 du travail placé en position i est donnée par $C_{\pi(i),2} = max_{1 \leq u \leq v \leq i}\{r_{\pi(u)} - u.p + \theta_{\pi(v)}^{min}\} + (i+2)p$

A partir des formules que nous venons d'établir, il est possible de construire des algorithmes polynomiaux pour résoudre de façon optimale plusieurs problèmes.

b) Règle fournissant l'ordonnancement optimal pour le problème $F3_\pi|p_{j,k}=p,\theta_{j,k}^{min}|C_{max}$

Théorème. 4 *Une permutation optimale pour le problème* $F3_\pi|p_{j,k}=p,\theta_{j,k}^{min}|C_{max}$ *est obtenue en exécutant les travaux dans l'ordre croissant des time lags entre les deux premières machines* ($\theta_{j,1}^{min}$). *Une seconde permutation optimale (éventuellement identique à la précédente) consiste à ordonnancer les travaux dans l'ordre décroissant des time lags entre les deux dernières machines* ($\theta_{j,2}^{min}$). *La valeur optimale correspondante est donnée par* $C_{max}^{\star}=max_{1\leq j\leq n}\{\theta_{j,1}^{min}+\theta_{j,2}^{min}\}+(n+2)p$.

Preuve. Soit π une séquence minimisant le makespan. D'après la formule établie précédemment, $C_{max}(\pi)=C_{\pi(n),3}=max_{1\leq u\leq v\leq n}\{\theta_{\pi(u),1}^{min}+\theta_{\pi(v),2}^{min}\}+(n+2)p$. Supposons qu'il existe un indice i, $1\leq i\leq n-1$, tel que $\theta_{\pi(i),1}^{min}>\theta_{\pi(i+1),1}^{min}$. Soit $\tilde{\pi}$ la séquence obtenue à partir de π en échangeant $\pi(i)$ et $\pi(i+1)$ (c'est-à-dire $\forall h\neq i,i+1,\tilde{\pi}(h)=\pi(h)$, $\tilde{\pi}(i)=\pi(i+1)$ et $\tilde{\pi}(i+1)=\pi(i)$). Le makespan associé à $\tilde{\pi}$ s'exprime par $C_{max}(\tilde{\pi})=max_{1\leq u\leq v\leq n}\{\theta_{\tilde{\pi}(u),1}^{min}+\theta_{\tilde{\pi}(v),2}^{min}\}+(n+2)p$. Soient r et s ($r\leq s$) les indices correspondant respectivement à u et v et maximisant l'expression précédente (c'est-à-dire $C_{max}(\tilde{\pi})=\theta_{\tilde{\pi}(r),1}^{min}+\theta_{\tilde{\pi}(s),2}^{min}+(n+2)p$). Nous allons distinguer les cas selon les valeurs prises par r et s :

- Cas 1 : si $r\leq i-1$, alors $C_{max}(\tilde{\pi})\leq max_{r\leq v\leq n}\{\theta_{\pi(r),1}^{min}+\theta_{\pi(v),2}^{min}\}+(n+2)p\leq C_{max}(\pi)$
- Cas 2 : si $r\geq i+2$, alors $s\geq i+2$ et $C_{max}(\tilde{\pi})=\theta_{\pi(r),1}^{min}+\theta_{\pi(s),2}^{min}+(n+2)p\leq C_{max}(\pi)$
- Cas 3 : si $r=i+1$, alors $s\geq i+1$ et $C_{max}(\tilde{\pi})\leq max_{i\leq v\leq n}\{\theta_{\pi(i),1}^{min}+\theta_{\pi(v),2}^{min}\}+(n+2)p\leq C_{max}(\pi)$
- Cas 4 : si $r=i$ et $s\neq i+1$, alors $C_{max}(\tilde{\pi})\leq max_{i+1\leq v\leq n}\{\theta_{\pi(i+1),1}^{min}+\theta_{\pi(v),2}^{min}\}+(n+2)p\leq C_{max}(\pi)$
- Cas 5 : si $r=i$ et $s=i+1$, on ne retrouve pas le terme correspondant parmi les combinaisons à considérer pour le calcul de $C_{max}(\pi)$. Cependant, puisque $\theta_{\pi(i),1}^{min}>\theta_{\pi(i+1),1}^{min}$, $C_{max}(\tilde{\pi})=\theta_{\pi(i+1),1}^{min}+\theta_{\pi(i),2}^{min}+(n+2)p<\theta_{\pi(i),1}^{min}+\theta_{\pi(i),2}^{min}+(n+2)p\leq C_{max}(\pi)$.

Ainsi, dans tous les cas, $C_{max}(\tilde{\pi})\leq C_{max}(\pi)$ et $\tilde{\pi}$ est donc également une séquence optimale. Il est possible de répéter une telle modification jusqu'à aboutir à une séquence dans laquelle les travaux sont rangés dans l'ordre croissant des $\theta_{j,1}^{min}$. Soit $\pi^{\star}$ une telle permutation. Supposons que $C_{max}(\pi^{\star})=\theta_{f,1}^{min}+\theta_{g,2}^{min}+(n+2)p$, où f précède g dans $\pi^{\star}$. Par définition de $\pi^{\star}$, $\theta_{f,1}^{min}\leq\theta_{g,1}^{min}$ donc $C_{max}(\pi^{\star})\leq\theta_{g,1}^{min}+\theta_{g,2}^{min}+(n+2)p$. D'après la formule du makespan, $C_{max}(\pi^{\star})=max_{1\leq u\leq v\leq n}\{\theta_{\pi^{\star}(u),1}^{min}+\theta_{\pi^{\star}(v),2}^{min}\}+(n+2)p\geq max_{1\leq j\leq n}\{\theta_{j,1}^{min}+\theta_{j,2}^{min}\}+(n+2)p$. Par conséquent,$C_{max}(\pi^{\star})=max_{1\leq j\leq n}\{\theta_{j,1}^{min}+\theta_{j,2}^{min}\}+(n+2)p$.

Une démonstration similaire permet de montrer que toute permutation dans laquelle les travaux sont rangés dans l'ordre décroissant des $\theta_{j,2}^{min}$ est optimale. □

c) Règles fournissant l'ordonnancement optimal pour les problèmes $F2_\pi|p_{j,k}=p,r_j,\theta_j^{min}|C_{max}$ *et* $F2_\pi|p_{j,k}=p,\theta_j^{min}|L_{max}$

Il est possible d'établir des résultats du même type que précédemment pour les problèmes $F2_\pi|p_{j,k}=p,r_j,\theta_j^{min}|C_{max}$ et $F2_\pi|p_{j,k}=p,\theta_j^{min}|L_{max}$, comme l'attestent les théorèmes suivants.

Théorème. 5 *Une permutation optimale pour le problème* $F2_\pi|p_{j,k}=p,r_j,\theta_j^{min}|C_{max}$ *est obtenue en exécutant les travaux dans l'ordre croissant de leur date de disponibilité* r_j.

Eléments de preuve. A partir de la formule démontrée précédemment, un argument d'échange de travaux consécutifs, analogue à celui utilisé pour démontrer le théorème précédent, est suffisant. Nous omettons ici les détails.

Théorème. 6 *Une permutation optimale pour le problème* $F2_\pi|p_{j,k} = p, \theta_j^{min}|L_{max}$ *est obtenue en exécutant les travaux dans l'ordre croissant de leur date de fin souhaitée* d_j.

Preuve. Le plus grand retard L_{max} peut s'exprimer sous la forme $L_{max} = max_{1\leq j\leq n}\{C_{\pi(j),2} - d_{\pi(j)}\} = max_{1\leq i\leq j\leq n}\{\theta^{min}_{\pi(i),1} - d_{\pi(j)} + j.p\} + p$. De nouveau, on peut utiliser un argument d'échange de travaux consécutifs.

D'autre part, ce problème peut également être considéré comme symétrique par rapport au problème précédent $F2_\pi|p_{j,k} = p, r_j, \theta_j^{min}|C_{max}$. En effet, une technique classique en ordonnancement consiste à inverser l'ordre des machines. Les dates de disponibilité r_j deviennent alors des durées de latence (ou queues) q'_j. Or, la date de fin d'un travail j, calculée en prenant en compte cette durée de latence, est égale au retard de ce travail si l'on remplace cette durée de latence par une date de fin souhaitée $d''_j = -q'_j$. Du fait de la définition choisie pour les time lags (de fin à début), leur valeur n'est pas affectée lorsque l'on inverse l'ordre des machines. L'ordre croissant des r_j dans le premier problème avec des dates de disponibilité est équivalent à l'ordre décroissant des q'_j dans le problème avec des durées de latence (puisque l'on inverse l'ordre des machines) donc à l'ordre croissant des d''_j dans le problème avec des dates de fin souhaitées. Ainsi, cette transformation permet également de prouver le théorème. □

d) Algorithme exact pour le problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$

En ce qui concerne le problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$, nous proposons un algorithme proche de celui de Moore-Hodgson [Moore,68], qui est optimal pour le problème $1||\sum U_j$. Nous rappelons ici son principe de fonctionnement :

Algorithme. Moore-Hodgson [Moore,68]
0- Soit $J = \{1, 2, \ldots, n\}$
1- Exécuter les travaux de J dans l'ordre EDD, i.e. dans l'ordre croissant de leur date de fin souhaitée d_j.
2- Si aucun travail n'est en retard alors STOP. Sinon aller en 3.
3- Trouver le premier travail en retard. Soit h sa position dans la séquence.
4- Parmi les h premiers travaux, trouver celui avec la plus grande durée de traitement. Retirer ce travail de J et le placer en fin de séquence.
5- Si $J = \emptyset$ alors STOP. Sinon aller en 2.
La séquence finale minimise le critère $\sum U_j$.

Avant de présenter notre algorithme pour le problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$, nous établissons deux lemmes préliminaires définissant des conditions de dominance. Sans perte de généralité, supposons que les travaux sont numérotés dans l'ordre croissant de leur time lag θ_j^{min} et, à time lags égaux, dans l'ordre croissant de leur date de fin souhaitée d_j : $\forall i, j$ si $i < j$ alors $(\theta_i^{min}, d_i) \leq_{lex} (\theta_j^{min}, d_j)$, où $\leq_{lex}$ désigne la relation d'ordre lexicographique sur les couples. Pour un ordonnancement donné, nous notons E l'ensemble des travaux non en retard, L l'ensemble des travaux en retard et N l'ensemble des travaux qui n'ont pas encore été placés : $E = \{j/C_{j,2} \leq d_j\}$ et $L = \{j/C_{j,2} > d_j\} = \{1, \ldots, n\} \setminus E$. Enfin, $SG(J)$ désigne la permutation des travaux de l'ensemble J dans laquelle les travaux sont placés dans l'ordre croissant de

leurs dates de fin souhaitées (en cas d'égalité dans l'ordre de leurs indices) et $A(J)$ désigne une séquence arbitraire des travaux de J.

Lemme. 2 *Les ordonnancements où tous les travaux de E précèdent tous les travaux de L sont dominants pour le problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$.*

Preuve. Un argument d'échange de travaux consécutifs suffit pour démontrer ce résultat. Soit S un ordonnancement ne vérifiant pas la condition précédente. Alors il existe deux travaux consécutifs i et j tels que i précède j dans S, i est en retard et j n'est pas en retard. En échangeant les positions de i et j, on avance l'exécution de j, qui se termine toujours avant sa date de fin souhaitée, et on retarde i, qui était déjà en retard. Les autres travaux sont inchangés. Donc le nouvel ordonnancement est au moins aussi bon que S. On peut répéter cette transformation jusqu'à obtenir une solution telle que tous les travaux de E précèdent tous les travaux de L.□

Il est possible de restreindre encore davantage l'ensemble des ordonnancements dominants.

Lemme. 3 *Pour le problème $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$, il existe un ordonnancement optimal dans lequel les travaux de E sont exécutés dans l'ordre croissant de leur date de fin souhaitée (en cas d'égalité, dans l'ordre de leur indice), puis les travaux de L dans un ordre quelconque.*

Preuve. De nouveau, il est possible d'utiliser un argument d'échange de travaux consécutifs. Nous omettons les détails de la preuve.

Désormais, nous nous limitons à des solutions vérifiant les conditions précédentes. Considérons maintenant l'algorithme suivant :

Algorithme. A1.
0- Initialiser les ensembles E, L et N : $E = \emptyset$, $L = \emptyset$ et $N = \{1, 2, \ldots, n\}$.
1- Pour i allant de 1 à n répéter
1-1- Considérer l'ordonnancement partiel correspondant à $SG(E \cup \{i\})$.
1-2- Si tous les travaux respectent leur date de fin souhaitée, alors $E = E \cup \{i\}$. Sinon $L = L \cup \{i\}$.
1-3- $N = N \setminus \{i\}$
2- Retourner l'ordonnancement final $SG(E) \oplus A(L)$, où $\oplus$ désigne l'opérateur de concaténation.

Théorème. 7 *L'algorithme A1 fournit une solution optimale pour le problème de flowshop $F2_\pi|p_{j,k} = p, \theta_j^{min}|\sum U_j$ en $O(n^2)$ opérations.*

Preuve. Notons tout d'abord que la solution S obtenue en appliquant l'algorithme proposé vérifie les conditions des lemmes précédents.

Nous employons une démonstration par récurrence sur l'indice i de la boucle de l'algorithme pour prouver le théorème. L'hypothèse de récurrence associée au rang i est la suivante :

H_i : il existe une solution optimale $S^\star(i)$ telle que tous les travaux d'indice inférieur ou égal à i ont le même statut (en retard / non en retard) dans S et dans $S^\star(i)$. Autrement dit : $\forall h \leq i, U_h = U_h^\star$.

Vérifions si cette hypothèse est vraie pour $i = 1$. Le travail 1 est placé dans S lors de la première itération, sans les autres travaux.

- Cas 1 : Si le travail 1 est en retard dans S, alors il est nécessairement en retard dans toute solution, en particulier dans une solution optimale.
- Cas 2 : Le travail 1 n'est pas en retard dans S. Considérons une solution $\sigma^\star$ optimale. Si le travail 1 respecte sa date de fin souhaitée dans cette solution, alors la condition est vérifiée. Supposons que ce ne soit pas le cas. La solution est de la forme $SG(E) \oplus A(L)$ avec $SG(E) = \sigma_1 \oplus \sigma_2$, où σ_1 (respectivement σ_2) est une séquence de travaux j tels que $d_j < d_1$ (respectivement $d_j \geq d_1$). Considérons la solution S^{star} définie à partir de $\sigma^\star$, mais dans laquelle on a échangé les positions de 1 et du dernier travail de σ_1, que l'on note x. Par définition, $\theta_1^{min} \leq \theta_x^{min}$ donc $C_{1,2}(S^\star) \leq C_{x,2}(\sigma^\star)$ et le travail 1 ne retarde pas les travaux suivants dans $S^\star$. D'autre part, puisque x appartient à σ_1 dans $\sigma^\star$, $d_x \leq d_1$. Etant donné que x n'est pas en retard dans $\sigma^\star$, le travail 1 n'est pas en retard dans $S^\star$. Par conséquent, $\sum U_j(S^\star) \leq \sum U_j(\sigma^\star)$. Donc $S^\star$ est une solutions optimale vérifiant la condition de l'hypothèse H_1.

Supposons que l'hypothèse de récurrence soit vraie jusqu'au rang i, $i \leq n-1$. Au moment de placer le travail $i+1$ dans S, les i premiers travaux sont déjà séquencés. Nous distinguons alors trois cas :

- Cas 1 : lors de l'ajout de $i+1$ à l'étape 1-1, le travail $i+1$ est en retard dans $SG(E \cup \{i+1\})$, sans retarder d'autre travail. Alors nécessairement, tous les travaux de $E \cup \{i+1\}$ ne peuvent simultanément respecter leur date de fin souhaitée. D'après l'hypothèse de récurrence H_i, il existe une solution optimale $S^\star(i)$ dans laquelle tous les travaux de E respectent leur date de fin souhaitée. Donc $i+1$ est forcément en retard dans $S^\star(i)$. Par conséquent, $S^\star(i)$ est une solution optimale vérifiant la condition de l'hypothèse H_{i+1}.
- Cas 2 : lors de l'ajout de $i+1$ à l'étape 1-1, un travail j de E est retardé au-delà de sa date de fin souhaitée. De même que dans le cas précédent, tous les travaux de $E \cup \{i+1\}$ ne peuvent simultanément respecter leur date de fin souhaitée et on peut trouver une solution optimale vérifiant la condition de l'hypothèse H_{i+1}.
- Cas 3 : lors de l'ajout de $i+1$ à l'étape 1-1, aucun travail de $E \cup \{i+1\}$ n'est en retard. En procédant de la même manière que pour le travail 1 lors de l'établissement de l'hypothèse H_1, on peut montrer qu'une solution optimale $S^\star(i)$ vérifiant la condition de l'hypothèse H_i et dans laquelle $i+1$ est en retard peut être transformée en une solution $S^\star(i+1)$ vérifiant la condition de l'hypothèse H_{i+1}.

Finalement, l'hypothèse de récurrence est vraie pour le rang n. La seule solution $S^\star(n)$ telle que $\forall h \leq n, U_h^\star = U_h$ est la solution S fournie par l'algorithme. Celui-ci est donc optimal.

Par ailleurs, cet algorithme comporte n itérations, chacune d'elles nécessitant la construction d'un ordonnancement partiel de taille inférieure ou égale à n, et autant de tests sur les dates de fin souhaitées. Sa complexité est donc en $O(n^2)$. □

L'algorithme présenté fait appel à deux relations d'ordres distinctes : une première définie par les time lags, pour le classement initial des travaux, qui détermine l'ordre dans lequel ils seront considérés, et une seconde définie par les dates de fin souhaitées, pour le placement des travaux dans l'ordonnancement. L'exemple qui suit illustre le déroulement de l'algorithme et met en évidence le rôle de ces deux relations d'ordres.

Exemple. Considérons le problème à 4 travaux, de durées identiques unitaires, dont les time lags et les dates de fin souhaitées sont donnés dans le tableau 2.1.

On peut d'abord s'assurer que les travaux sont bien numérotés dans l'ordre croissant de leur time lag. Lors des 3 premières étapes, l'algorithme considère successivement les 3 premiers

Travail	$\theta_{j,1}^{min}$	d_j
1	0	13
2	2	13
3	4	13
4	10	12

TAB. 2.1 – Exemple d'instance pour $F2_\pi|p_{j,k}=1,\theta_j^{min}|\sum U_j$

travaux, et construit les ordonnancements partiels correspondant aux séquences (1), (1, 2) et (1, 2, 3) dans lesquels tous les travaux respectent leur date de fin souhaitée. Ainsi, ces 3 travaux sont placés dans l'ensemble E. A la dernière itération, le travail 4 est pris en compte. Dans la séquence $SG(1,2,3,4)$, il est placé en première position car sa date de fin souhaitée est la plus petite : $SG(1,2,3,4) = (4,1,2,3)$. Dans l'ordonnancement correspondant, les travaux 2 et 3 sont en retard. Le travail 4 est alors placé dans L et l'algorithme renvoie la séquence $(1,2,3,4)$, pour laquelle seul le travail 4 est en retard. En fait, si l'on impose que le travail 4 ne soit pas en retard, celui-ci doit nécessairement être exécuté en premier, et seul un autre travail parmi les 3 travaux restants peut être traité de manière à respecter sa date de fin souhaitée. Il y aura forcément deux travaux en retard, ce qui n'est pas optimal.

D'une manière plus générale, un travail avec un time lag de faible valeur n'a pas d'influence sur les travaux ayant des time lags plus grands, tandis qu'un travail avec un grand time lag peut empêcher les travaux suivants d'être terminés à temps. C'est la raison pour laquelle il est important de considérer les travaux dans l'ordre croissant de leurs time lags.

e) Résultats concernant la minimisation de la somme des dates de fin sur les machines

Nous terminons cette section par les problèmes de flowshop avec time lags minimaux où le critère est la somme (éventuellement pondérée) des dates de fin sur les machines.

Théorème. 8 *Nous avons les résultats de complexité suivants :*

- $F2_\pi|\theta_j^{min}|\sum w_k MC_k$ *est polynomial et peut être résolu avec l'algorithme de Mitten-Johnson [Mitten,59], proposé pour le problème* $F2_\pi|\theta_j^{min}|C_{max}$
- $F2|p_{j,k}=1,\theta_j^{min}|\sum MC_k$ *est NP-difficile au sens fort.*
- $F2|p_{j,1}=p_{j,2},\theta_j^{min}\in\{\theta_1,\theta_2\}|\sum MC_k$ *est NP-difficile au sens fort.*
- $F2_\pi|r_j,p_{j,k}=p,\theta_j^{min}|\sum w_k MC_k$ *est polynomial et peut être résolu avec l'algorithme que nous avons proposé pour le problème* $F2_\pi|r_j,p_{j,k}=p,\theta_j^{min}|C_{max}$.
- $F3_\pi|p_{j,k}=p,\theta_{j,k}^{min}|\sum w_k MC_k$ *est polynomial et peut être résolu avec l'algorithme que nous avons proposé pour le problème* $F3_\pi|p_{j,k}=p,\theta_{j,k}^{min}|C_{max}$.

Preuve. Les trois premiers problèmes vérifient la condition de la propriété 1. On retrouve donc les mêmes résultats de complexité que pour la minimisation du makespan. Concernant les deux derniers problèmes, il est possible d'utiliser un argument similaire à celui de la preuve de la propriété 1, avec des valeurs particulières pour les dates de fin sur les premières machines :

- pour le problème $F2_\pi|r_j,p_{j,k}=p,\theta_j^{min}|\sum w_k MC_k$, $MC_1 = max_{1\leq j\leq n}\{r_j+(n+1-j)p\}$, en supposant que les r_j sont indicés dans l'ordre croissant

– pour le problème $F3_\pi|p_{j,k} = p, \theta_{j,k}^{min}| \sum w_k MC_k$, $MC_1 = n.p$ et $MC_2 = max_{1 \leq j \leq n}\{\theta_{j,1}^{min} + (n+1)p$

Ces valeurs sont issues des formules établies pour les problèmes de minimisation du makespan dans les flowshops de permutation avec durées identiques et time lags minimaux. □

Tous ces résultats permettent de mieux identifier la frontière entre problèmes polynomiaux, dont la résolution ne pose, en règle générale, pas de difficultés dans la pratique, et problèmes NP-difficiles, pour lesquels il convient d'élaborer des méthodes de résolution astucieuses et efficaces.

2.5 Problèmes de flowshop avec contraintes d'attentes maximales ($\theta_{j,k}^{min} = 0$)

Dans cette section, nous ne considérons que des time lags maximaux, c'est-à-dire que nous supposons que tous les time lags minimaux sont nuls ($\forall j,k, \theta_{j,k}^{min} = 0$). Les problèmes avec time lags maximaux se révèlent être des cas intermédiaires entre deux situations extrêmes : les problèmes classiques, sans time lags maximaux ($\forall j,k, \theta_{j,k}^{max} = +\infty$) et les problèmes sans attente ($\forall j,k, \theta_{j,k}^{max} = 0$) pour lesquels les opérations successives des travaux doivent se succéder immédiatement, sans temps d'attente. Dans le cas de la minimisation du makespan dans un flowshop à deux machines, ces deux problèmes extrêmes peuvent être résolus optimalement par des algorithmes polynomiaux.

2.5.1 Problèmes de flowshop sans attente

a) Minimisation du makespan

Nous avons vu dans la section 2.3.1 que l'algorithme de Johnson [Johnson,54] est optimal pour le problème $F2||C_{max}$. Le problème sans attente $F2|no-wait|C_{max}$ peut quant à lui être résolu par l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64]. En fait, ce problème peut se mettre sous la forme d'un problème de voyageur de commerce (Traveling Salesman Problem ou TSP) dont les distances vérifient des propriétés particulières. Au même titre que l'algorithme de Johnson, que nous avons rappelé dans la section 2.3.1, nous présentons également l'algorithme de Gilmore et Gomory.

Algorithme. Gilmore et Gomory.

On suppose que les travaux sont numérotés dans l'ordre croissant des durées sur la machine 2 ($\forall j \leq n-1, p_{j,2} \leq p_{j+1,2}$).

0- Ajouter un travail artificiel 0 dont les durées opératoires $p_{0,1}$ et $p_{0,2}$ sont nulles (les travaux sont toujours numérotés dans l'ordre croissant des durées sur la machine 2).

1- Trouver une permutation ϕ de $\{0, 1, \ldots, n\}$ telle que $\forall j \leq n-1, p_{\phi(j),1} \leq p_{\phi(j+1),1}$. Définir, pour $0 \leq j \leq n$, $\gamma_j = min(p_{j,2}, p_{\phi(j),1})$ et $\delta_j = max(p_{j,2}, p_{\phi(j),1})$.

2- Construire un graphe G à $n+1$ sommets où chaque sommet est associé à un travail. Il existe un arc entre les sommets i et j si et seulement si $i = \phi(j)$ ou $\phi(i) = j$.

3- Si G est connexe, alors $\psi = \phi$ et aller à l'étape 13. Sinon aller à l'étape 4.

4- Construire le graphe G', sans arc, dans lequel chaque sommet correspond à une composante connexe de G.

5- Pour i allant de 0 à $n-1$ faire

5-1- Si les sommets i et $i+1$ n'appartiennent pas aux mêmes composantes connexes dans G, alors relier les sommets de G' correspondant par un arc a_i de poids $max(\gamma_{i+1} - \delta_i, 0)$.

6- Déterminer un arbre de recouvrement de poids minimal de G'. Définir deux graphes intialement vides G_1 et G_2.

7- Pour tout arc a_i de cet arbre, faire

7-1- Si $p_{\phi(i),1} \geq p_{i,2}$, alors ajouter l'arc a_i à G_1. Sinon ajouter l'arc a_i à G_2.

8- Classer les indices i des arcs a_i de G_1 dans l'ordre décroissant en une liste L_1.

9- Classer les indices i des arcs a_i de G_2 dans l'ordre croissant en une liste L_2.

10- Concaténer les deux listes en une liste $L = L_1 \oplus L_2$. Définir la permutation $\sigma = (0, 1, \ldots, n)$.

11- Pour h allant de 1 à $|L|$ (longueur de la liste L) faire

11-1- Echanger dans σ les éléments occupant les positions $L(h)$ et $L(h)+1$.

12- Déterminer la permutation ψ en appliquant ϕ à la permutation obtenue : $\psi = (\phi(\sigma(0)), \phi(\sigma(1)), \ldots, \phi(\sigma(n))$.

13- $\pi(0) = 0$. Pour i allant de 0 à $n-1$ faire

13-1- $\pi(i+1) = \psi(\pi(i))$.

14 Retirer le travail 0 au début de la permutation π.

π est la séquence de traitement des travaux sur les deux machines.

Remarques concernant l'algorithme :

- L'idée sous-jacente à cet algorithme est d'essayer de minimiser les temps morts sur chaque machine, en affectant les travaux de la manière suivante : le successeur du travail avec la plus petite durée sur la deuxième machine est le travail avec la plus petite durée sur la première machine, et ainsi de suite. Il procède ensuite à des échanges éventuels de travaux en minimisant le "coût" de ces échanges.
- L'ajout du travail fictif permet de se ramener à un cycle, au lieu d'une séquence classique : en effet, ce travail est en quelque sorte le prédécesseur du premier travail réel et le successeur du dernier travail.
- L'algorithme de Prim [Prim,57] ou l'algorithme de Kruskal [Kruskal,56] permettent de trouver rapidement un arbre de recouvrement de poids minimal (étape 6). L'algorithme de Prim peut également être utilisé pour tester la connexité du graphe (étape 3).
- Vairaktarakis [Vairaktarakis,03] propose une simplification de l'algorithme de Gilmore et Gomory. Le nouvel algorithme proposé fait appel à des cycles hamiltoniens complémentaires pour représenter le problème de voyageur de commerce, et utilise des règles de couplage simples exploitant la structure d'une solution optimale. Ainsi, les échanges d'éléments pour "corriger" la permutation ne sont plus nécessaires.

On peut facilement vérifier que l'algorithme précédent est polynomial. La minimisation du makespan dans un flowshop sans attente à deux machines est donc un problème que l'on sait efficacement résoudre. Par contre, la plupart des autres problèmes de flowshop sans attente sont NP-difficiles au sens fort : $F2|r_j, no-wait|C_{max}$ [Roeck,84b], $F3|no-wait|C_{max}$ [Roeck,84a], $F2|no-wait|L_{max}$ [Roeck,84b] et $F2|no-wait|\sum C_j$ [Roeck,84b].

b) Minimisation de la somme pondérée des dates de fin sur les machines

Si nous considérons le problème $F2|no-wait|\sum w_k MC_k$, nous pouvons établir le résultat suivant :

Théorème. 9 *$F2|no-wait|\sum w_k MC_k$ peut être résolu de manière optimale en appliquant l'algorithme de Gilmore et Gomory à n problèmes de voyageur de commerce.*

Preuve. Soit π une permutation quelconque des travaux. Supposons que ceux-ci soient traités sur les machines 1 et 2 dans l'ordre $\pi(1), \pi(2), \ldots, \pi(n)$. Les dates de fin sur les machines 1 et 2 s'expriment de la manière suivante :

$MC_1 = C_{\pi(n),1} = \sum_{1 \leq i \leq n} p_{i,1} + \sum_{1 \leq i \leq n-1} max(0, p_{\pi(i),2} - p_{\pi(i+1),1})$

$MC_2 = C_{\pi(n),2} = \sum_{1 \leq i \leq n} p_{i,1} + \sum_{1 \leq i \leq n-1} max(0, p_{\pi(i),2} - p_{\pi(i+1),1}) + p_{\pi(n),2}$

Ainsi, la valeur du critère est donnée par :

$\sum w_k MC_k = (w_1 + w_2)(\sum_{1 \leq i \leq n} p_{i,1}) + \sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)[p_{\pi(i),2} - p_{\pi(i+1),1}]) + w_2 p_{\pi(n),2}$.

La première partie de cette expression $(w_1+w_2)(\sum_{1 \leq i \leq n} p_{i,1})$ ne dépend pas de la permutation π. L'objectif revient donc à minimiser la partie restante :

$\sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)[p_{\pi(i),2} - p_{\pi(i+1),1}]) + w_2 p_{\pi(n),2}$.

Dans le cas de la minimisation du makespan, la valeur de la fonction objectif serait $C_{max} = C_{\pi(n),2}$ et le problème serait équivalent à minimiser $\sum_{1 \leq i \leq n-1} max(0, p_{\pi(i),2} - p_{\pi(i+1),1}) + p_{\pi(n),2}$. Ce dernier problème peut être considéré comme un problème de voyageur de commerce avec $n+1$ villes $0, 1, \ldots, n$ où chaque ville i correspond au travail i et où la ville 0 correspond à un travail fictif de durées nulles sur les deux machines. La distance entre la ville i et la ville j est donnée par $d_{i,j} = max(0, p_{i,2} - p_{j,1})$. On retrouve l'équivalence entre le problème sans attente et ce voyageur de commerce particulier que nous avons mentionnée en présentant l'algorithme de Gilmore et Gomory.

Dans le cas qui nous intéresse ici, nous introduisons n problèmes de voyageur de commerce, chacun d'eux correspondant au sous-problème dans lequel le dernier travail de la permutation est fixé. Soit h l'indice du problème de voyageur de commerce ($1 \leq h \leq n$). Le problème TSP_h est défini par $n+1$ villes $0, 1, \ldots, n$, chacune d'elles étant associée à deux valeurs $P_{i,1}$ et $P_{i,2}$:

$\forall i, 1 \leq i \leq n, i \neq h, P_{i,1} = (w_1 + w_2)(p_{i,1} + L), P_{i,2} = (w_1 + w_2)(p_{i,2} + L)$

$P_{h,1} = (w_1 + w_2)(p_{h,1} + L), P_{h,2} = (w_1 + w_2)p_{h,2}$

$P_{0,1} = w_1 p_{h,2}, P_{0,2} = 0$

où L est un entier très grand (par exemple, $L = 2\sum_{1 \leq i \leq n}(p_{i,1} + p_{i,2})$).

La distance entre la ville i et la ville j est donnée par $D_{i,j} = max(0, P_{i,2} - P_{j,1})$.

En raison de la très grande valeur de L, nous pouvons montrer qu'une solution optimale du problème contient nécessairement l'arc $(h, 0)$.

Supposons que T est une solution optimale qui ne contient pas l'arc $(h, 0)$. Alors T contient un arc $(j, 0)$ avec $j \neq h$. Puisque toutes les distances sont positives ou nulles, la longueur totale $C(T)$ de T est supérieure ou égale à la distance entre j et 0, c'est-à-dire :

$C(T) \geq max(0, P_{j,2} - P_{0,1}) = (w_1 + w_2)(p_{j,2} + L) - w_1 p_{h,2} > (w_1 + w_2)\sum_{1 \leq i \leq n} p_{i,2}$.

Soit $U = (u(0), u(1), \ldots, u(n), u(0))$ une solution contenant l'arc $(h, 0)$. Sans perte de généralité, on peut supposer que $u(0) = 0$ et $u(n) = h$. Alors on a :

$C(U) = max(0, P_{u(0),2} - P_{u(1),1}) + \sum_{1 \leq i \leq n-1} max(0, P_{u(i),2} - P_{u(i+1),1}) + max(0, P_{u(n),2} - P_{u(0),1}) = max(0, 0 - (w_1 + w_2)(p_{u(1),1} + L)) + \sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)(p_{u(i),2} - p_{u(i+1),1})) + max(0, w_2 p_{h,2}) = \sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)(p_{u(i),2} - p_{u(i+1),1})) + w_2 p_{h,2}$.

Par conséquent, $C(U) \leq (w_1 + w_2)(\sum_{1 \leq i \leq n} p_{i,2}) < C(T)$ ce qui contredit l'optimalité de T. Ainsi, une solution optimale du problème TSP_h doit contenir l'arc $(h, 0)$. De plus, la quantité $\sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)(p_{u(i),2} - p_{u(i+1),1}))$ doit être minimale.

Si nous gardons la solution de plus petite distance totale parmi toutes les solutions optimales des n problèmes TSP_h, nous obtenons une solution $S = (0, \pi(1), \pi(2), \ldots, \pi(n), 0)$ qui minimise l'expression :

$\sum_{1 \leq i \leq n-1} max(0, (w_1 + w_2)(p_{\pi(i),2} - p_{\pi(i+1),1}) + w_2 \times p_{\pi(n),2}$. La permutation correspondante $\pi = \pi(1), \pi(2), \ldots, \pi(n)$ constitue une solution optimale pour le problème de flowshop initial. On peut remarquer que le fait d'imposer l'arc $(h, 0)$ dans une solution du problème de voyageur de commerce correspond au placement de h en dernière position sur les machines. □

2.5.2 Problèmes à deux machines avec deux types de travaux

Avant de considérer le cas de time lags maximaux strictement positifs, il peut être intéressant de s'attarder sur une situation intermédiaire, correspondant à un problème de minimisation du makespan dans un flowshop à deux machines, entre le flowshop classique, qu'on peut résoudre avec l'algorithme de Johnson [Johnson,54] et le flowshop sans attente, qu'on peut résoudre avec l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64]. Supposons que deux types de travaux coexistent : les travaux dits "réguliers", pour lesquels il n'y a pas de contrainte d'attente maximale ($\theta_j^{max} = +\infty$) et les travaux sans attente ($\theta_j^{max} = 0$). Finke et al. [Finke et al.,02] étudient ce problème, en formulant les contraintes non pas en termes de time lags, mais en termes d'espace de stockage. Les travaux "réguliers" sont en fait des produits de petite taille que l'on peut stocker entre les machines successives, tandis que les travaux sans attente ont une taille trop importante pour pouvoir être immobilisés entre deux machines. Les auteurs montrent que le problème est NP-difficile au sens fort, qu'il s'agisse de la version générale ($F2|\theta_j^{max} \in \{0, +\infty\}|C_{max}$) ou de la version restreinte aux ordonnancements de permutation ($F2_\pi|\theta_j^{max} \in \{0, +\infty\}|C_{max}$). Pour ce dernier problème, le résultat a également été démontré par Bouquard [Bouquard,04]. Celui-ci considère aussi les cas où le nombre de travaux d'un type donné est borné, et aboutit aux résultats suivants :

- Si le nombre de travaux sans attente est quelconque et que le nombre de travaux réguliers est borné par q, le problème peut être résolu de manière optimale en appliquant $O(n^q)$ fois l'algorithme de Gilmore et Gomory.
- Si le nombre de travaux réguliers est quelconque et qu'il y a un nombre donné s de travaux sans attente, le problème est NP-difficile au sens ordinaire, mais un algorithme pseudo-polynomial permet de le résoudre.

2.5.3 Problèmes avec contraintes d'attentes maximales positives

Dans cette partie, nous nous focalisons sur la minimisation du makespan et nous supposons qu'il existe au moins un time lag maximal de valeur finie strictement positive ($\exists j, k, 0 < \theta_{j,k}^{max} < +\infty$). Yang et Chern [Yang et Chern,95] montrent que dans ce cas, le problème du flowshop de permutation à deux machines est NP-difficile au sens ordinaire. Néanmoins, la preuve qu'ils utilisent fait appel à des travaux de durée nulle sur la deuxième machine, et les contraintes de time lags ne sont pas prises en compte pour ces travaux. La démonstration est donc un peu ambiguë. Nous revenons sur la complexité des problèmes à deux machines par la suite.

a) Résultats de dominance

Une des premières questions qui se posent est celle de la dominance éventuelle des solutions de permutation. L'exemple suivant nous montre que la réponse à cette question est négative.

	Travail 1	Travail 2	Travail 3	Travail 4
$p_{j,1}$	5	2	5	8
$p_{j,2}$	3	10	5	2
θ_j^{max}	1	3	8	3

TAB. 2.2 – Durées et time lags de l'exemple

Exemple.
Soit un ensemble de 4 travaux à ordonnancer sur deux machines. Les durées opératoires, ainsi que les time lags maximaux sont fournis dans le tableau 2.2.

Si nous considérons la séquence $\sigma 1 = (2, 3, 1, 4)$ sur la première machine et la séquence $\sigma 2 = (2, 1, 3, 4)$ sur la seconde machine, l'ordonnancement associé $S^\star$ a un makespan $C_{max}(S^\star) = 22$. Cette valeur est atteinte par une borne inférieure $\sum_{1 \leq j \leq n} p_{j,1} + min_{1 \leq i \leq n}\{p_{i,2}\}$, donc $S^\star$ est optimal (Figure 2.4.a).

On peut facilement s'assurer que tout ordonnancement optimal commence par le travail 2 sur la première machine et se termine par le travail 4 sur la seconde machine. Si ce n'est pas le cas, tout ordonnancement S, pour lequel le premier travail traité sur la première machine n'est pas le travail 2, a un makespan $C_{max}(S) \geq min_{i \neq 2}\{p_{i,1}\} + \sum_{1 \leq j \leq n} p_{j,2} = 25$ et n'est donc pas optimal. De manière analogue, un ordonnancement dont le dernier travail sur la deuxième machine n'est pas le travail 4 ne peut pas être optimal.

Supposons maintenant qu'il existe un ordonnancement de permutation qui soit optimal. D'après les conditions établies précédemment, seuls deux ordonnancements $S1$ et $S2$ sont possibles, correspondant aux permutations $\pi 1 = (2, 1, 3, 4)$ et $\pi 2 = (2, 3, 1, 4)$ respectivement. A cause du time lag du travail 1, on a $C_{max}(S1) = 26$ et $C_{max}(S2) = 26$ (Figure 2.4.b et 2.4.c).

Ainsi, les ordonnancements de permutation ne sont pas dominants pour cet exemple.

Pour les problèmes comptant jusqu'à 3 travaux, les ordonnancements de permutation sont dominants, comme le montre le résultat suivant.

Propriété. 5 *Pour le problème $F2|\theta_j^{max}|C_{max}$, les ordonnancements où le même travail est placé en première position sur les deux machines sont dominants. De même, les ordonnancements où le même travail est placé en dernière position sur les deux machines sont dominants.*

Preuve. Nous présentons la preuve pour la première partie de la propriété, elle est similaire pour la seconde partie. Soit S un ordonnancement optimal pour une instance donnée de $F2|\theta_j^{max}|C_{max}$. Soit h le travail séquencé en premier sur la seconde machine dans S. Supposons que h n'est pas le premier travail à être traité sur la première machine (Figure 2.5.a). Nous construisons l'ordonnancement S' de la manière suivante : sur la première machine, le travail h est inséré en première position et tous les travaux qui le précédaient sont décalés vers la droite de $p_{h,1}$ (Figure 2.5.b).

Si le temps d'attente entre les deux opérations du travail h dépasse le time lag maximal $\theta_{h,1}^{max}$, on décale vers la gauche la seconde opération de h de telle sorte que la contrainte de time lag soit finalement respectée. Toutes les autres opérations sur la seconde machine restent inchangées (Figure 2.5.c).

On peut facilement vérifier que les contraintes de précédence pour les travaux qui ont été décalés sont toujours satisfaites. Par conséquent, on obtient un ordonnancement optimal ayant

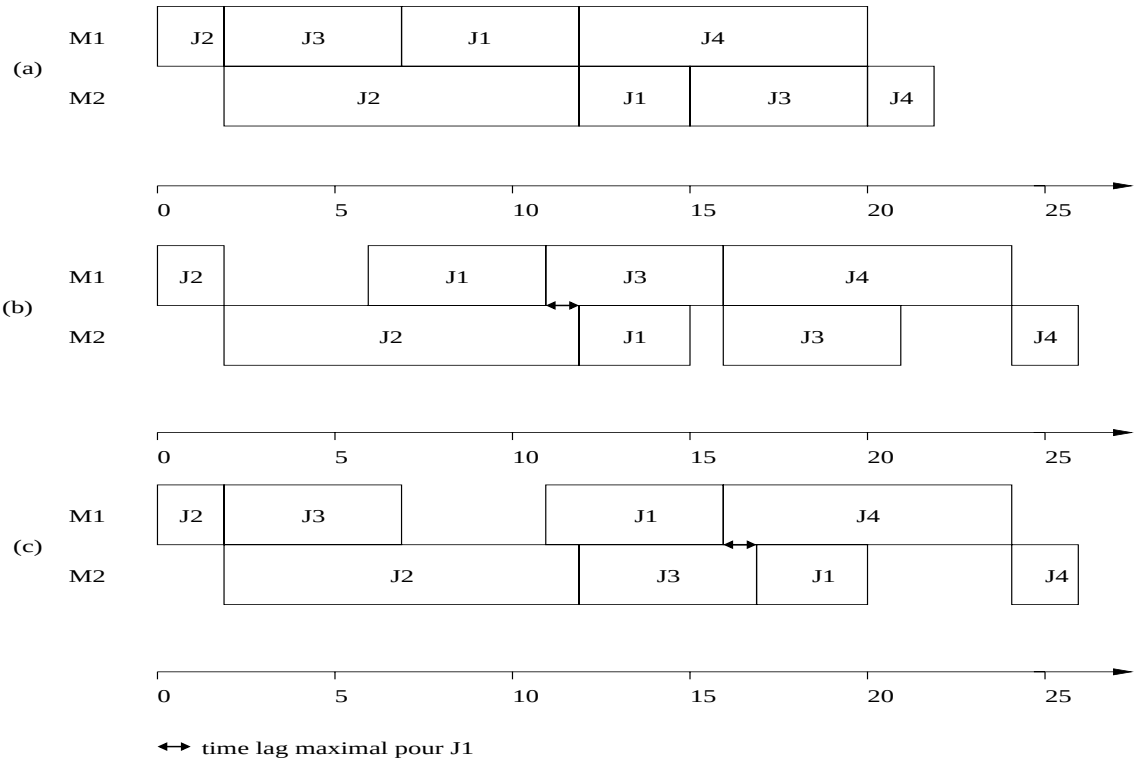

FIG. 2.4 – Non dominance des ordonnancements de permutation

le même travail placé en première position sur les deux machines. □

Tout comme pour les time lags minimaux, il existe des cas particuliers de problèmes de flowshop à deux machines avec time lags maximaux pour lesquels les solutions de permutation sont dominantes.

Propriété. 6 *Dans le cas où* $max_{1\leq j\leq n}\{\theta_j^{max}\} < min_{1\leq j\leq n}\{p_{j,1}+p_{j,2}\}$*, seuls les ordonnancements de permutation respectent les contraintes de time lags. Ils sont donc dominants.*

Preuve. Supposons que dans un ordonnancement S, le travail i précède le travail j sur la première machine et le travail j précède le travail i sur la seconde machine. On a alors les inégalités suivantes :

1) $t_{j,1} \geq C_{i,1}$
2) $t_{i,2} \geq C_{j,2} = t_{j,2} + p_{j,2}$ et
3) $t_{j,2} \geq C_{j,1} = t_{j,1} + p_{j,1}$.

Le temps d'attente pour le travail i est $t_{i,2} - C_{i,1} \geq t_{j,2} + p_{j,2} - t_{j,1} \geq p_{j,1} + p_{j,2}$. Or, par hypothèse, $t_{i,2} - C_{i,1} > max_{1\leq r\leq n}\{\theta_r^{max}\} \geq \theta_i^{max}$. Par conséquent, la contrainte de time lag maximal pour le travail i n'est pas respectée et l'ordonnancement S n'est pas réalisable. □

b) Résultats de complexité

Nous allons maintenant montrer que le problème $F2_\pi|\theta_j^{max}|C_{max}$ est NP-difficile au sens fort.

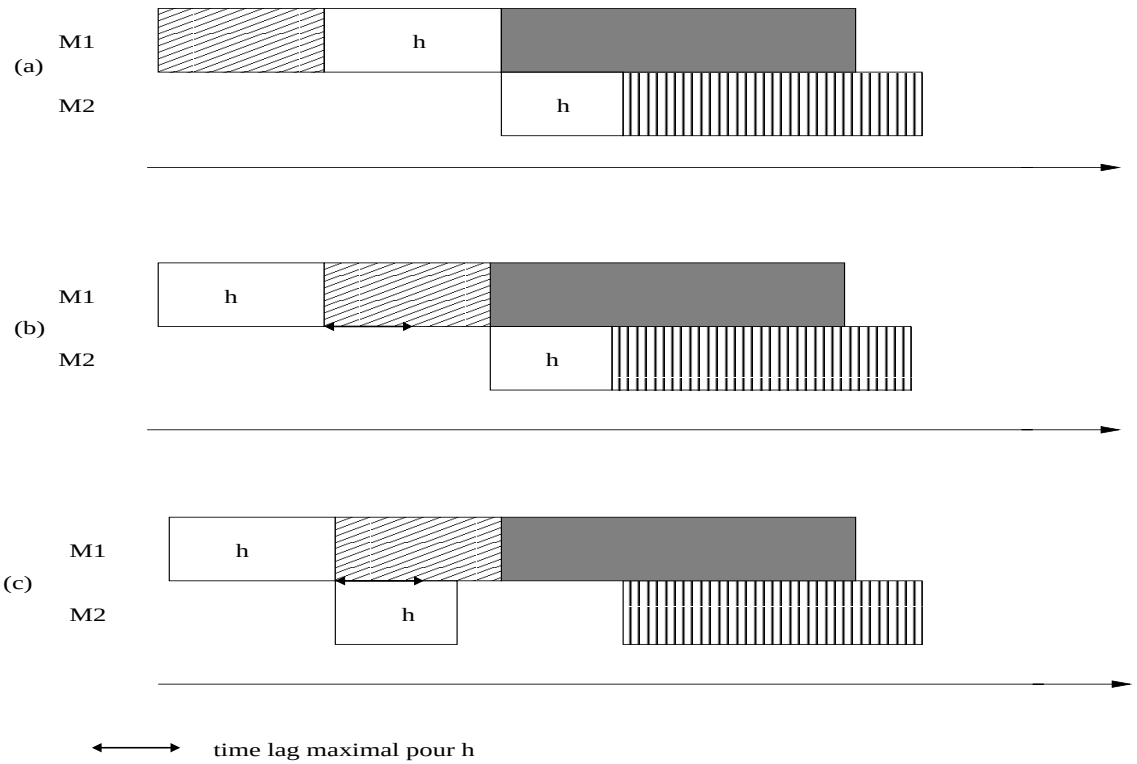

FIG. 2.5 – Illustration de la propriété 5

Théorème. 10 *Le problème de minimisation du makespan dans un flowshop de permutation à deux machines en présence de time lags maximaux de même valeur* $F2_\pi|\theta_j^{max} = \theta|C_{max}$ *est NP-difficile au sens fort.*

Preuve. Considérons le problème sous sa forme de problème de décision $F2_\pi|\theta_j^{max} = \theta|C_{max} \leq Z$. Celui-ci est clairement dans la classe NP. Nous allons utiliser une réduction polynomiale du problème 3-PARTITION, qui est connu pour être NP-complet au sens fort [Garey et Johnson,79].

3-PARTITION : Etant donnés un entier W et un ensemble N de $3n$ entiers positifs $\{a_1, \ldots, a_{3n}\}$, tels que $\frac{W}{4} < a_j < \frac{W}{2}$, $1 \leq j \leq 3n$ et $\sum_{j=1}^{3n} a_j = nW$, existe-t-il une partition $N_1, \ldots, N_n$ de l'ensemble N telle que $|N_k| = 3$ et $\sum_{a_j \in N_k} a_j = W$, $1 \leq k \leq n$?

A partir d'une instance de 3-PARTITION, nous construisons une instance de notre problème $F2_\pi|\theta_j^{max} = \theta|C_{max} \leq Z$ avec $4n+1$ travaux de la manière suivante :

$\forall j, \theta_j^{max} = \frac{W}{2}$

$Z = 2nW + 3W/4$

Nous allons montrer que 3-PARTITION a une solution si et seulement s'il existe un ordonnancement réalisable S pour le problème $F2_\pi|\theta_j^{max} = \theta|C_{max} \leq Z$.
Supposons que 3-PARTITION a une solution, définie par les sous-ensembles $N_1, \ldots, N_n$. Nous construisons un ordonnancement réalisable S pour notre problème de la façon suivante : le L-job J_{4n+1} est placé en premier sur les deux machines, suivi par les travaux associés aux éléments de

O-Jobs : $p_{j,1} = a_j, \quad p_{j,2} = \frac{W}{2}, \quad j = 1, \ldots, 3n$

E-Jobs : $p_{j,1} = W, \quad p_{j,2} = \frac{W}{2}, \quad j = 3n+1, \ldots, 4n$

L-Job : $p_{4n+1,1} = \frac{W}{4}, \quad p_{4n+1,2} = \frac{W}{2},$

N_1. Ces O-jobs sont exécutés sur la première machine durant la période $[W/4, W + W/4]$ et sur la seconde machine pendant $[3W/4, 2W + W/4]$. J_{3n+1} est ensuite placé, sans attente entre les deux machines. D'une manière générale, les O-jobs correspondant au sous-ensemble N_i passent sur la première machine entre $(2i-2)W + W/4$ et $(2i-1)W + W/4$ et sur la seconde machine entre $(2i-2)W + 3W/4$ et $2iW + W/4$, suivis par J_{3n+i} qui se termine à $2iW + W/4$ sur la première machine et commence aussitôt sur la seconde machine, de sorte qu'il se termine à la date $2iW + 3W/4$. Ainsi, la durée totale de l'ordonnancement S est $2nW + 3W/4$. Cet ordonnancement est représenté sur la figure 2.6.

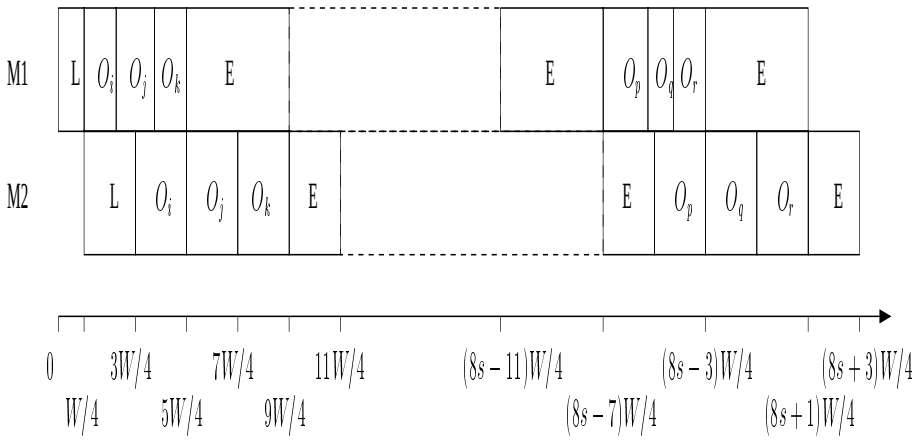

FIG. 2.6 – Ordonnancement S

Réciproquement, supposons qu'il existe un ordonnancement réalisable S pour l'instance du problème $F2_\pi|\theta_j^{max} = \theta|C_{max} \leq Z$ définie précédemment. Puisque la durée totale de traitement de tous les travaux sur la première machine est $2nW + W/4$, la durée maximale des temps morts cumulés sur cette machine jusqu'à la date Z est $W/2$. Cette durée correspond exactement au temps de traitement minimum d'un travail sur la seconde machine. Ainsi, il n'y a pas de temps mort sur la première machine avant la fin du dernier travail. De manière analogue, la durée totale d'exécution de tous les travaux sur la seconde machine est $(4n+1)W/2 = 2nW + W/2$, donc la durée maximale des temps morts cumulés sur cette machine entre 0 et Z est $W/4$. Tous les travaux ont une durée sur la première machine strictement supérieure à cette valeur, sauf le L-job J_{4n+1} dont la durée est égale à $W/4$. La première machine doit donc débuter par ce travail et il n'y a pas de temps mort une fois que celui-ci a commencé sur la seconde machine.

Supposons maintenant que le successeur de J_{4n+1} est un E-job. Ce travail va provoquer un temps mort sur la seconde machine d'au moins $W - W/2 = W/2$, ce qui conduit à une contradiction. De manière similaire, si seulement un O-job J_i ou deux O-jobs J_i et J_j sont placés

entre J_{4n+1} et le E-job suivant, cela va provoquer un temps mort sur la seconde machine d'au moins $W + a_i - W = a_i$ dans le premier cas, et $W + a_i + a_j - 3W/2 = a_i + a_j - W/2 > 0$ dans le second cas, ce qui aboutit aussi à une contradiction. Il y a donc au moins trois O-jobs placés immédiatement après J_{4n+1}. En utilisant le même argument pour tous les E-jobs, on peut montrer qu'au moins trois O-jobs doivent être placés entre deux E-jobs consécutifs. Puisque les nombres de O-jobs et E-jobs sont respectivement $3n$ et n, il doit y avoir exactement trois O-jobs entre deux E-jobs consécutifs. On obtient ainsi n groupes N_i, $i = 1, \ldots, n$, composés de trois O-jobs associés aux éléments de N.

Nous démontrons maintenant que les E-jobs sont ordonnancés sans attente entre les deux machines. Supposons qu'il existe un E-job J_k avec un temps d'attente $\delta_k (0 < \delta_k \leq W/2)$. Soit J_j le O-job qui précède J_k dans S. D'après les résultats précédents, il n'y a pas de temps mort sur les machines entre J_j et J_k. D'où $C_{j,2} - W/2 - C_{j,1} = C_{k,1} + \delta_k - W/2 - (C_{k,1} - W) = \delta_k + W/2 > W/2$. La contrainte de time lag maximal pour J_j n'est pas respectée et l'ordonnancement S n'est pas réalisable. Par conséquent, il ne peut pas y avoir de temps d'attente entre les machines pour les E-jobs.

Considérons l'un des groupes N_i : nous désignons les travaux correspondant par J_{i_1}, J_{i_2} et J_{i_3} et le prédécesseur (respectivement successeur) immédiat de la séquence $(J_{i_1}, J_{i_2}, J_{i_3})$ dans S par J_{k_1} (respectivement J_{k_2}). Puisqu'il n'y a pas de temps mort sur les deux machines entre J_{k_1} et J_{k_2}, on a :

$$C_{k_2,1} = W + a_{i_1} + a_{i_2} + a_{i_3} + C_{k_1,1} \text{ et } C_{k_2,2} = 2W + C_{k_1,2}.$$

D'après ce qui précède, J_{k_1} et J_{k_2} sont tous les deux exécutés sans attente entre les machines. D'où : $C_{k_2,2} = C_{k_2,1} + W/2$ et $C_{k_1,2} = C_{k_1,1} + W/2$. On en déduit que $2W + C_{k_1,2} = W + a_{i_1} + a_{i_2} + a_{i_3} + C_{k_1,1} + W/2$. Donc $W = a_{i_1} + a_{i_2} + a_{i_3}$. Ceci est vrai pour tout sous-ensemble $N_i, i = 1, ..., n$.

Ainsi, on obtient une partition de N en n sous-ensembles N_i de trois éléments tels que $\sum_{a_j \in N_i} a_j = W$, $\forall i = 1, ..., n$, ce qui définit une solution au problème 3-PARTITION. □

S'il on considère le problème plus général $F2|\theta_j^{max} = \theta|C_{max}$ sans se restreindre aux ordonnancements de permutation, c'est-à-dire qu'on accepte éventuellement les solutions avec des séquences de travaux différentes sur les deux machines, la démonstration précédente reste valable pour montrer que ce problème est NP-difficile au sens fort. En effet, l'instance construite vérifie $min\{p_{i,1} + p_{i,2}\} = 3W/4 > W/2 = max\{\theta_i^{max}\}$, et d'après la propriété 6, seuls les ordonnancements de permutation sont réalisables. Par conséquent, $F2|\theta_j^{max} = \theta|C_{max}$ est NP-difficile au sens fort.

En outre, l'instance construite pour la réduction de 3-PARTITION est telle que $\forall j, p_{j,2} = \theta_j^{max} = \theta$. Le problème reste donc NP-difficile au sens fort si les durées opératoires sur la seconde machine sont identiques, et de même valeur que les time lags maximaux. Par symétrie, en inversant les rôles des deux machines et l'axe des temps, on obtient le même résultat si les durées opératoires sur la première machine sont identiques, et de même valeur que les time lags maximaux. Ceci est possible en raison de la définition choisie pour les time lags (stop-start time lags).

Corollaire. 1 *Les problèmes $F2|\theta_j^{max} = \theta|C_{max}$, $F2|p_{j,2} = \theta_j^{max} = \theta|C_{max}$ et $F2|p_{j,1} = \theta_j^{max} = \theta|C_{max}$ sont NP-difficiles au sens fort.*

En ce qui concerne le critère $\sum MC_k$, le résultat de complexité est identique à celui associé au makespan. En effet, il est possible d'utiliser la même réduction, en notant que l'ordonnancement construit est sans temps mort sur la première machine (la date de fin sur cette machine est donc égale à la borne inférieure $\sum_{1 \leq j \leq n} p_{j,1}$).

c) Analyse de performance d'heuristiques dans le pire cas

Cette partie concerne plus particulièrement le problème $F2_\pi|\theta_j^{max}|C_{max}$. D'après le théorème 10, ce problème est NP-difficile au sens fort, mais on sait également que les deux cas particuliers extrêmes $F2_\pi||C_{max}$ (obtenu pour $\theta_j^{max} = +\infty, \forall j$) et $F2_\pi|no-wait|C_{max}$ (obtenu pour $\theta_j^{max} = 0, \forall j$) sont résolus de manière exacte par des algorithmes polynomiaux, respectivement celui de Johnson [Johnson,54] et celui de Gilmore et Gomory [Gilmore et Gomory,64].

Pour la résolution du problème plus général $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, nous proposons dans la section 4.4 une Procédure par Séparation et Evaluation, qui est par définition une méthode exacte mais exponentielle. Les heuristiques simples que nous employons pour obtenir une borne supérieure (voir la section 4.4.3) sont bien, quant à elles, polynomiales, mais on ne dispose d'aucune garantie sur la qualité du résultat. Enfin, en ce qui concerne les méthodes arborescentes tronquées présentées dans la section 4.4.4, l'approximation à paramètre ϵ permet effectivement de majorer l'erreur commise, mais l'algorithme reste exponentiel, tandis que le beam search n'offre aucune garantie de performance théorique.

Il peut être intéressant de chercher à développer des heuristiques fonctionnant en temps polynomial et assurant dans le même temps une certaine qualité sur le résultat obtenu. Pour y parvenir, nous proposons d'analyser le comportement des deux algorithmes bien connus que nous avons mentionnés auparavant : l'algorithme de Johnson [Johnson,54] et celui de Gilmore et Gomory [Gilmore et Gomory,64].

Pour uniformiser les notations dans ce qui suit, nous proposons de désigner par $C_H^v(I)$ le makespan de l'ordonnancement obtenu à partir de la séquence issue de l'algorithme H, appliquée à l'instance I, dans le cas où les time lags maximaux sont de type v :

- Les algorithmes considérés sont ceux de Johnson [Johnson,54], désigné par J, de Gilmore et Gomory [Gilmore et Gomory,64], désigné par GG, et un algorithme quelconque fournissant des ordonnancements calés au plus tôt, désigné par Q. La valeur *opt* pour le champ H indique par ailleurs l'optimum.
- Les durées opératoires sont fournies par l'instance I.
- En ce qui concerne les time lags maximaux, nous distinguons trois situations : sans attente (0), sans contraintes ($+\infty$) et quelconques (θ_j).

L'objectif de ce travail est de déterminer des ratios de performance dans le pire cas pour chacun des algorithmes, dans les différentes situations examinées. Il s'agit donc de valeurs théoriques, qui ne préjugent en rien du comportement empirique, en moyenne. Les trois théorèmes énoncés dans les sections qui suivent nous donnent des éléments de réponse à ce sujet.

Comparaison des cas sans attente et sans time lags

Théorème. 11 *Pour une instance donnée d'un problème de flowshop de permutation à deux machines, le makespan optimal dans le cas sans attente ($\forall j, \theta_j^{max} = 0$) est au plus le double du*

makespan optimal dans le cas sans time lags ($\forall j, \theta_j^{max} = +\infty$). De plus, il existe des instances pour lesquelles ce ratio de 2 est atteint asymptotiquement.

Preuve. Considérons une instance quelconque I. Nous allons essayer de déterminer une relation entre le makespan optimal obtenu pour cette instance dans le cas sans attente $C_{opt}^0(I)$ et le makespan optimal dans le cas classique, sans time lags $C_{opt}^{+\infty}(I)$. Ces valeurs correspondent respectivement aux solutions fournies par les algorithmes de Gilmore et Gomory [Gilmore et Gomory,64] et de Johnson [Johnson,54] : $C_{opt}^0(I) = C_{GG}^0(I)$ et $C_{opt}^{+\infty}(I) = C_J^{+\infty}(I)$.

Puisque le cas sans attente est plus contraint que le cas sans time lags, on a trivialement $C_{opt}^{+\infty}(I) \leq C_{opt}^0(I)$. A partir de l'ordonnancement optimal dans le cas sans time lags, on se ramène à une solution sans attente de la manière suivante : supposons que l'ordre de traitement (sur les deux machines) est donné par la séquence $(J(1), J(2), \ldots, J(n))$, et que l'ordonnancement est calé au plus tôt. Le premier travail $J(1)$ est déjà traité sans attente entre les deux machines, donc il est maintenu en l'état. Pour le travail suivant $J(2)$, on est amené à distinguer deux cas :

- Cas 1 : l'opération sur la deuxième machine est calée par rapport à l'opération sur la première machine ($t_{J(2),2} = C_{J(2),1}$). Dans ce cas, le travail est traité sans attente et on peut passer au suivant
- Cas 2 : l'opération sur la deuxième machine est calée par rapport à l'opération du travail précédent sur cette machine ($t_{J(2),2} = C_{J(1),2}$). Dans ce cas, pour se ramener à une situation sans attente, on décale l'opération du travail $J(2)$ sur la première machine d'au plus $p_{J(1),2}$ unités de temps. Par conséquent, l'ordonnancement des travaux suivants est éventuellement décalé d'au plus $p_{J(1),2}$ unités de temps

En procédant de la même manière pour les travaux suivants, ou en utilisant une démonstration par récurrence, on prouve que l'opération du travail $J(i)$ sur la première machine est retardée d'au plus $\sum_{1 \leq h \leq i-1} p_{J(h),2}$ unités de temps, les opérations sur la deuxième machine étant réalisées sans attente. En particulier, le dernier travail est décalé d'au plus $\sum_{1 \leq h \leq n-1} p_{J(h),2}$ unités de temps sur la première machine. La solution sans attente que l'on obtient finalement a donc un makespan de valeur $C \leq C_{opt}^{+\infty}(I) + \sum_{1 \leq h \leq n-1} p_{J(h),2} \leq C_{opt}^{+\infty}(I) + \sum_{1 \leq i \leq n} p_{i,2}$. La somme des durées opératoires sur la seconde machine constituent une borne inférieure pour le makespan, quelle que soit la situation, donc finalement $C \leq 2C_{opt}^{+\infty}(I)$. Puisque la solution construite est sans attente, son makespan est supérieur ou égal au makespan optimal dans le cas sans attente. On en déduit que $C_{opt}^0(I) \leq C \leq 2C_{opt}^{+\infty}(I)$. Ainsi, pour une instance donnée, le makespan optimal dans le cas sans attente est au plus le double du makespan optimal dans le cas sans time lags. L'exemple suivant nous montre que cette borne est atteinte asymptotiquement :

Exemple. Soient $s^2 + 1$ travaux dont les durées sont données dans le tableau 2.3 ($s > 2$). Dans le cas sans attente, le makespan, obtenu par exemple avec l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64] vaut $C_{opt}^0(I) = 2s^3 + 1$, alors que dans le cas sans time lags, l'optimum est de $C_{opt}^{+\infty}(I) = s^3 + s^2 + s$. Le ratio $C_{opt}^0(I)/C_{opt}^{+\infty}(I)$ tend bien vers 2 quand s tend vers l'infini.

Supposons désormais qu'on est dans le cas d'un problème $F2_\pi|\theta_j^{max}|C_{max}$. Quelles que soient les valeurs des times lags maximaux, la solution optimale de ce problème est réalisable pour le problème correspondant sans time lags, et constitue par ailleurs une borne inférieure pour le problème sans attente. D'où : $C_{opt}^{+\infty}(I) \leq C_{opt}^{\theta_j}(I) \leq C_{opt}^0(I)$. On en déduit que pour le problème $F2_\pi|\theta_j^{max}|C_{max}$, on peut déterminer un intervalle $[\lambda, \mu]$ dans lequel se trouve l'optimum et tel que $\mu \leq 2\lambda$. On a ainsi le résultat suivant :

Travail j	$p_{j,1}$	$p_{j,2}$
$i \leq s^2$	1	s
s^2+1	s^3	s

TAB. 2.3 – Durées opératoires pour l'exemple

Théorème. 12 *Pour le problème $F2_\pi|\theta_j^{max}|C_{max}$, l'heuristique HGG consistant à ordonnancer les travaux au plus tôt en respectant la séquence obtenue par l'algorithme de Gilmore et Gomory a un ratio de performance dans le pire cas inférieur ou égal à 2.*

Preuve. La solution optimale dans le cas sans attente constitue une solution approchée pour le problème avec time lags $F2_\pi|\theta_j^{max}|C_{max}$, et son makespan $C^0_{opt}(I)$ est au plus le double du makespan optimal $C^{\theta_j}_{opt}(I)$, d'après ce qui précède. Si, à partir de cette solution, on cale toutes les opérations au plus tôt, compte tenu des contraintes de time lags maximaux, on ne dégrade pas le makespan (qui est un critère régulier). Ainsi, $C^{\theta_j}_{HGG}(I) \leq C^0_{opt}(I) \leq 2.C^{\theta_j}_{opt}(I)$.□

Analyse de performance dans le cas sans attente

Nous nous plaçons désormais dans le cas sans attente ($\forall j, \theta_j^{max} = 0$). Nous avons bien évidemment $C^0_{opt}(I) = C^0_{GG}(I)$ et nous cherchons à déterminer le ratio de performance dans le pire cas d'une heuristique quelconque Q.

Théorème. 13 *Pour toute heuristique Q fournissant des solutions calées au plus tôt, le ratio de performance dans le pire cas pour le problème $F2|no-wait|C_{max}$ est inférieur ou égal à 2. En particulier, si Q est l'algorithme de Johnson [Johnson,54], ce ratio peut être atteint asymptotiquement.*

Preuve. Nous supposons que les ordonnancements résultant de Q sont calés au plus tôt, compte tenu des contraintes sans attente. Alors on vérifie aisément que $C^0_Q(I) \leq \sum_{1\leq i\leq n}(p_{i,1}+p_{i,2}) = \sum_{1\leq i\leq n} p_{i,1} + \sum_{1\leq i\leq n} p_{i,2}$. Chacun de ces deux termes constitue une borne inférieure pour le makespan optimal donc finalement : $C^0_Q(I) \leq 2C^0_{opt}(I)$. L'exemple suivant prouve que le ratio est atteint asymptotiquement pour l'algorithme de Johnson [Johnson,54].

Exemple. Soient $n = 4$ travaux dont les durées sont données dans le tableau 2.4. On suppose $s > 2$.

Travail j	$p_{j,1}$	$p_{j,2}$
1	1	s
2	s	2
3	2	s^2
4	s^2	1

TAB. 2.4 – Durées opératoires pour l'exemple

L'algorithme de Gilmore et Gomory [Gilmore et Gomory,64] conduit à la séquence $(1,2,3,4)$ et le makespan correspondant (optimal) est de $C^0_{opt}(I) = C^0_{GG}(I) = s^2+s+4$. En appliquant l'algorithme de Johnson [Johnson,54], on obtient l'ordonnancement respectant la séquence $(1,3,2,4)$

et de valeur $C_J^0(I) = 2s^2 + s + 2$. Lorsque s tend vers l'infini, le ratio $C_J^0(I)/C_{opt}^0(I)$ tend bien vers 2. □

Analyse de performance dans le cas sans time lags

Nous terminons cette partie en considérant le cas classique sans time lags ($\forall j, \theta_j^{max} = +\infty$). Nous avons bien évidemment $C_{opt}^{+\infty}(I) = C_J^{+\infty}(I)$ et nous cherchons à déterminer le ratio de performance dans le pire cas d'une heuristique quelconque Q.

Théorème. 14 *Pour toute heuristique Q fournissant des solutions calées au plus tôt, le ratio de performance dans le pire cas pour le problème $F2||C_{max}$ est inférieur ou égal à 2. En particulier, si Q est l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64], ce ratio peut être atteint asymptotiquement.*

Preuve. Nous supposons que les ordonnancements résultant de Q sont calés au plus tôt, compte tenu des contraintes sans attente. De la même manière que dans le cas précédent : $C_Q^{+\infty}(I) \leq \sum_{1 \leq i \leq n}(p_{i,1} + p_{i,2}) \leq 2C_{opt}^{+\infty}(I)$. L'exemple suivant prouve qu'il est atteint asymptotiquement pour l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64].

Exemple. Soient $s+1$ travaux dont les durées opératoires sont données dans le tableau 2.5. On suppose que $s > 2$.

Travail j	$p_{j,1}$	$p_{j,2}$
1	1	s^2
$2 \leq i \leq s$	i	$s^2 - s + i - 1$
$s+1$	s^3	s

TAB. 2.5 – Durées opératoires pour l'exemple

L'algorithme de Johnson [Johnson,54] conduit à la séquence $(1, 2, \ldots, s, s+1)$ et le makespan correspondant (optimal) est de $C_{opt}^{+\infty}(I) = C_J^{+\infty}(I) = s^3 + 1/2s^2 + 3/2s$.

En appliquant l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64], on obtient l'ordonnancement respectant la séquence $(1, s+1, 2, 3, \ldots, s)$ et de valeur $C_{GG}^{+\infty}(I) = 2s^3 - 3/2s^2 + 3/2s + 1$. Lorsque s tend vers l'infini, le ratio $C_{GG}^{+\infty}(I)/C_{opt}^{+\infty}(I)$ tend bien vers 2.□

2.6 Problèmes de flowshop avec contraintes d'attentes minimales et maximales

Les problèmes impliquant à la fois des time lags minimaux et des time lags maximaux étant plus généraux que les versions correspondantes avec uniquement un type de time lags, les résultats de NP-complétude établis dans les sections précédentes se transfèrent directement à ces problèmes. Néanmoins, une situation particulière mérite qu'on s'y attarde.

Il s'agit du cas où, pour chaque travail j, les time lags minimaux et maximaux sont égaux ($\forall j, k, \theta_{j,k}^{min} = \theta_{j,k}^{max}$). Ce cas, que nous désignons comme étant un problème avec time lags exacts, fait l'objet d'une étude particulière dans la section 4.7. En réalité, il peut être considéré comme

une généralisation du cas sans attente : en effet, tout comme dans ce dernier cas, la succession des opérations consécutives au sein des travaux est très fortement contrainte puisque la durée entre la fin de la première opération et le début de la seconde est imposée. Simplement, la valeur imposée à cette attente n'est pas forcément nulle. Une autre différence réside dans la question de la dominance des solutions de permutation. Il est clair que dans le cas sans attente, les ordonnancements de permutation sont dominants, puisque ce sont les seuls qui respectent les contraintes. Au contraire, dans le cas de time lags exacts, cette propriété n'est plus vérifiée. Pour s'en convaincre, il suffit de reprendre l'exemple illustré par la figure 2.3 en ajoutant aux contraintes de time lags minimaux des time lags maximaux de même valeur. Seul l'ordonnancement correspondant à la permutation $(1, 2)$ est modifié, sans pour autant que son makespan le soit, comme le montre la figure 2.7.b.

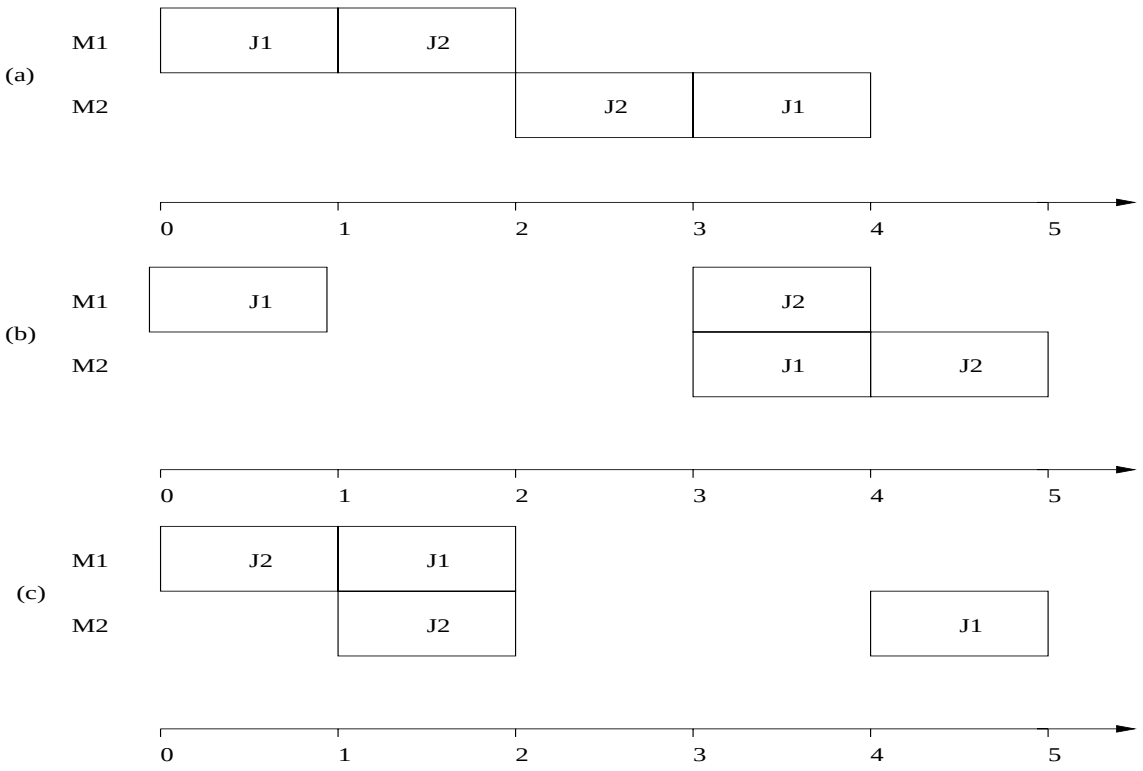

FIG. 2.7 – Non dominance des ordonnancements de permutation

Si nous considérons le problème de la minimisation du makespan pour les flowshops de permutation à deux machines $F2_\pi|\theta_j^{min} = \theta_j^{max}|C_{max}$, on retrouve un cas que l'on peut résoudre avec l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64], présenté dans la partie 2.5.1.

Théorème. 15 *Le problème $F2_\pi|\theta_j^{min} = \theta_j^{max}|C_{max}$ peut être résolu de manière optimale en appliquant l'algorithme de Gilmore et Gomory.*

Preuve. Pour tout j, on pose $\theta_j = \theta_j^{min} (= \theta_j^{max})$. Soit π une permutation des travaux. On considère l'ordonnancement S associé à cette permutation, où le travail $\pi(j)$ est exécuté en j-ième position sur les deux machines. Le makespan de S peut être exprimé ainsi :

$C_{max}(S) = C_{\pi(n),2} = C_{\pi(n),1} + \theta_{\pi(n)} + p_{\pi(n),2}$

Or, les dates de fin sur la première machine peuvent être déterminées par la formule récursive suivante :

$C_{\pi(j+1),1} = C_{\pi(j),1} + p_{\pi(j+1),1} + max(0, \theta_{\pi(j)} + p_{\pi(j),2} - (\theta_{\pi(j+1)} + p_{\pi(j+1),1}))$

La valeur intiale pour le premier travail est : $C_{\pi(1),1} = p_{\pi(1),1}$. Le makespan peut donc être calculé de la manière suivante :

$C_{max}(S) = \sum_{1 \leq i \leq n} p_{i,1} + \sum_{1 \leq i \leq n-1} max(0, \theta_{\pi(i)} + p_{\pi(i),2} - (p_{\pi(i+1),1} + \theta_{\pi(i+1)})) + \theta_{\pi(n)} + p_{\pi(n),2}$

La première somme $\sum_{1 \leq i \leq n} p_{i,1}$ ne dépend pas de la permutation π et peut donc être retirée de la fonction objectif à minimiser. Le second terme correspond à un problème de voyageur de commerce de type Gilmore et Gomory, que l'on peut résoudre optimalement avec l'algorithme du même nom. On peut remarquer que le second terme est équivalent à ce que l'on obtient dans le cas sans attente, avec des durées $P_{j,1} = p_{j,1} + \theta_j$ sur la première machine et $P_{j,2} = p_{j,2} + \theta_j$ sur la seconde machine.

Dans le cas général où l'on ne se restreint pas aux solutions de permutation, on retrouve le même résultat qu'avec les time lags minimaux : le problème $F2|\theta_j^{min} = \theta_j^{max}|C_{max}$ est NP-difficile au sens fort [Yu et al.,04].

2.7 Prise en compte d'une contrainte additionnelle : machine à traitement par fournées

Nous avons également cherché à incorporer une contrainte supplémentaire à notre modèle : le traitement par fournées (désignées par le terme *batch* en anglais, qui peut aussi signifier lot). Ce travail correspond au dernier semestre de préparation de cette thèse. Par conséquent, nous n'avons pas eu le temps d'être aussi exhaustifs pour les résultats existants dans la littérature. Il existe de nombreuses études sur les problèmes d'ordonnancement en présence de machines à traitement par fournées. Nous renvoyons en particulier le lecteur aux articles présentant l'état de l'art dans le domaine [Potts et Van Wassenhove,91], [Webster et Baker,95], [Potts et Kovalyov,00].

2.7.1 Généralités sur les machines à traitement par fournées

On suppose habituellement que chaque machine de l'atelier n'est capable d'exécuter qu'un travail à la fois, ce qui conduit à un traitement séquentiel des produits. La mise en parallèle de plusieurs machines identiques autorise un traitement parallèle, mais dans ce cas, les travaux traités sur des machines différentes le sont indépendemment : aucune contrainte ne lie leurs dates de début ou de fin respectives. Or, dans la pratique, il arrive que l'on rencontre des machines sur lesquelles il est non seulement possible, mais également souhaitable, pour ne pas dire obligatoire, de traiter plusieurs travaux à la fois : ceux-ci sont introduits au même instant dans la machine et en ressortent ensemble, à la même date. Tout se passe en fait comme si les travaux regroupés étaient assimilés à un macro-travail, dont la durée de traitement P_Ω sur la machine est fonction des durées p_j de chacun des travaux $j \in \Omega$, Ω désignant l'ensemble des travaux regroupés pour le traitement. On distingue généralement deux cas :

- Cas 1 : $P_\Omega = max_{j \in \Omega}\{p_j\}$. La durée est égale à la plus longue des durées des travaux. On parle alors de *max-batch machine*, ou de machine à traitement parallèle. C'est en particulier le cas des fours qui servent à réchauffer les pièces en sidérurgie, pour les amener aux températures souhaitées. Le regroupement permet alors de réduire considérablement les dépenses d'énergie, par rapport à un traitement individuel.

– Cas 2 : $P_\Omega = \sum_{j\in\Omega} p_j$. La durée est égale à la somme des durées des travaux. On parle alors de *sum-batch machine*, ou de machine à traitement séquentiel. Il convient de noter que ce cas est tout de même différent du cas représenté par une machine classique, effectuant les opérations des travaux les unes après les autres, dans la mesure où, avec un traitement par fournée, les travaux arrivent et quittent la machine simultanément.

Outre le type de traitement, ces machines sont également caractérisées par le nombre maximum de travaux qu'elles peuvent traitées simultanément. Ce paramètre, noté b, est appelé capacité de la machine, et on est souvent amené à distinguer le cas capacité limitée ($b < n$) et le cas capacité illimitée ($b \geq n$), qui peuvent s'avérer très différents, y compris en termes de complexité. Bien entendu, le cas $b = 1$ correspond aux machines classiques.

2.7.2 Problème considéré

Nous nous intéressons à un problème de flowshop à deux machines, où la première est une machine à traitement par fournées de type *max-batch*, de capacité b, tandis que la deuxième est une machine classique, en présence de time lags minimaux. L'objectif est de minimiser le makespan. D'après la notation générale proposée par Graham et al. [Graham et al.,79] pour les problèmes d'ordonnancement, étendue aux cas des machines à traitement par fournées [Brucker et al.,98], ce problème est désigné par $F2|p-batch(1), b, \theta_j^{min}|C_{max}$. Il est nécessaire ici d'adapter la définition des time lags au contexte de traitement par fournées. Il nous semble naturel de considérer, pour tout travail $j \in \Omega$ l'écart entre la date de fin commune sur la première machine $C_{j,1} = C_{\Omega,1}$ et la date de début sur la seconde machine $t_{j,2}$. Nous attirons l'attention du lecteur sur le fait que, en raison des nouvelles hypothèses considérées, les différents types de time lags (start-start, stop-stop, stop-start), ne sont plus équivalents puisque la durée sur la première machine n'est plus fixe mais varie selon les regroupements effectués.

Potts et al. [Potts et al.,01] considèrent le problème de flowshop avec deux machines à traitement par fournées parallèle et fournissent de nombreux résultats de complexité. D'autres résultats, dans le cas de machines à traitement séquentiel, sont donnés par Oulamara et Finke [Oulamara et Finke,01]. A notre connaissance, la prise en compte simultanée de traitement par fournées et de contraintes de time lags ne fait l'objet d'aucun article dans la littérature en ordonnancement d'atelier, si l'on excepte le cas particulier sans attente. Celui-ci est abordé par Oulamara et al. [Oulamara et al.,05] pour les problèmes avec machines à traitement parallèle et par Lin et Cheng [Lin et Cheng,01] et Hall et al. [Hall et al.,03] pour les problèmes avec machines à traitement séquentiel.

Dans les problèmes de flowshop avec machines classiques, la classe des ordonnancements de permutation joue un rôle important : d'une part, elle correspond à un cas particulier qui a une interprétation pratique, et d'autre part, elle est dominante et permet de réduire l'espace de recherche à explorer pour un certain nombre de problèmes. Il est possible d'adapter la définition de cette classe lorsque le problème comporte une ou plusieurs machines à traitement par fournée : supposons que pour chaque machine k, $B_k(j)$ indique la position du batch, dans l'ordre de traitement sur la machine k, dans lequel se trouve le travail j (si k est une machine classique, $B_k(j)$ représente la position à laquelle est traité le travail j). L'ordonnancement est "de permutation" si pour tout couple de travaux i et j, on a $\forall k, B_k(i) \leq B_k(j)$ ou $\forall k, B_k(i) \geq B_k(j)$. Autrement dit, i est traité avant ou en même temps que j sur toutes les machines, ou j est traité avant ou

en même temps que i sur toutes les machines. Si toutes les machines sont classiques, on retrouve la définition habituelle des ordonnancements de permutation.

En l'absence de time lags minimaux, on vérifie facilement que les solutions de permutation sont dominantes pour le problème $F2|p-batch(1),b|C_{max}$ (par un argument d'échange de travaux consécutifs sur la deuxième machine, par exemple). Par ailleurs, Potts et al. [Potts et al.,01] montrent que, dans le cas $b \geq n$ le problème peut être résolu par un algorithme de programmation dynamique en $O(n^2)$, en se ramenant au problème $1|p-batch, b \geq n|L_{max}$, alors que dans le cas $1 < b < n$, le problème est NP-difficile au sens ordinaire. La question de l'existence d'un algorithme pseudo-polynomial pour ce problème reste ouverte. Dans le cas particulier où la durée est constante sur la première machine, le problème $F2|p-batch(1), 1 < b < n, p_{j,1} = p|C_{max}$ devient polynomial [Ahmadi et al,92] : il suffit de traiter les travaux dans l'ordre décroissant des durées sur la seconde machine, en formant des fournées pleines avec b travaux sur la première machine.

2.7.3 Résultats préliminaires

Nous nous intéressons à ce que deviennent les résultats précédents lorsque l'on introduit des time lags minimaux. Tout d'abord, on peut facilement montrer la série de propriétés suivantes :

Lemme. 4 *Le problème avec des time lags minimaux constants ($F2|p-batch(1), \theta_j^{min} = \theta|C_{max}$) est équivalent au cas sans time lags. Les résultats de complexité sont donc identiques.*

Preuve. On passe d'un cas à l'autre en décalant l'ordonnancement sur la seconde machine de θ unités de temps. □

Lemme. 5 *Supposons que l'affectation des travaux aux fournées et que le séquencement de celles-ci sur la première machine soient réalisés. Alors le problème qui consiste à ordonnancer les travaux sur la seconde machine de manière à minimiser le makespan est polynomial et se ramène au problème $1|r_j|C_{max}$.*

Preuve. Pour tout travail j, on désigne par $B(j)$ la fournée dans laquelle le travail est placé, en supposant que celles-ci sont indicées selon leur position sur la première machine. La date à partir de laquelle j est disponible sur la seconde machine est donnée par $r_{j,2} = C_{B(j),1} + \theta_j^{min}$. Le problème consiste à placer les travaux sur la seconde machine, qui est une machine classique, de telle sorte que le makespan soit minimal. Il s'agit donc d'un problème à une machine avec dates de disponibilité des travaux ($1|r_j|C_{max}$) pour lequel une solution optimale est obtenue en traitant les travaux dans l'ordre croissant des dates de disponibilité.□

Lemme. 6 *Supposons que les travaux h et i sont regroupés dans la même fournée F, avec $\theta_h^{min} \leq \theta_i^{min}$ (il peut y avoir d'autres travaux dans la fournée). Alors il existe une solution optimale dans laquelle h est traité avant i sur la deuxième machine.*

Preuve. Soit C_F la date de fin de traitement de F sur la première machine. On a également $C_{h,1} = C_{i,1} = C_{F,1}$. Les dates de début au plus tôt de h et de i sur la seconde machine sont respectivement $r_{h,2} = C_{F,1} + \theta_h^{min}$ et $r_{i,2} = C_{F,1} + \theta_i^{min}$. D'après les hypothèses, il vient que

$r_{h,2} \leq r_{i,2}$. D'après la propriété précédente, on peut obtenir un ordonnancement optimal sur la seconde machine en plaçant les travaux dans l'ordre croissant de leur date de début au plus tôt $r_{j,2}$ donc dans un tel ordonnancement, h est traité avant i sur la seconde machine. □

Ces résultats peuvent nous aider à établir certains résultats de complexité que nous présentons maintenant dans les sections qui suivent.

2.7.4 Problème à capacité limitée ($1 < b < n$)

Le première question que l'on est en droit de se poser est celle de la dominance éventuelle des ordonnancements de permutation. On montre que :

Propriété. 7 *Pour le problème $F2|p-batch(1), 1 < b < n, \theta_j^{min}|C_{max}$, les ordonnancements de permutation ne sont pas dominants, même avec des durées opératoires unitaires.*

Preuve. Considérons l'exemple suivant, avec 4 travaux dont les time lags minimaux sont respectivement $\theta_1^{min} = 3, \theta_2^{min} = 4, \theta_3^{min} = 0, \theta_4^{min} = 1$, et une capacité $b = 2$. L'ordonnancement S consistant à traiter les travaux 1 et 2 ensemble puis les travaux 3 et 4 ensemble, sur la première machine, tandis que la séquence de traitement sur la seconde machine est $(3, 4, 1, 2)$ a un makespan de 6, qui est égal à la borne inférieure $p_{2,1} + \theta_2^{min} + p_{2,2}$. Cet ordonnancement est donc optimal. Pour atteindre cette valeur, il est nécessaire de traiter le travail 2 dans la première fournée sur la première machine et entre la date 5 et 6 sur la seconde machine. De plus, la date de fin au plus tôt du travail 1 est $p_{1,1} + \theta_1^{min} + p_{1,2} = 5$, donc dans tout ordonnancement optimal, ce travail doit être exécuté entre 0 et 1 sur la première machine (donc dans la première fournée), et entre 4 et 5 sur la seconde machine. La seule possibilité est alors de regrouper les travaux 3 et 4 dans la seconde fournée, traitée entre 1 et 2 sur la première machine, et de les exécuter sur la seconde machine entre 2 et 3, et 3 et 4, respectivement. Par conséquent, l'ordonnancement S est le seul ordonnancement optimal, et ce n'est pas un ordonnancement de permutation puisque $B_1(1) = B_1(2) = 1 < B_1(3) = B_1(4) = 2$, alors que $B_2(1) = 3 > B_2(3) = 1$. Cet ordonnancement est représenté sur la figure 2.8. □

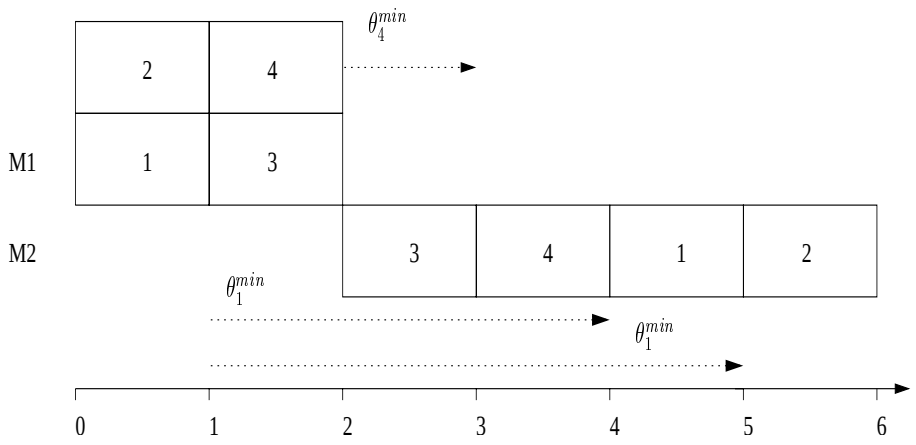

FIG. 2.8 – Non dominance des solutions de permutation

On peut donc être amené, comme dans un flowshop avec des machines classiques, à distinguer les problèmes généraux de ceux où l'on se restreint aux solutions de permutation.

En l'absence de time lags, le problème $F2|p-batch(1), 1 < b < n|C_{max}$ est NP-difficile au sens ordinaire [Potts et al.,01], et comme les ordonnancements de permutation sont dominants dans ce cas, on en déduit immédiatement que les problèmes $F2_{\pi}|p - batch(1), 1 < b < n, \theta_j^{min}|C_{max}$ et $F2|p - batch(1), 1 < b < n, \theta_j^{min}|C_{max}$ sont également NP-difficiles, au sens ordinaire. Nous montrons que :

Théorème. 16 *Le problème $F2|p - batch(1), 1 < b < n, p_{j,k} = 1, \theta_j^{min}|C_{max}$ est NP-difficile au sens fort.*

Preuve. Nous présentons la preuve dans le cas $b = 2$. Nous utilisons une réduction de la version de décision du problème $F2|p_{j,k} = 1, \theta^{min}|C_{max}$, qui est NP-difficile au sens fort [Yu et al.,04]. Soit une instance I de ce problème, comportant n travaux de durées unitaires, de time lags $\theta_1^{min}, \theta_2^{min}, \ldots, \theta_n^{min}$, et un entier Z. Nous construisons à partir de I une instance I' du problème qui nous intéresse, de la manière suivante : il y a $n' = 2n$ travaux de durées unitaires, dont les time lags sont définis par $\forall j \leq n, \theta_j^{min\prime} = \theta_j$ (anciens travaux) et $\forall j > n, \theta_j^{min\prime} = Z - 1$ (nouveaux travaux). On cherche à déterminer s'il existe un ordonnancement de makespan inférieur ou égal à $Z' = Z + n$.

Supposons qu'il existe un ordonnancement S pour l'instance I tel que $C_{max}(S) \leq Z$. On peut supposer sans perte de généralité que dans cet ordonnancement, les travaux sont traités sans interruption entre 0 et n sur la première machine (caractéristique des ordonnancements semi-actifs en présence de time lags minimaux uniquement). On construit un ordonnancement S' pour I' en plaçant chaque ancien travail $j \leq n$ à la même date que dans S : $\forall j \leq n, C'_{j,1} = C_{j,1}$ et $C'_{j,2} = C_{j,2}$. On regroupe chaque nouveau travail $j > n$ avec un ancien travail, de manière à constituer n fournées d'exactement 2 travaux chacune. Celles-ci sont traitées sur la première machine sans interruption, entre les dates 0 et n. Puisque tous les nouveaux travaux sont identiques, on peut supposer sans perte de généralité que le travail d'indice $i > n$ est placé dans la fournée $B_1(i-n)$, traitée entre les dates $i-n-1$ et $i-n$ sur la première machine. Il est donc disponible au plus tôt sur la deuxième machine à la date $i - n + Z - 1 \geq Z$. Tous les anciens travaux étant terminés avant la date Z d'après l'hypothèse faite sur S, il est donc possible d'exécuter le travail i entre $i - n + Z - 1$ et $i - n + Z$ sur la deuxième machine. Ainsi, le dernier travail termine son exécution à la date $C_{max}(S') = n + Z = Z'$.

Réciproquement, supposons qu'il existe un ordonnancement S' pour l'instance I' tel que $C_{max}(S') = Z'$. De même que précédemment, on peut supposer sans perte de généralité que S' comporte n fournées comportant chacune exactement 2 travaux et traitées entre les dates 0 et n sur la première machine. La date de début au plus tôt pour un nouveau travail i sur la deuxième machine est $p_{i,1} + \theta_i^{min\prime} = Z$. Etant donné qu'il y a n travaux de ce type, ceux-ci sont nécessairement exécutés sans interruption, entre Z et $Z' = Z + n$, sur la deuxième machine. On montre que l'on peut alors se ramener à un ordonnancement dans lequel chaque fournée contient exactement un nouveau travail. Supposons que ce ne soit pas le cas dans S', et soit r l'indice de la première fournée (dans l'ordre de traitement)qui ne contient pas exactement un nouveau travail. Celle-ci est traitée entre $r-1$ et r sur la première machine. Nous distinguons alors deux cas :

- Cas 1 : la fournée ne contient aucun nouveau travail. Les r nouveaux travaux exécutés sur la deuxième machine entre Z et $Z+r$ doivent être terminés sur la première machine au plus tard à la date $Z+r-1-(Z-1)=r$. Par conséquent, ces travaux appartiennent tous à l'une des r premières fournées sur la première machine. Or, par hypothèse, les $r-1$ premières ne contiennent qu'un nouveau travail et la r-ième n'en contient pas. On aboutit donc à une contradiction.
- Cas 2 : la fournée contient 2 nouveaux travaux. En utilisant un argument similaire au précédent, on peut montrer que les nouveaux travaux contenus dans les $r-1$ fournées précédentes sont ceux qui sont exécutés sur la seconde machine entre Z et $Z+r-1$. On dispose alors de deux travaux disponibles sur cette machine à partir de la date $Z+r-1$. On peut alors échanger l'un de ces deux travaux, disons j, avec un ancien travail h dans la première fournée qui comporte deux anciens travaux. Le fait d'avancer l'exécution du travail h sur la première machine ne modifie pas son exécution sur la deuxième (la contrainte de time lag minimale est toujours respectée).

En répétant le processus décrit dans le second cas chaque fois que celui-ci se présente, on se ramène en fin de compte à un ordonnancement dans lequel chaque fournée contient exactement un nouveau travail, et donc un ancien travail également. On peut alors construire un ordonnancement S pour l'instance I en plaçant les travaux sur la première machine dans le même ordre que les anciens travaux correspondant dans S'. Puisque ceux-ci occupent tous des fournées différentes, il n'y a pas de conflit sur la première machine. D'autre part, nous avons vu que les fournées occupent la première machine entre les dates 0 et n dans S', donc dans S, la première machine est occupée sans interruption entre 0 et n. Si l'on place les anciens travaux sur la seconde machine de la même manière que dans S', on peut vérifier que les time lags minimaux sont respectés (l'écart entre les opérations sur les deux machines ne varie pas). Puisque dans S', les nouveaux travaux occupent la seconde machine sans interruption entre Z et $Z'=C_{max}(S')$, tous les anciens travaux sont traités avant la date Z. On en déduit que $C_{max}(S) \leq Z$. S est donc un ordonnancement réalisable pour l'instance I, ce qui termine la preuve. □

Dans le cas où l'on se limite aux solutions de permutation, on sait que le problème $F2_{\pi}|p-batch(1), 1<b<n, p_{j,k}, \theta_j^{min}|C_{max}$ est au moins NP-difficile au sens ordinaire dans le cas général, puisque le problème sans time lags l'est [Potts et al.,01]. On peut par ailleurs se demander si le cas particulier où les durées sont identiques sur la première machine pour tous les travaux est toujours polynomial. Plaçons nous dans ce cas particulier : $\forall j, p_{j,1}=t$. On suppose de plus que le nombre de travaux est un multiple de la capacité des fournées b. Considérons une fournée F dans laquelle les travaux $j_1, j_2, \ldots, j_b$ sont regroupés, avec $\theta_{j_1}^{min} \leq \theta_{j_2}^{min} \leq \ldots \leq \theta_{j_b}^{min}$. D'après le lemme 6, les travaux de cette fournée sont traités dans l'ordre $(j_1, j_2, \ldots, j_b)$ sur la seconde machine. De plus, étant donné qu'on se limite aux solutions de permutation, aucun travail appartenant à une autre fournée n'est exécuté entre ces travaux. Par conséquent, les temps morts sur la seconde machine entre ces travaux sont inutilisés et il est possible de caler les travaux au plus tard par rapport au dernier j_b. On peut alors assimiler l'ensemble de ces travaux à un macro-travail $\bar{F}$, de durées $p_{\bar{F},1}=t$ et $p_{\bar{F},2}=\sum_{1\leq h\leq b} p_{j_h,2}$ sur la première et la deuxième machine respectivement, et de time lag minimal $\theta_{\bar{F}}^{min}=max_{1\leq h\leq b}\{\theta_{j_h}^{min}-\sum_{1\leq g\leq h-1} p_{j_g,2}\}$. Ceci est illustré sur la figure 2.9.

Ainsi, on aboutit à la propriété suivante.

Propriété. 8 *Si l'affectation des travaux aux fournées est réalisée, l'ordonnancement optimal de ces fournées sur la première machine, et des travaux sur la seconde machine, est obtenu en appliquant la règle de Mitten-Johnson [Mitten,59] aux macros-travaux.*

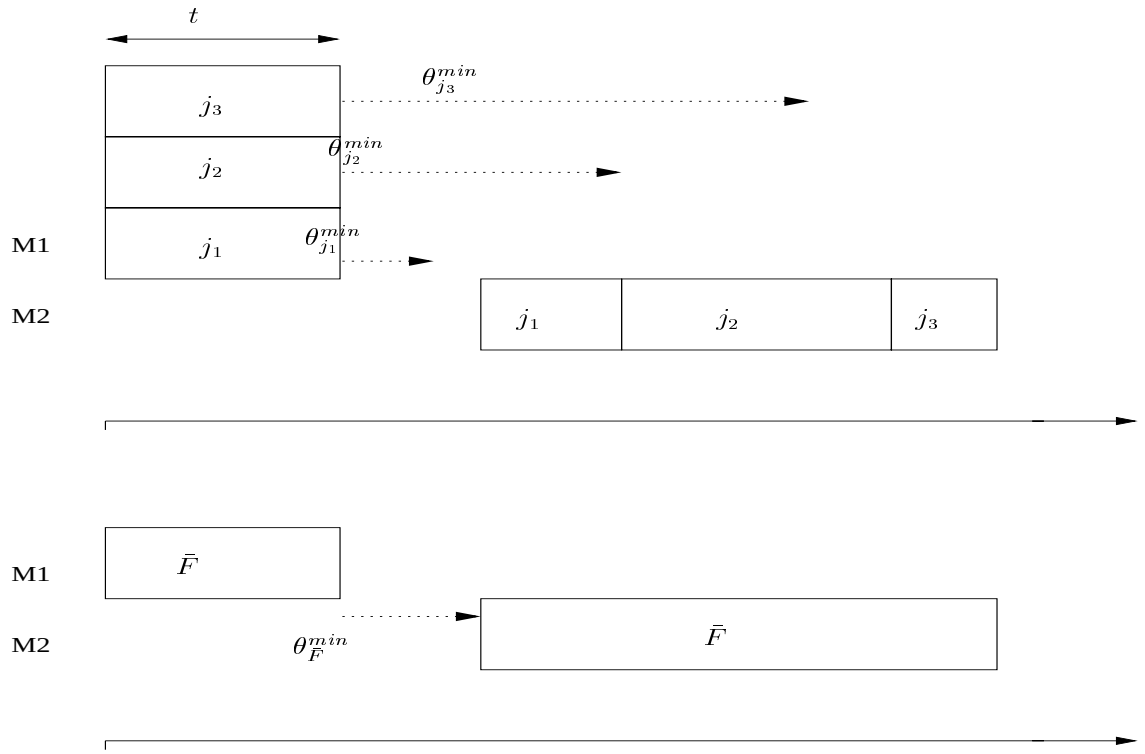

FIG. 2.9 – Construction du macro-travail

Preuve. D'après ce qui précède, on est ramené à ordonnancer des macro-travaux sur un flow-shop de permutation à deux machines avec time lags minimaux, dont les données sont calculées à partir des durées opératoires et des time lags des travaux initiaux. Ce problème $(F2_{\pi}|\theta_j^{min}|C_{max})$ est résolu de manière exacte par l'algorithme de Mitten-Johnson [Mitten,59]. □

2.7.5 Problème à capacité illimitée ($b \geq n$)

De la même manière que dans le premier cas, la première propriété énoncée concerne la dominance éventuelle des ordonnancements de permutation.

Propriété. 9 *Pour le problème $F2|p-batch(1), b \geq n, \theta_j^{min}|C_{max}$, les ordonnancements de permutation ne sont pas dominants, même quand les durées opératoires ne dépendent pas des machines, mais uniquement des travaux ($p_{j,1} = p_{j,2}$).*

Preuve. Considérons l'exemple suivant avec deux travaux 1 et 2, de durées $p_{1,1} = p_{1,2} = 1$ et $p_{2,1} = p_{2,2} = 5$, et de time lags $\theta_1^{min} = 10$ et $\theta_2^{min} = 5$. L'ordonnancement qui consiste à traiter d'abord le travail 1 seul puis le travail 2 seul sur la première machine, et le travail 2 puis le travail 1 sur la seconde machine, a un makespan de 12 (voir la figure 2.10a). Cette valeur est égale à la borne inférieure $p_{1,1} + \theta_1^{min} + p_{2,1}$ donc l'ordonnancement est optimal. Pour atteindre cette valeur du makespan, il est nécessaire que l'exécution du travail 1 sur la première machine se termine à la date 1. Donc le travail 1 doit être traité seul sur cette machine, en première position. La seule solution de permutation qui vérifie cette condition consiste à suivre la séquence $(1, 2)$ sur les

deux machines, ce qui conduit à un makespan de 17 (voir la figure 2.10b). Dans cet exemple, il n'est pas intéressant de regrouper les travaux en fournées, et on se retrouve finalement dans une situation analogue à celle qu'on aurait avec deux machines classiques. Ainsi, les ordonnancements de permutation ne sont pas dominants sur cet exemple. □

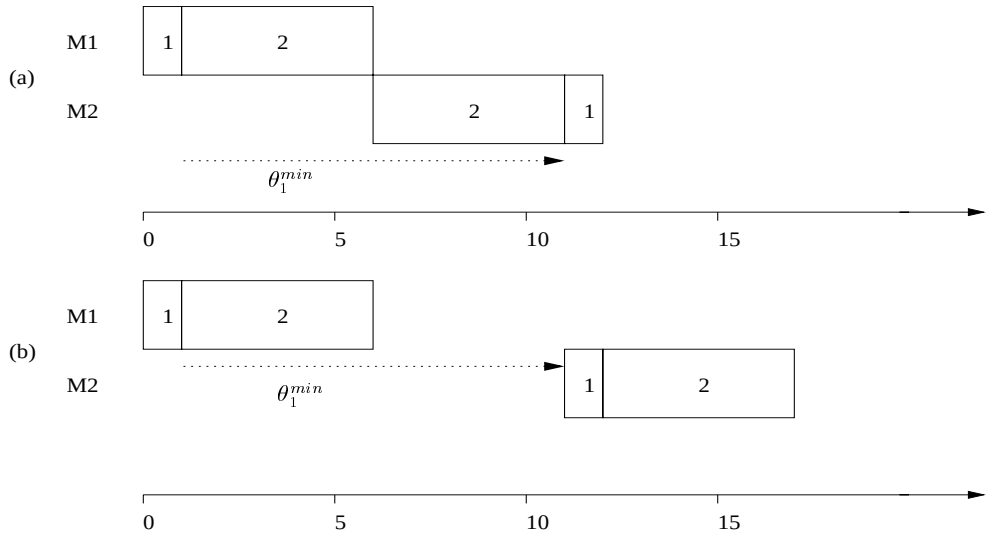

FIG. 2.10 – Non dominance des solutions de permutation

Cependant, il existe des cas particuliers pour lesquels les ordonnancements de permutation sont dominants. Nous en étudions deux dans les paragraphes qui suivent. Avant cela, nous présentons une propriété intéressante du cas où la capacité des fournées est illimitée.

Lemme. 7 *Pour le problème* $F2|p-batch(1), b \geq n, \theta_j^{min}|C_{max}$*, les ordonnancements dans lesquels les travaux sont placés dans les fournées dans l'ordre croissant des durées sur la première machine sont dominants (ces ordonnancements vérifient* $\forall i, j$*, si* $p_{i,1} < p_{j,1}$ *alors* $B(i) \leq B(j)$*).*

Preuve. Supposons que l'on dispose d'un ordonnancement optimal dans lequel la condition n'est pas vérifiée. Il existe i et j tels que $B(j) = B(i) - 1$ (j est contenu dans la fournée qui précède celle de i), et $p_{i,1} < p_{j,1}$. Puisque $p_{i,1} < p_{j,1}$, on peut déplacer i de la fournée $B(i)$ vers la fournée $B(j)$, sans qu'aucune fournée ne soit retardée sur la première machine. Aucun changement n'intervient sur la seconde machine. En outre, comme l'exécution de i est avancée sur la première machine, la contrainte de time lag minimal pour ce travail est toujours satisfaite. En répétant cette transformation, on se ramène à un ordonnancement vérifiant la condition et de même makespan, donc optimal. □

a) Cas où les durées sont identiques sur la première machine

On suppose que $\forall j, p_{j,1} = t$.

Théorème. 17 *Pour le problème* $F2|p-batch(1), b \geq n, p_{j,1} = t, \theta_j^{min}|C_{max}$*, les ordonnancements de permutation sont dominants et une solution optimale est obtenue en regroupant tous les travaux dans la première fournée sur la première machine, et en les traitant dans l'ordre croissant des time lags sur la seconde machine.*

Preuve. Il est suffisant de considérer les solutions pour lesquelles tous les travaux sont regroupés dans la première fournée. En effet, leur date de fin commune sur la première machine est, dans ce cas, minimale et égale à t. D'après le lemme 6, une solution optimale est obtenue en traitant les travaux dans l'ordre croissant de leur time lag θ_j^{min}. On peut en outre remarquer que, puisque tous les travaux sont contenus dans la première fournée, la solution est de permutation, par définition. Les ordonnancements de permutation sont donc dominants dans ce cas. □

b) Cas où les durées sur la première machine et les time lags sont en accord

Définition.
On dit que les durées sur la première machine et les time lags sont *en accord* lorsque $\forall i, j$, si $p_{i,1} < p_{j,1}$ alors $\theta_i^{min} \leq \theta_j^{min}$.

Nous montrons que :

Théorème. 18 *Pour le problème $F2|p-batch(1), b \geq n, \theta_j^{min}|C_{max}$, si les durées sur la première machine et les time lags sont en accord, les ordonnancements de permutation sont dominants et le problème peut être résolu en $O(n^2)$.*

Preuve. D'après le lemme 7, on peut se ramener à un ordonnancement dans lequel les travaux sont placés dans les fournées dans l'ordre croissant des durées sur la première machine, qui ici est le même que l'ordre croissant des time lags. On peut alors montrer que quelle que soit la répartition des travaux dans les fournées, ils sont traités sur la seconde machine dans l'ordre croissant des time lags. Supposons qu'on ait deux travaux i et j vérifiant $\theta_i^{min} < \theta_j^{min}$. Par hypothèse, $p_{i,1} \leq p_{j,1}$ et donc $B(i) \leq B(j)$. On en déduit que $C_{i,1} \leq C_{j,1}$. Par conséquent les dates de début au plus tôt sur la seconde machine vérifient $r_{i,2} = C_{i,1} + \theta_i^{min} < C_{j,1} + \theta_j^{min} = r_{j,2}$. De la même manière que dans la preuve du lemme 6, les ordonnancements où i précède j sur la seconde machine sont dominants. On a finalement montré que les ordonnancements dans lesquels les travaux sont placés dans les fournées et sur la seconde machine dans l'ordre croissant des time lags, sont dominants. Ce sont bien évidemment des ordonnancements de permutation.

Afin d'obtenir un ordonnancement optimal, il reste à déterminer comment répartir les travaux dans les fournées. Supposons désormais que les travaux sont numérotés dans l'ordre croissant des time lags. Le problème consiste à déterminer le nombre de fournées N et les indices $n_1, n_2, \ldots n_{N-1}$ tels que la fournée $i \leq N$ contienne les travaux d'indices $n_{i-1} + 1$ à n_i, avec $n_0 = 0$ et $n_N = n$. Pour le résoudre, on procède de la même manière que dans [Potts et al.,01]. Supposons désormais que les travaux sont renumérotés dans l'ordre croissant des time lags. Pour un ordonnancement dans lequel les travaux sont traités sur les deux machines dans l'ordre $(1, 2, \ldots, n)$, le makespan est donné par $C_{max} = max_{1 \leq j \leq n}\{C_{j,1} + \theta_j^{min} + \sum_{j \leq h \leq n} p_{h,2}\}$. On est ainsi ramené à un problème de minimisation du makespan sur une machine de type *max-batch*, où chaque travail j a une durée opératoire $p_{j,1}$ et une durée de latence $q_j = \theta_j^{min} + \sum_{j \leq h \leq n} p_{h,2}\}$. Ce problème $(1|p - batch, b \geq n, q_j|C_{max})$ est équivalent à la minimisation du plus grand retard $(1|p - batch, b \geq n|L_{max})$ avec des dates de fin souhaitées $d_j = -q_j$, pour lequel Brucker et al. [Brucker et al.,98] proposent un algorithme de programmation dynamique en $O(n^2)$. □

Dans le cas général, la question de la complexité du problème $F2|p-batch(1), b \geq n, \theta_j^{min}|C_{max}$ reste ouverte, qu'on se limite aux solutions de permutation ou non. On rappelle qu'en l'absence de time lags minimaux, les solutions de permutation sont dominantes et il existe un algorithme de résolution exacte de complexité $O(n^2)$ [Potts et al.,01].

Problèmes polynomiaux "maximaux"		
Problème	Dominance des permutations	Référence
$F2\|\|\sum w_k MC_k$	Oui	Théorème 2
$F\|\|p_{j,k}=1, r_j\|\sum w_j U_j$	Oui	[Brucker et Knust]
$F\|\|p_{j,k}=1, r_j\|\sum w_j T_j$	Oui	[Brucker et Knust]
$F\|\|p_{j,k}=1, r_j\|\sum w_k MC_k$	Oui	Voir précédemment
Problèmes NP-difficiles au sens fort "minimaux"		
Problème	Dominance des permutations	Référence
$F3\|\|C_{max}$	Oui	[Garey et al.,76]
$F3\|\|\sum MC_k$	Oui	Théorème 3
$F2\|r_j\|C_{max}$	Oui	[Lenstra et al.,77]
$F2\|\|L_{max}$	Oui	[Lenstra et al.,77]
$F2\|\|\sum C_j$	Oui	[Garey et al.,76]

TAB. 2.6 – Résultats de complexité des problèmes sans time lags

2.8 Synthèse des résultats de complexité

Cette section reprend de façon synthétique l'ensemble des résultats de complexité démontrés dans ce chapitre (à l'exception de ceux de la section 2.7), complétés par certains résultats fondamentaux établis dans la littérature. Une liste assez large de résultats de complexité concernant les problèmes d'ordonnancement d'ateliers peut être trouvée dans [Brucker et Knust]. De la même manière, nous présentons uniquement les problèmes polynomiaux "maximaux" (au sens de la réduction polynomiale des problèmes) et les problèmes NP-difficiles "minimaux", ainsi que les problèmes qui restent, à notre connaissance, ouverts.

Le tableau 2.6 concerne les problèmes sans time lags. La deuxième colonne indique si les ordonnancements de permutation sont dominants, dans le cas où cette caractéristique n'est pas imposée ; dans le cas d'un problème de flowshop de permutation, cette colonne est sans objet, ce qui est représenté par le symbole "/". Les problèmes à durées unitaires avec un critère classique présentent peu d'intérêt puisqu'ils se ramènent à des problèmes à une machine. En ce qui concerne le problème $F||p_{j,k}=1, r_j|\sum w_k MC_k$, c'est également vrai et l'on peut facilement montrer qu'une solution optimale consiste à traiter les travaux sur toutes les machines dans l'ordre croissant des dates de disponibilité r_j.

Le tableau 2.7 présente les résultats de complexité associés aux problèmes avec time lags minimaux uniquement. Comme nous l'avons déjà mentionné, la plupart des problèmes sont NP-difficiles au sens fort, à moins de se restreindre à des cas où les durées sont constantes. Il convient de remarquer que le problème $F||p_{j,k}=p, \theta^{min}_{j,k}=\theta^{min}_j|\sum C_j$, étudié dans [Brucker et al.,04a], considère des time lags minimaux indépendants des machines (pour un travail donné, les time lags sont les mêmes entre tout couple de machines consécutives). Nous n'avons pas fait figurer les problèmes pour lesquels les time lags sont indépendants des travaux, puisque ces problèmes peuvent se ramener aux cas classiques, sans time lags.

En ce qui concerne les problèmes avec time lags maximaux uniquement, les résultats sont résumés dans le tableau 2.8. Mis à part le problème sans attente avec la somme pondérée des dates

Problèmes polynomiaux "maximaux"		
Problème	Dominance des permutations	Référence
$F2_\pi\|\theta_j^{min}\|\sum w_k MC_k$	/	Théorème 8
$F\|\|p_{j,k}=p,\theta_{j,k}^{min}=\theta_j^{min}\|\sum C_j$	Oui	[Brucker et al.,04a]
$F3_\pi\|p_{j,k}=p,\theta_{j,k}^{min}\|\sum w_k MC_k$	/	Théorème 8
$F2_\pi\|p_{j,k}=p,r_j,\theta_j^{min}\|\sum w_k MC_k$	/	Théorème 8
$F2_\pi\|p_{j,k}=p,\theta_j^{min}\|\sum U_j$	/	Théorème 7
Problèmes NP-difficiles au sens fort "minimaux"		
Problème	Dominance des permutations	Référence
$F2\|p_{j,k}=1,\theta_j^{min}\|C_{max}$	Non	[Yu et al.,04]
$F2\|p_{j,1}=p_{j,2},\theta_j^{min}\in\{\theta_1,\theta_2\}\|C_{max}$	Non	[Yu,96]
$F2\|p_{j,k}=1,\theta_j^{min}\|\sum MC_k$	Non	Théorème 8
$F2\|p_{j,1}=p_{j,2},\theta_j^{min}\in\{\theta_1,\theta_2\}\|\sum MC_k$	Non	Théorème 8
$F2\|p_{j,k}=1,r_j,\theta_j^{min}\|\sum C_j$	Non	[Brucker et al.,04a]
$F2\|p_{j,k}=1,\theta_j^{min}\|\sum w_j C_j$	Non	[Brucker et al.,04a]
Problèmes ouverts "minimaux"		
Problème	Dominance des permutations	Référence
$F3\|\|p_{j,k}=1,\theta_{j,k}^{min}\|\sum C_j$	Non	
$F4_\pi\|p_{j,k}=1,\theta_{j,k}^{min}\|C_{max}$	/	
$F3_\pi\|p_{j,k}=1,r_j,\theta_{j,k}^{min}\|C_{max}$	/	
$F3_\pi\|p_{j,k}=1,\theta_{j,k}^{min}\|L_{max}$	/	
$F2_\pi\|p_{j,k}=1,\theta_j^{min}\|\sum w_j U_j$	/	
$F2_\pi\|p_{j,k}=1,\theta_j^{min}\|\sum T_j$	/	
$F2_\pi\|p_{j,k}=1,\theta_j^{min}\|\sum w_j C_j$	/	

TAB. 2.7 – Résultats de complexité des problèmes avec time lags minimaux uniquement

Problèmes polynomiaux "maximaux"		
Problème	Dominance des permutations	Référence
$F2\|no-wait\|\sum w_k MC_k$	Oui	Théorème 9
Problèmes NP-difficiles au sens fort "minimaux"		
Problème	Dominance des permutations	Référence
$F2\|r_j, no-wait\|C_{max}$	Oui	[Roeck,84b]
$F3\|no-wait\|C_{max}$	Oui	[Roeck,84a]
$F2\|no-wait\|L_{max}$	Oui	[Roeck,84b]
$F2\|no-wait\|\sum C_j$	Oui	[Roeck,84b]
$F2\|\theta_j^{max} \in \{0,+\infty\}\|C_{max}$	Non	[Finke et al.,02]
$F2_\pi\|\theta_j^{max} \in \{0,+\infty\}\|C_{max}$	/	[Finke et al.,02]
$F2\|p_{j,2} = \theta_j^{max} = \theta\|C_{max}$	Oui	Section 2.5.3
$F2\|p_{j,1} = \theta_j^{max} = \theta\|C_{max}$	Oui	Section 2.5.3
$F2\|p_{j,2} = \theta_j^{max} = \theta\|\sum MC_k$	Oui	Section 2.5.3
$F2\|p_{j,1} = \theta_j^{max} = \theta\|\sum MC_k$	Oui	Section 2.5.3

TAB. 2.8 – Résultats de complexité des problèmes avec time lags maximaux uniquement

de fin des machines pour critère, qui est une généralisation du problème résolu par l'algorithme de Gilmore et Gomory, tous les problèmes sont NP-difficiles au sens fort.

Les résultats de complexité pour les problèmes incluant à la fois des time lags minimaux et maximaux se déduisent des tableaux 2.7 et 2.8. Il convient simplement de noter que le cas particulier $F2_\pi|\theta_j^{min} = \theta_j^{max}|\sum w_k MC_k$ se ramène au cas sans attente, et que le problème $F2|\theta_j^{min} = \theta_j^{max}|C_{max}$ est NP-difficile au sens fort [Yu et al.,04].

2.9 Conclusions du chapitre

Ce chapitre expose une synthèse des résultats théoriques concernant les problèmes d'ordonnancement de type flowshop, en présence de contraintes de time lags minimaux et/ou maximaux. En ce qui concerne la complexité de ces problèmes, nous avons rappelé des résultats classiques, que ce soit dans les cas sans time lags, avec time lags minimaux uniquement ou sans attente. Nous avons aussi démontré de nouveaux résultats de complexité, en fournissant des algorithmes polynomiaux exacts pour une série de problèmes de flowshop de permutation avec durées unitaires et time lags minimaux, et en montrant que le flowshop à deux machines avec time lags maximaux identiques est NP-difficile au sens fort. La prise en compte d'un critère particulier, lié à l'utilisation des machines et motivé par des considérations pratiques, nous a conduit à généraliser des résultats qui concernaient la minimisation du makespan. L'ensemble de ces résultats de complexité nous permet de mieux appréhender les problèmes de flowshop avec time lags, qui sont pour la plupart NP-difficiles au sens fort, à moins de se limiter à des cas très particuliers.

Contrairement au cas classique, sans time lags, où il est suffisant de considérer des ordonnancements de permutation pour les problèmes de minimisation du makespan comportant jusqu'à trois machines, de tels ordonnancements ne sont plus dominants dès $m = 2$, en présence de time lags (minimaux ou maximaux). Néanmoins, il existe des cas particuliers pour lesquels il est suffisant de se limiter à certains types de solutions, ce qui permet de réduire l'espace de recherche

à explorer. Nous présentons ainsi des résultats de dominance dans le cas du flowshop à deux machines avec time lags maximaux.

Nous analysons également la performance dans le pire cas d'heuristiques, dans les cas sans time lags, sans attente, et avec time lags maximaux. En particulier, nous montrons que pour le problème $F2_{\pi}|\theta_j^{max}|C_{max}$, la valeur de la solution obtenue grâce à l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64] est au plus le double de l'optimum.

Ce chapitre aborde aussi la complexité des problèmes de flowshop impliquant à la fois des contraintes de time lags et des machines à traitement par fournées, ce qui, à notre connaissance, n'a jamais été entrepris, si ce n'est pour le cas particulier sans attente. Nous établissons des premiers résultats concernant le problème $F2|p - batch(1), b \geq n, \theta_j^{min}|C_{max}$. Ce type de problèmes devrait à l'avenir recevoir une attention accrue de la part des chercheurs, étant donné que les contraintes prises en compte traduisent des situations réellement rencontrées dans certains domaines industriels, et qu'un grand nombre de problèmes reste de complexité ouverte.

D'une manière générale, tous ces résultats théoriques constituent une étape préliminaire nécessaire avant le développement de méthodes de résolution pour des problèmes réels, NP-difficiles. Elles peuvent permettre d'orienter la mise au point de ces méthodes, en cherchant à faire apparaître des sous-problèmes ou des relaxations conduisant à des cas particuliers polynomiaux, et en intégrant les propriétés de dominance afin de se limiter à des ensembles de solutions intéressantes.

Dans le chapitre qui suit, nous décrivons des approches de résolution consacrées à une grande variété de problèmes d'ordonnancement avec time lags, issues de la littérature, tandis que dans le chapitre 4, nous exposons nos propres méthodes de résolution, que nous avons développées pour des problèmes de flowshop de permutation.

Chapitre 3

Etat de l'art sur les approches de résolution pour les problèmes avec time lags

Résumé

Dans ce chapitre, nous présentons les travaux existant dans la littérature concernant les problèmes d'ordonnancement en présence de time lags. A notre connaissance, il n'existe dans ce domaine aucune publication comportant une synthèse de tous ces travaux. Nous examinons à la fois les modèles proposés pour ces problèmes et les approches de résolution développées pour les traiter. Nous distinguons les cas à une machine, à machines parallèles, les problèmes d'atelier multi-machines (flowshop, jobshop, open shop), qui nous intéressent plus particulièrement et pour lesquels nous avons cherché à être le plus exhaustif possible, et considérons également les problèmes d'ordonnancement de projet. Le cas spécifique des problèmes sans attente est traité à part, étant donné les très nombreuses références sur ce sujet. Pour ces problèmes, nous nous contentons de citer quelques études significatives.

Sommaire

3.1 Introduction

Comme nous l'avons déjà mentionné, le concept de time lags en ordonnancement existe depuis près d'un demi-siècle. Malgré le grand nombre de situations industrielles pouvant être modélisées à l'aide des time lags, les travaux de recherche en ordonnancement consacrés à ces contraintes restent encore assez marginaux. Ils sont notamment menés de manière totalement indépendante, sans qu'il y ait de véritable effort de synthèse et d'unification. A ce titre, les nombreuses dénominations employées pour désigner les contraintes de type time lags sont significatives (voir la section 1.3). Il faut ajouter à ceci le fait que pour certaines applications particulières, telles que la sidérurgie ou l'informatique en temps réel, des problèmes d'ordonnancement avec time lags sont étudiés par des communautés spécifiques associées à ces applications, qui utilisent leurs propres modèles et méthodes de résolution. L'objectif de ce chapitre est de faire le point sur les travaux dédiés à ce sujet, quel que soit le cadre dans lequel ils sont menés (ordonnancement d'atelier, ordonnancement de projet). Nous limitons volontairement l'analyse des travaux concernant les problèmes sans attente, qui sont très abondants dans la littérature. Dans tout ce chapitre, nous convenons que lorsque le critère n'est pas explicitement mentionné, il s'agit du makespan.

3.2 Les problèmes à une machine

Pour les problèmes à une machine, il convient de distinguer deux cas en fonction de la définition des time lags :

- Cas 1 : le cas habituel, où chaque travail est composé d'une seule opération, à traiter sur la machine. Dans ce cas, la définition utilisée jusqu'à maintenant pour les time lags est caduque, puisqu'il n'y a pas d'opérations consécutives au sein des travaux. Par contre, on peut toujours envisager de considérer des contraintes de précédence entre les travaux, et d'associer à ces contraintes de précédence des time lags minimaux ou maximaux.
- Cas 2 : le modèle classique peut être modifié de sorte que chaque travail j soit composé d'un certain nombre n_j d'opérations, liées par des relations de précédence formant une chaîne (pas d'arbitrage entre les opérations d'un même travail). Dans ce cas, la définition des time lags entre les opérations consécutives au sein des travaux reste valable.

On pourrait être tenté de croire que le second modèle est un cas particulier du premier. Cependant, la définition rigoureuse des travaux reste fondamentale, et a une influence sur les critères qui s'expriment sous forme de somme. Par exemple, pour le critère de la somme des dates de fin des travaux $\sum C_j$, toutes les opérations seront prises en compte dans le premier cas, alors que l'on ne considère que les opérations finales de chaque travail (situées à l'extrémité terminale de chaque chaîne) dans le second cas. Cette distinction est d'ailleurs mentionnée dans [Munier et Sourd,03], où les auteurs se placent dans le premier cas en considérant la contribution de chaque opération au critère $\sum C_j$. Pour le second cas, ils suggèrent le nom de "jobshop à une machine".

3.2.1 Premier cas : une opération par travail et contraintes de précédence entre les travaux

Il s'agit du modèle que l'on rencontre le plus dans la littérature. En partant du constat que nous avons déjà fait, selon lequel les contraintes de précédence classiques ne sont pas suffisantes, il paraît nécessaire de disposer d'un modèle plus général dans lequel de telles contraintes pourraient être valuées, de manière à représenter les intervalles de temps associés à ces contraintes.

En particulier, la représentation des problèmes classiques sous forme de graphe disjonctif peut être très facilement adaptée à ce nouveau modèle. Nous allons brièvement rappeler comment est défini ce graphe et quel est son intérêt dans la modélisation, l'étude et la résolution des problèmes d'ordonnancement.

a) Rappels concernant le modèle du graphe disjonctif

Le modèle du graphe disjonctif, encore appelé graphe potentiel-tâche, est dû à Roy et Sussman [Roy et Sussman,64]. Les opérations à exécuter y sont représentées par des sommets, contrairement à la méthode PERT (potentiel-étape) où elles sont associées aux arcs du graphe. Plus précisément, le graphe $G = (N, A, E)$ est défini de la manière suivante :

- l'ensemble des sommets N correspond à l'ensemble des n opérations, auquel on ajoute deux opérations fictives (0 et $n+1$) associées au début et à la fin de l'ordonnancement
- l'ensemble des arcs A correspond à l'ensemble des relations de précédence entre opérations ; un arc qui représente une contrainte de précédence entre une opération i et une opération j est valué par la durée p_i de l'opération i : $t_j \geq t_i + p_i$. Il est alors possible d'associer au sommet i la date de début de l'opération correspondante t_i. Cette valeur est appelée, en termes de graphe, le potentiel. En ce qui concerne les opérations fictives, on considère une relation de précédence entre l'opération 0 et toute opération réelle, valuée par 0 (durée de l'opération fictive 0) ainsi qu'une relation de précédence entre toute opération réelle j et l'opération $n+1$, valuée par p_j (durée de l'opération j)
- l'ensemble des arêtes E correspond aux disjonctions à arbitrer, c'est-à-dire aux relations entre les opérations partageant la même ressource, pour lesquelles il faut déterminer l'ordre dans lequel on doit exécuter ces opérations. Dans le cas d'un problème à une machine, toutes les opérations doivent être exécutées sur la seule machine considérée, donc il existe potentiellement une arête entre tout couple d'opérations réelles. La valuation associée à une arête définie entre l'opération i et l'opération j dépend du sens dans lequel on parcourt cette arête : p_i si l'on va de i vers j (ce qui correspond à la décision d'exécuter l'opération i avant l'opération j) ou p_j si l'on va dans le sens inverse.

Le problème d'ordonnancement consiste alors à arbitrer toutes les disjonctions : une fois qu'une disjonction est arbitrée, l'arête correspondante est remplacée dans le graphe par un arc orienté dans l'ordre de traitement choisi. Lorsque tous les arbitrages ont été effectués, on dispose d'une solution (ordonnancement complet). Le calcul des dates associées à cette solution se fait par l'intermédiaire d'un calcul de plus long chemin dans le graphe, qui est désormais totalement orienté (toutes les arêtes ont été remplacées par des arcs). Pour toute opération i, la date de début t_i est déterminée par la longueur d'un plus long chemin entre le sommet 0 et le sommet i du graphe. Quant au makespan, il correspond à la longueur d'un plus long chemin entre le sommet 0 et le sommet $n+1$, c'est-à-dire entre le début et la fin de l'ordonnancement.

Remarques :

- Un tel graphe peut bien entendu être utilisé dans le cas de problèmes à plusieurs machines. Un sommet est alors défini pour toute opération et chaque machine correspond à un ensemble de sommets (les opérations devant être exécutées sur cette machine) et d'arêtes formant une clique dans le graphe initial. Les sommets associés aux opérations d'un même travail sont reliés par des arcs représentant la gamme opératoire du travail.
- Les relations de précédence associées aux arcs doivent être transitives. En particulier, il convient de les mettre à jour après tout arbitrage. De la même manière, s'il existe un

circuit formé uniquement d'arcs, toute solution issue du graphe est infaisable (ici, la valeur associée à tout arc est positive)
- Si l'on retire du graphe l'ensemble E des arêtes, ainsi que les valuations associées aux arcs, on retrouve le graphe de précédence du problème, dont la structure est parfois spécifiée dans le champ β de la notation standard.
- Nous avons présenté le modèle du graphe disjonctif dans une sous-section consacrée aux problèmes à une machine. Bien entendu, dans le cas classique, sans time lags, le problème de base $1||C_{max}$ est résolu trivialement : tout ordonnancement sans temps mort est optimal, de makespan $C^{\star}_{max} = \sum_{1 \leq j \leq n} p_j$, et il est inutile de passer par le graphe disjonctif pour traiter ce problème. Cependant, ce modèle permet également de représenter les dates de disponibilité éventuelles r_j des travaux, ainsi que des durées de latence (ou queues) q_j. Nous rappelons que ces contraintes agissent de la manière suivante : $\forall j \in \{1, \ldots, n\}, t_j \geq r_j$ (un travail ne peut débuter son exécution avant sa date de disponibilité) et $\forall j \in \{1, \ldots, n\}, C_j = t_j + p_j + q_j$ (la date de fin du travail est la date de fin sur la machine à laquelle on ajoute la durée de latence). Pour prendre en compte ces contraintes additionnelles, il suffit de modifier les valuations des arcs issus de 0 d'une part et allant vers $n+1$ d'autre part : entre le sommet 0 et tout sommet j $(1 \leq j \leq n)$, l'arc a pour valeur r_j, et entre tout sommet j $(1 \leq j \leq n)$, l'arc a pour valeur $p_j + q_j$. Il est alors possible de modéliser, par exemple, le problème $1|r_j, q_j|C_{max}$, qui est NP-difficile au sens fort.

Ce modèle peut être très naturellement adapté de manière à prendre en compte des contraintes de time lags minimaux et maximaux. Nous avons vu que, dans le cas classique, une contrainte de précédence entre deux opérations i et j se traduit par l'existence d'un arc entre les sommets correspondant, valué par p_i et qui exprime la relation $t_j \geq t_i + p_i$. En autorisant une valeur arbitraire $a_{i,j}$ pour un tel arc, on peut représenter des time lags minimaux et maximaux. En effet, la relation devient $t_j \geq t_i + a_{i,j}$, caractérisant ainsi un time lag de début à début (start-start time lag). En formulant ces contraintes selon les conventions que nous avons fixées, on peut distinguer les situations suivantes, représentées sur la figure 3.1 :

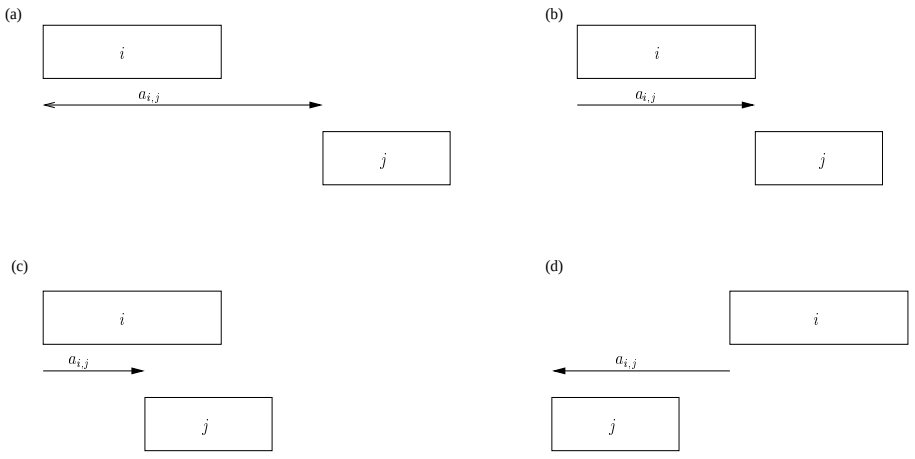

FIG. 3.1 – Différentes situations selon la valeur des arcs dans le graphe

- si $a_{i,j} > p_i$, il s'agit d'un time lag minimal entre i et j de valeur $a_{i,j} - p_i$ $(t_j \geq C_i + a_{i,j} - p_i)$

- si $a_{i,j} = p_i$, on retrouve le cas classique avec une contrainte de précédence simple entre i et j ($t_j \geq C_i$)
- si $0 \leq a_{i,j} < p_i$, il s'agit d'un time lag minimal négatif entre i et j, de valeur $a_{i,j} - p_i$, correspondant à une possibilité de chevauchement des deux opérations : l'opération j peut débuter dès que $a_{i,j}$ unités de temps se sont écoulées pour le traitement de l'opération i. La période de chevauchement dure au plus $p_i - a_{i,j}$ unités de temps
- si $a_{i,j} < -p_j$, il s'agit d'un time lag maximal entre j et i de valeur $-a_{i,j} - p_j$ ($t_i \leq C_j + (-a_{i,j} - p_j)$)

Il convient de noter que, pour avoir réellement un time lag maximal entre j et i tel que nous l'avons considéré jusqu'à maintenant, la dernière contrainte est nécessaire, mais il faut également une contrainte de précédence entre j et i du type $t_i \geq C_j$. On remarque ainsi que les time lags minimaux et maximaux ne sont pas tout-à-fait symétriques, dans la mesure où les time lags minimaux se substituent aux contraintes de précédence classiques alors que les time lags maximaux s'ajoutent à ces contraintes. Nous ne décrivons pas le cas où $-p_j \leq a_{i,j} < 0$, qui peut s'exprimer en termes de time lag minimal entre i et j ou en termes de time lag maximal entre j et i, avec dans les deux cas un chevauchement possible.

Dans le cas classique, il existe un ordonnancement réalisable associé au graphe disjonctif si et seulement si ce graphe ne possède pas de circuit (constitué uniquement d'arcs). En présence de time lags, la nouvelle condition nécessaire et suffisante de faisabilité est l'absence de circuit de longueur strictement positive [Bartusch et al.,88]. Des algorithmes classiques de plus long chemin en théorie des graphes, comme celui de Floyd-Warshall (voir par exemple [Lawler,76]), permettent de tester cette condition et le cas échéant de déterminer les dates de début au plus tôt de l'ensemble des opérations en $O(n^3)$, où n désigne le nombre d'opérations à ordonnancer.

Comme nous avons pu le constater dans le chapitre précédent, la présence de time lags a une influence considérable sur la complexité du problème. Quand le graphe a une structure quelconque, la question de l'existence d'une solution réalisable pour le problème à une machine avec time lags minimaux et maximaux est déjà en soi un problème NP-difficile au sens fort [Bartusch et al.,88]. Il suffit en effet de considérer le problème $1|r_j, q_j|$ avec des dates de disponibilité et des queues, qui, comme nous l'avons signalé, peuvent être traduites par des time lags minimaux entre les travaux réels et les travaux fictifs de début et de fin de l'ordonnancement, et d'ajouter une contrainte de time lag maximal entre les deux travaux fictifs 0 et $n+1$, de valeur égale au makespan optimal $C^\star$. Ainsi, une solution réalisable pour ce problème est une solution optimale du problème $1|r_j, q_j|C_{max}$, qui est NP-difficile au sens fort [Lenstra et al.,77].

b) Algorithmes polynomiaux et résultats de complexité quand le graphe de précédence a une structure particulière

Wikum et al. [Wikum et al.,94] sont parmi les premiers à s'intéresser à la complexité des problèmes à une machine en présence de contraintes de précédence généralisées. Ils considèrent à la fois des time lags minimaux et maximaux mais se limitent aux cas où ces contraintes sont sous forme de chaînes, ayant toutes pour extrémité finale le travail fictif noté $\star$ (voir figure 3.2).

L'objectif à minimiser est le makespan, et les auteurs identifient trois cas particuliers polynomiaux :

- Cas 1 : si chaque chaîne est composée d'un seul travail, lié au travail final par un time lag minimal, le problème est équivalent à $1||L_{max}$: il suffit d'exécuter les travaux dans l'ordre décroissant des time lags (qui correspond à la règle Earliest Due Date)

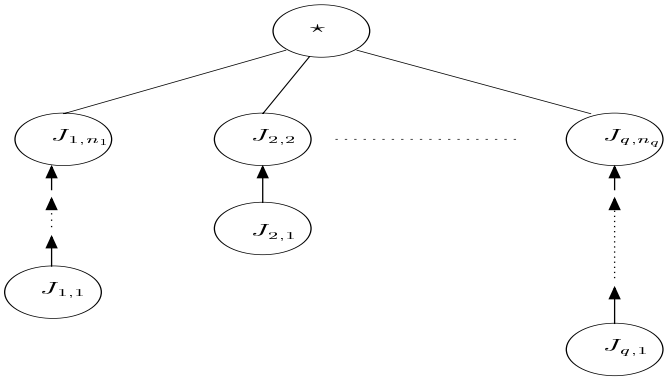

FIG. 3.2 – Structure des time lags considérés par Wikum et al.

- Cas 2 : si chaque chaîne est composée d'un seul travail, lié au travail final par un time lag maximal, le problème est équivalent à $1|r_j|C_{max} \leq \sum p_j$: il suffit d'exécuter les travaux dans l'ordre croissant des dates de disponibilité $r_j = \sum_{i \neq j} p_i - \theta_j^{max}$ et de tester s'il y a des temps morts avant la fin de l'ordonnancement
- Cas 3 : si chaque chaîne est composée d'un seul travail, lié au travail final par un time lag minimal ou maximal, il suffit d'ordonnancer d'abord les travaux concernés par les time lags minimaux, puis les travaux concernés par les time lags maximaux, en suivant les règles énoncées dans les cas précédents.

Il convient toutefois de noter que dans le cas des time lags maximaux, il n'existe pas forcément de solution réalisable.

Mis à part ces cas polynomiaux, les auteurs fournissent les résultats de NP-complétude suivants :

- Cas 4 : si chaque chaîne est composée d'un seul travail, lié au travail final par un time lag minimal et un time lag maximal, le problème est NP-difficile au sens fort
- Cas 5 : dans le cas d'un seul type de time lags (minimaux ou maximaux), dès qu'il existe une chaîne avec au moins 2 travaux, le problème est NP-difficile (au sens fort à partir de 3 travaux)
- Cas 6 : dans le cas de time lags minimaux uniquement, si toutes les chaînes possèdent au moins 2 travaux, le problème est NP-difficile au sens fort

Enfin, pour le cas particulier où seuls des time lags minimaux sont considérés, et où toutes les chaînes sont composées d'un seul travail, sauf une qui est composée de 2 travaux, une heuristique avec un ratio de performance dans le pire cas de 3/2 est proposée. D'autre part, dans le sous-cas où seule la chaîne composée de 2 travaux possède des time lags minimaux, il existe un algorithme pseudo-polynomial, mais ce sous-cas reste NP-difficile au sens ordinaire. Le problème est en fait ramené à deux problèmes de sac à dos qui peuvent être résolus à l'aide de la programmation dynamique.

Finta et Liu [Finta et Liu,96] considèrent le problème à une machine, avec dates de disponibilité, queues et time lags minimaux positifs entre les travaux, essentiellement du point de vue de la complexité. Ils montrent d'abord comment utiliser ce type de contraintes pour traiter l'indisponibilité de la machine, puis démontrent que le problème est NP-difficile même si toutes les opérations ont des durées unitaires. Si par contre les time lags minimaux sont tous égaux à 1, le problème est polynomial, en affectant un label particulier à chaque travail et en utilisant un

algorithme de liste se référant à un ordre lexicographique sur ce label [Coffman et Graham,72]. Une extension de cet algorithme est proposée pour traiter d'autres cas particuliers, avec des durées unitaires pour certains travaux, certaines contraintes de précédence sous forme de chaînes, et des dates de disponibilité et des queues dans $\{0, 1\}$. Finta et Liu s'intéressent également au cas où la préemption est autorisée : on retrouve un cas particulier polynomial similaire au cas non préemptif. En revanche, ils fournissent un exemple pour illustrer le fait que l'algorithme de liste n'est plus optimal dès qu'on considère deux machines parallèles.

Brucker et Knust [Brucker et Knust,99] fournissent de nombreux résultats de complexité pour les problèmes à une machine avec time lags minimaux : ils montrent tout d'abord que les problèmes à m machines parallèles avec préemption et d'éventuelles contraintes de précédence se réduisent à des problèmes à une machine à durées unitaires et time lags minimaux de valeur constante $m - 1$. Une réduction du même genre est présentée pour les problèmes $Pm|p_j = 1, r_j, intree|L_{max}$ et $Pm|p_j = 1, r_j, outtree|L_{max}$. D'autre part, les auteurs démontrent également que les problèmes à une machine avec contraintes de précédence se réduisent à des problèmes analogues, où les contraintes de précédence sont remplacées par des time lags minimaux de valeur constante arbitraire L. Enfin, ils fournissent des algorithmes polynomiaux optimaux pour deux problèmes de minimisation de la somme des dates de fin des travaux $\sum C_j$ sur une machine avec des travaux de durées unitaires et des time lags minimaux constants, et prouvent que le problème devient NP-difficile au sens fort dès que les durées sont quelconques, même si les time lags restent constants et forment des chaînes.

Ces dernières années, plusieurs études ont été menées au sujet de la complexité des problèmes à une machine en présence de time lags minimaux, dont le graphe associé est sous forme de chaînes. Munier et Sourd [Munier et Sourd,03] montrent ainsi que la recherche d'un ordonnancement de makespan minimal, lorsque les durées des opérations sont fixes, et que les time lags sont également constants, est pseudo-polynomiale, alors que si on cherche uniquement à déterminer la valeur minimale du makespan, le problème est polynomial. En fait, la question du codage des données en entrée du problème se pose, puisque celui-ci peut se faire de manière compacte, en représentant uniquement le nombre d'opérations dans chaque chaîne et les deux constantes pour les durées et les time lags. D'autre part, lorsque les time lags sont plus petits que les durées, la minimisation du makespan est polynomiale et la minimisation de la somme des dates de fin des travaux, avec des durées constantes, peut aussi être résolue en temps polynomial grâce à la programmation dynamique. Il ne faut pas oublier que pour ce dernier problème, chaque opération constitue un travail à part entière et contribue à ce titre au calcul de la fonction objectif.

Brucker et al. [Brucker et al.,03] [Brucker et al.,04b] fournissent des résultats complémentaires. Ils proposent un algorithme pseudo-polynomial pour le problème de la minimisation de la somme des dates de fin des travaux, lorsque les durées d'une part, et les time lags d'autre part, sont constants, en utilisant la structure particulière des ordonnancements optimaux. En outre, ils montrent que cet algorithme peut être ramené à un algorithme polynomial, grâce à une description compacte des solutions, et signalent que ce résultat est également valable pour l'algorithme de Munier et Sourd [Munier et Sourd,03] pour la minimisation du makespan.

c) Méthodes de résolution pour les problèmes avec time lags minimaux uniquement

Balas et al. [Balas et al.,95] s'intéressent au problème à une machine, avec dates de disponibilité, queues et time lags minimaux positifs entre les travaux. Ils démontrent que le problème

est NP-difficile au sens fort même si toutes les dates de disponibilité d'une part, et toutes les queues d'autre part, sont égales, que ce soit dans le cas non préemptif ou dans le cas préemptif. Ils justifient l'étude d'un tel problème par son lien avec le problème relaxé qui apparaît lors de l'utilisation de la Shifting Bottleneck Procedure pour la résolution du problème de jobshop [Adams et al.,88].

Cette méthode considère chaque machine successivement, et séquence les opérations sur la machine en ignorant les machines qui n'ont pas encore été traitées tout en maintenant fixes les séquences sur les machines traitées précédemment. En termes de graphe disjonctif, cela signifie que les ensembles d'arêtes correspondant aux machines non encore traitées sont supprimés, tandis que ceux qui sont associés aux machines déjà traitées sont remplacés par des arcs orientés. La machine choisie à chaque itération est la machine "goulot", pour laquelle le makespan du problème à une machine correspondant est maximal. Le problème à résoudre au cours du déroulement de la Shifting Bottleneck Procedure est le problème $1|r_j, q_j|C_{max}$. Bien que NP-difficile au sens fort [Lenstra et al.,77], il peut, dans la pratique, être résolu efficacement grâce à la Procédure par Séparation et Evaluation proposée par Carlier [Carlier,82]. De nombreuses études ont montré l'efficacité de cette méthode, et sa supériorié par rapport notamment aux règles de priorité traditionnelles. Néanmoins, l'ordonnancement effectué sur une machine au cours de la Shifting Bottleneck Procedure peut introduire de nouvelles contraintes de précédence généralisées entre des opérations devant être exécutées sur une machine qui n'a pas encore été exécutée. Au niveau du graphe disjonctif, cela se traduit par le phénomène suivant : lors de l'arbitrage d'une arête, un chemin est créé entre deux sommets appartenant à la même clique disjonctive (correspondant à deux opérations devant être effectuées sur la même machine). Cette nouvelle contrainte de précédence peut être assimilée à un time lag minimal.

La question qui se pose alors est la suivante : n'est-il pas judicieux de chercher à résoudre le problème à une machine en tenant compte de ces contraintes additionnelles ? On dispose en effet d'un problème plus contraint que le simple problème $1|r_j, q_j|C_{max}$, dont la solution pourrait conduire à de meilleures performances de la part de la Shifting Bottleneck Procedure. Dauzère-Peres et Lasserre [Dauzère-Peres et Lasserre,93] ont proposé une heuristique pour ce problème et ont comparé les résultats obtenus entre leur nouvelle Shifting Bottleneck Procedure et la méthode initiale. Dans l'ensemble, ceux-ci indiquent que la nouvelle méthode est plus performante, ce qui incite à étudier le problème à une machine en présence de time lags minimaux.

Balas et al. proposent d'aller plus loin, en développant une méthode exacte pour résoudre ce problème, et en intégrant cette méthode au schéma de la Shifting Bottleneck Procedure. La PSE de Carlier [Carlier,82] reposait sur les propriétés de l'heuristique de la plus grande queue [Potts,80]. En adaptant et en analysant cette heuristique en présence de time lags minimaux, les auteurs proposent une PSE pour le nouveau problème, avec deux types de branchement possibles, selon les cas : la règle de branchement "forte" est similaire à celle qui est utilisée dans la PSE de Carlier [Carlier,82]. Si les conditions d'application de cette règle ne sont pas vérifiées, une règle de branchement "faible", moins efficace, est utilisée. Pour chaque nouveau sous-problème créé, une mise à jour des dates de disponibilité et des queues est réalisée. Enfin, la borne inférieure employée, issue de la résolution du problème préemptif $1|pmtn, r_j, q_j|C_{max}$, est la même que celle de Carlier, et fait appel à la règle de Horn [Horn,74]. Les résultats expérimentaux obtenus indiquent que les améliorations apportées au sein de la Shifting Bottleneck Procedure grâce à la nouvelle PSE sont relativement importantes.

Le travail de Lourenço [Lourenço,98] est complémentaire de celui de Balas et al. [Balas et al.,95] : il considère le problème à une machine avec dates de disponibilités, queues et time lags minimaux, dans le cas particulier où ceux-ci sont sous forme de chaînes et où la préemption est

autorisée. Un algorithme polynomial est proposé pour résoudre de manière optimale ce problème : il consiste à appliquer la version préemptive de l'heuristique de la plus grande queue un nombre polynomial de fois, et peut être considéré comme une procédure par séparation et évaluation pour laquelle un nombre polynomial de noeuds sont explorés. L'algorithme est ensuite utilisé pour fournir des bornes inférieures au problème classique du jobshop $Jm||C_{max}$. Par rapport à la borne habituellement utilisée, qui repose sur la règle de Horn [Horn,74], pour résoudre le problème $1|pmtn, r_j, q_j|C_{max}$, la borne proposée permet d'éliminer d'avantage de noeuds, mais requiert un temps de calcul plus important. Selon les instances, la PSE intégrant cette borne est donc plus ou moins efficace que la PSE prise comme référence, issue de [Brucker et al.,94].

d) Méthodes de résolution pour les problèmes avec time lags minimaux et maximaux

Chu et Proth [Chu et Proth,96] introduisent un problème rencontré par un laboratoire automatisé d'analyses médicales : il s'agit d'ordonnancer des travaux sur une machine de manière à minimiser le makespan, tout en respectant des contraintes de time lags minimaux et maximaux formant des chaînes. L'application concrète est décrite dans la section 1.5. Outre la difficulté à obtenir une solution faisable, que nous avons déjà mentionnée, les auteurs précisent également que la relaxation lagrangienne ne peut pas être appliquée pour ce problème, du fait des time lags maximaux. Une preuve de NP-complétude est présentée, et un algorithme de calcul des dates d'exécution des opérations, pour une séquence donnée, est proposé : en raison des time lags maximaux, ce calcul n'est pas trivial, et les auteurs ont recours à un algorithme de plus long chemin dans le graphe correspondant. Deux approches de résolution sont proposées :

- Approche 1 : pour les problèmes de grande taille, en particulier ceux que rencontre le laboratoire d'analyses médicales, des heuristiques à base de règles de priorité sont testées. Les chaînes d'opérations sont séquencées successivement, selon trois règles différentes, ce qui conduit à trois heuristiques. Les auteurs montrent que le ratio de performance dans le pire cas est au moins de 2, pour ces heuristiques.
- Approche 2 : pour les problèmes de petite taille, une Procédure par Séparation et Evaluation est développée. Chaque noeud correspond à une séquence partielle et les descendants d'un noeud sont obtenus en ajoutant à la fin de la séquence partielle un travail parmi tous les travaux disponibles restants (c'est-à-dire ceux dont le prédécesseur éventuel est déjà placé). Une borne inférieure est obtenue en résolvant un problème à une machine avec des dates de disponibilité et des queues où la préemption est autorisée ($1|pmtn, r_j, q_j|C_{max}$).

Les expériences effectuées semblent indiquer que les problèmes qui nécessitent le plus grand temps de calcul pour la PSE sont ceux pour lesquels les contraintes de time lags sont "moyennes" : lorsque celles-ci sont fortes, le problème devient très contraint et de nombreuses solutions deviennent irréalisables, ce qui réduit rapidement l'espace de recherche. A l'inverse, lorsque les contraintes sont assez lâches, de nombreuses solutions sont proches de l'optimum, et les heuristiques permettent d'atteindre celui-ci rapidement. En ce qui concerne les heuristiques, les auteurs constatent que le temps de calcul augmente lorsque les contraintes deviennent de plus en plus lâches. Celles-ci ont par ailleurs été utilisées pour résoudre des problèmes réels rencontrés par le laboratoire, et ont conduit à des améliorations significatives de la productivité.

Le problème de la minimisation du makespan sur une machine où les travaux sont liés par des time lags minimaux et maximaux est étudié par Brucker et al. [Brucker et al.,99b]. Les auteurs montrent comment de nombreux problèmes d'ordonnancement à plusieurs machines avec time lags peuvent se réduire à ce problème à une machine, notamment :

1- le problème à machines parallèles dédiées, pour lequel chaque travail doit être exécuté sur une machine spécifiée (il ne s'agit pas d'un problème à machines parallèles classique, pour lequel chaque travail peut être traité par n'importe quelle machine)
2- le problème d'atelier généralisé, pour lequel les travaux sont composés d'opérations affectées à une machine spécifiée, et qui est une généralisation des problèmes classiques de flowshop, de jobshop et d'openshop.
3- le problème avec des travaux multi-processeurs, dans lequel chaque travail requiert une ou plusieurs machines simultanément pour son traitement (comme dans [Blazewicz et al.,92], [Hoogeven et al.,94], par exemple)
4- le problème avec machines multi-fonctions : à chaque travail est associé un ensemble de machines capables de le traiter et le travail doit être affecté à l'une d'entre elles (voir par exemple [Brucker et Schlie,90])
5- le problème à une machine avec temps de changement entre groupes de travaux : si le travail j, appartenant au groupe ν est exécuté immédiatement après le travail i, appartenant au groupe μ, le traitement nécessite entre ces deux travaux un temps de changement $s_{\mu,\nu}$ (voir par exemple [Monma et Potts,89]). Ce type de contraintes est équivalent à des temps de réglage dépendant de la séquence (*Sequence-Dependent Setup Times* ou SDST)

Il convient de noter que, puisque le problème à une machine avec time lags minimaux et maximaux est NP-difficile au sens fort, et que tous les problèmes cités précédemment appartiennent à la classe NP, l'existence de ces réductions polynomiales est évidente. L'intérêt des résultats présentés est essentiellement d'obtenir une manière explicite de transformer tous ces problèmes en problèmes à une machine. En particulier, les problèmes de flowshop de permutation avec time lags minimaux et maximaux, que nous considérons dans cette thèse, peuvent être ramenés à des problèmes à une machine. L'approche de résolution proposée par Brucker et al. est donc applicable à ces problèmes. Elle repose sur une Procédure par Séparation et Evaluation que nous décrivons maintenant.

Etant donné que la construction d'une solution faisable est déjà un problème NP-difficile, seules les feuilles de l'arbre d'exploration seront associées à des solutions faisables. Le schéma de séparation est basé sur le graphe disjonctif, et consiste à arbitrer une disjonction. En outre, des règles de sélection immédiate sont établies, afin d'arbitrer directement certaines disjonctions, à partir des choix précédents. Celles-ci utilisent des calculs de plus longs chemins dans le graphe, et définissent des fenêtres temporelles relatives à un travail. Une PSE de base (procédure notée α) est dans un premier temps développée, afin de tester l'existence d'une solution réalisable dont la valeur est inférieure à un seuil UB, correspondant à une borne supérieure de l'optimum. Des tests d'infaisabilité sont ajoutés pour détecter plus rapidement l'absence de solution réalisable : outre ceux qui sont déjà intégrés à la procédure de sélection immédiate, une recherche de circuit de longueur strictement positive est effectuée, ainsi que la résolution, pour chaque travail i, d'un problème $1|r_j, pmtn|L_{max} \leq 0$, où les dates de disponibilité r_j et les dates de fin souhaitées d_j correspondent aux fenêtres temporelles relatives à i.

Une seconde PSE, plus sophistiquée, est également proposée (procédure β). Celle-ci incorpore la procédure α et procède en trois phases :
- Phase 1 : amélioration de la borne inférieure
- Phase 2 : réduction de la borne supérieure
- Phase 3 : recherche de l'optimum

Une borne inférieure initiale est obtenue en résolvant le problème $1|r_j|C_{max}$, où les dates de disponibilité r_j sont déterminées à partir des fenêtres temporelles relatives au travail fictif de

début 0. En ce qui concerne la borne supérieure initiale, deux méthodes sont utilisées : la première calcule la somme des valeurs des plus grands arcs issus de chaque sommet dans le graphe, tandis que la seconde utilise la fermeture transitive du graphe disjonctif et calcule le plus long chemin de 0 à $n+1$ en passant par un seul travail. Il est important de noter que cette PSE fonctionne de manière "destructive", dans le sens où elle cherche essentiellement à prouver l'absence de solution réalisable pour une valeur donnée. En particulier, comme nous l'avons déjà mentionné, au cours de l'exécution de la procédure α, on ne dispose pas de bornes supérieures intermédiaires. Il est donc totalement inutile de stopper cette procédure avant son terme. Il s'agit là d'un inconvénient majeur dans la mesure où, pour les problèmes de grande taille, on ne pourra pas avoir recours à cette méthode en tant qu'heuristique, en fixant une limite de temps.

En ce qui concerne l'expérimentation des algorithmes proposés, les auteurs se sont heurtés à un problème : il n'existe en effet aucun jeu d'essai classique dans la littérature, pour les problèmes avec time lags minimaux et maximaux. En outre, la première série de problèmes, générés aléatoirement comme des problèmes à une machine, s'est révélée relativement facile à résoudre, l'optimum étant, dans la quasi-totalité des cas, égal à la borne inférieure initiale. Les auteurs se sont alors tournés vers des problèmes générés à partir de réductions de jeux d'essais classiques pour le jobshop [Fisher et Thompson,63] [Lawrence,84] et l'openshop [Taillard,93]. Les résultats obtenus sont plutôt satisfaisants, même si ils n'atteignent pas les performances des méthodes spécialement conçues pour ces problèmes. On peut donc penser qu'il en serait de même si l'on comparait cette approche de résolution aux méthodes que nous avons développées pour le flowshop de permutation avec time lags, et qui font l'objet du chapitre 4. D'autre part, les auteurs constatent qu'en plus du nombre de travaux, la structure des time lags (c'est-à-dire la structure du problème initial avant réduction éventuelle) a un impact sur la difficulté des problèmes.

Pour le même problème, Hurink et Keuchel [Hurink et Keuchel,01] proposent un algorithme de recherche locale basé sur une méthode tabou. Ils présentent dans un premier temps une méthode de plus long chemin permettant de déterminer la meilleure solution réalisable respectant une séquence donnée, si une telle solution existe. Ils définissent également la notion d'ordonnancement partiellement faisable, qui correspond à un ordonnancement respectant les contraintes de time lags minimaux, mais pas nécessairement les time lags maximaux. De la même manière que Brucker et al. [Brucker et al.,99b], plusieurs tests d'infaisabilité sont proposés (nombre d'itérations maximal dépassé lors de la construction d'une solution, mise à jour d'un nombre trop élevé de dates de début de travaux, existence de circuits de longueur positive composé de deux arcs). En ce qui concerne la méthode de recherche locale en elle-même, elle enrichit une méthode tabou par un processus de diversification. Etant donné que la question de l'existence d'une solution faisable est NP-difficile [Bartusch et al.,88], l'espace de recherche considéré est celui des ordonnancements partiellement faisables et l'objectif est défini en fonction des contraintes de time lags maximaux non respectées. Les solutions réalisables sont rendues irréalisables en ajoutant un time lag maximal entre les travaux fictifs de début et de fin, de valeur égale au makespan de la solution moins un. Les auteurs introduisent une décomposition des circuits dans le graphe associé en blocs de travaux et définissent le voisinage d'une solution relativement à ces blocs. La connexité de ce voisinage est démontrée. La solution initiale est déterminée selon des règles de priorité simples. Les tests expérimentaux concernent les mêmes instances que celles de Brucker et al. [Brucker et al.,99b], obtenues à partir de réductions de benchmarks pour le jobshop et l'openshop sans time lags. Là encore, les résultats sont assez encourageants, bien que la méthode soit moins performante que les algorithmes spécialement conçus pour les problèmes en question, qu'il s'agisse du jobshop ou de l'openshop.

e) Problèmes reposant sur un modèle différent de time lags

Tous les travaux présentés jusqu'ici concernent des problèmes pour lesquels les time lags induisent des contraintes de précédence, c'est-à-dire où chaque contrainte de time lag est implicitement associée à une contrainte de précédence. Ainsi, s'il existe entre deux opérations i et j un time lag minimal (positif), alors i précède forcément j dans toute solution faisable. Il pourrait être envisageable de considérer des contraintes de type time lags, qui ne préjugent en rien de la précédence entre les opérations, voire qui dépendent des décisions prises concernant l'ordre d'exécution des opérations, du type : si i est traitée avant j, alors il faut respecter un time lag minimal (maximal) entre i et j, de valeur $\theta^{min}_{(i,j)}$ ($\theta^{max}_{(i,j)}$) mais si au contraire j est traitée avant i, il faut respecter un time lag minimal (maximal) entre j et i, de valeur $\theta^{min}_{(j,i)}$ ($\theta^{max}_{(j,i)}$).

Janczewski et Kubale [Janczewski et Kubale,01] s'intéressent à ce genre de problèmes, dans le cas d'une machine avec opérations de durée unitaire, et en présence de time lags uniquement minimaux et symétriques ($\theta^{min}_{(i,j)} = \theta^{min}_{(j,i)}$). Le critère à minimiser est le makespan. Les contraintes sont également représentées sous forme d'un graphe pondéré. Les auteurs montrent que le problème est équivalent au problème de T-coloration et à la minimisation du T-span. Celui-ci, étudié dans le cadre de l'affectation de fréquence (Frequency Assignment Problem ou FAP), consiste à affecter à chaque sommet x du graphe une couleur $c(x)$, c'est-à-dire un entier positif, de sorte que si deux sommets i et j sont adjacents, on ait $|c(i) - c(j)| \geq r_{i,j}$, où $r_{i,j}$ est le poids (entier) de l'arête (i, j). Le T-span du graphe est la distance minimale, pour toutes les T-colorations possibles du graphe, entre la plus grande et la plus petite couleur utilisées. Pour plus de détails concernant ces problèmes, le lecteur pourra se reporter à [Roberts,91]. Grâce à l'équivalence entre les deux problèmes, les auteurs déduisent des résultats de complexité pour le problème d'ordonnancement, en fonction de la structure du graphe :

- si le graphe est biparti, le problème est polynomial
- si le graphe a un nombre cyclomatique borné, le problème est pseudo-polynomial
- si le graphe est complet, le problème est NP-difficile au sens fort (le problème est équivalent au problème de voyageur de commerce [Janczewski,99]).

Nous renvoyons le lecteur à [Gondran et Minoux,94] pour tout ce qui concerne les structures de graphes et les concepts associés.

3.2.2 Deuxième cas : plusieurs opérations par travail

Nous nous limitons dans cette section au cas où chaque travail est composé de deux opérations ($\forall j, n_j = 2$), puisque c'est la seule situation que nous avons rencontrée dans la littérature. Nous proposons d'étendre les notations employées pour le flowshop à ce type de problèmes, puisqu'il n'y a aucune ambiguïté : $p_{j,1}$, θ^{min}_j, θ^{max}_j et $p_{j,2}$ désignent respectivement la durée de la première opération, le time lag minimal entre les deux opérations, le time lag maximal entre les deux opérations et la durée de la seconde opération, pour le travail j.

a) Problèmes avec time lags minimaux uniquement

Dans sa thèse, essentiellement consacrée au problème $F2|\theta^{min}_j|C_{max}$, Yu [Yu,96] présente également quelques résulats de complexité pour le problème $1|n_j = 2, \theta^{min}_j|C_{max}$. En particulier, par analogie avec le flowshop à deux machines, il montre que ce problème reste NP-difficile au sens fort lorsque les durées sont égales pour les deux opérations de chaque travail, et que les time

lags ne peuvent prendre que deux valeurs arbitraires ($1|n_j = 2, p_{j,1} = p_{j,2}, \theta_j^{min} \in \{\theta_1, \theta_2\}|C_{max}$). De la même manière, le problème est toujours NP-difficile au sens fort si toutes les durées sont unitaires, les time lags étant quelconques ($1|n_j = 2, p_{j,k} = 1, \theta_j^{min}|C_{max}$). Ce résultat est repris dans [Yu et al.,04].

Lin et Haley [Lin et Haley,94] s'intéressent au problème de la minimisation du makespan sur une machine, où les deux opérations de chaque travail sont liées par un time lag minimal $1|n_j = 2, \theta_j^{min}|C_{max}$. Ils proposent un programme mathématique en variables mixtes pour modéliser le problème et présentent un certain nombre de propriétés :

- Propriété 1 : les ordonnancements dans lesquels les secondes opérations des travaux sont traitées après toutes les premières opérations, celles-ci étant exécutées sans temps morts, sont dominants
- Propriété 2 : pour une séquence donnée des premières opérations, il suffit de traiter les secondes opérations selon leurs dates de disponibilité (autrement dit, on est ramené au problème $1|r_j|C_{max}$ qui est résolu en séquençant les travaux dans l'ordre croissant des r_j). Le problème revient donc à déterminer l'ordre optimal d'exécution des premières opérations
- Propriété 3 : le problème étudié est équivalent au problème $F2|\theta_j^{min}|C_{max}$, qui est NP-difficile au sens fort.

Plusieurs cas particuliers polynomiaux sont présentés :

- Cas 1 : $p_{j,2} = 0$
- Cas 2 : $max_j\{\theta_j^{min}\} \leq min_j\{p_{j,1} + \theta_j^{min}\}$
- Cas 3 : $\sum_j(p_{j,1}) + max_j\{\theta_j^{min}\} \leq min_j\{\theta_j^{min} + p_{j,2}\}$.

On y retrouve notamment des cas équivalents à ceux décrits dans la section 2.4.1. Une heuristique à plusieurs variantes, issue de ces cas polynomiaux, est proposée. Une Procédure par Séparation et Evaluation est également développée pour résoudre le problème de manière exacte. Celle-ci utilise les heuristiques précédentes afin d'obtenir une borne supérieure initiale. Chaque noeud de l'arborescence est associé à une séquence partielle des premières opérations des travaux (l'ordre d'exécution des premières opérations des travaux est suffisant pour caractériser une solution, d'après les résultats précédents). En ce qui concerne les bornes inférieures, des relaxations conduisant aux problèmes polynomiaux $1|r_j|C_{max}$, $1|q_j|C_{max}$ et $F2||C_{max}$ sont employées. Une autre borne inférieure utilise également le ratio de performance dans le pire cas de l'heuristique conçue. Indépendemment de l'étude expérimentale, les auteurs décrivent les points forts et les points faibles de chaque heuristique, en termes de qualité de la solution fournie et de temps de calcul. D'autre part, ils signalent également pour chaque borne inférieure le type d'instances sur lesquelles elle est susceptible d'être précise. Les tests expérimentaux, qui concernent des problèmes comportant jusqu'à 11 travaux, indiquent que, lorsque le nombre de travaux augmente, l'influence de la borne inférieure sur le temps de calcul se ressent, tandis que le choix de l'heuristique initiale n'a pas d'impact significatif.

Pour le même problème $1|n_j = 2, \theta_j^{min}|C_{max}$, Gupta [Gupta,96] énonce des résultats similaires en ce qui concerne la complexité, les ensembles de solutions dominantes et les bornes inférieures, et propose plusieurs heuristiques, basées sur des règles de priorité ou sur l'algorithme de Johnson [Johnson,54] : celles-ci génèrent dans un premier temps des ordonnancements de "permutation", c'est-à-dire pour lesquels l'ordre des secondes opérations des travaux est le même que celui des premières opérations, puis appliquent des procédures d'amélioration, qui conduisent à des solutions qui ne sont pas nécessairement de permutation (l'ensemble des ordonnancements

de permutation n'étant pas dominant). D'après les résultats expérimentaux, le gain de qualité apporté par ces procédures est significatif. D'autre part, l'efficacité des heuristiques croît avec le nombre de travaux.

b) Problèmes avec time lags minimaux et maximaux

La plupart des articles considérant des problèmes à une machine, avec deux opérations par travail, liées non seulement par des time lags minimaux, mais aussi des time lags maximaux, sont ceux qui traitent le cas du *coupled task*. Ce problème a été introduit par Shapiro [Shapiro,81], en relation avec le problème d'ordonnancement d'un système radar (voir la section 1.5). Les travaux sont constitués de deux opérations séparées par un délai exact, c'est-à-dire par un time lag minimal et un time lag maximal égaux ($\theta_j^{min} = \theta_j^{max} = \theta_j$). Orman et Potts [Orman et Potts,97] présentent une étude approfondie de ce problème, du point de vue de la complexité, dans le cas de la minimisation du makespan. Ils montrent tout d'abord que le problème est réversible, c'est-à-dire qu'on obtient un problème équivalent en inversant le rôle des deux opérations pour tous les travaux. En outre, ce problème peut aussi être considéré comme étant un problème de jobshop sans attente à deux machines, dans lequel chaque travail est composé de trois opérations et la seconde machine n'est pas goulet ($J2|n_j = 3, no-wait, M2\ non\ bottleneck|C_{max}$). Parmi tous les cas particuliers envisageables, les auteurs démontrent que très peu sont polynomiaux : $p_{j,1} = \theta_j = p$ et son inverse, $p_{j,1} = p_{j,2} = p, \theta_j = \theta$, ainsi que tous les cas particuliers qui en sont issus. Un seul problème est répertorié comme étant ouvert : $p_{j,1} = p_1, \theta_j = \theta, p_{j,2} = p_2$, et tous les autres problèmes sont NP-difficiles au sens fort. On peut noter que le cas $p_{j,1} = p_{j,2}$ figure aussi dans [Yu,96] [Yu et al.,04], mais avec des opérations de durées unitaires $p_{j,k} = 1$.

Le problème ouvert, mentionné précédemment, est examiné par Ahr et al. [Ahr et al.,04]. Les auteurs définissent un type de motifs pour représenter l'état de la machine pendant le délai du dernier travail séquencé, en utilisant un bit pour chaque unité de temps (0 indiquant que la machine est inactive et 1 qu'elle est occupée). Ils majorent le nombre total de motifs possibles et introduisent un graphe pour représenter les transitions entre ces motifs, où chaque transition correspond à l'ajout d'un travail et est valuée par l'augmentation de makespan résultant de cet ajout. Le problème est alors équivalent à la recherche d'un plus court chemin possédant exactement $n-1$ arcs dans le graphe. Une condition de dominance est présentée afin de réduire le nombre d'arcs dans le graphe. Il en résulte un algorithme exact, linéaire selon le nombre n de travaux, pour une valeur fixée de θ. Enfin, les auteurs proposent une analyse de la structure des ordonnancements optimaux quand n tend vers l'infini, en relation avec les circuits de poids moyen minimal dans le graphe.

Récemment, Potts et Whitehead [Potts et Whitehead,05] se sont intéressés au cas où le délai entre les deux opérations d'un travail n'est plus exact, mais borné par un time lag minimal et un time lag maximal. Ce problème étant plus général que le *coupled task*, il est également NP-difficile au sens fort. Pour sa résolution, les auteurs proposent des algorithmes de recherche locale associant à chaque solution la séquence de traitement des $2n$ opérations. L'espace de recherche est réduit en éliminant les solutions invalides, qui sont trivialement irréalisables, et l'évaluation d'une solution est réalisée grâce à un calcul de plus long chemin dans un graphe, de complexité $O(n^2)$. Parmi les algorithmes proposés, on retrouve des algorithmes de descente et des méthodes tabou réactives, c'est-à-dire pour lesquelles la taille de la liste des solutions tabou varie afin de privilégier l'intensification dans certains cas et la diversification dans d'autres. Un mécanisme de réparation est incorporé dans les heuristiques susceptibles d'aboutir à des solutions

irréalisables. Par ailleurs, une méthode approchée par construction est également développée, en deux versions : une version déterministe et une version aléatoire. Elle consiste à fusionner certains travaux, choisis en fonction d'indicateurs spécifiques, en macro-travaux puis à les ordonnancer sur la machine. Les auteurs affirment que les expériences menées montrent qu'une application répétée de l'heuristique par construction sous sa forme aléatoire est nettement plus performante que les méthodes de recherche locale, notamment de type tabou. Ils justifient ce résultat par le coût élevé de l'évaluation d'une solution, dans le cas de la recherche locale.

3.2.3 Bilan concernant les problèmes à une machine

A la lumière de tous ces travaux, on constate que le problème d'ordonnancement à une machine avec time lags minimaux a fait et continue de faire l'objet de nombreuses études. En ce qui concerne la prise en compte supplémentaire de time lags maximaux, les recherches se développent selon deux directions :

1- La conception de méthodes de résolution efficaces pour le cas général, où les contraintes de time lags sont définies entre des couples d'opérations quelconques [Brucker et al.,99b] [Hurink et Keuchel,01]. Ces méthodes font généralement appel au graphe disjonctif, qui permet une représentation et une modélisation intéressantes en vue de sa résolution. Etant donnée la généralité du problème traité, les algorithmes développés ne prennent pas en compte la structure des time lags, et sont donc moins performants que des méthodes dédiées à des problèmes particuliers.

2- L'analyse des problèmes de *coupled task*, pour lequels les contraintes de time lags ont une structure spécifique, puisque celles-ci ne sont définies qu'entre les deux opérations appartenant à chaque travail, et sont (en général) plus "fortes" dans le sens où, pour chaque couple d'opérations liées, le time lag maximal est égal au time lag minimal.

3.3 Les problèmes à machines parallèles

Les problèmes d'ordonnancement sur machines parallèles avec time lags sont, à notre connaissance, essentiellement traités dans le cadre de l'ordonnancement de tâches informatiques. Les machines correspondent alors aux processeurs, et les time lags minimaux représentent des délais de communication. Cependant, comme nous l'avons souligné dans la section 1.5, le modèle considéré est dans ce cas différent, dans la mesure où on fait généralement l'hypothèse que de tels délais n'interviennent qu'entre des processeurs différents. Nous nous contentons donc d'évoquer l'existence de travaux sur ces problèmes, sans être aussi exhaustif que pour les problèmes d'atelier. Un état de l'art sur le sujet est proposé par Chrétienne et Picouleau [Chrétienne et Picouleau,95].

Le problème considéré par Munier et al. [Munier et al.,98] est plus classique : il s'agit de minimiser une fonction objectif en ordonnançant des travaux de durées quelconques sur m machines parallèles identiques, de manière à satisfaire les contraintes de time lags minimaux définies entre certains couples d'opérations. Les valeurs associées ne dépendent pas de l'affectation des travaux aux machines. Le paramètre ρ, défini comme étant le rapport entre le plus grand time lag et la plus petite durée, est introduit : il représente la "granularité" du problème (les travaux éventuels à durée nulle ne sont pas pris en compte dans sa définition). Les performances théoriques de deux types d'algorithmes de listes sont analysées :

– Type 1 : les algorithmes de listes de Graham [Graham,66], qui génèrent des ordonnancements sans délai, c'est-à-dire sans temps morts évitables. A partir d'une liste des travaux, la solution est construite progressivement de la manière suivante : dès qu'une machine se

libère, le premier travail disponible dans la liste est placé sur cette machine et est supprimé de la liste. Un travail est disponible si tous ses prédécesseurs ont déjà été placés et si les time lags associés sont respectés.
- Type 2 : les algorithmes de listes orientés travaux, qui séquencent les travaux un par un dans l'ordre d'une liste, sans remettre en cause l'ordonnancement des travaux déjà placés. La liste utilisée est supposée être une extension de l'ordre partiel induit par les contraintes de précédence.

Même si la classe des ordonnancements sans délai n'est pas dominante pour les fonctions régulières, les algorithmes du premier type sont efficaces pour les critères liés à l'utilisation des machines, comme le makespan par exemple, du fait qu'ils évitent autant que possible les temps morts. Ainsi, Munier et al. déterminent le ratio de performance de tout algorithme de Graham [Graham,66], utilisant une liste de travaux quelconque, pour le problème sans time lags $Pm|prec|C_{max}$. Cependant, les auteurs fournissent un exemple pour lequel un algorithme de Graham a un ratio non borné pour le problème $1|r_j|\sum w_jC_j$, qui est un cas particulier de problème à machines parallèles avec time lags minimaux. Ce résultat peut s'expliquer par le rôle bénéfique que peuvent jouer les temps morts pour le critère $\sum w_jC_j$: en effet, ils peuvent permettre à des travaux ayant un poids élevé d'être exécutés assez tôt et de ne pas être retardés par des travaux disponibles plus tôt, mais de poids plus faible.

En ce qui concerne les algorithmes de listes orientés travaux, les listes construites à partir des dates de fin des travaux issues d'une relaxation peuvent produire des solutions de très mauvaise qualité pour les problèmes à machines parallèles, même quand les dates de fin sont optimales (ratio de l'ordre de m sur certaines instances, pour le critère du makespan). Plutôt que les dates de fin, les auteurs proposent de trier les travaux dans l'ordre croissant de leur point médian, c'est-à-dire de la date à partir de laquelle la moitié du traitement de chaque travail est réalisée. Dans le cas non préemptif, le point médian du travail j est donné par $C_j - p_j/2$. Une relaxation linéaire simple, qui ne tient pas compte de l'affectation des travaux aux machines, est décrite pour le problème de minimisation de la somme pondérée des dates de fin des travaux $\sum w_jC_j$. L'algorithme faisant appel à la liste des travaux triés selon leur point médian constitue une heuristique avec un ratio de performance de 4, atteint asymptotiquement, tandis que la borne inférieure issue de la relaxation est supérieure à 1/4 de la valeur optimale. En réalité, la méthode employée permet d'obtenir une borne reliant, pour chaque travail, la date de début calculée par l'heuristique et son point médian obtenu par la relaxation. Les ratios pour les cas particuliers sans time lags (contraintes de précédence simples), et à une machine, sont également fournis.

3.4 Les problèmes de flowshop

Nous considérons ici les problèmes de flowshop, qui nous intéressent plus particulièrement. Nous passons très brièvement sur les résultats de complexité, que nous avons déjà abordés dans le chapitre 2, avant de mentionner les méthodes de résolution proposées pour différentes situations : restriction aux ordonnancements de permutation, prise en compte de time lags maximaux ...

3.4.1 Résultats de complexité

On retrouve dans la littérature un certain nombre de références présentant des résultats de complexité sur le flowshop avec time lags minimaux [Yu,96] [Rayward-Smith et Rebaine,96] [Yu et al.,04] [Brucker et al.,04a]). Outre ceux que nous avons déjà mentionnés dans le chapitre 2 et qui concernent essentiellement le makespan, des résultats portant sur d'autres critères sont donnés par Brucker et al. [Brucker et al.,04a] :

- pour $F|p_{j,k} = p, \theta_{j,k}^{min} = \theta_j^{min}| \sum C_j$: il est optimal d'exécuter les travaux dans l'ordre croissant des time lags θ_j^{min}
- pour $F2|p_{j,k} = 1, \theta_j^{min}| \sum w_j C_j$: le problème est NP-difficile au sens fort
- pour $F2|p_{j,k} = 1, \theta_j^{min}, r_j| \sum C_j$: le problème est NP-difficile au sens fort

Par ailleurs, en ce qui concerne les deux derniers problèmes, les auteurs présentent des conditions sous lesquelles l'ordre croissant des time lags est optimal, y compris pour un nombre quelconque de machines.

3.4.2 Problèmes à deux machines avec time lags minimaux

Depuis les travaux de Mitten [Mitten,59], beaucoup d'articles ont été publiés au sujet du flowshop à deux machines en présence de time lags minimaux, et éventuellement de temps de réglage et/ou démontage indépendants de la séquence. Cette section leur est consacrée.

a) Problèmes restreints au flowshop de permutation

Maggu et al. [Maggu et al.,82] montrent comment se ramener à des time lags minimaux de type stop-start (définis entre la fin de la première opération du couple et le début de la seconde), lorsque le problème comprend à la fois des time lags minimaux de types start-start, stop-stop et des temps de transport (c'est-à-dire stop-start).

Khurana et Bagga [Khurana et Bagga,84] considèrent le problème de la minimisation du makespan dans un flowshop de permutation à deux machines, en présence de temps de réglage et de time lags minimaux de type start-start et stop-stop, avec l'hypothèse que pour chaque travail, les time lags start-start et stop-stop sont égaux. Ils présentent des conditions sur les temps de réglage, qui, si elles sont vérifiées, permettent de construire un ordonnancement optimal.

Szwarc [Szwarc,86] démontre que, dans le cas général, le problème peut être résolu à partir d'une extension de la règle de Johnson et que la méthode proposée par Khurana et Bagga n'en est qu'un cas particulier. Pour les problèmes comportant plus de deux machines, il propose la même heuristique que dans [Szwarc,83].

Baker [Baker,90] s'intéresse au problème du flowshop de permutation à deux machines avec time lags minimaux, dans le cadre de la technologie de groupe : des groupes de travaux à traiter ensemble sont prédéterminés, et lors d'un changement de groupe, des temps de réglage (supposés indépendants de la séquence) interviennent. L'auteur exprime la règle de Mitten-Johnson [Mitten,59] en fonction des temps morts initiaux sur la seconde machine et finaux sur la première machine, lorsque les opérations sur la première (respectivement seconde) machine sont calées à gauche (respectivement à droite). Une méthode exacte polynomiale est proposée, qui procède en deux étapes : au sein de chaque groupe, les travaux sont exécutés selon la règle de Mitten-Johnson [Mitten,59], tandis qu'entre les groupes, des time lags minimaux négatifs sont utilisés pour représenter le chevauchement autorisé et la règle de Mitten-Johnson [Mitten,59] permet de séquencer les pseudo-travaux correspondant aux groupes. Ce travail rejoint en fait celui de Sekiguchi [Sekiguchi,82] pour le jobshop (voir la section 3.5.4).

Tous ces articles montrent finalement que la règle de Johnson, qui est probablement l'un des résultats les plus classiques en ordonnancement, ne se limite pas au problème de base du flowshop à deux machines, mais peut être étendue afin de prendre en compte un certain

nombre de contraintes supplémentaires (à condition de se limiter aux solutions de permutation) : time lags minimaux, temps de réglage, dates de disponibilité des machines, technologie de groupe... D'autre part, on retrouve deux résultats que nous avons déjà énoncés :

- les différents types de time lags (start-start, stop-start, stop-stop) conduisent à des problèmes équivalents
- les temps de réglage et de démontage peuvent être représentés par des time lags minimaux.

Des résultats analogues figurent dans [Espinouse et al.,00].

b) Problèmes de flowshop général

Dell'Amico [Dell'Amico,96] s'intéresse au problème $F2|\theta_j^{min}|C_{max}$. Il montre que le problème est NP-difficile au sens fort, y compris dans sa version préemptive $F2|pmtn, \theta_j^{min}|C_{max}$, mais que si la séquence de traitement des travaux est fixée sur la première machine, le problème devient polynomial. On est en effet ramené à résoudre le problème à une machine avec dates de disponibilité $1|r_j|C_{max}$ dont une solution optimale consiste à ordonner les travaux dans l'ordre croissant des r_j. Nous revenons sur ce résultat dans le prochain chapitre.

Une condition de dominance est proposée, pour laquelle il suffit de rechercher la solution optimale parmi les ordonnancements de permutation : le problème peut alors être résolu en temps polynomial grâce à la règle de Mitten [Mitten,59]. Plusieurs bornes inférieures sont présentées, avec leur ratio de performance dans le pire cas. Outre les bornes simples obtenues à partir de la charge des machines et de la durée totale des travaux, on retrouve des bornes calculées en relâchant la contrainte de capacité sur l'une des machines ou en réduisant les time lags (pour plus de détails, se référer à la section 4.8.3). A partir des séquences obtenues par relaxation, il est aussi possible de construire des solutions réalisables fournissant des bornes supérieures, dont les performances dans le pire cas sont calculées.

Enfin, l'auteur propose une approche de résolution basée sur une recherche tabou : chaque solution est caractérisée par la séquence de traitement des travaux sur la première machine, et le voisinage d'une solution est défini comme étant l'ensemble des solutions obtenues en sélectionnant deux positions i et j et en insérant à la position i le travail situé en position j dans la séquence de départ. L'évaluation complète de ce voisinage, de taille $O(n^2)$, nécessite $O(n^3)$ calculs. L'auteur propose une amélioration qui consiste à ne prendre en compte que les voisins susceptibles de réduire immédiatement le makespan de la solution courante, en vérifiant un ensemble de conditions nécessaires.

Les expériences menées permettent de comparer la solution obtenue par la méthode tabou et la solution exacte, fournie par la Procédure par Séparation et Evaluation de Brucker et al. [Brucker et Jurisch,92] pour le jobshop classique. Chaque instance pour le problème $F2|\theta_j^{min}|C_{max}$ est transformée en une instance de jobshop à n travaux et $n+2$ machines : à chaque travail du problème initial est associé un nouveau travail, composé de 3 opérations , correspondant à la première opération, au time lag minimal et à la seconde opération du flowshop. L'ensemble des premières (respectivement deuxièmes) opérations est traité sur la machine 1 (respectivement $n+2$) et le time lag de chaque travail j est traité sur une machine particulière $j+1$. On retrouve ici l'interprétation des time lags minimaux en tant qu'opérations effectuées sur des ressources non goulets.

Le schéma de génération de jeux d'essais utilisé et les résultats expérimentaux permettent de distinguer les classes d'instances en fonction de leur difficulté :

- Classe 1 : les instances pour lesquelles les durées sur l'une des machines sont plus grandes que les durées sur l'autre machine et les time lags sont les plus faciles à résoudre
- Classe 2 : les instances pour lesquelles les durées sur les deux machines sont du même ordre de grandeur et sont plus grandes (ou du même ordre de grandeur) que les time lags sont assez faciles à résoudre
- Classe 3 : les instances avec des time lags relativement grands par rapport aux durées opératoires sont les plus difficiles à résoudre, et la difficulté croît avec la taille des time lags.

Dans sa thèse, Yu [Yu,96] considère le même problème. Mis à part les nombreux résultats théoriques qui y figurent, qu'il s'agisse de complexité ou de dominance, des bornes inférieures, qui reposent sur le découpage des travaux et sur les propriétés du problème à durées unitaires $F2|p_{j,k}=1,\theta_j^{min}|C_{max}$, sont également présentées. Celles-ci sont détaillées dans la section 4.8.3.

3.4.3 Problèmes à m machines avec time lags minimaux

Szwarc [Szwarc,83] présente un modèle mathématique pour une généralisation du flowshop de permutation à m machines. Une borne inférieure et une heuristique, utilisant la règle de Johnson [Johnson,54] sont proposées. L'équivalence entre les différents types de time lags minimaux est également démontrée, de même que l'utilisation des time lags minimaux pour représenter des temps de réglage et de démontage.

Kim et al. [Kim et al.,96] étudient un problème plus complexe de type $Fm_\pi|SDST,\theta_j^{min}<0|\sum T_j$, rencontré dans une entreprise de fabrication de circuits imprimés, au niveau du processus d'assemblage. Les time lags minimaux négatifs sont dus à la présence de lots (*batches*) de transfert, de taille inférieure à celle du lot de traitement et modélisent une possibilité de chevauchement des opérations d'un même travail. Ils sont déterminés de manière à ce qu'il n'y ait aucun temps mort lors du traitement des produits d'un même lot sur une machine. Etant donné qu'une solution peut être entièrement caractérisée par la séquence de traitement des travaux, plusieurs heuristiques par voisinage utilisant cette propriété sont proposées et une analyse comparative est effectuée sur ces méthodes, ainsi que sur la règle EDD et une adaptation de l'heuristique NEH [Nawaz et al.,83]. Parmi les approches de résolution proposées, on retrouve une méthode itérative de plus grande pente, une méthode tabou, un recuit simulé et un algorithme de permutation de blocs de travaux. Une première série d'expériences est menée afin de permettre un réglage judicieux des paramètres intervenant dans ces méthodes, puis une seconde série d'expériences, impliquant un plus grand nombre de jeux d'essais, est effectuée afin de comparer les versions les plus efficaces de chaque approche de résolution. Les résultats semblent indiquer que pour le type d'instances considérées, l'approche basée sur le recuit simulé est la plus performante, suivie par la recherche tabou.

3.4.4 Problèmes avec time lags maximaux

Depuis quelques années, des méthodes de résolution ont également été proposées pour traiter les problèmes de flowshop en présence de time lags maximaux. Yang et Chern [Yang et Chern,95] proposent une Procédure par Séparation et Evaluation pour le problème $F2_\pi|\theta_j^{max}|C_{max}$. Le schéma de branchement est le schéma classique utilisé pour les problèmes de permutation : à chaque noeud est associé une séquence partielle et les fils d'un noeud donné sont obtenus en

ajoutant à la fin de la séquence partielle un élément (ici un travail) qui n'a pas encore été placé. Nous revenons sur ce schéma dans le chapitre suivant.

La borne inférieure employée est une borne simple tournée machine, qui relâche toutes les contraintes de précédence et de time lags maximaux entre les opérations des travaux qui restent à placer. La solution correspondant à la règle de Johnson [Johnson,54] fournit une borne supérieure initiale, et une condition sur les time lags permet de vérifier si cette première solution est optimale. Les auteurs utilisent le même principe pour compléter les séquences partielles à chaque noeud, et éliminer ainsi davantage de branches dans l'arborescence. La stratégie d'exploration consiste à traiter le noeud dont la borne inférieure associée est la plus petite (en cas d'égalité, le noeud de plus grande profondeur est choisi). L'algorithme est d'autant plus performant que les time lags maximaux sont grands, c'est-à-dire que les contraintes sont "lâches". On se rapproche en effet de plus en plus du cas classique, pour lequel la règle de Johnson [Johnson,54] est optimale. Les auteurs signalent qu'il est possible d'adopter une approche similaire pour des problèmes à plus de deux machines, sans pour autant expliquer comment généraliser la borne inférieure et l'heuristique.

3.4.5 Problèmes avec time lags minimaux et maximaux

Bouquard [Bouquard,04] présente une application de l'algèbre Max-Plus à certains problèmes d'ordonnancement. Il montre comment cette théorie permet de modéliser des problèmes de minimisation du makespan dans un flowshop de permutation à m machines impliquant des contraintes variées telles que des temps de montage et démontage, des time lags minimaux, des dates de disponibilité des travaux et des durées de latence, des travaux sans attente, des time lags maximaux et des stocks de capacité limitée entre les machines. Les travaux sont représentés par des matrices, et les produits de telles matrices correspondent à des séquences de travaux.

Cette représentation particulière permet d'exprimer, pour le flowshop de permutation à deux machines, la règle de Johnson [Johnson,54] dans un cadre beaucoup plus général, incluant les cas avec temps de montage et démontage, et avec des time lags minimaux. Ce résultat utilise le fait que les matrices des travaux sont triangulaires, ce qui n'est plus vraie en présence de time lags maximaux. Pour résoudre le problème à deux machines, qui est NP-difficile, une Procédure par Séparation et Evaluation est proposée : le schéma de séparation est classique, avec ajout de travail en fin de séquence partielle et les deux bornes inférieures font appel respectivement à la règle de Johnson [Johnson,54], optimale pour le problème sans time lags, et à l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64], optimale pour le problème sans attente. Les séquences issues des bornes inférieures sont également employées en tant que solutions heuristiques pour obtenir des bornes supérieures.

D'après les résultats expérimentaux, la borne supérieure issue de l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64] est meilleure que celle de Johnson [Johnson,54] tandis que c'est plutôt l'inverse pour les bornes inférieures, qui sont cependant complémentaires et beaucoup plus performantes que celle de Yang et Chern [Yang et Chern,95]. Enfin, il est intéressant de noter qu'un autre schéma de séparation, original, est présenté, bien qu'il ne soit pas expérimenté pour le problème avec time lags maximaux : il s'agit d'agréger progressivement les matrices correspondant aux travaux, sans nécessairement procéder par ajout en fin de séquence partielle, mais en considérant également la possibilité de regrouper des travaux non encore placés de manière à obtenir un macro-travail.

Deppner [Deppner,04] [Deppner,03] considère des problèmes d'ordonnancement d'atelier beau-

coup plus généraux, avec des gammes de fabrication quelconques, pas nécessairement linéaires, des contraintes de calendrier sur les machines, et des time lags minimaux et maximaux. Comme nous l'avons déjà mentionné, pour ce type de problèmes, la construction d'une solution réalisable est déjà un problème NP-difficile. L'objectif est de concevoir des méthodes de résolution efficaces, capables de fournir des solutions respectant l'ensemble des contraintes, pour des instances issues de situations réelles. Nous avons choisi de présenter ces travaux dans la section consacrée au flowshop, étant donné qu'ils se concentrent plus particulièrement sur ce type d'atelier, notamment au niveau des expériences menées. Néanmoins, la méthodologie générale reste applicable à une grande variété de problèmes d'ordonnancement.

Des algorithmes de construction à base de liste de priorité sont développés, qui prennent en compte notamment les time lags maximaux. L'auteur illustre les difficultés rencontrées du fait de la présence de telles contraintes, à travers le phénomène du "saute-mouton" qui survient lorsque deux couples d'opérations liées par des time lags maximaux sont en concurrence pour être placées. Si aucune solution n'est apportée à ce problème, les dates de disponibilité des opérations augmentent inutilement, l'exécution de ces opérations est donc retardée et la solution obtenue est fortement dégradée. Pour y remédier, le placement d'opérations individuelles dans l'algorithme de construction est remplacé par un placement de grappes d'opérations. L'ordonnancement des grappes les unes par rapport aux autres peut alors se faire facilement, à partir de règles simples et le problème de la construction d'une solution réalisable se pose uniquement à l'intérieur de chaque grappe. Sous certaines hypothèses, les algorithmes proposés convergent et se révèlent beaucoup plus efficaces que les algorithmes de base en ce qui concerne la recherche d'une solution réalisable.

En outre, dans le cadre du flowshop, si les travaux sont indépendants, la décomposition en grappes conduit naturellement à regrouper les opérations appartenant à un même travail. Les méthodes proposées consistent alors à placer les travaux les uns après les autres. L'auteur soulève la question de la dominance des ordonnancements de permutation : en présence de time lags maximaux, il donne une condition suffisante de dominance, qui est cependant plus restrictive que celle de la Propriété 6 de la section 2.5.3, et dans le cas de time lags minimaux uniquement, il montre que si ceux-ci sont bornés, tout comme les durées opératoires, le rapport entre le meilleur makespan parmi les ordonnancements de permutation et le makespan optimal parmi tous les ordonnancements tend vers 1 lorsque le nombre de travaux tend vers l'infini.

Au niveau de la résolution du problème, un algorithme génétique est développé. Un codage de permutation est adopté, qui représente l'ordre de placement des grappes, c'est-à-dire des travaux, avec deux variantes possibles pour la construction de la solution : la première se contente de déterminer l'ordonnancement de permutation correspondant à la séquence codée et se limite donc au flowshop de permutation, tandis que la seconde cherche à exploiter, lors du placement d'un travail, les trous générés durant les étapes précédentes. L'algorithme fait appel à plusieurs opérateurs de croisement, certains étant relativement classiques pour les problèmes de permutation et d'autres cherchant à prendre en compte les spécificités du problème, notamment les contraintes de time lags maximaux. A chaque croisement, un opérateur est choisi aléatoirement, avec une probabilité qui rend compte des améliorations fournies par cet opérateur lors des utilisations précédentes. D'après les études expérimentales menées, ce processus d'auto-détermination se révèle plus efficace que le choix définitif d'un unique opérateur. D'autre part, il permet également d'introduire une certaine diversification dans la recherche et peut à ce titre remplacer l'opérateur de mutation. Enfin, l'algorithme génétique est complété par une recherche locale de plus forte pente utilisant le voisinage de Nowicki et Smutnicki [Nowicki et Smutnicki,96] qui repose sur l'identification des blocs critiques dans un ordonnancement. La méthode de recherche locale n'est appliquée qu'aux enfants, générés par croisement, avec une certaine probabilité, afin d'une part de limiter les effets de convergence prématurée et d'autre part de ne pas augmenter

de manière trop importante les temps de calcul. Les résultats expérimentaux obtenus montrent l'intérêt d'une telle hybridation.

3.4.6 Bilan concernant les problèmes de flowshop

De la même manière que pour les problèmes à une machine, on note pour le flowshop un certain déséquilibre entre les références qui traitent de problèmes avec time lags minimaux et celles qui prennent en compte des time lags maximaux. Nous avons déjà pu constater cette tendance au niveau des résultats théoriques (abordés dans le chapitre 2), il en est de même en ce qui concerne les méthodes de résolution. On peut tout de même noter que de nombreux articles s'intéressent à la généralisation de la règle de Mitten-Johnson pour le flowshop de permutation avec time lags minimaux, de manière à prendre en compte des contraintes supplémentaires.

Le manque d'attention porté jusqu'à maintenant aux problèmes de flowshop avec time lags minimaux et surtout maximaux justifie donc qu'on s'intéresse à ces problèmes, notamment au niveau du développement de méthodes de résolution efficaces. Le chapitre 4 est consacré à la présentation de nos propres approches de résolution pour ces problèmes.

3.5 Les autres problèmes d'ordonnancement d'atelier

Cette section dresse un panorama des différents travaux portant sur les autres problèmes d'ordonnancement d'atelier en présence de time lags. Parmi les organisations considérées, on retrouve des flowshops avec des travaux à opérations multiples, des flowshops hybrides, des openshops, et des jobshops.

3.5.1 Flowshops avec travaux à opérations multiples

Dans notre modèle, nous avons considéré, comme c'est le cas dans la majorité des études concernant le flowshop, que chaque travail était composé de m opérations, telles que la k-ième opération doit être exécutée sur la machine k. Autrement dit, chaque travail possède exactement une opération par machine (la possibilité de considérer des durées nulles permet de prendre en compte des travaux avec des opérations manquantes, à condition de ne pas se restreindre aux ordonnancements de permutation, pour lesquels une telle hypothèse pose problème [Sridhar et Rajendran,96]).

Riezebos et al. [Riezebos et al.,95] supposent que chaque travail j possède un nombre $N_{j,k}$ d'opérations devant être exécutées sur la machine k. Puisqu'il s'agit toujours d'un flowshop, la première de ces opérations a pour prédécesseur, dans la gamme de fabrication, la dernière opération sur la machine $k-1$, et la dernière de ces opérations a pour successeur la première opération sur la machine $k+1$. D'autre part, des time lags minimaux sont définis entre tout couple d'opérations successives au sein des travaux, que ces opérations soient affectées à la même machine ou pas. Ce problème est issu des systèmes de production flexibles, pour lesquels les produits subissent plusieurs traitements sur une même station de travail, séparés par des temps de transport et d'autres activités effectuées sur des ressources non critiques. Les auteurs considèrent que ce problème se situe au premier niveau d'une approche hiérarchique, le second niveau consistant à vérifier si les activités annexes mentionnées précédemment peuvent effectivement être réalisées durant le temps disponible. Ils proposent une formulation mathématique et plu-

sieurs bornes inférieures, qui utilisent la structure particulière du problème. Parmi ces bornes, on peut distinguer :

- Borne 1 : une borne tournée travaux, qui minore les dates de fin d'exécution des travaux
- Bornes 2 et 3 : deux bornes tournées machines, qui minorent les dates de fin d'utilisation des machines. Celles-ci peuvent être appliquées à l'ensemble des travaux, ou à des sous-ensembles
- Borne 4 : une borne basée sur la définition de dates de fin souhaitées artificielles : après avoir déterminé la machine goulet, il est possible, à partir d'une borne inférieure sur la date de fin d'utilisation de cette machine, de minorer les dates de fin au plus tard des opérations sur cette machine. On obtient ainsi des dates de fin souhaitées artificielles qui, combinées aux dates de début au plus tôt sur la machine, conduisent à un problème du type $1|r_j|T_{max}$. Ce problème étant NP-difficile, les auteurs ont recours à sa version préemptive, qui elle est polynomiale, pour obtenir une borne inférieure pour le plus grand retard réel. Si cette valeur est strictement positive, on améliore la borne inférieure initiale pour le makespan. Là encore, il est possible de travailler sur des sous-ensembles d'opérations.

Des heuristiques par construction sont également proposées, certaines utilisant les bornes inférieures précédentes pour la sélection des opérations à ajouter. Les auteurs distinguent en particulier les méthodes qui génèrent des ordonnancements actifs, et celles qui génèrent des ordonnancements sans délai.

Ces travaux sont poursuivis par [Riezebos et Gaalman,98]. Le même type d'approche est proposé, et une étude expérimentale est menée afin d'estimer l'impact de la taille des time lags sur la performance des heuristiques, à partir de techniques d'analyse de la variance (ANOVA) : si le rapport entre la valeur moyenne des time lags et la durée opératoire moyenne est supérieure à 20%, les heuristiques générant des ordonnancements sans délai sont meilleures que celles qui génèrent des ordonnancements actifs, alors que c'est l'inverse pour un rapport inférieur à 20%. Ce résultat peut paraître surprenant puisque l'ensemble des ordonnancements sans délai n'est pas dominant, tandis que celui des ordonnancements actifs l'est. Les auteurs proposent une explication pour interpréter un tel phénomène : les heuristiques construisant des ordonnancements actifs peuvent avoir tendance, lorsqu'une opération vient d'être placée, à donner la priorité à l'opération suivante du même travail, imposant ainsi un temps mort correspondant au time lag entre ces opérations. Ce n'est pas le cas des heuristiques construisant des ordonnancements sans délai qui évitent les temps morts quand cela est possible. Enfin, d'une manière générale, les heuristiques faisant appel aux bornes inférieures pour la sélection des opérations sont beaucoup plus efficaces que les heuristiques par construction de base, qui reposent sur des règles de priorité simples.

3.5.2 Flowshops hybrides

Hodson et al. [Hodson et al.,85] décrivent un problème industriel dans le domaine agro-alimentaire dans lequel des contraintes de type time lags maximaux interviennent. La solution retenue se concentre sur les deux dernières étapes du process, la cuisson et le refroidissement. L'atelier peut ainsi être considéré comme un flowshop hybride à deux étages, avec des opérations multi-processeurs (bien que ce modèle ne soit pas mentionné dans l'article). L'objectif n'est pas d'optimiser un critère mais simplement de déterminer une solution réalisable pour laquelle tous les travaux sont traités durant une journée de travail, si une telle solution existe. On est donc ramené à un problème de recherche d'un ordonnancement S tel que $C_{max}(S) \leq K$, où K représente la durée (constante) d'une journée de travail. Deux paramètres majeurs sont pris en compte dans

la construction de l'ordonnancement : la priorité affectée à chaque travail par l'utilisateur, et la date de cuisson au plus tôt, qui peut être assimilée à une date de disponibilité du travail (celle-ci dépend des décisions prises en amont, lors de l'étape de préparation). Des règles d'affectation des travaux aux machines à chaque étage sont proposées.

Botta et Guinet [Botta et Guinet,96], [Botta-Genoulaz,00] s'intéressent au problème de flowshop hybride à m étages avec des contraintes de précédence entre les travaux de type *outtree*, des temps de réglage et de démontage indépendants de la séquence et des time lags minimaux entre les opérations successives au sein des travaux. L'objectif est de minimiser le makespan (C_{max}) ou le plus grand retard (L_{max}), ce dernier étant plus général. Les auteurs proposent une formulation du problème comme un programme linéaire avec des variables binaires.

La méthode de résolution proposée procède en deux étapes : séquencement des travaux sur m machines en résolvant un problème de type $Fm|prec(outtree), NSDST, NSDRT, \theta_j^{min}|L_{max}$, dans lequel la durée d'une opération est divisée par le nombre de machines présentes à l'étage correspondant, puis affectation des opérations aux machines à partir d'un algorithme de liste. On retrouve cette approche, efficace notamment dans le cas de machines identiques à chaque étage, dans de nombreux travaux de la littérature portant sur le flowshop hybride [Guinet,96] [Vignier et al.,99].

Lors de la première étape, le problème $Fm|prec(outtree), NSDST, NSDRT, \theta_j^{min}|L_{max}$ est transformé en $Fm|prec(outtree), \theta_j^{min}|L_{max}$ en redéfinissant les durées et les time lags minimaux de manière à intégrer les temps de réglage et les temps de démontage. Quatre heuristiques classiques sont adaptées au problème spécifique considéré :

- les méthodes CDS [Campbell et al.,70] et NEH [Nawaz et al.,83], qui font partie des heuristiques les plus efficaces pour le problème $Fm_\pi||C_{max}$. Ces méthodes sont détaillées dans les sections 4.4.3 et 4.3.5
- la procédure de Szwarc [Szwarc,83], proposée initialement pour le problème $Fm_\pi|\theta_j^{min}|C_{max}$ dont nous avons déjà parlé dans la section 3.4
- la méthode de Townsend [Townsend,77] conçue pour le problème $Fm_\pi||L_{max}$, qui construit itérativement la solution en ajoutant à chaque étape, à la fin de la séquence partielle, le travail qui minimise une borne inférieure. Il s'agit en fait d'une méthode d'exploration d'une arborescence en profondeur, sans retour en arrière. Les contraintes de précédence entre les travaux permettent de réduire le nombre de branches issues de chaque noeud.

Pour les heuristiques cherchant à minimiser le makespan, le problème avec plus grand retard est transformé en problème avec makespan en ajoutant une machine fictive en dernière position $(m+1)$ sur laquelle la durée du travail j est donnée par $p_{j,m+1} = max_i\{d_i\} - d_j$. Nous présentons une technique assez proche dans la section 4.8.1.

Outre l'adaptation de ces heuristiques, deux nouvelles méthodes de séquencement sont proposées. La première, appelée VBA, tient particulièrement compte des contraintes de précédence, en attribuant à chaque travail un indice dépendant du nombre de ses successeurs dans toute l'arborescence. Les travaux sont alors placés selon l'ordre décroissant de cet indice et en cas d'égalité selon la règle EDD. L'autre méthode consiste à résoudre optimalement m problèmes $1|SDST, SDRT|L_{max}$ avec une adaptation de la règle EDD, où le k-ième problème correspond à l'étage k. Parmi toutes les séquences obtenues, celle qui minimise la fonction objectif du problème de flowshop est conservée. Une fois le séquencement réalisé, les travaux sont affectés aux machines, suivant un algorithme qui s'inspire des règles de la première machine disponible et de la dernière machine occupée, de manière à minimiser les temps morts sur les machines et les temps d'attente pour les travaux.

Une étude expérimentale est menée afin d'évaluer l'efficacité de chaque heuristique, selon le type de données et le critère : outre le plus grand retard L_{max}, le plus grand retard absolu, la somme des retards et la somme des dates de fin des travaux sont examinés. Plusieurs configurations sont testées, en fonction du signe des time lags modifiés. Selon le type de problème considéré, l'heuristique la plus performante est identifiée. Pour le flowshop hybride, la méthode CDS modifiée [Campbell et al.,70] semble être la meilleure, quelle que soit la configuration, tandis que pour le flowshop classique, la meilleure méthode dépend de la situation considérée.

Ruiz et Serifoglu [Ruiz et Serifoglu,05a], [Ruiz et Serifoglu,05b] cherchent à considérer des problèmes réalistes, prenant en compte un grand nombre de contraintes rencontrées dans l'industrie. Ils s'intéressent ainsi à un problème de flowshop hybride avec :

- des machines non identiques à chaque étage (la durée d'une opération dépend alors de la machine à laquelle elle est affectée)
- des contraintes de compatibilité opération-machine, qui limitent les possibilités d'affectation
- des dates de disponibilité initiales pour les machines
- des travaux ne visitant pas tous les étages
- des temps de montage dépendant de la séquence, anticipatoires ou pas
- des time lags minimaux positifs ou négatifs, dépendant des machines auxquelles les opérations sont affectées
- des contraintes de précédence entre les travaux

La recherche se limite à des solutions "de permutation", pour lesquelles l'ordre relatif des travaux les uns par rapport aux autres est conservé, lorsqu'ils sont traités sur la même machine. Les auteurs proposent un programme linéaire avec des variables binaires pour modéliser le problème. Celui-ci est résolu pour une série d'instances de petite taille, et les résultats sont analysés de manière à identifier l'impact de chaque contrainte sur la difficulté du problème. Il semblerait que les principaux facteurs agissant sur la difficulté soient ceux qui limitent le nombre de solutions réalisables, et que les distributions des données temporelles (durées, time lags, temps de montage, dates de disponibilité) soient presque sans effet. Ces travaux devraient se poursuivre avec le développement d'heuristiques, et notamment d'un algorithme génétique, pour traiter le problème.

3.5.3 Problèmes d'openshop

Les travaux concernant les problèmes d'openshop en présence de time lags se limitent presque exclusivement au cas de deux machines avec time lags minimaux uniquement. Les principaux résultats concernent la complexité de ces problèmes, avec le makespan comme critère.

Rayward-Smith et Rebaine [Rayward-Smith et Rebaine,92] fournissent un algorithme polynomial exact, lorsque les durées opératoires sont unitaires et les time lags sont constants. Ils montrent également que si les durées sont quelconques, le problème est NP-difficile au sens ordinaire. Ce résultat reste valable même si les durées opératoires sur une des deux machines sont nulles [Strusevich et Hall,97].

Rebaine et Strusevich [Rebaine et Strusevich,99] s'intéressent aussi au cas particulier où les time lags minimaux ne dépendent pas des travaux. L'heuristique qu'ils proposent, pour le cas de deux machines, a un ratio de performance de 3/2 si les time lags minimaux sont effectivement constants, et de 8/5 s'ils dépendent de l'ordre de visite des machines, ces bornes étant atteintes dans les deux situations. Il est en effet important de remarquer que dans le cas de l'openshop,

puisque l'ordre de visite des machines n'est pas déterminé à l'avance pour les travaux, il est tout-à-fait possible d'envisager des time lags qui varient selon cet ordre. Cette situation correspond, par exemple, à des temps de transport d'une machine vers une autre non symétriques. Cependant, la plupart des études considèrent des time lags minimaux symétriques.

A partir des résultats démontrés pour le flowshop à deux machines, Yu [Yu,96] prouve que les problèmes suivants à deux machines, pourtant très contraints, sont NP-difficiles au sens fort :
- Problème 1 : si les durées opératoires ne dépendent pas des machines mais uniquement des travaux ($p_{j,1} = p_{j,2}$), et que les time lags ne peuvent prendre que deux valeurs différentes
- Problème 2 : si les durées opératoires sont unitaires, et les time lags différents

Un cas particulier polynomial est cependant présenté dans [Rebaine et Strusevich,99] : celui pour lequel le plus grand des time lags est inférieur à la plus petite durée de traitement.

Pour le problème général à deux machines, Strusevich décrit une heuristique dont il analyse la performance théorique. Celle-ci partitionne dans un premier temps les travaux en fonction de l'ordre de visite des machines qu'on leur attribue, puis ordonnance les ensembles obtenus comme des flowshops de permutation. Les deux ordonnancements partiels sont alors combinés de manière à minimiser le makespan. Cette démarche peut être considérée comme une extension de l'algorithme de Gonzalez et Sahni [Gonzalez et Sahni,76], optimal pour le problème sans time lags $O2||C_{max}$. En ce qui concerne le partitionnement des travaux, deux algorithmes sont développés, selon qu'il existe ou pas une opération longue, c'est-à-dire telle que $p_{j,k} > (1/2)\ LB$ où LB désigne une borne inférieure naturelle, basée sur la charge des machines et la longueur totale des travaux. Dans les deux cas, la complexité des algorithmes est en $O(n.log\ n)$ et leur ratio de performance vaut $3/2$ et est atteint.

Enfin, on peut signaler les résultats de complexité décrits par Brucker et al. [Brucker et al.,04a] pour des problèmes d'openshop à deux machines avec time lags minimaux et d'autres critères que le makespan.

3.5.4 Problèmes de jobshop

Sekiguchi [Sekiguchi,82] présente des généralisations de la règle de Johnson [Johnson,54] pour le cas du jobshop de "permutation", avec time lags minimaux, dates de disponibilité initiale pour les machines, groupes de travaux prédéterminés et temps de réglage avant le traitement de chaque groupe. Dans le cas du jobshop, on peut classer les travaux en quatre catégories :
- Catégorie A : les travaux qui sont traités uniquement sur la première machine
- Catégorie B : les travaux qui sont traités uniquement sur la seconde machine
- Catégorie AB : les travaux qui sont traités d'abord sur la première machine, puis sur la seconde
- Catégorie BA : les travaux qui sont traités d'abord sur la seconde machine, puis sur la première

L'expression "jobshop de permutation" fait référence aux ordonnancements pour lesquels l'ordre d'exécution des travaux AB d'une part, et des travaux BA d'autre part, est le même sur les deux machines. La contrainte de disponibilité initiale des machines signifie que, contrairement à la situation classique, chacune des machines n'est pas nécessairement libre au début de la période d'ordonnancement, à la date 0, mais ne le devient qu'à partir d'une date donnée.

Dans son article consacré au flowshop à deux machines avec time lags minimaux, Dell'Amico [Dell'Amico,96] démontre une propriété intéressante du problème $J2|\theta_j^{min}|C_{max}$: soient $J_{1,2}$ et $J_{2,1}$ les ensembles des travaux qui visitent les machines dans l'ordre $(1,2)$ et $(2,1)$ respectivement. On définit les ensembles $\Omega_{1,2} = \{j \in J_{1,2}/p_{j,1} > 0, p_{j,2} > 0\} \cup \{j \in J_{1,2}/\theta_j^{min} > 0, p_{j,2} > 0\} \cup \{j \in J_{1,2}/p_{j,1} > 0, \theta_j^{min} > 0\}$, $\Omega_1 = \{j/\theta_j^{min} = 0, p_{j,2} = 0\}$, $\Omega_2 = \{j/p_{j,1} = 0, \theta_j^{min} = 0\}$ et $\Omega_{2,1} = \{1, 2, \dots n\} \setminus (\Omega_{1,2} \cup \Omega_1 \cup \Omega_2)$. Soient $\Pi_{1,2}^1$ et $\Pi_{1,2}^2$ les séquences, sur la première et la deuxième machine respectivement, correspondant à une solution optimale du problème de flowshop limité aux travaux de l'ensemble $\Omega_{1,2}$. De la même manière, soient $\Pi_{2,1}^2$ et $\Pi_{2,1}^1$ les séquences, sur la deuxième et la première machine respectivement, correspondant à une solution optimale du problème de flowshop limité aux travaux de l'ensemble $\Omega_{2,1}$ (qui visitent les machines dans l'ordre $(2,1)$). Une solution optimale du problème de jobshop $J2|\theta_j^{min}|C_{max}$ est obtenue en exécutant les travaux sur la machine 1 dans l'ordre $\Pi_{1,2}^1$, suivi d'une séquence arbitraire de Ω_1, suivie de la séquence $\Pi_{2,1}^1$, et sur la machine 2 dans l'ordre $\Pi_{2,1}^2$, suivi d'une séquence arbitraire de Ω_2, suivie de la séquence $\Pi_{2,1}^1$. Ainsi, pour résoudre le problème $J2|\theta_j^{min}|C_{max}$, on est ramené à traiter le problème $F2|\theta_j^{min}|C_{max}$. Ce dernier problème est NP-difficile au sens fort, étant donné qu'on ne se limite pas aux ordonnancements de permutation (voir chapitre précédent).
Si l'on exclut les deux articles précédents, et le cas particulier sans attente, la littérature sur les problèmes de jobshop avec time lags est très réduite. A notre connaissance, les seuls travaux véritablement consacrés à ce sujet précis, sont ceux de Deppner [Deppner,04], que nous avons déjà détaillés dans la section 3.4.5, et ceux du Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS) de Clermont-Ferrand, développés notamment dans le cadre de la thèse d'Anthony Caumond [Caumond et al.,04a] [Caumond et al.,04b] [Caumond et al.,05a] [Caumond et al.,05b]. Nous présentons ici une synthèse de ces travaux.

Le problème considéré est celui de la minimisation du makespan dans un atelier de type jobshop, dans lequel les opérations consécutives au sein des travaux sont liées par des contraintes de time lags minimaux et maximaux. Ce problème est issu d'un cas réel rencontré dans l'industrie sidérurgique, plus précisément dans un atelier de forge. Les auteurs proposent un modèle linéaire [Caumond et al.,04b], qui est une généralisation de celui qu'a proposé Manne pour le jobshop classique [Manne,60]. Etant donnée la difficulté du problème, une méthode exacte reposant sur ce programme n'est capable de traiter que des instances de petite taille dans un laps de temps raisonnable. Par conséquent, plusieurs approches heuristiques sont développées, dans lesquelles la même démarche est adoptée : le problème est représenté sous la forme d'un graphe disjonctif (voir la section 3.2). En combinant ce graphe avec une séquence de traitement des opérations sur les différentes machines, il est possible d'orienter le graphe, et si celui-ci ne possède aucun circuit de longueur strictement positive, on obtient un ordonnancement semi-actif respectant à la fois les contraintes de time lags et la séquence de traitement. Le makespan associé peut être déterminé par un calcul de plus long chemin. Pour générer la séquence des opérations, les auteurs proposent plusieurs algorithmes.

- Une méthode tabou, avec plusieurs exécutions [Caumond et al.,04a] [Caumond et al.,04b]. La solution initiale est construite en plaçant les travaux les uns après les autres, puis en calant les opérations au plus tôt, compte tenu des contraintes de time lags, afin d'obtenir un ordonnancement semi-actif. Il convient de noter que cette solution, qualifiée de canonique par les auteurs, est toujours réalisable, alors que dans le cas général où les time lags sont définis entre opérations quelconques de travaux différents, le problème de l'existence d'une solution faisable est NP-complet. Le voisinage retenu est celui de Nowicki et Smutnicki

[Nowicki et Smutnicki,96], qui repose sur la notion de bloc, c'est-à-dire de sous-séquence maximale d'opérations exécutées sur la même machine, et situées sur un plus long chemin dans le graphe.

- Un algorithme génétique hybridé avec une recherche locale [Caumond et al.,05a]. Le codage des chromosomes est un codage de permutation généralisée, qui fait intervenir des valeurs dupliquées [Bierwirth,95] : la k-ième apparition du travail j dans la séquence correspond à sa k-ième opération dans la gamme. L'opérateur de croisement utilisé est un opérateur GOX à deux points, inspiré de OX pour le problème de voyageur de commerce, qui cherche à conserver l'ordre relatif des opérations et qui est bien adapté au codage retenu. Il est important de noter que compte tenu de ce codage, des chromosomes différents peuvent représenter la même solution. La détection de tels "doubles" est réalisée en comparant la signature des solutions, définie à partir des dates de début des opérations. Si une nouvelle solution possède la même signature qu'une solution existante, elle n'est pas introduite dans la population. D'autre part, l'espace de recherche est constitué de toutes les solutions faisables semi-actives, ainsi que des solutions qui ne respectent pas les time lags maximaux. Afin de remédier à ce problème, la fonction de fitness f choisie pour évaluer une solution est définie par $f = w_1 f_1 + w_2 f_2$, où w_1 et w_2 ($w_1 >> w_2$) sont les poids affectés respectivement au nombre de contraintes de time lags maximaux violées f_1 et au makespan f_2. Une heuristique est proposée pour estimer la valeur de f_1. Plutôt que de partir du graphe totalement arbitré, on retire tous les arcs correspondant aux time lags maximaux et on les rajoute un par un, en effectuant à chaque itération un calcul de plus long chemin ; si l'ajout d'un arc conduit à l'apparition d'un circuit de longueur strictement positive, l'arc est supprimé et f_1 est incrémentée. La population initiale est générée suivant le même principe que pour la solution initiale de la méthode tabou, dans l'ensemble des ordonnancements canoniques. En ce qui concerne la mutation, celle-ci est réalisée à travers une recherche locale qui sélectionne le premier voisin améliorant rencontré, et qui est limitée à un nombre maximal d'itérations. Ce critère d'arrêt est introduit pour éviter la convergence prématurée de l'algorithme. Le voisinage retenu est identique à celui qui intervient dans la méthode tabou. Enfin, un processus de redémarrage est intégré, afin de diversifier la recherche.

- des heuristiques par construction, avec des règles de priorité simples [Caumond et al.,05b]. A chaque itération, on ajoute à l'ordonnancement partiel une opération choisie, parmi un ensemble d'opérations candidates, selon une règle. Celle-ci choisit en priorité, s'il y en a, une opération conduisant au non respect d'un time lag maximal. Dans ce cas, l'opération est ajoutée et une procédure de réparation est exécutée, de manière à décaler vers la droite ses prédécesseurs. Les auteurs démontrent que la solution partielle ainsi obtenue est réalisable. De la même manière que pour les approches précédentes, seuls les ordonnancements semi-actifs sont considérés.

En ce qui concerne l'évaluation des performances de ces différents algorithmes, les auteurs se heurtent à un problème important : l'absence de jeux d'essais dans la littérature pour le problème considéré. Ils proposent de générer des jeux d'essais à partir des instances connues, proposées pour le jobshop classique, en ajoutant des time lags maximaux : pour chaque travail, les time lags sont obtenus en multipliant la durée moyenne des opérations de ce travail par un coefficient x pour les time lags minimaux et y pour les time lags maximaux. Il convient de noter que les instances ainsi générées sont telles que les time lags ne dépendent que des travaux, et pas des machines. L'analyse menée se concentre essentiellement sur les time lags maximaux, qui ont

une forte influence sur la proportion de solutions réalisables. Pour les instances de petite taille, la comparaison avec l'optimum est possible, tandis que si le nombre d'opérations est élevé, les auteurs se réfèrent à une borne inférieure, qui correspond à la solution optimale du problème sans time lags (connue pour ces instances). L'algorithme génétique est légèrement plus efficace que la méthode tabou, et la performance des heuristiques basées sur des règles de priorité est assez médiocre. On constate par ailleurs que la qualité des méthodes décroît de manière significative quand les time lags maximaux diminuent, tout en restant positifs. La difficulté du problème en présence de faibles time lags maximaux est également illustrée par l'incapacité à générer aléatoirement un ordonnancement réalisable. En ce qui concerne le cas classique, l'algorithme génétique atteint dans la quasi totalité des cas l'optimum, et pour les problèmes sans attente, il se révèle meilleur que la méthode à voisinage variable VNS mais un peu moins bon que l'algorithme génétique couplé au recuit simulé, proposés par Schuster et Framinan [Schuster et Framinan,03]. Par contre, les temps de calcul nécessaires sont beaucoup plus importants que pour ces deux méthodes de résolution consacrées spécialement au cas sans attente.

3.5.5 Bilan concernant les autres problèmes d'atelier

Etant donné le très faible nombre de publications sur le sujet, il est difficile de tirer des conclusions significatives. Si l'on excepte les travaux portant sur la complexité, notamment pour l'openshop à deux machines avec time lags minimaux, la plupart des études portent sur des problèmes particuliers, pour lesquels il n'existe aucune autre référence. Néanmoins, ces articles sont dans l'ensemble (très) récents, ce qui tendrait à montrer que les recherches s'orientent de plus en plus sur ce type de problèmes.

3.6 L'ordonnancement de projet

3.6.1 Spécificités des problèmes d'ordonnancement de projet

Contrairement à l'ordonnancement d'atelier, le concept de time lags est assez répandu dans le domaine de l'ordonnancement de projet, et on trouve ainsi un nombre important d'articles dans lesquels de telles contraintes sont prises en compte. Le modèle le plus couramment rencontré est celui de l'ordonnancement de projet à contraintes de ressources ou RCPSP (*Resource Constrained Project Scheduling Problem*), pour lequel un ensemble d'activités doit être exécuté, de manière à minimiser la durée totale du projet (le makespan), tout en vérifiant un certain nombre de contraintes : à chaque durée est associé un ensemble de ressources dont elle a besoin pour sa réalisation, selon des quantités spécifiées. On distingue généralement les ressources renouvelables, qui sont restituées une fois l'activité terminée (machines, outils, opérateurs) et les ressources consommables (argent, matières premières). Toutes les ressources sont disponibles en quantité limitée. Les problèmes de RCPSP ne prennent en compte généralement que les ressources renouvelables. L'autre grand type de contraintes considérées dans le cadre de l'ordonnancement de projet correspond aux précédences entre activités. Outre la précédence classique, il est possible, à l'aide des time lags, de représenter des contraintes plus sophistiquées, régissant la réalisation temporelle des activités. De la même manière que pour l'ordonnancement d'atelier, on peut modéliser ces contraintes à l'aide d'un graphe. Pour une description de l'état de l'art sur le RCPSP, on consultera l'article de Herroelen et al. [Herroelen et al.,98], celui de Brucker et al. [Brucker et al. 99a] et celui de Neumann et al. [Neumann et al.,02].

D'un point de vue général, on peut considérer tout problème d'ordonnancement d'atelier comme un cas particulier de RCPSP, où les seules ressources sont constituées par les machines, et où les contraintes temporelles correspondent à la gamme opératoire des travaux. Il est alors naturel de chercher à employer les méthodes développées pour le RCPSP pour tenter de résoudre les problèmes d'ordonnancement d'atelier. Néanmoins, les spécificités de ces derniers les rendent extrêmement complexes : les contraintes de précédence sont peu nombreuses, puisqu'elles se limitent aux opérations des mêmes travaux, ce qui conduit à un grand nombre de séquences de traitement possibles et tend à augmenter la combinatoire du problème. Par ailleurs, les contraintes liées à la capacité très limitée des machines (en général unitaire, sauf dans le cas de machines parallèles) sont très fortes. Pour caractériser les contraintes du RCPSP, plusieurs facteurs ont été introduits :

- OS (pour *Order Strength*) défini initialement comme le rapport entre le nombre de relations de précédence (y compris transitives) et le nombre maximal de relations possibles ($n(n-1)/2$) [Mastor,70] ; dans le cas de problèmes avec time lags, on remplace le numérateur par le nombre de contraintes de time lags minimaux. La difficulté des problèmes a tendance à décroître lorsque OS augmente [Schwindt,96].
- RF (pour *Resource Factor*) qui représente la proportion moyenne de ressources requise par activité [Pascoe,66]. Plus la valeur est proche de 1, plus les activités ont tendance à utiliser un nombre important de ressources, et plus le problème est difficile à résoudre.
- RS (pour *Resource Strength*) défini pour chaque ressource k par $(a_k - b_k^{min})/(b_k^{max} - b_k^{min})$, où a_k, b_k^{min} et b_k^{max} désignent respectivement la capacité totale, la quantité minimale et la quantité maximale requises de la ressource k [Kolisch et al.,95]. L'influence des RS sur la difficulté du problème peut être représentée par une courbe en forme de cloche.

Comme nous l'avons signalé au début de la section, de nombreux articles s'intéressent au RCPSP en présence de time lags, à la fois minimaux et maximaux. Nous n'allons pas analyser chacun dans le détail, mais nous allons plutôt nous arrêter sur les plus significatifs d'entre eux.

3.6.2 Méthodes de résolution proposées pour le RCPSP avec time lags

Le travail de Bartusch et al. [Bartusch et al.,88] est certainement l'un des plus essentiels d'un point de vue théorique. Les auteurs présentent une approche structurelle pour traiter le RCPSP avec time lags, qui reprend le modèle du graphe disjonctif et les méthodes reposant sur une relation d'ordre, utilisées pour l'ordonnancement de projet avec des contraintes de précédence simple. L'ensemble des contraintes temporelles est représenté par une matrice de distances, qui sont en réalité les longueurs des plus longs chemins entre tout couple d'activités. Les contraintes de ressources considérées sont caractérisées par des demandes et des disponibilités variables dans le temps, mais les auteurs montrent comment se ramener à des grandeurs fixes dans le temps :

- pour les demandes, il suffit de décomposer chaque travail en sous-travaux en accord avec les périodes où la demande est constante, et d'utiliser des time lags entre les sous-travaux pour assurer la non-préemption ;
- en ce qui concerne les disponibilités, on peut introduire des travaux fictifs qui consomment une quantité de ressource correspondant à l'écart entre la disponibilité maximale et la disponibilité effective pendant chaque période.

Les auteurs font appel au concept d'ensembles interdits pour représenter les contraintes de ressources. Ces ensembles regroupent des activités qui ne peuvent pas être exécutées toutes simultanément sans quoi la quantité totale d'une ressource consommée dépasserait la capacité de celle-ci. L'équivalence entre les ensembles d'activités potentiellement parallèles et les antichaînes

d'une relation d'ordre partiel est également montrée, ce qui conduit à la définition d'ensembles interdits réduits. Le modèle ainsi construit permet de formaliser une grande classe de problèmes d'ordonnancement avec des contraintes de temps et de ressources, en utilisant une représentation standard. Deux résultats fondamentaux sont par ailleurs démontrés.

- Il suffit de se restreindre aux ordonnancements au plus tôt, qui correspondent aux ordonnancements semi-actifs, pour rechercher une solution optimale.
- L'existence d'une solution faisable est un problème NP-complet au sens fort, même dans le cas où toutes les durées sont unitaires.

Sur le plan pratique, une Procédure par Séparation et Evaluation est développée. Celle-ci prend en compte les contraintes temporelles d'origine et des contraintes additionnelles de précédence appliquées au sein des ensembles interdits. Deux branchements sont alors possibles :

- Branchement 1 : à chaque étape, on détermine un ensemble interdit à supprimer, et on considère toutes les contraintes permettant d'y parvenir.
- Branchement 2 : on considère directement toutes les contraintes qu'on peut ajouter, quel que soit l'ensemble interdit sur lequel elles agissent.

La matrice des distances est mise à jour après chaque branchement, et la procédure adopte une recherche en profondeur d'abord, qui facilite la manipulation des données. La borne inférieure utilisée repose sur l'ordonnancement au plus tôt, qu'on obtient en relaxant les contraintes de ressources ; on ne tient donc pas compte des ensembles interdits restant. En outre, les auteurs montrent l'importance des travaux qualifiés d' "essentiels", dans la mise à jour de la matrice des distances et dans la sélection des contraintes à ajouter lors de la séparation. Il s'agit en fait des travaux qui nécessitent une ressource critique ou qui interviennent directement dans le calcul de la fonction objectif.

Les expériences numériques effectuées indiquent que la méthode proposée est surtout efficace si le nombre d'ensembles interdits est petit, et que le cardinal de chacun d'eux est faible. Elle est notamment particulièrement adaptée à l'ordonnancement de projets rencontrés dans l'industrie du bâtiment. Si les contraintes de time lags prennent la forme de chaînes parallèles, comme c'est le cas pour les problèmes d'ordonnancement d'atelier, les temps de calcul requis sont très longs.

Dans la lignée de ces travaux, de nombreuses Procédures par Séparation et Evaluation ont été proposées pour le RCPSP avec time lags [De Reyck et Herroelen,98],[Fest et al.,99]. Le principe général de résolution est similaire et consiste à énumérer les ordonnancements réalisables vis-à-vis des contraintes temporelles (les time lags), en relaxant les contraintes de ressources. La prise en considération de celles-ci conduit à identifier des conflits sur ces ressources, qui sont résolus en ajoutant des contraintes.

Ainsi, la méthode présentée par De Reyck et Herroelen [De Reyck et Herroelen,98] associe à chaque noeud de l'arbre de recherche le graphe initial des contraintes temporelles, auquel des relations de précédence additionnelles ont été ajoutées afin de résoudre des conflits de ressources. Le conflit suivant (dans l'ordre chronologique) est traité en utilisant la notion d'alternative retardant minimale : si un tel conflit apparaît, c'est parce qu'un ensemble d'activités nécessitant une même ressource ne peuvent pas être exécutées simultanément, du fait de la capacité limitée de la ressource. Une alternative retardant minimale correspond alors à un ensemble minimal (au sens de l'inclusion) d'activités qui, si elles sont retardées, permettent aux autres activités d'être réalisées en même temps. Un tel décalage est obtenu en ajoutant des contraintes de précédence. Il convient de noter que pour chaque alternative retardant minimale, il existe plusieurs ensembles de contraintes possibles. Ces ensembles sont qualifiés de modes retardant minimaux. La séparation est effectuée selon tous les modes de toutes les alternatives permettant de résoudre

le conflit identifié (extension d'une stratégie pour le RCPSP classique). La borne inférieure employée est relativement simple et repose sur le calcul de la longueur d'un chemin critique dans le graphe associé au noeud. D'autre part, trois règles de dominance sont élaborées, basées respectivement sur la recherche d'alternatives redondantes, la recherche de modes redondants, et le calcul d'une borne inférieure plus sophistiquée, issue du RCPSP. La sélection du noeud depuis lequel le branchement sera réalisé dépend de l'un des deux critères suivants :

- la plus petite valeur de la borne inférieure (critère classique)
- la plus grande marge par rapport aux contraintes temporelles, c'est-à-dire le plus grand intervalle entre les dates de début au plus tôt et au plus tard.

Ce second critère conduit à une plus forte probabilité de parvenir à une solution réalisable, et est utilisé au début de l'algorithme, tant qu'aucune solution réalisable n'a été obtenue. Pour le même genre de raisons, les auteurs optent pour une recherche en profondeur d'abord. On peut également ajouter qu'un traitement préliminaire est effectué en amont de la PSE pour réduire l'espace de recherche : après analyse des contraintes de ressources, des relations de précédence sont ajoutées au graphe initial.

Les expériences numériques indiquent que la procédure est d'autant plus efficace que le nombre de time lags maximaux est élevé, que les valeurs associées sont petites (contraintes fortes) et que le nombre de time lags minimaux négatifs, représentant des possibilités de chevauchement, est élevé.

Franck et al. [Franck et al.,01a] s'intéressent aussi au RCPSP avec time lags. Ils proposent une Procédure par Séparation et Evaluation dont le schéma de séparation est identique à celui de De Reyck et Herroelen [De Reyck et Herroelen,98] et repose sur l'identification des alternatives retardant minimales. Pour chaque noeud créé, un algorithme pseudo-polynomial de point fixe permet de déterminer l'ordonnancement optimal lorsque seules les contraintes temporelles sont prises en compte (ordonnancement courant). Plusieurs bornes inférieures sont envisagées :

- une borne inférieure destructive, qui procède par dichotomie en cherchant à déterminer la plus grande valeur pour laquelle on peut prouver qu'il n'existe pas de solution réalisable
- une borne inférieure standard pour laquelle les contraintes de ressources sont relaxées
- une borne inférieure qui relaxe les contraintes de time lags et décompose les activités en sous-activités de durées et demandes unitaires
- une borne inférieure reposant sur un raisonnement énergétique

C'est cette dernière borne, souvent de meilleure qualité que les autres, qui est employée dans la PSE. De la même manière que pour les précédentes approches de résolution exacte, la recherche s'effectue en profondeur d'abord et une phase de pré-traitement est menée, dans laquelle des contraintes de précédence sont ajoutées afin de supprimer certains ensembles interdits. Afin de limiter les temps de calcul et de pouvoir traiter des problèmes de grande taille, des versions tronquées de cette approche sont présentées : une méthode d'approximation à paramètre ϵ, pour laquelle la borne supérieure courante correspond à la meilleure valeur obtenue divisée par $1+\epsilon$, un beam search, et une méthode par décomposition.

Cette dernière est basée sur un résultat de Neumann et Zhan [Neumann et Zhan,95], selon lequel le problème initial possède une solution faisable si et seulement si chaque sous-problème, défini par les composantes fortement connexes du graphe des contraintes temporelles, possède lui-même une solution faisable. Ainsi, une manière de traiter le problème de RCPSP avec time lags consiste à identifier les composantes fortement connexes du graphe, appelées structures de cycles, de résoudre chaque sous-problème correspondant de façon indépendante, puis d'aggréger les sous-ordonnancements obtenus. Les auteurs proposent d'appliquer la méthode d'approximation à paramètre ϵ à chaque sous-problème, d'ajouter éventuellement des time lags afin de

conserver strictement les ordonnancements partiels obtenus, puis d'appliquer un beam search sans retour en arrière pour obtenir une solution approchée. D'autres heuristiques sont également décrites : des méthodes à base de règles de priorité, qu'elles travaillent directement avec les activités ou par décomposition, avec les structures de cycles. Des listes d'activités sont générées par l'application d'une règle, de manière à être compatibles avec les contraintes de précédence et les structures de cycles (les activités appartenant à une même structure sont placées consécutivement dans la liste). Ainsi, à chaque itération lors de la construction de la solution, une seule activité est candidate. Le placement d'une activité peut conduire au non respect d'une contrainte de time lag maximal. Dans ce cas, un ré-ordonnancement est effectué, pour lequel une partie des activités déjà séquencées sont retirées de l'ordonnancement partiel, et les dates de début correspondantes sont incrémentées. En réalité, cette approche à base de règles de priorité, ainsi que la décomposition en structures de cycles, sont analogues aux méthodes proposées par Deppner [Deppner,04] pour les problèmes d'ordonnancement d'atelier, où les grappes sont assimilées aux structures de cycles. Outre toutes ces méthodes, un algorithme génétique et une procédure de recherche tabou sont également développées. Dans les deux cas, le codage d'une solution est réalisé sous forme d'une liste d'activités. L'application de l'algorithme de construction précédent permet alors d'obtenir un ordonnancement. Il convient de noter que les auteurs suggèrent de limiter le nombre total de ré-ordonnancements effectués. Si la limite est atteinte, l'algorithme de construction est stoppé sans avoir trouvé de solution réalisable. L'algorithme génétique accepte les solutions pour lesquelles des contraintes de time lags sont violées, en intégrant des pénalités dans la fonction de fitness. Il utilise des croisements à un ou deux points, et la mutation retenue consiste à échanger dans la liste deux activités consécutives, tout en continuant à respecter la précédence et les structures de cycles. En ce qui concerne la méthode tabou, quatre opérateurs de voisinage, adaptés d'opérateurs classiques, sont proposés, ainsi qu'une procédure de réparation afin de conserver des listes vérifiant les contraintes de précédence et les structures de cycles. Selon les situations, le voisinage peut éventuellement être réduit pour limiter les temps de calcul, ou à l'inverse être étendu pour obtenir une diversification suffisante. Les ordonnancements qui ne respectent pas les contraintes de ressources sont considérés dans l'espace de recherche, mais ils reçoivent une pénalité lors de leur évaluation. A la suite d'études expérimentales, Franck et al. parviennent aux conclusions suivantes : la règle de priorité la plus performante est celle qui classe les activités dans l'ordre croissant des dates de début au plus tard. Pour les instances de grande taille, les méthodes à base de règles de priorité parviennent plus facilement à des solutions réalisables que les méthodes arborescentes. Ceci est sans doute dû au fait que le ré-ordonnancement est effectué plus rapidement que le retour arrière (backtracking). L'utilisation de la décomposition en structures de cycles conduit dans la plupart des cas à une augmentation de l'écart avec une borne inférieure, c'est-à-dire à une dégradation des solutions. Elle peut néanmoins s'avérer utile lorsqu'elle est couplée à des PSE tronquées, puisqu'elle conduit alors à des temps d'exécution plus faibles. En ce qui concerne les méta-heuristiques, l'algorithme génétique semble légèrement plus performant que la recherche tabou.

La Procédure par Séparation et Evaluation proposée par Fest et al. [Fest et al.,99] pour le même problème se différencie essentiellement de la précédente par la manière de résoudre les conflits dus aux contraintes de ressources. Plutôt que d'ajouter des contraintes de précédence, l'algorithme procède par une mise à jour dynamique des dates de disponibilité des activités à retarder. Le principal inconvénient provient du fait que la suppression des conflits est faite temporairement, c'est-à-dire qu'un même conflit peut apparaître plusieurs fois, pour des noeuds appartenant à un même chemin dans l'arborescence. Cependant, les opérations qui en résultent sont moins coûteuses que celles de la méthode précédente, que ce soit en termes de nombre de

calculs ou en termes d'espace mémoire utilisé. En effet, la mise à jour des dates de disponibilité des activités s'effectue en $O(n^2)$ et la matrice des distances demeure inchangée, tandis que la mise à jour de cette matrice dans l'algorithme de De Reyck et Herroelen nécessite $O(n^3)$ opérations. Mis à part cette différence, le schéma de séparation est le même et les auteurs prouvent qu'il est correct, c'est-à-dire qu'il conduit nécessairement à une solution optimale s'il en existe. La sélection du noeud à partir duquel le branchement est effectué incorpore une part d'aléa et ne repose pas sur la valeur de la borne inférieure, mais sur d'autres paramètres liés aux marges des activités. Des heuristiques à base de règles de priorité permettent d'obtenir une borne supérieure initiale et la borne inférieure utilisée est la même que dans la méthode précédente : on calcule la longueur d'un chemin critique, les contraintes de ressources étant relaxées. Les auteurs justifient ce choix par la rapidité des calculs engendrés. Trois règles de dominance sont également intégrées dans la PSE :

- Règle 1 : cette règle élimine les ordonnancements dans lesquels les dates de début des activités sont toutes supérieures ou égales à celles d'une autre solution.
- Règle 2 : cette règle tient compte des conflits de ressources qui apparaissent, alors qu'ils ont déjà été résolus dans un noeud prédécesseur.
- Règle 3 : cette règle permet de ne considérer que des ordonnancements calés gauche.

En outre, une phase de pré-traitement permet de réduire la taille de l'arbre de recherche : certaines contraintes de précédence peuvent être ajoutées, des dates de disponibilité sont augmentées grâce à un calcul de borne inférieure, basée sur une réduction à un problème à une machine. Enfin, les auteurs proposent deux variantes de leur procédure initiale :

- un ordonnancement "bi-directionnel" : pour une limite de temps fixée, la moitié du temps est consacrée à résoudre le problème initial, tandis que le reste est passé sur le problème inverse, obtenu par symétrie
- un beam search, ou recherche par faisceau : il s'agit en fait d'une PSE tronquée, pour laquelle, à chaque séparation, seul un certain nombre λ de noeuds fils sont générés. Parmi ceux-ci, seuls les μ meilleurs, en ce qui concerne leurs bornes inférieures, sont conservés.

Dorndorf et al [Dorndorf et al.,00] adoptent une démarche très différente dans leur Procédure par Séparation et Evaluation : les contraintes de time lags et de ressources sont prises en compte simultanément, au sein d'un algorithme de propagation de contraintes. Chaque noeud de l'arborescence correspond à un ensemble de domaines pour les dates de début des activités. L'idée est d'appliquer les techniques de propagation de contraintes afin de réduire ces ensembles. Lorsque le domaine associé à une activité est réduit à un singleton, cette activité est considérée comme étant placée. L'objectif est de parvenir à un noeud pour lequel toutes les activités sont placées (solution réalisable). Dès qu'un domaine est vide, le noeud est éliminé. Le processus de séparation procède de la manière suivante : une activité est sélectionnée, telle que cette activité vérifie certaines conditions, et deux noeuds fils sont créés. Dans le premier, l'activité choisie est placée au plus tôt, tandis que dans le second, sa date de début au plus tôt est augmentée.

Afin de se restreindre aux ordonnancements actifs qui sont dominants, les critères de sélection de l'activité sont judicieusement déterminés, de même que la valeur dont on augmente la date de début au plus tôt dans le deuxième fils. Les auteurs montrent que ce schéma de séparation permet de générer implicitement tous les ordonnancements actifs, prouvant ainsi qu'il est correct. La recherche est menée en profondeur d'abord, afin de parvenir rapidement à une solution réalisable. De plus, des règles de dominance sont également ajoutées de manière à éliminer tous les noeuds conduisant à des ordonnancements non actifs. En ce qui concerne la propagation de contraintes, la procédure applique de façon itérative un certain nombre de tests de consistance qui éliminent des valeurs au sein des domaines de dates de début des activités. Ces tests concernent

les contraintes de time lags entre tout couple d'activités, et les contraintes de ressources sur certains intervalles (approche similaire à un raisonnement énergétique [Lopez,91]). Enfin, de la même manière que pour la méthode précédente, un ordonnancement bi-directionnel est proposé. Les auteurs insistent sur une caractéristique importante de leur algorithme : celui-ci n'a en effet pas recours explicitement à une borne inférieure. Il semblerait que les résultats expérimentaux obtenus en ajoutant une telle borne à la procédure n'indiquent qu'un gain très faible. Bien entendu, la propagation des contraintes au cours de la recherche peut conduire implicitement au calcul de bornes inférieures permettant d'éliminer certaines valeurs pour les dates de début des activités.

3.6.3 Extensions du RCPSP avec time lags

a) Contraintes additionnelles de calendriers

Plusieurs extensions du RCPSP avec time lags ont aussi fait l'objet d'études. Franck et al. [Franck et al.,01b] considèrent la contrainte supplémentaire de calendriers, c'est-à-dire d'indisponibilités des ressources. Ces contraintes, correspondant dans la pratique aux situations de week-ends ou de congés, varient selon les ressources et peuvent avoir un impact sur les time lags : selon les cas, les périodes d'indisponibilité peuvent ou non être prises en compte dans le calcul de l'intervalle de temps séparant deux activités. Les auteurs proposent alors d'introduire, pour chaque time lag, un calendrier, à l'instar de ce qui se fait pour les ressources.

La dernière particularité du modèle concerne l'interruption des activités. Par souci d'uniformité, toutes les activités sont supposées avoir une partie incompressible ϵ_i, devant être effectuée avant l'apparition d'une période d'indisponibilité. Le cas $\epsilon_i = 0$ correspond aux activités qui peuvent être interrompues à n'importe quel moment et reprendre normalement après la période d'indisponiblité, tandis que le cas $\epsilon_i = p_i$ est associé aux activités ininterruptibles. Il convient de noter que ce modèle est différent d'un modèle avec activités préemptives, puisqu'ici, les interruptions d'activités ne peuvent survenir qu'aux instants de début des périodes d'indisponibilité.

Les auteurs s'intéressent dans un premier temps à la situation où la capacité des ressources n'est pas limitée. Outre les contraintes de calendriers, seules les contraintes liées aux time lags sont donc prises en compte. Un algorithme modifié de plus long chemin dans le graphe associé aux time lags est proposé. Celui-ci permet d'obtenir les ordonnancements au plus tôt et au plus tard en temps polynomial, en décalant progressivement l'exécution des activités. Dans le cas où des contraintes de ressources sont considérées, une méthode à base de règles de priorité est adaptée de manière à prendre en compte la présence de calendriers. Celle-ci fonctionne sur le même principe que dans Franck et al. [Franck et al.,01a], avec ré-ordonnancement lorsque les contraintes de time lags ne sont pas satisfaites. Parmi toutes les règles de priorité testées, la plus performante lors des expériences est celle de la plus petite date de début au plus tard.

Schwindt et Trautmann [Schwindt et Trautmann,00] étudient un problème d'ordonnancement par lots dans le domaine de l'industrie des procédés. De nombreuses contraintes sont considérées telles que des contraintes temporelles (contraintes de précédence, time lags minimaux et maximaux, durées variables entre deux bornes), des contraintes de ressources (capacités limitées, temps de réglage dépendant de la séquence, stockages limités) et des contraintes de calendriers. Les auteurs insistent essentiellement sur le modèle, issu du RCPSP avec time lags.

L'approche de résolution, exposée brièvement, procède de manière itérative. L'aspect temporel, obtenu par relaxation des autres contraintes, est résolu par un algorithme de plus long chemin du même type que celui de Franck et al. [Franck et al.,01b]. Afin de résoudre les conflits qui apparaissent lorsque l'on prend de nouveau en compte les autres contraintes, des relations de précédence dites "disjonctives" sont ajoutées entre des ensembles d'activités judicieusement sélectionnés. Le processus est répété jusqu'à ce que le problème temporel n'ait plus de solution ou que l'ordonnancement obtenu respecte toutes les contraintes.

b) Activités pouvant être exécutées selon plusieurs modes

Une extension classique du RCPSP consiste à considérer pour l'exécution de chaque activité non pas un mais plusieurs modes possibles (ensembles de contraintes temporelles et de ressources pour la réalisation d'une activité). Le problème est alors noté MRCPSP (*Multi-mode Resource Constrained Project Scheduling Problem*). Dans ce contexte, trouver une affectation de modes faisable est déjà un problème NP-difficile [Kolisch,95]. Il est possible d'ajouter des contraintes de time lags minimaux et maximaux, ce qui conduit au problème du MRCPSP avec time lags. Toutes les caractéristiques des activités dépendent du mode choisi pour leur réalisation : la durée opératoire, les ressources utilisées et les demandes associées. Les time lags, quant à eux, dépendent des modes des deux activités concernées.

Heilmann [Heilmann,03] décrit une Procédure par Séparation et Evaluation intégrée, c'est-à-dire qui détermine en parallèle l'affectation des modes aux activités et les dates d'exécution de celles-ci. Cette méthode se distingue donc des approches par décomposition, telles que la recherche tabou de De Reyck [De Reyck,98], qui effectuent dans un premier temps l'affectation des modes, ce qui conduit ensuite à résoudre un RCPSP avec time lags. Une relaxation du problème consiste à affecter à chaque activité un mode fictif, équivalent à un ensemble de modes : la durée opératoire, les demandes en ressources et les time lags sont obtenus en conservant les minimums de chacun d'eux.

Les noeuds de l'arbre de recherche sont associés au graphe correspondant aux modes fictifs ainsi définis, auquel sont ajoutées des relations de précédence disjonctives. A chaque itération, l'algorithme détermine une décision (affectation d'un mode à une activité ou résolution d'un conflit sur une ressource) et effectue le branchement selon son type (modes possibles ou alternatives retardant minimales), ce qui conduit à l'ajout de précédences disjonctives. La décision sélectionnée est la plus difficile, c'est-à-dire la plus discriminante, afin de tenter de réduire l'espace de recherche le plus tôt possible. Un indicateur, basé sur le calcul de bornes inférieures pour chaque choix résultant d'une décision, permet de mesurer cette notion de "difficulté". De manière classique, les bornes inférieures sont obtenues en relaxant les contraintes de ressources et plusieures règles d'élimination sont présentées (suppression des décisions inefficaces, sélection immédiate). Les résultats issus d'expériences numériques indiquent que cette méthode intégrée est nettement plus performante que les quelques approches par décomposition décrites dans la littérature.

Pour le même problème, Brucker et Knust [Brucker et Knust,03] proposent une borne inférieure destructive, qui consiste à déterminer par dichotomie la plus grande valeur du makespan pour laquelle on peut prouver qu'il n'existe pas d'ordonnancement réalisable correspondant. Dans une phase de pré-traitement, un certain nombre de modes irréalisables ou inefficaces, ainsi que des ressources redondantes sont supprimés. Puis un algorithme de propagation de contraintes

est exécuté. De manière similaire à Heilmann [Heilmann,03], on se ramène à un RCPSP avec time lags par l'intermédiaire des modes fictifs. La propagation de contraintes permet de réduire les fenêtres temporelles pour l'exécution de chaque activité. Si l'algorithme n'est pas en mesure de déterminer l'absence de solution réalisable, une borne inférieure basée sur un programme linéaire est employée : la préemption est autorisée, les time lags sont relaxés et on suppose qu'une activité peut être réalisée par différents modes. Par contre, le programme tient compte des fenêtres temporelles obtenues par propagation de contraintes. Bien que la borne finale améliore les bornes proposées jusque là, et qu'elle se révèle être de bonne qualité pour le MRCPSP classique, les auteurs considèrent que l'écart moyen par rapport à la meilleure solution connue reste assez élevé en présence de time lags.

3.6.4 Bilan concernant les problèmes d'ordonnancement de projet

Le nombre de publications portant sur les problèmes d'ordonnancement de projet en présence de time lags est beaucoup plus élevé que pour les problèmes d'ordonnancement d'atelier. Le concept d'écart temporel est nettement plus familier et il est important de noter que les modèles considérés incluent à la fois des time lags minimaux et des time lags maximaux. Une explication possible provient des applications considérées, pour lesquelles il est plus fréquent et surtout plus naturel de considérer des contraintes temporelles fortes.

Au niveau des approches de résolution, il peut paraître surprenant de constater que de nombreux travaux privilégient des méthodes de résolution exacte, de type PSE, pour ces problèmes NP-difficiles. En réalité, ces méthodes générales d'optimisation combinatoire permettent d'incorporer des techniques variées pour orienter et faciliter la recherche. D'autre part, elles servent assez souvent de base à des heuristiques pour lesquelles on peut envisager de résoudre des problèmes de taille plus importante.

3.7 Problèmes avec une contrainte sans attente

Comme nous l'avons déjà mentionné, les problèmes où les opérations de chaque travaux doivent se succéder sans interruption entre les machines (contrainte *no-wait*), sont des cas particuliers de problèmes avec time lags maximaux, pour lesquels chaque time lag est nul. Il existe une littérature très abondante sur le sujet, notamment en ordonnancement d'atelier, où ce type de contraintes se rencontre assez fréquemment. Nous avons délibérément choisi de ne pas détailler l'ensemble des travaux sur les problèmes d'ordonnancement sans attente, pour deux raisons principales :

1- D'une part, de nombreux articles s'intéressent à ce type de problèmes. L'étude bibliographique menée au début des années 90 par Hall et Sriskandarajah [Hall et Sriskandarajah,96] compte environ 130 références (même si une petite partie d'entre elles ne concernent pas directement le cas sans attente), et une recherche sur la base INSPEC à partir des mots clés *scheduling* et *no wait* aboutit à plus de 120 articles, alors qu'il n'y en a qu'une quarantaine pour les mots clés *scheduling* et *time lags*. Il nous a donc semblé trop fastidieux de chercher à étudier toutes ces publications, relatives à un cas très particulier de notre problème.

2- D'autre part, un certain nombre de problèmes sans attente, notamment pour des ateliers de type flowshop, peuvent se ramener à des problèmes de voyageur de commerce [Wismer,72], pour lesquels de très nombreuses méthodes de résolution ont été proposées. Nous renvoyons le lecteur à l'ouvrage de Lawler et al. [Lawler et al.,85] pour plus de précisions sur ce type de problèmes classiques en optimisation combinatoire.

Nous nous contentons donc de rappeler quelques points fondamentaux tirés de l'étude de Hall et Sriskandarajah [Hall et Sriskandarajah,96], et de donner deux exemples de travaux récents traitant de problèmes d'ordonnancement sans attente. Hall et Sriskandarajah considèrent à la fois les problèmes sans attente et les problèmes avec blocage. Dans le cas sans attente, il est impossible d'exécuter un travail j sur une machine k si, à fin de ce traitement ($C_{j,k}$), la machine suivante l n'est pas disponible (en particulier si celle-ci est occupée par un autre travail i). On est alors obligé de retarder le traitement du travail j. Dans le cas avec blocage, qui peut être considéré comme une relaxation du cas sans attente, le travail j peut être exécuté sur la machine k même si la machine suivante l n'est pas disponible à l'instant où ce traitement s'achève. Le travail j continue alors d'occuper la machine k, l'empêchant d'exécuter une autre opération, jusqu'à ce que la machine l se libère pour traiter j.

Parmi les applications possibles des problèmes sans attente, on peut distinguer trois catégories :

- les situations où des contraintes technologiques imposent un traitement continu des travaux, comme par exemple en métallurgie, en plasturgie, en chimie, ou dans les ateliers automatisés. On retrouve ainsi les mêmes domaines d'applications que pour les problèmes avec time lags maximaux, qui sont une généralisation des problèmes sans attente
- les cas pour lesquels les coûts liés à l'attente des produits sont très élevés, voire rédhibitoires, ce qui conduit à exécuter les opérations d'un même travail sans la moindre interruption. Ces coûts peuvent par exemple être relatifs aux en-cours, et s'exprimer en fonction de la durée de séjour réelle de chaque travail dans l'atelier (date de fin sur la dernière machine moins date de début sur la première). On pourrait d'ailleurs aussi avancer une telle explication pour justifier l'étude des problèmes avec time lags maximaux, à la différence que, dans ce cas, on autorise une certaine attente, limitée. En outre, la situation sans attente peut également être considérée comme un cas particulier de problème d'ordonnancement où les durées opératoires sur chaque machine dépendent de l'attente en amont de cette machine. On est alors dans le cas où la durée croît extrêmement rapidement avec l'attente, ce qui pousse à exécuter les travaux de manière continue sur toutes les machines
- les problèmes avec absence d'espace de stockage, même si ceux-ci correspondent plutôt à la situation de blocage

Hall et Sriskandarajah [Hall et Sriskandarajah,96] fournissent de nombreux résultats de complexité, pour les problèmes sans attente ou avec blocage, mais également pour des situations proches (espace de stockage limité entre les machines, durées opératoires dépendant de l'attente), ou impliquant des contraintes additionnelles (ressources supplémentaires limitées, vitesse des machines contrôlable). Ils considèrent également des contextes de production cyclique. Le rôle de l'algorithme de Gilmore et Gomory [Gilmore et Gomory,64] pour les problèmes de flowshop est mis en évidence. Hall et Sriskandarajah examinent aussi les ateliers de flowshop hybride, de jobshop et d'openshop, ainsi que les travaux menés dans le cadre de l'ordonnancement stochastique.

Parmi les publications récentes portant sur les problèmes sans attente, on retrouve des résultats théoriques, par exemple d'approximabilité [Sviridenko,03], des études de problèmes avec prise en compte de contraintes additionnelles, comme la présence de temps de réglage et de démontage (voir la section 4.6 consacrée à de tels problèmes) et des propositions de méthodes de résolution. Nous présentons maintenant brièvement deux de ces méthodes développées pour des problèmes de flowshop.

Bertolissi [Bertolissi,00] s'intéresse à la minimisation de la somme des dates de fin des travaux sur un flowshop à m machines ($Fm|no-wait|\sum C_j$). Du fait de la contrainte sans attente, la position des opérations d'un même travail les unes par rapport aux autres est fixe. Pour chaque paire de travaux i et j, l'auteur compare la contribution à la fonction objectif des situations où i et j sont exécutés l'un après l'autre (i avant j et j avant i). Il établit alors, à l'aide d'un indicateur, une liste des travaux, et procède de manière analogue à NEH [Nawaz et al.,83], en construisant la solution par insertions successives des travaux, selon l'ordre de la liste, dans une séquence partielle. Les résultats expérimentaux indiquent que cette heuristique est plus performante que d'autres méthodes approchées par construction, surtout quand le nombre de machines m devient grand. Les temps d'exécution, bien qu'un peu plus élevés, restent raisonnables.

Shyu et al. [Shyu et al.,04] proposent une approche de résolution par colonie de fourmis pour le problème $F2|no-wait, NSDST|\sum C_j$. En l'absence de temps de réglage, le problème est équivalent à un problème de voyageur de commerce cumulatif, où l'on cherche à minimiser la somme des distances de la ville sourceà toutes les autres villes, et est NP-difficile au sens fort. Les auteurs montrent comment se ramener à un tel problème en présence de temps de réglage. Ils parviennent ainsi à un problème de graphe, pour lequel ils peuvent utiliser un algorithme de colonie de fourmis. Le taux de phéromone déposé initialement sur chaque arc est déterminé par une heuristique gloutonne. Puis à chaque itération, les fourmis parcourent indépendemment le graphe de manière à construire des solutions, en suivant des probabilités de transition d'un noeud vers un autre. Ces probabilités intègrent à la fois des préférences locales (représentées par le concept de visibilité) et des préférences globales, liées à l'intensité de phéromone sur les arcs. L'algorithme propose ainsi un compromis entre intensification et diversification. A l'issue d'une itération, le taux de phéromone sur chaque arc est mis à jour, en fonction de la qualité des solutions comportant cet arc et d'un facteur d'évaporation. Cette approche est hybridée à une recherche locale, afin d'améliorer les solutions obtenues à la fin de chaque cycle. D'après les expériences réalisées, cette méthode de résolution se révèle très efficace.

3.8 Conclusion de ce chapitre

Ce chapitre consiste en une étude bibliographique approfondie, concernant les problèmes d'ordonnancement en présence de time lags, ce qui, à notre connaissance, n'a jamais été entrepris auparavant dans la littérature. L'absence d'une expression consacrée pour désigner ce type de contraintes, que nous avons déjà évoquée au chapitre 1, n'a pas facilité les recherches que nous avons menées. La synthèse que nous proposons regroupe les travaux publiés selon la structure de problème qu'ils considèrent. Nous considérons tout particulièrement les problèmes d'ordonnancement d'atelier, pour lesquels nous avons essayé d'être exhaustif. Les études les plus significatives en ordonnancement de projet avec time lags sont également présentées.

A l'issue de ce chapitre, nous pouvons faire les constats suivants :

- Les problèmes d'atelier à une machine avec time lags font l'objet de nombreuses études, notamment sur le plan théorique. Si le problème général de la minimisation du makespan est NP-difficile au sens fort, il existe de nombreux cas particuliers polynomiaux, et certains problèmes restent ouverts en termes de complexité. Par ailleurs, les réductions proposées par Brucker et al. [Brucker et al.,99b] permettent de transformer la plupart des problèmes d'ordonnancement d'atelier avec ou sans time lags en problèmes à une machine avec time

lags, ce qui peut justifier qu'on cherche à développer des méthodes de résolution efficaces pour ces problèmes.
- Les autres problèmes d'atelier en présence de time lags sont étudiés de façon marginale et surtout isolée, même si une attention croissante leur est portée. De même que pour le cas à une machine, on trouve essentiellement des résultats de complexité, pour des problèmes avec time lags minimaux uniquement.
- Au contraire, dans le domaine de l'ordonnancement de projet, les contraintes de time lags sont assez couramment prises en compte et de nombreuses méthodes de résolution ont été proposées, qu'il s'agisse de simples heuristiques à base de règles de priorité, de procédures par séparation et évaluation ou de méta-heuristiques de type algorithme génétique ou recherche tabou. D'autre part, les contraintes de time lags minimaux et maximaux sont toujours considérées simultanément.
- Le graphe disjonctif joue un rôle primordial dans la grande majorité des approches de résolution proposées. Il permet en effet de modéliser aisément les contraintes d'écarts temporels. Les techniques employées sont alors issues de la théorie des graphes.
- En ce qui concerne les fonctions objectifs, la plupart des travaux traitent du makespan, ou du plus grand retard. La somme, éventuellement pondérée, des dates de fin, n'est étudiée que du point de vue de la complexité.

Si l'on met en perspective toutes ces remarques avec les conclusions faites à la fin du premier chapitre, quant à l'importance de la prise en compte des time lags, il semble normal, pour ne pas dire nécessaire, d'étudier les problèmes d'ordonnancement d'atelier incluant de telles contraintes. Bien que le flowshop de permutation avec time lags, qui fait l'objet de cette thèse, puisse être ramené à un problème à une machine avec time lags, ou à un problème d'ordonnancement de projet particulier, les méthodes déjà proposées ne semblent pas les plus adaptées. En effet, dans les deux cas, les expériences menées indiquent que ces approches ne sont pas aussi efficaces que des méthodes spécifiques, développées pour des structures d'atelier particulières. Il paraît donc judicieux de chercher à prendre en compte au mieux la structure du problème lors de la conception d'algorithmes de résolution. C'est ce que nous réalisons, pour des problèmes de flowshop de permutation, dans le chapitre qui suit.

Chapitre 4

Approches de résolution développées pour les problèmes de flowshop avec time lags

Résumé

Nous décrivons dans ce chapitre les méthodes que nous avons développées pour la résolution de problèmes de type flowshop en présence de time lags minimaux et maximaux. Nous nous concentrons essentiellement sur le cas du flowshop de permutation, pour lequel l'ordre de traitement des travaux est le même pour toutes les machines. Nous introduisons le problème restreint, pour lequel cet ordre est fixé et donné en entrée. Le résultat fondamental associé à ce problème nous conduit à développer une classe de Procédures par Séparation et Evaluation utilisant un schéma de branchement classique pour le flowshop de permutation. Nous présentons les éléments communs de ces méthodes, en particulier les stratégies d'exploration, ainsi que des heuristiques génériques valables pour des problèmes où les solutions sont représentées par des permutations. Nous insistons également sur les considérations qui nous ont poussé à générer des jeux d'essais particuliers. Puis nous détaillons, selon les problèmes considérés, les spécificités de chacune d'elles, qu'il s'agisse des bornes inférieures utilisées ou des règles de dominance éventuelles. Parmi les problèmes examinés, on retrouve la minimisation du makespan pour le flowshop de permutation à m machines avec time lags minimaux et maximaux, qui constitue le problème central de notre étude, et pour lequel nous avons effectué une série d'expériences approfondies. Nous nous intéressons aussi à la minimisation de la somme pondérée des dates de fin sur les machines, pour le même type de problème, et à la minimisation du plus grand retard, dans un premier temps pour un flowshop à deux machines avec traitement sans attente et temps de montage et démontage, et dans un second temps pour un problème plus général de flowshop de permutation à m machines avec time lags exacts. Pour chacun de ces problèmes, nous décrivons et interprétons les résultats numériques issus d'expériences. Enfin, nous analysons des extensions possibles de ces approches de résolution. Nous montrons comment les utiliser pour prendre en compte des contraintes additionnelles de dates de disponibilité et de durées de latence pour les travaux. L'intégration de time lags généralisés, c'est-à-dire définis entre des couples d'opérations quelconques au sein des travaux, est aussi étudiée. Enfin, nous proposons une réflexion sur l'utilisation possible d'un schéma de séparation similaire, pour des problèmes de flowshop général, où l'on ne se limite pas aux solutions de permutation. Nous montrons en particulier que le problème restreint, défini à partir de la séquence de traitement des travaux

sur la première machine uniquement, devient NP-difficile au sens fort, sauf dans le cas à deux machines avec time lags minimaux uniquement. Pour ce cas particulier, nous proposons plusieurs bornes inférieures envisageables.

Sommaire

4.1 Introduction

La présence de contraintes de type time lags rend la plupart des problèmes NP-difficiles au sens fort, comme nous avons pu le constater dans le chapitre 2. Face à de tels problèmes, plusieurs approches de résolution sont alors envisageables : d'une part, on peut chercher à résoudre le problème de manière optimale, à l'aide d'un algorithme dont la complexité sera exponentielle, d'autre part, on peut tenter de mettre au point des heuristiques plus ou moins sophistiquées de manière à obtenir, en un temps de calcul raisonnable, une solution de bonne qualité. Ces deux types d'approches sont complémentaires et il ne s'agit pas de chercher à les opposer complétement. Il est évident qu'une méthode exacte fournira toujours une solution au moins aussi bonne qu'une heuristique mais en contrepartie, l'obtention de la solution optimale dans le premier cas nécessitera le plus souvent beaucoup plus de temps de calcul. La taille des problèmes considérés intervient également dans le choix de l'approche de résolution. La question est alors de savoir de combien de temps on dispose pour calculer une solution et si cette durée est suffisante pour permettre une résolution exacte. Etant donné la taille des problèmes rencontrés dans le monde industriel, des heuristiques semblent plus adaptées à ce type de problématique et il serait facile de penser que les méthodes exactes n'ont qu'un intérêt "théorique". Cependant, l'évaluation de performance d'une heuristique, en termes de qualité des solutions fournies, est loin d'être évidente. Pour une instance donnée, il existe plusieurs manières de réaliser cette estimation, mais dans tous les cas, cela nécessite une comparaison entre la valeur obtenue par l'heuristique et une valeur de référence (en calculant, par exemple, l'écart relatif).

1- Pour certaines heuristiques, il est possible de calculer théoriquement un majorant, voire la borne supérieure, de l'écart relatif entre la valeur obtenue et l'optimum. On parle alors de ratio de performance dans le pire cas. Néanmoins, cette propriété se limite la plupart du temps à des heuristiques simples (par exemple, des algorithmes de liste avec règles de priorité) et n'est pas valable pour des méthodes plus sophistiquées, telles que les métaheuristiques. En outre, le pire cas théorique correspond souvent à une instance très particulière pour laquelle l'heuristique est particulièrement mauvaise, et qui ne reflète pas une situation réaliste.

2- Si l'on dispose de plusieurs heuristiques, on peut comparer ces méthodes entre elles ou calculer pour chacune un écart relatif par rapport à la meilleure valeur trouvée. Le principal inconvénient provient du fait que l'on n'a dans ce cas aucune idée des performances de chaque méthode dans l'absolu, c'est-à-dire par rapport à l'optimum réel, et l'on peut très bien se retrouver avec des heuristiques classées précisément les unes par rapport aux autres mais qui sont toutes de mauvaise qualité !

3- La comparaison peut également se faire vis-à-vis d'une borne inférieure de la valeur optimale. Mais pour avoir une idée précise de la qualité de la borne inférieure, on retrouve le même genre de problèmes que précédemment.

4- La dernière possibilité consiste à comparer, sur des instances de taille raisonnable, la valeur calculée par l'heuristique avec le véritable optimum, obtenu à l'aide d'une méthode exacte. Il n'est dans ce cas pas nécessaire de déterminer des ratios d'approximation théoriques.

Ainsi, les méthodes de résolution exacte, pour des problèmes NP-difficiles, peuvent avoir un intérêt pratique pour la mesure de la qualité des heuristiques qui seront ensuite utilisées sur des

instances réelles de grande taille. D'autre part, il est également possible de faire appel à des méthodes exactes au sein d'heuristiques, pour résoudre des sous-problèmes de petite taille. Les méthodes par décomposition s'y prêtent particulièrement bien.

La manière la plus immédiate de concevoir une méthode de résolution exacte est d'énumérer l'ensemble des solutions et de garder celle qui conduit à la meilleure valeur. Afin d'être plus efficace, il est nécessaire de définir des règles permettant d'éviter de considérer des solutions inintéressantes (solutions dominées), c'est-à-dire dont on sait qu'elles ne peuvent pas être optimales, ou de se limiter à des sous-ensembles de solutions dont on a la garantie qu'ils contiennent au moins une solution optimale (sous-ensembles dominants). Les Procédures par Séparation et Evaluation (PSE) forment une classe générale de méthodes exactes fonctionnant sur ce principe. Nous avons choisi de développer de telles méthodes pour résoudre les problèmes de flowshop avec time lags. La section suivante présente la définition générale des PSE, ainsi que le schéma que nous avons adopté. Il est à noter que nos travaux ne tiennent pas compte de l'article publié très récemment par Ladhari et Haouari [Ladhari et Haouari,05], qui propose une PSE pour le problème de flowshop de permutation classique ($Fm_{\pi}||C_{max}$), utilisant une nouvelle borne inférieure basée sur le problème $F2_{\pi}|r_j, \theta_j^{min}, q_j|C_{max}$. Bien que ce dernier problème soit NP-difficile, la résolution exacte de celui-ci permet d'aboutir à une PSE dont les résultats semblent prometteurs.

Nous précisons qu'une version antérieure de la PSE de la section 4.4, pour des problèmes avec time lags maximaux uniquement, a été présentée dans [Fondrevelle,02b], [Fondrevelle,02a] et [Fondrevelle,03a]. La version actuelle est décrite également dans [Fondrevelle et al.]. Le problèmes des sections 4.5, 4.6 et 4.7 sont abordés respectivement dans [Fondrevelle et al.,05b], [Fondrevelle et al.,04b] et [Fondrevelle et al.,05d]. Enfin, les extensions proposées en fin de chapitre ont fait l'objet d'une présentation dans [Fondrevelle et al.,05a] et [Fondrevelle et al.,05c].

4.2 Présentation générale des Procédures par Séparation et Evaluation

Dans cette section, nous supposons que l'on cherche à minimiser une fonction objectif. Dans le cas contraire (critère à maximiser), il suffit de considérer l'opposé de la fonction. Les Procédures par Séparation et Evaluation sont des méthodes générales de résolution de problèmes combinatoires. Elles consistent à énumérer implicitement l'ensemble des solutions, en procédant par décomposition. Puisque toutes les solutions sont (implicitement) considérées, le résultat fourni par une PSE est nécessairement optimal et cette méthode est donc exacte. Cependant, contrairement à un algorithme d'énumération explicite, qui se contenterait d'examiner toutes les solutions envisageables, les unes après les autres, et qui conserverait la meilleure, une PSE cherche à éliminer les régions de l'espace des solutions qui ne contiennent pas de solutions intéressantes, c'est-à-dire de solutions potentiellement optimales. Elle procède pour cela par décomposition : l'ensemble global des solutions est décomposé successivement en sous-ensembles stricts qui le recouvrent ou le partitionnent. Si on connaît la meilleure solution dans chaque sous-ensemble, il suffit de garder, parmi toutes ces solutions, celle qui a la plus petite valeur, pour obtenir la solution optimale du problème. On procède ainsi, de manière récursive, jusqu'à parvenir

1- soit à un ensemble dont on peut déterminer facilement une meilleure solution,

2- soit à un ensemble de solutions dominées, que l'on élimine.

Si l'ensemble initial considéré est fini, puisque le cardinal de chaque sous-ensemble généré lors d'une décomposition décroît strictement, la procédure s'arrête au bout d'un nombre fini d'ap-

pels récursifs. La manière d'effectuer la décomposition, à chaque étape, constitue le principe de séparation (ou de branchement), qui est un élément fondamental d'une PSE, et qui conditionne à ce titre sa performance. Généralement, la séparation conduit à une partition de l'ensemble de départ (les sous-ensembles sont disjoints deux à deux et leur union redonne l'ensemble de départ), mais dans certains cas, il peut être plus simple, au niveau de l'implémentation, de recourir à un recouvrement de l'ensemble par des sous-ensembles non disjoints.

Comme nous l'avons mentionné, la séparation s'arrête dans deux cas. Si on parvient à un ensemble dont on peut déterminer facilement une meilleure solution, il n'est plus nécessaire de poursuivre la décomposition. En général, ce cas correspond aux situations où l'ensemble est réduit à un singleton ; la meilleure solution est alors évidemment l'unique élément de l'ensemble. Mais il est possible, dans certaines situations, d'arriver à un sous-ensemble pour lequel le problème devient polynomial. On peut alors déterminer la meilleure solution au sein du sous-ensemble et terminer ainsi l'exploration de celui-ci, sans poursuivre la décomposition. L'autre cas est celui des ensembles que l'on élimine, sachant qu'ils ne conduisent pas à une solution optimale. Pour déterminer si un ensemble vérifie cette condition, on lui applique un ou plusieurs tests de dominance. Les PSE reposent sur une règle principale de dominance que nous présentons maintenant :

Règle générale de dominance : Chaque sous-ensemble considéré est évalué par le calcul d'une borne inférieure. La valeur obtenue est en fait un minorant de la valeur d'une meilleure solution appartenant à l'ensemble. Si l'on dispose par ailleurs d'une borne supérieure pour la solution optimale du problème global, il est possible de comparer les deux valeurs. Dans le cas où la borne inférieure associée à l'ensemble est strictement supérieure à la borne supérieure de l'optimum global, on en déduit que l'ensemble considéré ne contient pas de solution optimale et peut donc être éliminé. L'élimination d'un ensemble conduit directement à l'élimination de tous ses sous-ensembles, qu'ils aient déjà été générés ou pas.

Notons que si la borne supérieure correspond à la valeur d'une solution réalisable dont on dispose déjà, et si l'on recherche une seule solution optimale au problème, il est aussi possible d'éliminer l'ensemble considéré en cas d'égalité des deux valeurs. Plus la borne inférieure est précise, c'est-à-dire plus elle est proche de la valeur réelle de la meilleure solution contenue dans l'ensemble, plus elle est susceptible d'éliminer de nombreux ensembles. Il en va de même pour la borne supérieure. Celle-ci correspond généralement à la valeur de la meilleure solution obtenue jusque là, qu'elle provienne d'un ensemble exploré ou qu'elle ait été fournie par une heuristique, avant ou pendant l'exécution de la PSE.

Il est possible d'incorporer dans la procédure d'autres tests de dominance, qui reposent cette fois sur des propriétés particulières du problème étudié. Par exemple, si l'on est capable de démontrer que les solutions ayant une structure particulière sont dominantes, on peut se restreindre aux solutions de ce type et ainsi éliminer de la recherche les sous-ensembles qui ne contiennent aucune solution ayant cette structure.

La dernière caractéristique des PSE concerne l'ordre dans lequel on sépare les sous-ensembles, lorsque plusieurs sont candidats. Elle est définie par la stratégie d'exploration. On distingue en général deux types de stratégies. D'un côté, les PSES ou PSE Séquentielles explorent les sous-ensembles dans un ordre établi a priori : en profondeur d'abord, en largeur d'abord. De l'autre côté, on retrouve les PSEP ou PSE Progressives, pour lesquelles on considère à chaque étape le sous-ensemble de plus petite valeur. Cette stratégie repose sur l'hypothèse qu'une borne infé-

rieure de petite valeur a une plus grande probabilité de conduire à une solution optimale.

Une manière commode de représenter la recherche d'une solution lors de l'exécution d'une PSE est de recourir à un arbre de recherche, dans lequel chaque noeud est associé à un sous-ensemble de l'espace des solutions. La racine correspond à l'ensemble global initial et les fils d'un noeud donné représentent les sous-ensembles issus de la séparation. C'est la raison pour laquelle les PSE font aussi partie de la famille des méthodes de recherche arborescentes.

Pour une description plus détaillée des PSE, on peut consulter l'ouvrage de Lawler et Wood [Lawler et Wood,66].

4.3 Schéma générique des Procédures par Séparation et Evaluation proposées

Nous avons vu dans la section précédente qu'une Procédure par Séparation et Evaluation était caractérisée par les éléments suivants :

- le schéma de séparation
- la ou les bornes inférieures
- la ou les bornes supérieures
- les autres règles de dominance éventuelles
- la stratégie d'exploration

Nous nous sommes intéressés à plusieurs problèmes de flowshop de permutation avec time lags minimaux et maximaux, pour lesquels nous avons développé des approches de résolution basées sur des PSE. Nous présentons dans cette section les caractéristiques communes de ces PSE, puis nous détaillons dans la section suivante les particularités de chacune.

4.3.1 Définition et analyse du problème restreint

Quels que soient le nombre de machines et les contraintes additionnelles éventuelles, un problème d'ordonnancement dans un flowshop de permutation peut être formulé de la manière suivante :

Déterminer l'ordre de traitement des travaux sur les machines, et les dates d'exécution associées, de manière à satisfaire toutes les contraintes du problème et à optimiser la fonction objectif.

Considérons maintenant le problème restreint qui consiste, pour une permutation donnée $\pi = (\pi(1), \pi(2), \ldots \pi(n))$ des travaux, à calculer les dates d'exécution de façon à obtenir un ordonnancement qui respecte à la fois la séquence de traitement π sur toutes les machines et les contraintes du problème, et qui optimise la fonction objectif. Nous allons voir qu'il existe un algorithme polynomial exact pour ce problème, pour une grande classe de critères, et que ce résultat nous conduit naturellement à adopter un schéma de séparation spécifique dans nos PSE.

Les méthodes de résolution usuelles pour des problèmes d'ordonnancement génèrent le plus souvent des solutions réalisables appartenant à l'une des trois grandes catégories suivantes [Baker,74] :

- Les ordonnancements sans délai. Ceux-ci sont caractérisés par la propriété qu'aucune ressource n'est laissée inactive alors qu'elle pourrait exécuter une opération disponible.

- Les ordonnancements actifs, pour lesquels aucune opération ne peut être exécutée plus tôt sans en retarder une autre.
- Les ordonnancements semi-actifs, pour lesquels aucune opération ne peut être exécutée plus tôt sur la même machine, sans modifier l'ordre de traitement sur au moins une machine.

On peut aisément vérifier que l'ensemble des ordonnancements semi-actifs contient l'ensemble des ordonnancements actifs, qui contient lui-même l'ensemble des ordonnancements sans délai.

Le fait de se restreindre à l'un de ces types d'ordonnancement conduit normalement à des solutions de bonne qualité. Plus précisément, il est connu que l'ensemble des ordonnancements actifs est dominant quand le critère est régulier [Baker,74]. Nous rappelons qu'un critère $f(C_1, C_2, \ldots, C_n)$, qui s'exprime en fonction des dates de fin des travaux C_j, est dit régulier, s'il est croissant selon chaque composante, les autres restant fixes. C'est le cas des critères classiques cités dans la section 1.2. Le critère de la somme pondérée des dates de fin sur les machines $\sum w_k MC_k$, que nous avons introduit, peut être considéré comme régulier, bien qu'il ne s'exprime pas en fonction des dates de fin des travaux C_j : il suffit en fait de généraliser la définition à des critères qui sont fonctions des dates de fin des opérations $C_{j,k}$, ce qui est le cas de $\sum w_k MC_k = \sum_k w_k max_j\{C_{j,k}\}$. La propriété de dominance des ordonnancements actifs pour tout critère régulier signifie donc qu'une solution optimale peut être recherchée parmi ces ordonnancements. Puisque l'ensemble des ordonnancements semi-actifs contient celui des ordonnancements actifs, il en est de même pour cette catégorie de solutions. Par contre, les ordonnancements sans délai ne sont pas dominants pour les critères réguliers [Baker,74]. Cependant, de nombreuses méthodes de résolution se contentent de générer de tels ordonnancements, pour des raisons de simplicité, en faisant l'hypothèse qu'ils sont de bonne qualité étant donné qu'ils évitent certains temps morts.

Si ces définitions ne posent pas de problème quand on ne considère que des time lags minimaux, l'introduction de time lags maximaux peut conduire à des difficultés, illustrées dans l'exemple simple suivant, proposé par Deppner [Deppner,04].

Exemple. Considérons un flowshop à deux machines et un seul travail j, composé de deux opérations de durées unitaires devant être réalisées sans attente ($\theta_j^{max} = \theta_j^{min} = 0$). Supposons que l'on fixe la date de début de la première opération $t_{j,1}$ à $t > 0$. On a alors $C_{j,1} = t_{j,2} = t + 1$ et $C_{j,2} = t + 2$ (Figure 4.1(a)). Cet ordonnancement n'est pas optimal puisqu'en débutant la première opération à l'instant 0, on obtiendrait une date de fin de 1 sur la première machine et 2 sur la deuxième (Figure 4.1(b)).

Pourtant, d'après la définition donnée précédemment, l'ordonnancement peut d'une certaine manière être considéré comme semi-actif, puisque, en raison du time lag maximal, chaque opération ne peut individuellement être avancée, si l'autre reste fixe.

Pour remédier à ce problème, Sprecher et al. [Sprecher et al.,95] ont introduit des versions généralisées des concepts d'ordonnancements actifs et semi-actifs. Leur définition est similaire aux versions initiales, à ceci près que l'on peut considérer des sous-ensembles d'opérations plutôt que des opérations individuelles.

Définition
Un ordonnancement est semi-actif, au sens généralisé, s'il est réalisable et s'il est impossible de décaler un sous-ensemble d'opérations vers la gauche, tout en conservant les mêmes séquences de traitement sur les différentes machines, de manière à obtenir un ordonnancement réalisable.

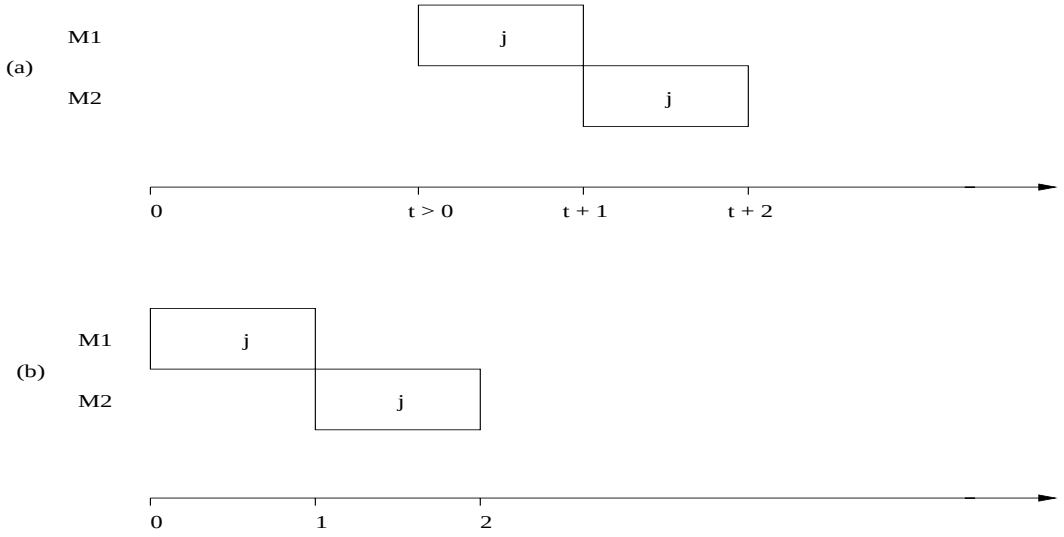

FIG. 4.1 – Limite de la définition classique des ordonnancements semi-actifs

Définition
Un ordonnancement est actif, au sens généralisé, s'il est réalisable et s'il est impossible de décaler un sous-ensemble d'opérations vers la gauche, sans en retarder d'autres, de manière à obtenir un ordonnancement réalisable.

Si le problème implique des time lags maximaux, les termes semi-actif et actif font donc référence aux versions généralisées, afin d'éviter toute ambiguïté. Tout comme dans le cas classique, ces ordonnancements sont dominants quand le critère est régulier.

Revenons au problème restreint. Pour une séquence donnée des travaux, dans un flowshop de permutation, il n'existe qu'un seul ordonnancement semi-actif respectant cette séquence : c'est l'ordonnancement au plus tôt, pour lequel toutes les opérations sont calées à gauche, compte tenu des contraintes. L'algorithme suivant permet de construire cet ordonnancement.

Algorithme. A2.
1 *Placer le premier travail* $\pi(1)$ *au plus tôt*
  1-1 $C_{\pi(1),1} = p_{\pi(1),1}$
  1-2 $k = 1$. Tant que $k < m$ répéter
  – $C_{\pi(1),k+1} = C_{\pi(1),k} + \theta^{min}_{\pi(1),k} + p_{\pi(1),k+1}$
  – $k = k + 1$
2 *Placer les autres travaux au plus tôt, dans l'ordre de la séquence.* $j = 2$. Tant que $j \leq n$ répéter
  2-1 *Placée la première opération au plus tôt sur la première machine.* $C_{\pi(j),1} = C_{\pi(j-1),1} + p_{\pi(j),1}$
  2-2 *Placer les autres opérations successivement au plus tôt, en ne tenant compte que des contraintes de précédence et des time lags minimaux.* $k = 1$. Tant que $k < m$ répéter
  – $C_{\pi(j),k+1} = max\{C_{\pi(j),k} + \theta^{min}_{\pi(j),k}, C_{\pi(j-1),k+1}\} + p_{\pi(j),k+1}$
  – $k = k + 1$
  2-3 *Vérifier si les contraintes de time lags maximaux sont satisfaites, en commençant par*

la dernière machine. Tant que $k > 1$ répéter
- Si $C_{\pi(j),k} - C_{\pi(j),k-1} > \theta^{max}_{\pi(j),k-1} + p_{\pi(j),k}$
- *Retarder l'exécution de la première opération du couple concerné afin de respecter le time lag maximal* $C_{\pi(j),k-1} = C_{\pi(j),k} - p_{\pi(j),k} - \theta^{max}_{\pi(j),k-1}$
- $k = k - 1$

Théorème. 19 *L'algorithme précédent (A2) fournit une solution optimale pour le problème restreint en* $O(n.m)$.

Preuve. L'ordonnancement construit par l'algorithme respecte la séquence π fixée. De plus, on peut aisément vérifier, par récurrence, qu'il est réalisable et que toutes les opérations sont ordonnancées au plus tôt. Aucune opération, ni aucun groupe d'opérations, ne peut débuter son exécution plus tôt sans modifier la séquence de traitement. L'ordonnancement est donc semi-actif, et optimal pour le problème restreint. En ce qui concerne la complexité de l'algorithme, il suffit de remarquer que l'ordonnancement d'un travail (deux boucles sur k, sauf pour le premier travail) est effectué en $O(m)$ opérations élémentaires. Le traitement des n travaux se fait donc en $O(n.m)$ opérations.□

Remarques sur l'algorithme :

1- L'algorithme que nous avons présenté travaille avec les dates de fin des opérations $C_{j,k}$. De manière équivalente, on aurait pu procéder avec les dates de début $t_{j,k}$. Etant donné que le problème est non-préemptif, on peut passer d'un type de variables à l'autre grâce aux relations $\forall j, k\ C_{j,k} = t_{j,k} + p_{j,k}$.

2- L'étape [2-3] de prise en compte des time lags maximaux considère les machines dans l'ordre inverse de l'ordre de parcours par les travaux, afin de minimiser le nombre de décalages. De cette manière, chaque opération (mis à part la dernière) est retardée au plus une seule fois. En procédant différemment, par exemple en considérant les machines dans l'ordre de parcours par les travaux, on pourrait être amené à retarder plusieurs fois une même opération. En effet, retarder l'opération sur la machine $k > 1$ peut conduire à ne plus respecter le time lag maximal entre cette opération et l'opération précédente, obligeant ainsi à retarder également cette dernière. Il est donc judicieux de ne considérer l'opération sur la machine k que lorsque toutes les opérations suivantes sont définitivement placées.

3- L'étape [2-3] n'est pas nécessaire pour le premier travail. En effet, celui-ci n'a aucun prédécesseur et ses opérations sont par conséquent calées exactement par rapport aux time lags minimaux, ce qui garantit que les time lags maximaux sont respectés ($\theta^{max}_{j,k} \geq \theta^{min}_{j,k}$).

4.3.2 Schéma de séparation adopté

Le fait de disposer d'un algorithme polynomial qui résout de manière exacte le problème restreint est crucial : ceci implique que l'on peut passer "facilement" d'une séquence de travaux à l'ordonnancement optimal selon cette séquence. Ainsi, pour déterminer la solution optimale au problème initial, c'est-à-dire l'ordonnancement réalisable qui minimise le critère étudié, il suffit de rechercher la séquence de travaux correspondante. Le problème peut alors être vu comme un problème de recherche d'une permutation optimale de l'ensemble $\{1, 2, \ldots, n\}$, dans lequel la fonction d'évaluation est obtenue en appliquant l'algorithme précédent et en calculant la valeur du critère pour l'ordonnancement obtenu. On est donc ramené à un problème très général, que l'on rencontre souvent en optimisation combinatoire.

On peut se demander si chaque permutation conduit à une solution réalisable, c'est-à-dire à un ordonnancement vérifiant toutes les contraintes. D'après le théorème de décomposition établi par Neumann et Zhan [Neumann et Zhan,95], il suffit de considérer les "structures de cycles", correspondant aux composantes fortement connexes du graphe représentant les contraintes temporelles. Dans notre cas, puisque nous nous limitons à des time lags définis entre opérations successives au sein des travaux, chacun de ces ensembles contient (au plus) toutes les opérations d'un même travail. Si l'on se restreint aux opérations d'un travail, on obtient aisément une solution réalisable en ordonnançant les opérations les unes après les autres, dans l'ordre de la gamme opératoire. Il n'y a en effet aucun conflit de ressource. Ainsi, chaque structure de cycle admet un ordonnancement réalisable et par conséquent le problème général en admet un aussi. Toutes les permutations de l'ensemble $\{1, 2, \ldots, n\}$ conduisent donc à une solution réalisable. L'espace de ces solutions contient donc $n!$ éléments. Il convient de noter que l'application du théorème de décomposition [Neumann et Zhan,95] conduit naturellement à appliquer l'algorithme que nous avons présenté, qui consiste à ordonnancer les travaux, c'est-à-dire les structures de cycles, les uns après les autres. On retrouve la même idée dans les algorithmes de construction par grappes développés par Deppner [Deppner,04].

La représentation des solutions d'un problème sous forme de permutations de l'ensemble $\{1, 2, \ldots, n\}$ permet d'adopter des approches de résolution générales dont les éléments sont basés sur cette représentation. En ce qui nous concerne, nous avons développé des Procédures par Séparation et Evaluation avec un schéma de séparation classique pour ce type de problèmes. A tout noeud de l'arbre de recherche, situé à la profondeur i, est associé une séquence σ de i éléments, qui désignent les travaux placés aux i premières positions dans un ordonnancement partiel. Un tel noeud possède $n - i$ fils, chacun correspondant à une séquence de $i + 1$ éléments composée des éléments de σ suivis d'un travail j non placé. La racine est associée à la séquence vide, tandis que chaque feuille de profondeur n correspond à une permutation complète, c'est-à-dire à une solution du problème. Dans l'arbre complet, le nombre de noeuds à la profondeur $i \geq 1$ est $n \times (n-1) \times \ldots \times (n-i+1) = n!/(n-i)!$ et le nombre total de noeuds est donc $1 + \sum_{1 \leq i \leq n} n!/(n-i)!$.

Un tel schéma a été proposé pour le flowshop de permutation classique par Ignall et Schrage [Ignall et Schrage,65] et a été repris avec succès dans la PSE de Lageweg et al. [Lageweg et al.,78]. Il a également été utilisé dans de nombreuses PSE consacrées à des problèmes de flowshop de permutation en présence de contraintes additionnelles : deux machines et time lags maximaux [Yang et Chern,95], temps de réglage dépendant de la séquence [Rios-Mercado et Bard,99], trois machines et contraintes représentées dans l'algèbre Max-Plus par des matrices triangulaires (généralisant les contraintes de time lags minimaux, de temps de réglage et démontage, dates de disponibilités des travaux et durées de latence) [Bouquard,04].

4.3.3 Stratégie d'exploration

Pour les méthodes arborescentes que nous avons développées, nous avons retenu trois stratégies d'exploration possibles :

- Stratégie MFP (Meilleur Fils en Profondeur) : lors de la séparation d'un noeud, tous les fils sont générés et leur borne inférieure est calculée. Le nouveau noeud à traiter est le premier fils de valeur minimale, à condition, bien entendu, que cette valeur soit inférieure à la borne supérieure courante. Dans le cas contraire, on effectue un retour arrière, jusqu'à parvenir à un noeud dont un frère au moins reste à considérer. Parmi tous les noeuds frères

non traités, on sélectionne le premier à avoir été généré. Cette méthode n'est pas exactement une stratégie en profondeur d'abord, dans la mesure où tous les fils d'un noeud sont générés lors de la même étape. Pourtant, le principe s'en rapproche, puisque l'on atteint rapidement les feuilles de d'arbre. L'idée sous-jacente est d'obtenir assez tôt une solution qu'on espère de bonne qualité, ce qui permettrait d'éliminer de nombreuses branches dans l'arbre de recherche. Simplement, par rapport à une exploration en profondeur d'abord au sens strict, on choisit en premier un noeud susceptible de conduire à une bonne solution, plutôt qu'un noeud quelconque. Par ailleurs, de la même manière que pour la PSE de Bartusch et al. [Bartusch et al.,88], une telle stratégie permet une manipulation plus aisée des données. En effet, l'exploration repose uniquement sur des déplacements d'un noeud vers un de ses fils ou, inversement, d'un noeud vers son père, ce qui correspond à l'ajout ou au retrait d'un travail à la fin de la séquence partielle.

- Stratégie PFP (Partition des Fils en Profondeur) : le principe est le même que pour la stratégie précédente, la seule différence résidant au niveau de l'ordre de traitement des noeuds frères, c'est-à-dire issus d'un même noeud père. Plutôt que de considérer le premier noeud avec une valeur minimale, puis tous les autres dans l'ordre dans lequel ils ont été générés, les noeuds sont "partiellement" triés. Plus précisément, l'ordre dans lequel tous les noeuds générés à partir d'un même père situé à une profondeur i sont traités est donné par une liste $L = (L(1), \ldots, L(nb_{meil}), L(nb_{meil}+1), \ldots, L(nb_{meil}+nb_{classs}), \ldots, L(n-i))$ où les nb_{meil} premiers éléments de la liste sont tous les noeuds pour lesquels la borne inférieure prend une valeur minimale, les nb_{classs} éléments suivants sont les autres noeuds de valeur inférieure à $valeur_{lim}$, classés dans un ordre quelconque, et les $n-i-nb_{meil}-nb_{classs}$ derniers éléments sont les noeuds restants, dont la valeur est supérieure ou égale à $valeur_{lim}$, placés dans un ordre quelconque. Autrement dit, les noeuds sont partitionnés en deux groupes en situant leur borne inférieure par rapport à une certaine valeur $valeur_{lim}$. Celle-ci est calculée à partir des valeurs minimale et maximale prises par la borne inférieure pour les noeuds considérés et d'un paramètre $x \in [0,1]$ fixé par l'utilisateur : $valeur_{lim} = valeur_{min} + x \times (valeur_{max} - valeur_{min})$. $x = 1$ et $x = 0$ correspondent au cas où seuls les noeuds de valeur minimale sont placés en tête de la liste, les autres étant séquencés dans un ordre arbitraire. Cette situation n'est pas exactement la même que celle qu'on obtient avec la stratégie précédente, pour laquelle un seul noeud de valeur minimale est considéré à part. En outre, il convient de noter que le traitement d'un noeud n'implique pas nécessairement sa séparation. Celle-ci n'intervient que si la borne inférieure calculée pour le noeud est inférieure à la borne supérieure courante. En particulier, si ce n'est pas le cas pour un noeud situé dans les $nb_{meil}+nb_{classs}$ premières positions, les noeuds placés au-delà de la position $nb_{meil}+nb_{classs}$ dans la liste sont immédiatement éliminés, compte tenu du tri effectué. La procédure de constitution des groupes a une complexité en $O(n)$, puisqu'il suffit de parcourir une fois l'ensemble des valeurs pour déterminer $valeur_{min}$ et $valeur_{max}$, de calculer $valeur_{lim}$ puis de répartir chaque noeud dans le groupe adéquat (en début de liste, au milieu, à la fin). Un véritable classement des noeuds, dans l'ordre de leur borne inférieure, nécessiterait $O(n.log(n))$ opérations.

- Stratégie CNL (Classement des Noeuds en Largeur) : contrairement aux deux stratégies précédentes, l'exploration se fait en largeur d'abord : les noeuds sont traités niveau par niveau, c'est-à-dire selon la profondeur à laquelle ils sont situés. Cette stratégie est moins adapatée au schéma de séparation retenu et la manipulation des données qui en résulte s'en trouve alourdie. En réalité, nous adoptons cette approche uniquement dans le cadre

d'un beam search. Il s'agit d'une méthode arborescente approchée, qui peut être considérée comme une Procédure par Séparation et Evaluation tronquée. Parmi tous les noeuds générés à la profondeur i, seuls les f_i meilleurs, c'est-à-dire de plus petite valeur, sont conservés. f_i, qui peut dépendre de i, est appelé la largeur du faisceau ("beam" en anglais) et est spécifié par l'utilisateur avant l'exécution de l'algorithme. Seule une exploration en largeur d'abord peut permettre de mettre en place un beam search. Récemment, des algorithmes plus sophistiqués, mais incorporant des idées similaires, ont été proposés, comme le *recovering beam search* [Della Croce et T'kindt,02].

Les figures 4.2, 4.3, 4.4 et 4.5 illustrent les trois stratégies précédentes : on représente dans la figure 4.2 un arbre de recherche, dans lequel à chaque noeud est associée une valeur, qui est utilisée pour déterminer l'ordre de traitement des noeuds (à l'instar de la borne inférieure dans une PSE).

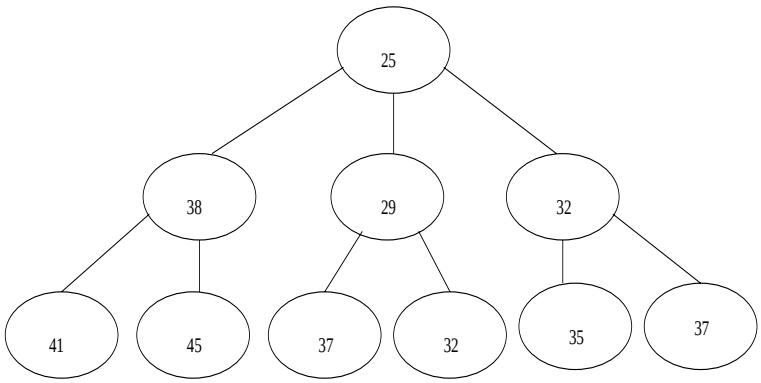

FIG. 4.2 – Arbre de recherche

Dans les figures 4.3, 4.4 et 4.5, les valeurs indiquées entre parenthèses indiquent l'ordre dans lequel les noeuds sont traités, en fonction de la stratégie adoptée : MFP, PFP avec $x = 2/3$ et CNL respectivement. On suppose qu'il n'y a pas de règles d'élimination, c'est-à-dire que tous les noeuds sont examinés. Dans le cas PFP, puisque $x = 2/3$, on a, pour les noeuds issus de la racine, $valeur_{min} = 29$, $valeur_{max} = 38$ et donc $valeur_{lim} = 35$.

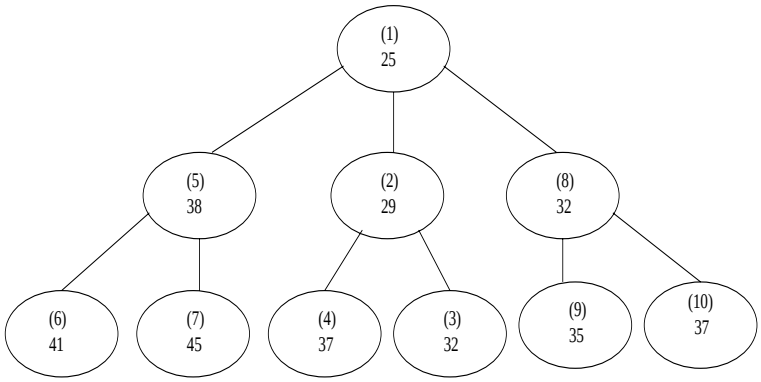

FIG. 4.3 – Ordre d'exploration des noeuds dans le cas MFP

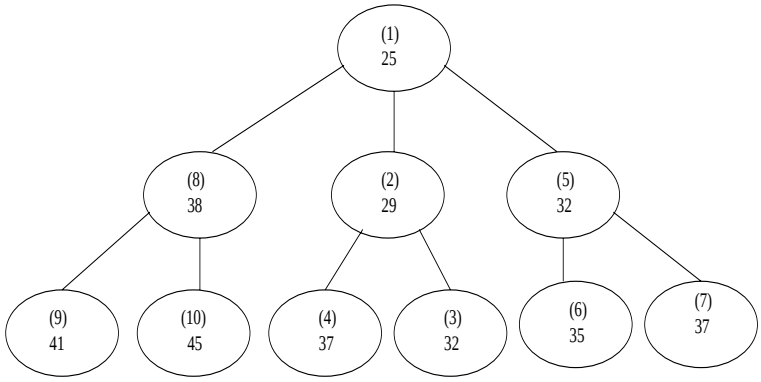

FIG. 4.4 – Ordre d'exploration des noeuds dans le cas PFP

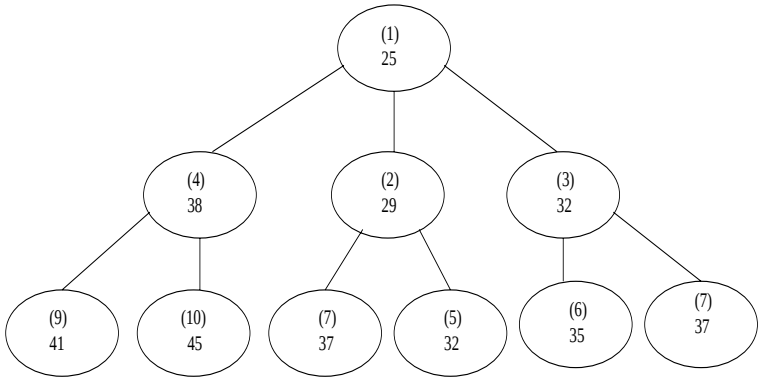

FIG. 4.5 – Ordre d'exploration des noeuds dans le cas CNL

4.3.4 Bornes inférieures et règles de dominance

Etant donné un noeud dans l'arbre de recherche, l'objectif est de déterminer un minorant de la plus petite valeur parmi toutes les solutions issues de ce noeud. D'une manière générale, les bornes inférieures procèdent par relaxation de contraintes. Le calcul exact de la plus petite valeur $v^\star$ réellement atteinte par une solution issue du noeud courant passe normalement par la résolution d'un problème Π NP-difficile (si le problème est polynomial, on dispose directement de cette solution et l'ensemble du sous-arbre est traité). En ne tenant pas compte de certaines contraintes, on se ramène à un problème polynomial Π' dont on peut calculer la solution optimale v'. La solution correspondant à la valeur $v^\star$ que l'on recherche satisfait toutes les contraintes de Π donc de Π'. On en déduit que $v' \leq v^\star$, autrement dit que v' est une borne inférieure pour le noeud courant.

La performance d'une borne inférieure est évaluée selon deux critères :
- d'une part, sa complexité, c'est-à-dire le nombre d'opérations élémentaires de calcul qu'elle requiert. Etant donné que l'algorithme correspondant est susceptible d'être appliqué à un grand nombre de noeuds, ce facteur contribue grandement au temps d'exécution total de la PSE ;
- d'autre part, sa précision, c'est-à-dire l'écart par rapport à la meilleure valeur d'une solution issue du noeud courant. Ce critère a un impact sur le nombre de branches éliminées

L'idéal serait de disposer d'une borne inférieure qui soit à la fois rapide à calculer et précise. Malheureusement, ces deux critères sont généralement opposés. On est par conséquent ramené à trouver un compromis satisfaisant entre eux, de manière à ce que le temps total d'exécution de la PSE soit le plus petit possible.

Le développement d'une borne inférieure étant intrinsèquement lié aux caractéristiques du problème que l'on cherche à résoudre, nous ne pouvons décrire de manière unifiée et générique celles que nous proposons. Chacune d'elles est présentée dans la section consacrée au problème pour lequel elle est employée, et est désignée par LB_v^u, où u indique le numéro du problème et v le numéro de la borne. De la même façon, les tests de dominance additionnels sont exposés séparément pour chaque problème.

4.3.5 Bornes supérieures

Toutes les méthodes que nous avons développées utilisent comme borne supérieure la valeur de la meilleure solution obtenue jusque là. Une borne initiale est fournie par une ou plusieurs heuristiques exécutées en amont de la PSE. Outre les heuristiques spécifiques que nous utilisons pour certains problèmes, nous nous sommes intéressés à une approche de résolution basée sur les travaux de Nawaz et al. [Nawaz et al.,83], proposée pour le problème de flowshop de permutation classique à n machines ($Fm_\pi||C_{max}$). Le principe de cette méthode appelée communément NEH est très général et peut être adapté à tout problème de recherche d'une permutation optimale. Dans un premier temps, on construit une liste ordonnée des éléments de $\{1, \ldots, n\}$. La solution est ensuite déterminée progressivement : à chaque itération le travail suivant dans la liste est inséré dans la séquence partielle à la meilleure position, de manière à minimiser le critère. Framinan et al. [Framinan et al.,03] proposent une analyse intensive de NEH et de différentes règles permettant de générer la liste des travaux, pour les problèmes de minimisation du makespan (C_{max}), des temps morts (équivalents à la somme des dates de fin sur les machines $\sum MC_k$), et de la somme des dates de fin des travaux ($\sum C_j$). Parmi les 177 règles testées, la règle initiale, qui consiste à trier les travaux dans l'ordre décroissant de la durée opératoire totale ($\sum_k p_{j,k}$) se révèle être la meilleure pour les critères du makespan et des temps morts, mais pas pour la somme des dates de fin des travaux, où elle semble être moins bonne qu'une génération aléatoire. Les auteurs proposent également pour chaque critère un ensemble de cinq règles complémentaires permettant d'obtenir une solution de très bonne qualité : NEH est appliquée successivement avec chaque règle, et la meilleure solution parmi les cinq obtenues est conservée. Les expériences menées indiquent que le gain par rapport à l'utilisation d'une seule règle est significatif. Par ailleurs, les auteurs mentionnent d'autres généralisations possibles de la méthode NEH, décrites dans la littérature :

- plutôt que de tester toutes les positions possibles lors de l'insertion d'un travail dans la séquence partielle, on peut se restreindre à un sous-ensemble de positions [Rajendran,93]
- à chaque itération, il est possible de tester l'insertion de plusieurs travaux de la liste dans la séquence partielle [Woo et Yim,98]
- à chaque itération, on peut conserver les $s > 1$ meilleures séquences partielles

Nous employons une autre extension, qui repose sur une application itérative de la méthode : à chaque étape, la séquence finale obtenue à la fin de l'étape précédente est utilisée en tant que liste ordonnée de travaux, pour la construction de la nouvelle séquence. La procédure s'arrête après un nombre fixé d'itérations et on conserve la meilleure séquence obtenue.

Pour représenter les différentes bornes supérieures utilisées, nous employons une notation analogue à celle des bornes inférieures : H_v^u, où u indique le problème et v le numéro de la borne.

4.3.6 Remarques concernant la génération des jeux d'essais

Pour chacun des problèmes étudiés, qui font l'objet des sections suivantes, nous avons mené des séries d'expériences afin de tester l'efficacité des méthodes proposées. Ne disposant pas de problèmes réels, issus du monde industriel, nous nous sommes tournés vers des jeux d'essais générés aléatoirement. Néanmoins, contrairement à ce qui se fait assez souvent dans la littérature, nous avons cherché à prendre en compte un certain nombre de paramètres de manière à représenter, à défaut de cas réels, des situations réalistes. Plutôt que de considérer des classes où seule la taille des instances varie, et où les durées opératoires sont par exemple toutes générées uniformément entre 1 et 100, nous nous sommes intéressés à la charge des machines, à l'existence d'opérations critiques, aux types de travaux traités, ... Toutefois, nous avons essayé de limiter raisonnablement le nombre de classes de jeux d'essais. D'autre part, bien que nous n'ayons pas approfondi cette question, et que nous nous soyons limités à des distributions uniformes pour la génération aléatoire de données, on peut s'interroger sur la pertinence de ce choix pour représenter des problèmes industriels, dont les données sont probablement plus proches de valeurs issues de lois log-normales.

4.4 Problème de minimisation du makespan

Dans cette section, nous considérons le problème $Fm_\pi|\theta_{j,k}^{max}, \theta_{j,k}^{min}|C_{max}$, pour lequel nous avons développé une Procédure par Séparation et Evaluation. Le schéma de séparation, ainsi que la stratégie d'exploration sont décrits en détails dans la section précédente. Nous nous concentrons donc ici sur les éléments spécifiques de la PSE : bornes inférieures, règle de dominance, et heuristiques fournissant une borne supérieure. De plus, nous présentons deux méthodes de résolution approchées, correspondant à des PSE tronquées : un beam search et une méthode d'approximation à paramètre ϵ. Nous terminons la section par la description des expériences numériques que nous avons menées. Parmi les quatre problèmes pour lesquels nous avons développé et testé des méthodes de résolution, celui-ci est le premier que nous avons étudié. Il est en outre plus général que les problèmes qui font l'objet des sections 4.6 et 4.7, et, par rapport au problème examiné dans la section 4.5, il considère un critère plus classique (le makespan). Il peut ainsi être considéré comme le problème central de cette thèse, pour lequel nous avons effectué une série d'expériences plus intensive.

4.4.1 Bornes inférieures

Supposons que le noeud courant N, pour lequel on souhaite déterminer une borne inférieure, corresponde à la séquence partielle σ, de longueur i (le noeud est alors situé à une profondeur i dans l'arbre de recherche). Nous avons développé quatre bornes tournées machines, c'est-à-dire pour lesquelles on relaxe les contraintes de précédence (et de time lags) au sein des travaux, mais on continue à prendre en compte les contraintes de capacité des machines (par opposition aux bornes tournées travaux). Ce choix est motivé par le fait que nous nous intéressons surtout à des problèmes pour lesquels les time lags minimaux ne sont pas trop élevés. Sous cette condition, la restriction à des ordonnancements de permutation ne dégrade pas trop la qualité des solutions pour le problème du flowshop général. Deppner [Deppner,04] a montré que si les durées opératoires et les time lags minimaux sont bornés, le ratio entre les valeurs optimales obtenus

en se limitant et sans se limiter aux ordonnancements de permutation tend vers 1 lorsque le nombre de travaux n tend vers l'infini. La restriction aux ordonnancements de permutation est donc raisonnable pour des problèmes avec de petits time lags minimaux. Pour de tels problèmes, si le nombre de travaux est grand par rapport au nombre de machines, la charge maximale des machines $max_k\{\sum_j p_{j,k}\}$ a de fortes chances d'être plus élevée que la longueur maximale des travaux $max_j\{\sum_k p_{j,k} + \theta^{min}_{j,k}\}$.

La première borne développée LB^1_1 consiste simplement à tasser les unes contre les autres, sans tenir compte des contraintes de time lags, les opérations des travaux qui n'ont pas encore été placés à la fin de l'ordonnancement partiel associé à la séquence σ, et à conserver la plus petite date de fin sur toutes les machines : $LB^1_1(N) = max_k\{C_{\sigma(i),k} + \sum_{j\in\bar{\sigma}} p_{j,k}\}$, où $\bar{\sigma}$ représente l'ensemble des travaux non encore placés (complémentaire de l'ensemble support de la séquence σ).

Afin de raffiner cette borne, le travail $j^\star$ qui sera placé en dernière position de l'ordonnancement complet peut être considéré séparément : les opérations des autres travaux non encore placés sont tassées comme précédemment, et $j^\star$ est calé au plus tôt sur le profil obtenu (l'ordonnancement de ce travail se fait comme dans l'algorithme de résolution du problème restreint). Plus précisément, la date de début au plus tôt sur la machine k de $j^\star$ est donnée par $C_{\sigma(i),k} + \sum_{j\in\bar{\sigma}\setminus\{j^\star\}} p_{j,k}$. Le makespan qui résulte du placement de $j^\star$ est une borne inférieure pour toutes les solutions commençant par la séquence partielle σ et se terminant par $j^\star$. Pour déterminer une nouvelle borne inférieure LB^1_2 pour le noeud courant, il suffit de considérer successivement tous les travaux qui ne figurent pas dans σ dans le rôle de $j^\star$, et de conserver le minimum parmi les $n - i$ valeurs obtenues.

On parvient à une borne plus précise LB^1_3 en utilisant une estimation de meilleure qualité pour l'ordonnancement sur les deux premières machines. Plutôt que de se contenter d'additionner les durées des travaux restants pour définir un profil sur lequel le travail $j^\star$ placé en dernière position est calé, on relaxe uniquement les contraintes de time lags maximaux entre ces deux machines pour les travaux de $\bar{\sigma}\setminus\{j^\star\}$ et on applique la règle de Mitten-Johnson [Mitten,59]. Celle-ci détermine un ordonnancement optimal pour le flowshop de permutation à deux machines avec time lags minimaux $F2_\pi|\theta^{min}_j|C_{max}$. En calant l'ordonnancement donné par cette règle sur l'ordonnancement partiel associé à σ, on obtient une plus grande date de début au plus tôt pour $j^\star$ sur la deuxième machine. Sur la première machine, la date de début au plus tôt pour $j^\star$ ne change pas, puisque toutes les opérations s'enchaînent sans temps morts avec la règle de Mitten-Johnson.

En généralisant l'idée précédente à tous les couples de machines consécutives, on peut définir une borne LB^1_4 : on obtient ainsi $m - 1$ bornes inférieures, dont on garde la plus élevée.

Les bornes précédentes sont numérotées dans l'ordre croissant de leur précision. Autrement dit, pour tout noeud N de l'arbre de recherche, on a : $LB^1_1(N) \leq LB^1_2(N) \leq LB^1_3(N) \leq LB^1_4(N)$. Il est donc inutile de calculer les valeurs obtenues par plusieurs d'entre elles pour un même noeud. Toutes ces bornes sont illustrées sur un exemple représenté sur les figures 4.6 4.7 4.8 4.9. L'ordonnancement partiel est représenté par le profil hachuré, et on suppose qu'il reste trois travaux h, i et j à placer. De plus, on ne considère qu'un seul time lag minimal positif $\theta^{min}_{i,1}$ et un seul time lag maximal fini $\theta^{max}_{j,1}$. Sur cet exemple, $LB^1_2 = LB^1_3$: l'application de la règle de Mitten-Johnson [Mitten,59] sur les deux premières machines conduit aux séquences (i, j), (h, j) et (i, h)

respectivement, mais il n'en résulte qu'un décalage (la deuxième opération j dans le premier cas, lorsque h est placé en dernier), et celui-ci n'influence pas la plus petite valeur obtenue (lorsque i est placé en dernier). La figure 4.9 ne représente que les ordonnancements construits en appliquant la règle de Mitten-Johnson sur les deux dernières machines, puisque les cas correspondant aux deux premières machines sont déjà illustrés sur la figure 4.8. Les séquences déterminées sont respectivement (j, i), (j, h) et (h, i). On obtient trois décalages (troisième opération de h lorsque i est placé en dernier et troisièmes opérations de h et de i lorsque j est placé en dernier), mais ils ne contribuent pas à faire augmenter la borne inférieure.

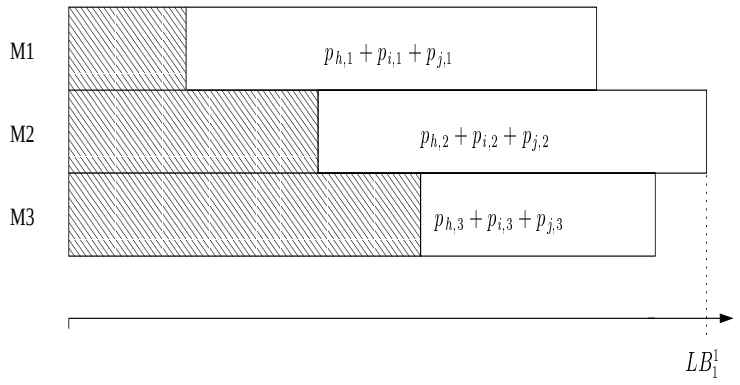

FIG. 4.6 – Borne inférieure LB_1^1

Dans la PSE que nous avons développé, un seul type de borne est sélectionné avant l'exécution et est appliqué à tous les noeuds que l'on doit évaluer.

4.4.2 Règle de dominance

Indépendemment de l'utilisation de la borne inférieure, une règle de dominance simple a été testée, à la racine de l'arbre. Elle consiste à comparer, pour tous les couples de travaux (i, j) les dates de fin obtenues pour les ordonnancements partiels correspondant aux séquences (i, j) et (j, i). Si pour toutes les machines, la date de fin est plus petite (respectivement plus grande) pour (i, j) que pour (j, i), alors les solutions commençant par (j, i) (respectivement (i, j)) sont dominées par celles qui commencent par (i, j) (respectivement (j, i)) et il est inutile de rechercher un ordonnancement optimal parmi elles. En réalité, ce test de dominance nous amène à construire tous les noeuds situés à la profondeur 2 dans l'arbre de recherche et à essayer d'en éliminer. Etant donné la faible profondeur, la suppression d'un noeud revient à réduire considérablement l'espace de recherche ; dans ce cas précis, $(n-2)!$ solutions sont issues d'un seul noeud.

4.4.3 Bornes supérieures

Cinq heuristiques fournissant des bornes supérieures en amont de la PSE ont été développées. La première, désignée par H_1^1, construit progressivement un ordonnancement réalisable en ajoutant à chaque étape, en fin de l'ordonnancement partiel, le travail non encore placé qui minimise le makespan. Soit σ_i la séquence des i premiers travaux $(i < n)$. On a alors : $\sigma_{i+1} = \sigma_i \oplus j^\star$, où $j^\star$ vérifie $C_{max}(\sigma_i \oplus j^\star) = min_{j \notin \sigma_i}\{C_{max}(\sigma_i \oplus j)\}$. Cette méthode peut être considérée comme une recherche sans retour en arrière dans l'arbre associé au schéma de séparation de la PSE, où le noeud choisi à chaque étape correspond à l'ajout du travail $j^\star$ défini par la relation précédente.

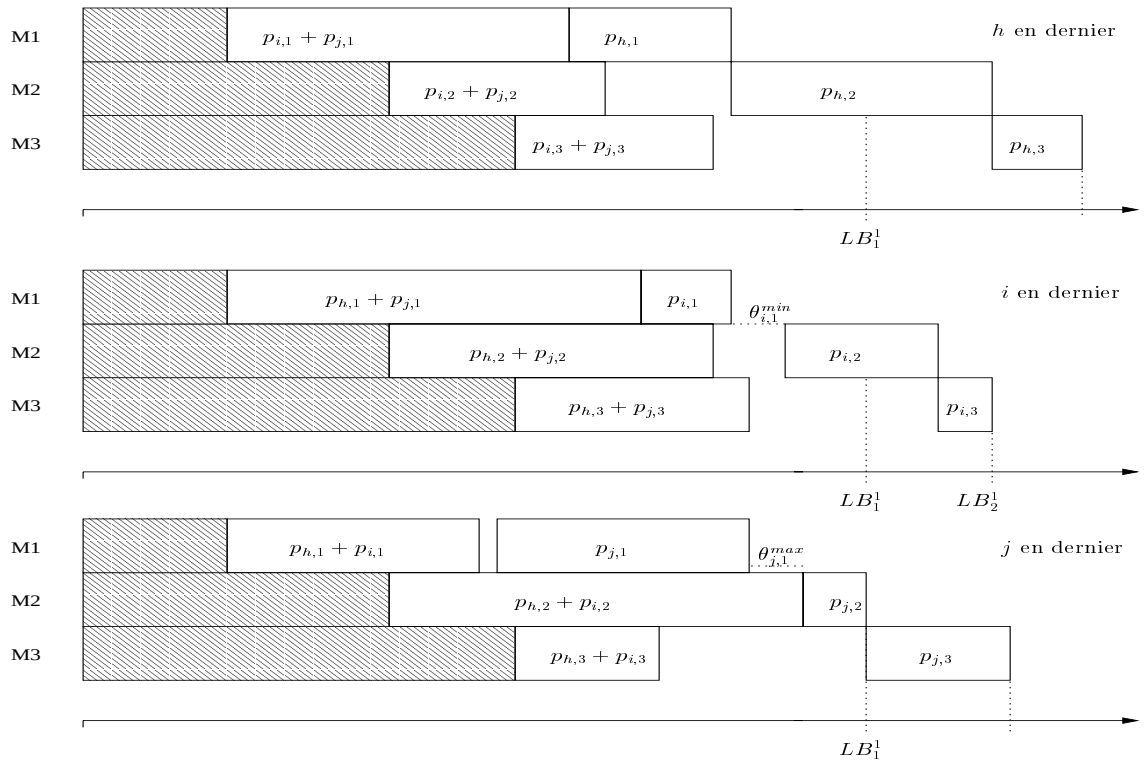

FIG. 4.7 – Borne inférieure LB_2^1

La seconde heuristique H_2^1 fonctionne sur un principe similaire. A chaque itération, on ajoute le travail $j^\star$ qui minimise le temps mort total sur toutes les machines. Puisque l'ajout d'un travail ne modifie pas l'ordonnancement des travaux placés lors des étapes précédentes, il suffit de considérer le temps mort additionnel provoqué par l'ajout de chaque travail non encore placé. Soit σ_i la séquence des i premiers travaux ($i < n$). On a alors : $\sigma_{i+1} = \sigma_i \oplus j^\star$, où $j^\star$ vérifie $\sum_k (t_{j^\star,k} - C_{\sigma_i(i),k}) = min_{j \notin \sigma_i}\{\sum_k (t_{j,k} - C_{\sigma_i(i),k})\}$.

La troisième heuristique, notée CDS^1 est l'adaptation de la méthode CDS [Campbell et al.,70] au problème de flowshop de permutation en présence de time lags. Initialement, cette méthode approchée est conçue pour résoudre le problème $Fm_\pi||C_{max}$ en faisant appel à la règle de Johnson [Johnson,54]. Celle-ci est utilisée pour résoudre des problèmes à deux machines, pour lesquels elle est optimale, générés de la manière suivante : pour chaque travail j du problème initial, on définit un travail j du nouveau problème avec une durée $P_{j,1}$ (respectivement $P_{j,2}$) sur la première (respectivement deuxième) machine égale à la somme des durées opératoires du travail initial de la machine 1 à la machine k_1 (respectivement de la machine k_2 à la machine m) : $\forall j, P_{j,1} = \sum_{1 \leq k \leq k_1} p_{j,k}, P_{j,2} = \sum_{k_2 \leq k \leq m} p_{j,k}$. En faisant varier k_1 entre 1 et $m-1$ et k_2 entre 2 et m, on obtient $(m-1)^2$ problèmes du type $F2_\pi||C_{max}$ qui peuvent être résolus avec la règle de Johnson. Parmi les $(m-1)^2$ séquences déterminées, on conserve celle qui conduit à la plus petite valeur du critère pour notre problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$. On peut remarquer que l'utilisation de cette heuristique telle quelle est initialement proposée ne tient pas compte des time lags dans la détermination des séquences. Ceux-ci n'interviennent que dans la construction des ordonnancements correspondants, c'est-à-dire dans l'évaluation de ces séquences. On pourrait

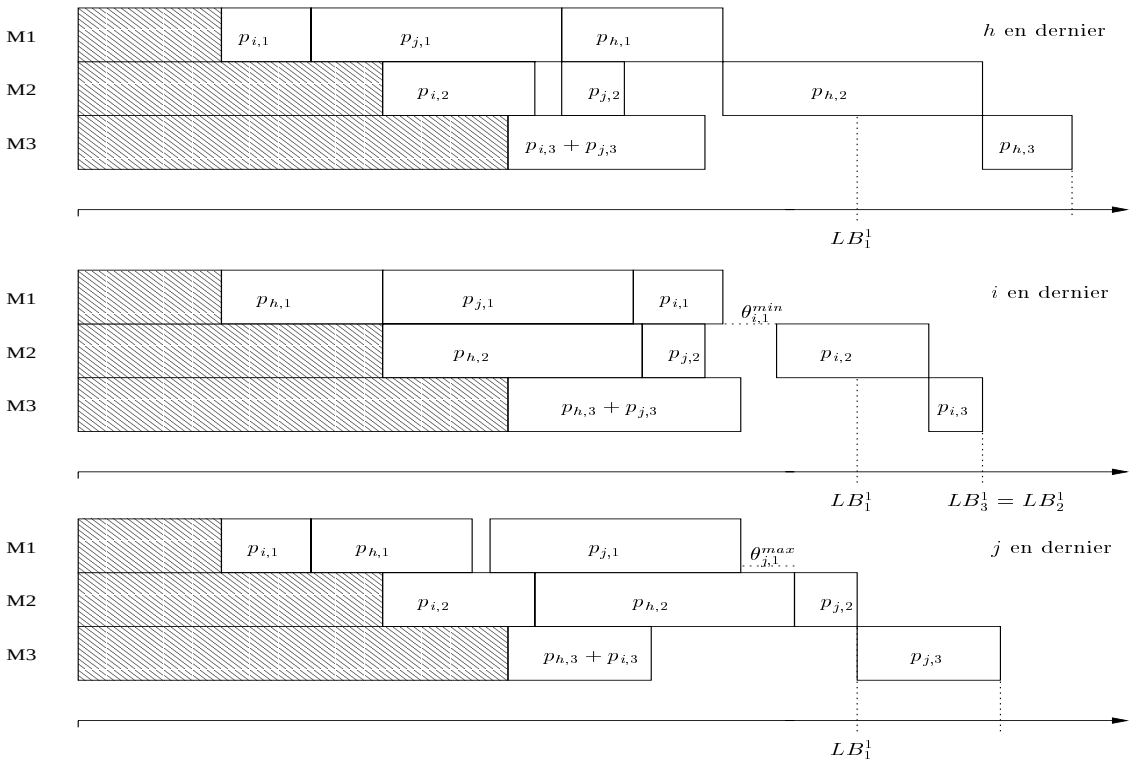

FIG. 4.8 – Borne inférieure LB_3^1

tout-à-fait envisager d'inclure les time lags minimaux dans la définition des durées artificielles $P_{j,1}$ et $P_{j,2}$, et/ou d'introduire un time lag minimal entre les deux machines du problème artificiel, ce qui conduirait à utiliser la règle de Mitten-Johnson [Mitten,59] plutôt que celle de Johnson. Le principal avantage de l'heuristique CDS provient du fait qu'elle fournit un nombre relativement important de séquences susceptibles d'être de bonne qualité.

Les deux dernières heuristiques proposées sont issues de la méthode NEH [Nawaz et al.,83] (voir Section 4.3.5). Elles sont désignées par $NEH^1_{\sum p}$ et $NEH^1_{\sum(p+\theta^{min})}$ et fonctionnent avec une seule itération. Les listes initiales sont obtenues en classant les travaux dans l'ordre décroissant de leur durée de traitement totale $\sum_k p_{j,k}$ (comme dans la version initiale) et de leur "longueur totale" $\sum_k (p_{j,k} + \theta^{min}_{j,k})$ respectivement.

4.4.4 Résolution approchée du problème : beam search et approximation à paramètre ϵ

Du fait qu'il s'agit d'une méthode de résolution exacte consacrée à un problème NP-difficile, une Procédure par Séparation et Evaluation conduit généralement à des temps de calculs très élevés et par conséquent rédhibitoires lorsque la taille du problème augmente. Pour pouvoir résoudre efficacement de tels problèmes, des méthodes approchées, dérivées de la PSE, peuvent être adoptées. Outre la possibilité de stopper l'exécution de la PSE lorqu'une limite de temps est atteinte, nous avons envisagé deux types de procédures qu'on rencontre assez souvent dans la littérature. L'approximation à paramètre ϵ peut être employée pour surmonter un obstacle récurrent : dans de nombreux cas, une grande partie du temps de calcul de la PSE est consacré

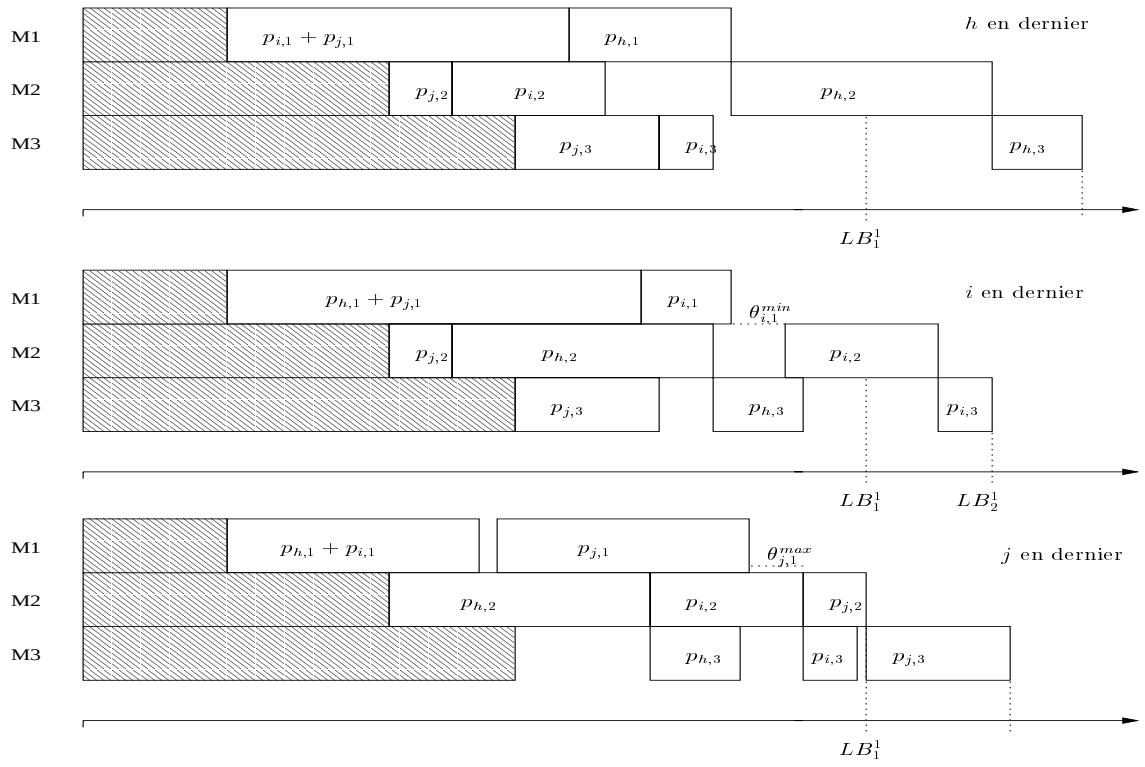

FIG. 4.9 – Borne inférieure LB_4^1

non pas à rechercher une solution optimale, mais à prouver que la meilleure solution courante est optimale. On peut facilement s'en rendre compte en mettant en place un compteur de noeuds et en étudiant le numéro du noeud correspondant à la solution optimale fournie par la procédure par rapport au nombre total de noeuds visités. L'approximation à paramètre ϵ consiste à réduire la borne supérieure de manière à supprimer davantage de noeuds. Plus précisément, la borne supérieure courante correspond à la meilleure valeur obtenue divisée par $1+\epsilon$. Avec cette heuristique, on dispose d'une majoration de l'écart relatif entre la solution fournie v et l'optimum $v^\star$. En effet, lorsque la procédure s'arrête, tous les noeuds non séparés ont une valeur supérieure ou égale à $v/(1+\epsilon)$. On en déduit que $v/(1+\epsilon) \leq v^\star$ c'est-à-dire $(v - v^\star)/v^\star \leq \epsilon$. Il s'agit donc d'une heuristique avec garantie de performance. Par contre, on n'a aucune assurance concernant le temps d'exécution et le nombre de noeuds visités, qui, dans le pire cas, peuvent être les mêmes qu'avec une PSE. En particulier, l'algorithme n'est pas polynomial.

Nous avons également développé un beam search, dont le principe de fonctionnement a déjà été présenté dans la section 4.3.3. Pour des raisons de simplicité, nous avons choisi d'utiliser une largeur de faisceau f constante ($\forall i, f_i = f$).

4.4.5 Génération des jeux d'essais

Comme nous l'avons expliqué dans la section 4.3.6, nous avons cherché à générer des instances susceptibles de traduire des situations réalistes, en prenant en compte un certain nombre de paramètres. Pour le problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, nous avons retenu les facteurs suivants :

- la charge des machines
- l'homogénéité ou l'hétérogénéité des durées opératoires des travaux
- le nombre de contraintes de time lags minimaux et/ou maximaux, en fonction des travaux et des machines
- l'amplitude des time lags

Ces paramètres sont liés à certaines caractéristiques du système de production, telles que la vétusté des équipements, l'existence d'opérations critiques dans le processus de production, ... Par exemple, si la chaîne de production est récente, les machines utilisées sont parfaitement adaptées aux technologies employées et aux produits fabriqués, et leurs charges sont plutôt équilibrées. Au fur et à mesure que la fabrication de nouveaux produits est lancée, les machines sont de moins en moins adaptées à ces produits. Il en résulte un déséquilibre au niveau des charges, ainsi qu'une différence sensible de durées opératoires entre les nouveaux produits et les anciens.

Nous avons généré 16 groupes de problèmes, chacun d'eux contenant 10 instances. Afin de réaliser des comparaisons significatives, le nombre de travaux a été fixé à 12, et le nombre de machines à 5, à l'exception d'une classe (la classe 9), pour laquelle $m = 10$. Puisque le problème consiste à déterminer la permutation de traitement optimale, le nombre de solutions (intéressantes), est de $n!$. Le nombre de machines n'intervient qu'au niveau de l'algorithme de construction d'un ordonnancement à partir d'une séquence de travaux, dont la complexité est en $O(m.n)$. Le paramètre limitant la taille des jeux d'essais pour lesquels le temps de calcul reste raisonnable est donc le nombre n de travaux. Etant donné que la toute première version de l'algorithme que nous avons implémentée a été écrite en Visual Basic, nous nous sommes limités à des instances à 12 travaux, pour lesquelles les temps d'exécution étaient de l'ordre de la seconde ou de la dizaine de secondes. Le passage à un programme codé en langage C a permis de réduire considérablement les temps de calcul, et par conséquent d'envisager une résolution exacte pour des problèmes de plus grande taille (jusqu'à 18 travaux). Néanmoins, nous avons conservé les classes de jeux d'essais initiales pour évaluer la performance de nos méthodes.

Le tableau 4.1 présente ces classes de problèmes, en utilisant les conventions suivantes :

- les trois premières colonnes indiquent respectivement le numéro de la classe, le nombre de travaux n et le nombre de machines m
- la quatrième colonne traduit l'hétérogénéité des durées opératoires : dans la plupart des cas, les durées $p_{j,k}$ sont générées d'après une distribution uniforme entre 20 et 50 ("Non" dans la colonne), tandis que pour les classes restantes, les 6 premiers travaux ont des durées comprises entre 20 et 50 et les 6 autres ont des durées comprises entre 20 et 100 ("Oui" dans la colonne)
- la cinquième colonne présente les charges des machines : celles-ci peuvent être homogènes ("Non" dans la colonne) ou hétérogène ("Oui" dans la colonne). Dans ce dernier cas, les durées opératoires sur les machines indiquées dans la case sont réduites de 25%
- les quatre dernières colonnes traitent des contraintes de time lags : les deux dernières colonnes indiquent pour quelles machines et quels travaux ces contraintes sont définies. Par exemple, la notation 2-4 dans l'avant-dernière colonne signifie que les time lags sont définis uniquement entre les machines 2 et 3 et entre les machines 4 et 5. Les valeurs correspondantes sont obtenues en fonction des paramètres λ et μ (colonnes 5 et 6). Les time lags maximaux sont générés d'après une distribution uniforme entre 0 et $\Theta^{max} = 70\lambda$ et les time lags minimaux ont des valeurs comprises entre 0 et $\Theta^{min} = 70\mu$. Si le time lag minimal $\theta^{min}_{j,k}$ d'une opération est supérieur au time lag maximal correspondant $\theta^{max}_{j,k}$, on

fixe $\theta_{j,k}^{min} = \theta_{j,k}^{max}$.

Classe	n	m	durées	charges	λ	μ	machines	travaux
1	12	5	Non	Non	0.1	0	Toutes	Tous
2	12	5	Non	1-3-5	0.1	0	Toutes	Tous
3	12	5	Non	Non	0.1	0	Toutes	1-2-3-4
4	12	5	Non	Non	$+\infty$	0	Non	Non
5	12	5	Non	1-3-5	0.1	0	2-4	Tous
6	12	5	Non	1-3-5	0.1	0	1-3	Tous
7	12	5	Oui	Non	0.1	0	Toutes	Tous
8	12	5	Non	Non	0.05	0	Toutes	Tous
9	12	10	Non	Non	0.1	0	Toutes	Tous
10	12	5	Oui	1-3-5	0.1	0	Toutes	Tous
11	12	5	Non	Non	0.2	0.1	Toutes	Tous
12	12	5	Non	Non	0.2	0.2	Toutes	Tous
13	12	5	Non	Non	0.1	10	Toutes	Tous
14	12	5	Non	1-3-5	0.2	0.1	Toutes	Tous
15	12	5	Non	1-3-5	0.2	0.1	Toutes	1-2-3-4
16	12	5	Non	1-3-5	0.2	0.1	2-4	Tous

TAB. 4.1 – Caractéristiques des classes de jeux d'essais utilisés

On peut remarquer, d'après le tableau 4.1, que les 10 premières classes contiennent des problèmes sans time lags minimaux ($\mu = 0$). Par ailleurs, la classe 4 correspond à des instances de flowshop classique, sans time lags ($\lambda = +\infty$ et $\mu = 0$). Enfin, les jeux d'essais de la classe 13 sont tels que la plupart des time lags minimaux sont égaux aux time lags maximaux (on parle alors de time lags exacts). Il est évident que nos expériences ne sont pas exhaustives. Par exemple, nous aurions pu changer les machines goulets (en comparant les situations où elles sont au début, au milieu, à la fin, au lieu d'être alternées). Il est à noter que, pour ce problème, nous n'avons pas généré de familles correspondant à des time lags négatifs, alors que nos algorithmes permettent de traiter ce cas particulier. Nous aurions pu également dupliquer toutes les familles d'exemples en utilisant des dates de disponibilité pour les travaux (voir la section 4.8.1). Intuitivement, plus celles-ci sont dispersées, plus le problème est facile. De telles expériences peuvent éventuellement être réalisées ultérieurement.

Les résultats obtenus sur ces différentes classes sont exposés dans la section suivante.

4.4.6 Résultats expérimentaux

Nous présentons ici les résultats obtenus à partir de programmes codés en langage C, sur un PC équipé d'un processeur à 1200 MHz.

a) Performance des heuristiques

Nous nous intéressons tout d'abord à la qualité des heuristiques employées pour obtenir une borne supérieure initiale. Pour chaque instance, nous avons calculé l'écart relatif (en %) entre le makespan C_H de la solution fournie par l'heuristique H et la valeur optimale C_{opt}, obtenue grâce à la PSE : $(C_H - C_{opt})/C_{opt} \times 100$. Le tableau 4.2 présente les valeurs moyennes pour chaque classe d'instances. Pour les 10 premières classes, sans time lags minimaux, les heuristiques $NEH^1_{\sum p}$ et $NEH^1_{\sum (p+\theta^{min})}$ sont similaires. Ces deux méthodes sont assez nettement meilleures

Classe	$NEH^1_{\sum p}$	$NEH^1_{\sum(p+\theta^{min}}$	CDS^1	H^1_1	H^1_2
1	1.63		4.82	9.65	5.75
2	1.10		4.21	9.12	5.44
3	1.75		5.37	13.53	7.10
4	0.97		2.42	13.00	7.87
5	0.41		1.67	9.02	6.17
6	0.53		1.39	12.52	6.51
7	1.18		4.57	12.43	7.05
8	1.34		5.26	11.69	6.95
9	1.91		3.59	12.53	6.10
10	2.26		4.74	16.42	10.53
11	1.95	1.70	4.35	10.78	4.56
12	2.09	2.29	5.88	10.13	5.66
13	1.21	1.01	5.68	9.46	4.91
14	2.00	1.87	5.45	9.80	6.12
15	1.27	0.87	4.13	11.78	7.10
16	0.27	0.29	1.15	10.53	6.85

TAB. 4.2 – Performance des heuristiques

que les autres : dans plus de 90 % des cas, la meilleure solution fournie par une heuristique est issue de $NEH^1_{\sum p}$ ou de $NEH^1_{\sum(p+\theta^{min})}$. L'écart relatif moyen est inférieur à 2.3 % en moyenne, sur toutes les instances, et il n'excède dans aucun cas 5.3 %. Les temps d'exécution sont trop courts pour être mesurés correctement (inférieurs à 10 millisecondes). Ils ne sont donc pas présentés.

b) Comparaison des bornes inférieures

Afin d'apprécier la performance des bornes inférieures, nous avons pris en compte deux critères : le temps d'exécution mis par la PSE pour atteindre la solution optimale et le nombre de noeuds évalués, c'est-à-dire le nombre d'appels à la borne inférieure. Il convient de noter que ce nombre est différent du nombre de séparations effectuées, qu'on retrouve parfois comme indicateur dans certains articles. Ces deux critères permettent de comparer les bornes inférieures selon les deux aspects importants, que nous avons évoqués en section 4.3.4. Le nombre de noeuds évalués varie selon la précision de la borne : plus celle-ci fournit une valeur proche de l'optimum et plus le nombre de noeuds éliminés dans l'arborescence est grand, donc moins il reste de noeuds à évaluer. La durée d'exécution, quant à elle, dépend à la fois du nombre d'appels à la procédure de calcul de la borne inférieure, donc de la précision de celle-ci, et de la complexité de la borne. Nous précisons qu'à l'issue d'une série d'expériences préliminaires que nous ne présentons pas ici, la borne LB^1_1 s'est révélée plutôt inefficace. Par conséquent, nous avons décidé de ne pas l'inclure dans les expériences que nous exposons à présent. Sauf mention contraire, les versions de la PSE employées utilisent la stratégie d'exploration MFP et ne font pas appel à la règle de dominance.

Le tableau 4.3 permet de comparer les trois bornes testées en ce qui concerne le temps d'exécution global. Pour chaque classe de problèmes, nous faisons figurer les temps de calcul moyens T_2, T_3 et T_4 obtenus avec les différentes versions de la PSE, lorsque les bornes LB^1_2, LB^1_3 et LB^1_4 sont utilisées, l'unité étant la milliseconde. Nous désignons par t_2, t_3 et t_4 les temps d'exécution obtenus sur une instance donnée. Nous avons également calculé les différences relatives

Classe	T_2	T_3	$\Delta t_3(\%)$	T_4	$\Delta t_4(\%)$
1	104	231	112	638	494
2	71	148	138	415	590
3	47	93	109	256	462
4	21	50	132	115	407
5	16	28	87	54	310
6	5	8	76	16	213
7	1368	2798	116	7066	432
8	352	807	138	2262	564
9	826	1483	82	7907	888
10	615	1287	117	3075	422
11	296	639	109	1503	426
12	414	909	123	2388	513
13	1119	2356	118	6263	500
14	202	400	116	1018	485
15	118	216	87	496	361
16	211	383	101	628	304

TAB. 4.3 – Comparaison des temps de calculs en fonction de la borne inférieure employée

moyennes, entre LB_3^1 et LB_2^1 (désignée par Δt_3 et définie comme la moyenne de $(t_3 - t_2)/t_2 \times 100$ sur les 10 instances de chaque classe), et entre LB_4^1 et LB_2^1 (avec une définition similaire). Dans tous les cas, nous utilisons comme borne supérieure initiale la meilleure valeur fournie par les heuristiques de la section 4.4.3.

La PSE utilisant la borne LB_2^1 est beaucoup plus rapide que les variantes qui ont recours aux autres bornes : pour chaque instance, elle requiert moins de temps pour atteindre une solution optimale. Le temps d'exécution est environ deux fois plus important quand on utilise LB_3^1, et entre 3 et presque 10 fois plus important lorsque l'on choisit LB_4^1.

Pourtant, d'après la définition des bornes inférieures, LB_4^1 est plus précise que LB_3^1, qui est elle-même plus précise que LB_2^1. Ce résultat théorique ne permet cependant pas de savoir si la différence entre ces bornes, en termes de précision du résultat, est importante ou pas. Le tableau 4.4 peut nous renseigner à ce sujet. Il présente le nombre moyen N_i de noeuds évalués en fonction de la borne inférieure LB_i^1 employée, ainsi que les différences relatives moyennes entre LB_3^1 et LB_2^1 (désigné par Δn_3, et défini comme la moyenne sur les 10 instances de chaque classe de $(n_3 - n_2)/n_2 \times 100$), et entre LB_4^1 et LB_2^1 (Δn_4, défini de manière similaire).

D'après le tableau 4.4, on peut remarquer que l'apport de LB_3^1 par rapport à LB_2^1 en termes de précision est relativement faible, inférieur à 4.5 % en moyenne pour chaque classe. Au contraire, LB_4^1 permet de réduire le nombre de noeuds évalués de 20 % en moyenne. Malgré tout, comme nous pouvons le constater avec les durées d'exécution présentées dans le tableau 4.3, cette réduction n'est pas suffisante pour compenser les temps de calculs supplémentaires engendrés par la complexité de cette borne. La borne LB_2^1 représente ainsi le meilleur compromis entre précision et complexité et nous la conservons comme borne inférieure de référence pour les autres expériences.

c) Intérêt de la borne supérieure initiale

Classe	N_2	N_3	$\Delta n_3(\%)$	N_4	$\Delta n_4(\%)$
1	49491	48607	-2.5	39294	-20.1
2	36306	35929	-1.2	32761	-8.8
3	23178	21164	-4.5	17293	-24.6
4	11284	11249	-1.9	7542	-38.8
5	9009	9009	0.0	5439	-6.7
6	2349	1847	-4.1	1291	-10.7
7	673400	636951	-4.1	498347	-29.2
8	154207	150889	-2.0	127189	-19.8
9	219831	218815	-0.6	181250	-17.9
10	302390	291857	-3.0	201637	-34.7
11	142609	138032	-3.6	95270	-28.0
12	196190	190038	-2.7	147388	-21.2
13	538696	514742	-4.4	418970	-21.1
14	102080	99507	-1.4	77031	-21.9
15	64861	64262	-1.3	49978	-25.6
16	118063	118063	0	61749	-16.9

TAB. 4.4 – Comparaison des nombres de noeuds évalués en fonction de la borne inférieure employée

Les heuristiques présentées en section 4.4.3 ont pour but de fournir une borne supérieure initiale à la PSE, ce qui peut permettre d'éliminer certains noeuds de l'arbre de recherche assez tôt, sans avoir besoin d'atteindre une solution de bonne qualité. Afin d'évaluer quantitativement le gain apporté par cette technique, nous avons comparé les temps d'exécution moyens T_{UB} et T et les nombres moyens de noeuds N_{UB} et N pour lesquels la borne inférieure est calculée, selon qu'on fait appel ou non à une borne supérieure initiale. Dans le cas où l'on emploie une telle borne, on utilise comme valeur la meilleure valeur fournie par l'ensemble des heuristiques. En ce qui concerne la borne inférieure, nous avons sélectionné la borne LB_2^1, qui s'est révélée être la plus performante, d'après les résultats précédents. Le tableau 4.5 présente les valeurs de T, T_{UB} (en millisecondes), N et N_{UB}, ainsi que les gains relatifs moyens correspondants Δt et Δn.

D'une manière générale, les gains en termes de temps et en termes de nombre de noeuds évalués sont du même ordre de grandeur pour chaque classe d'instances. Si l'on excepte les trois classes de problèmes "faciles" (classes 4, 5 et 6), pour lesquelles les valeurs sont relativement petites et les gains moyens pas nécessairement significatifs, l'apport reste modéré, les gains atteignant tout au plus 26% (pour la classe 16). L'utilisation d'une borne supérieure initiale ne permet donc pas de traiter des instances de plus grande taille. Néanmoins, étant donné que les temps d'exécution des heuristiques sont négligeables devant ceux de la PSE, il peut être intéressant d'avoir recours à ces heuristiques.

Nous n'avons pas essayé d'utiliser des heuristiques, éventuellement plus simples que celles que nous avons présentées, pour améliorer la borne supérieure au cours de l'exploration, en complétant les solutions partielles "prometteuses" de manière à obtenir des solutions complètes. Cette possibilité a le même inconvénient que la sophistication des bornes inférieures : cela augmente le coût global de la recherche, puisque le calcul est effectué pour un (très) grand nombre de noeuds. En outre, on peut supposer que une telle technique n'apporterait pas un gain significatif, comme semblent l'indiquer les expériences menées en ce qui concerne l'apport de la borne supérieure

Classe	T	T_{UB}	$\Delta t(\%)$	N	N_{UB}	$\Delta n(\%)$
1	119	104	22	55176	49491	12
2	75	71	9	38286	36306	11
3	70	47	16	30486	23178	17
4	32	21	49	15209	11284	36
5	19	16	33	10634	9009	56
6	18	5	50	4048	2349	32
7	1460	1368	11	720568	673400	11
8	377	352	12	178592	154207	12
9	845	826	4	225234	219831	4
10	621	615	2	304972	302390	2
11	309	296	4	149359	142609	7
12	419	414	2	198287	196190	1
13	1163	1119	6	560650	538696	7
14	232	202	12	117961	102080	9
15	125	118	18	67576	64861	9
16	234	211	26	131464	118063	46

TAB. 4.5 – Comparaison des temps d'exécution et des nombres de noeuds évalués en fonction de l'utilisation d'une borne supérieure initiale

initiale, à la racine de l'arbre de recherche.

Classe	T_{MFP}	T_{PFP}	$\Delta_{strat}t(\%)$	N_{MFP}	N_{PFP}	$\Delta_{strat}n(\%)$
1	104	109	-6	49491	48980	0
2	71	74	-14	36306	35153	3
3	47	47	-6	23178	21616	6
4	21	22	-6	11284	11579	-10
5	16	19	-12	9009	9630	0
6	5	4	50	2349	2302	1
7	1368	1411	-3	673400	669122	0
8	352	345	3	154207	157617	5
9	826	823	0	219831	216137	2
10	615	637	-1	302390	301810	2

TAB. 4.6 – Comparaison des temps d'exécution et des nombres de noeuds évalués en fonction de la stratégie d'exploration

d) Rôle de la stratégie d'exploration

Nous avons jusque là considéré des versions de notre PSE suivant la stratégie d'exploration MFP (voir section 4.3.3). Le tableau 4.6 expose les résultats obtenus, en termes de temps de calculs et en termes de noeuds, lorsque la procédure a recours à la stratégie MFP et à la stratégie PFP avec un paramètre $x = 0.5$, respectivement. Pour chaque classe de 1 à 10, T_{MFP} et T_{PFP} indiquent les durées d'exécution moyennes en millisecondes, N_{MFP} et N_{PFP} indiquent les nombres de noeuds évalués, et $\Delta_{strat}t$ et $\Delta_{strat}n$ représentent les gains moyens apportés par la stratégie PFP par rapport à la stratégie MFP. Dans tous les cas, on utilise comme borne supérieure initiale la meilleure valeur fournie par les heuristiques et comme borne inférieure la borne LB_2^1.

En ce qui concerne les durées d'exécution, la stratégie PFP conduit à une procédure légèrement moins rapide : dans l'ensemble, on note une perte de quelques pourcents au niveau des temps de calculs, par rapport à la PSE utilisant la stratégie MFP. Le comportement est inverse en termes de noeuds évalués : la stratégie PFP permet de réduire le nombre d'appels à la borne inférieure de quelques pourcents. Ceci tendrait donc à montrer que l'exploration est légèrement mieux menée, mais que cet apport n'est pas suffisant pour compenser les temps de calculs supplémentaires, nécessaires au partitionnement des fils dans les trois catégories. Par ailleurs, nous rappelons que les gains relatifs moyens sont obtenus en calculant la moyenne des gains relatifs pour l'ensemble des instances d'une classe (ce qui est différent du gain relatif global, qui serait calculé à partir de la moyenne des durées d'exécution). Par conséquent, on a $\Delta t_{strat} \neq (T_{PFP} - T_{MFP})/T_{MFP} \times 100$, ce qui peut expliquer certaines valeurs à première vue surprenantes dans le tableau 4.6.

Classe	T	T_{dom}	$\Delta t_{dom}(\%)$
1	104	93	19
2	71	77	-18
3	47	50	-3
4	21	23	0
5	16	16	0
6	5	4	50
7	1368	1356	-3
8	352	390	-8
9	826	1039	-17
10	615	610	0

TAB. 4.7 – Comparaison des temps d'exécution en fonction du recours à la règle de dominance

e) Apport de la règle de dominance

De manière analogue à la section précédente, nous analysons l'intérêt d'employer la règle de dominance exposée en section 4.4.2. Dans le tableau 4.7, nous présentons les résultats obtenus selon qu'on inclut ou pas la règle de dominance au sein de la PSE. Pour chaque classe de 1 à 10, T et T_{dom} indiquent les durées d'exécution moyennes en millisecondes, et $\Delta_{dom}t$ représente le gain moyen apporté par la règle de dominance. Dans tous les cas, on utilise comme borne supérieure initiale la meilleure valeur fournie par les heuristiques, comme borne inférieure la borne LB_2^1 et comme stratégie d'exploration la stratégie MFP.

Si l'on met de côté la classe 6, pour laquelle les durées d'exécution sont trop petites pour être représentatives, l'apport de la règle de dominance n'est pas évident : le gain relatif moyen varie entre -18% et 19% selon la classe d'instances, et il est pour 5 d'entre elles proche de 0. On peut donc conclure là encore que l'apport de la règle de dominance n'est que marginal, et que dans certains cas il ne compense pas le temps de calcul supplémentaire.

f) Evaluation des méthodes arborescentes approchées

Dans cette section, nous cherchons à apprécier la performance des méthodes présentées en 4.4.4, notamment par rapport aux heuristiques moins sophistiquées utilisées pour obtenir une borne supérieure initiale. Nous avons constaté que les meilleures d'entre elles sont $NEH^1_{\sum p}$ et $NEH^1_{(\sum p+\theta^{min})}$, pour lesquelles l'écart relatif moyen par rapport à l'optimal est de l'ordre de 2.5 % en moyenne, et ne dépasse jamais 5.3 %. Pour ces méthodes, les temps d'exécution sont négligeables. Il peut être intéressant de rechercher les durées nécessaires pour atteindre le même niveau de performance, en ayant une garantie sur celle-ci, avec une approximation à paramètre ϵ. Ainsi, le tableau 4.8 présente les durées d'exécution moyennes $T_{\epsilon=5\%}$ et $T_{\epsilon=2.5\%}$ (en millisecondes), ainsi que les écarts relatifs moyens $C_{\epsilon=5\%}$ et $C_{\epsilon=2.5\%}$ (en %), pour chaque classe d'instance de 1 à 10.Cette fois-ci, aucune borne supérieure initiale n'est employée.

En termes de précision, l'approximation à paramètre ϵ s'avère plutôt performante : l'écart relatif moyen par rapport à l'optimum est aux alentours de $\epsilon/2$, où ϵ représente la garantie de performance dans le pire cas. Ainsi, ce type de méthodes permet d'atteindre une précision plus importante que les heuristiques par construction. Les temps de calculs, bien que plus élevés,

Classe	$C_{\epsilon=5\%}$	$T_{\epsilon=5\%}$	$C_{\epsilon=2.5\%}$	$T_{\epsilon=2.5\%}$
1	2.1	4	1.1	18
2	2.1	0	1.2	7
3	2.8	5	1.4	5
4	3.3	15	1.2	16
5	1.6	0	0.7	3
6	1.4	0	0.7	0
7	2.3	502	0.9	759
8	1.9	41	0.8	99
9	2.5	111	1.2	196
10	2.1	73	0.8	175

TAB. 4.8 – Performance de l'approximation à paramètre ϵ

restent assez faibles pour les instances considérées. Cependant, les résultats obtenus pour les classes 7, 8, 9 et 10, constituées de problèmes plus "difficiles" que les autres, semblent indiquer que l'approximation à paramètre ϵ est très sensible au type de problème considéré. Ce résultat paraît naturel dans la mesure où il s'agit d'une procédure fonctionnant de manière similaire à la PSE, mais pour laquelle les conditions pour conserver un noeud sont plus restrictives. On retrouve donc un comportement analogue à celui de la méthode exacte. Nous rappelons d'ailleurs que l'approximation à paramètre ϵ repose sur un algorithme exponentiel, bien qu'il s'agisse d'une heuristique. C'est pourquoi une telle méthode ne permet pas, pour une faible valeur d'ϵ, d'augmenter significativement la taille des problèmes traités en un temps raisonnable, par rapport à une PSE.

Pour l'évaluation du beam search, nous avons procédé d'une manière différente : pour chaque classe d'instances de 1 à 10, le tableau 4.9 présente l'écart relatif moyen C_{BS} en %, entre la solution obtenue avec un beam search et l'optimum, ainsi que la durée moyenne d'exécution T_{BS} (en millisecondes). La procédure de beam search retenue conserve l'ensemble des noeuds de niveaux 1 et 2, et les 99 meilleurs noeuds aux niveaux suivants. Comme expliqué dans la section 4.3.3, on utilise une stratégie d'exploration de type CNL. D'autre part, les heuristiques par construction sont exécutées en amont afin de disposer d'une solution initiale, et la borne inférieure employée est LB_2^1. Celle-ci n'est utilisée dans notre beam search que pour le classement des noeuds et non pas pour leur élimination, par comparaison avec une borne supérieure. Avec le beam search, on pourrait mener d'autres séries d'expériences sur des problèmes de plus grande taille, mais dans ce cas, on ne disposerait plus des valeurs optimales fournies par la PSE et on devrait se contenter des solutions issues des heuristiques pour effectuer des comparaisons. Nous n'avons donc pas suivi cette possibilité.

D'après les résultats figurant dans le tableau 4.9, l'utilisation conjointe des heuristiques par construction et du beam search se révèle particulièrement efficace, puisque les écarts relatifs moyens sont inférieurs à 1%, sauf pour la classe 10. Le beam search améliore donc sensiblement la solution fournie par les heuristiques initiales. En outre, étant donné le principe de fonctionnement du beam search, le rôle de la borne supérieure est assez limité : celle-ci sert uniquement à fournir une solution de bonne qualité, dans le cas où le beam search n'en trouve pas, mais elle n'intervient en rien dans l'exploration de l'arbre de recherche. Ce sont donc les caractéristiques du beam search qui permettent d'atteindre des écarts relatifs à l'optimum si faibles. En ce qui concerne les durées d'exécution, on peut considérer, aux imprécisions de mesure près, qu'elles

Classe	C_{BS}	T_{BS}
1	0.3	34
2	0.2	38
3	0.5	36
4	0.3	36
5	0.1	46
6	0	45
7	0.7	33
8	0.3	37
9	0.8	59
10	1.4	36

TAB. 4.9 – Performance du beam search

sont constantes pour toutes les classes, sauf la classe 9. En effet, le beam search tel que nous l'avons utilisé évalue et traite toujours le même nombre de noeuds, quelle que soit l'instance. Il n'y a que pour les problèmes de la classe 9, pour lesquels le nombre de machines est différent, que les durées d'exécution sont nettement plus grandes. Pour les classes 7, 8 et 10, bien que les problèmes soient plus "difficiles" à résoudre que ceux des classes 1 à 6, les durées sont du même ordre. Un tel comportement est intéressant dans la mesure où l'on est capable, avant d'exécuter la procédure sur une instance, d'estimer assez précisément le temps de calculs, à partir uniquement de la taille du problème (nombre de travaux n et nombre de machines m).

g) Caractérisation des classes d'instances selon leur difficulté

L'ensemble des expériences dont nous avons exposé les résultats précédemment nous permettent de discuter de la difficulté des instances dans le cas d'une résolution exacte, notamment en fonction des paramètres utilisés pour leur génération (voir section 4.4.5). A partir des durées d'exécution et des nombres de noeuds évalués pour les différentes versions de la PSE, on peut émettre les conclusions suivantes :

- d'une manière générale, les problèmes incluant à la fois des time lags minimaux et des time lags maximaux sont plus difficiles que les problèmes avec time lags maximaux uniquement (si l'on compare les résultats pour la classe 1 et pour les classes 11, 12 et 13, ou pour la classe 2 et pour la classe 14). Plus les time lags minimaux et maximaux sont proches et plus les problèmes requièrent d'efforts en termes de calculs (comparaison entre les classes 1 et 8, et entre les classes 11 et 12). Enfin, les problèmes sont plus faciles lorsque les time lags ne sont définis que pour certains travaux, ou entre certaines machines (comparaison entre les classes 1 et 3, entre les classes 2 et 5-6, entre les classes 14 et 15)
- l'hétérogénéité des durées opératoires des travaux tend à rendre les problèmes plus difficiles (comparaison entre les classes 1 et 7)
- l'homogénéité des charges sur les différentes machines contribuent également à la difficulté des problèmes (comparaison entre les classes 1 et 2, entre les classes 11 et 14)

Les conclusions précédentes vont un peu à l'encontre des constatations formulées par Ruiz et Serifoglu [Ruiz et Serifoglu,05a] pour le flowshop hybride, qui indiquent que les distributions des données "temporelles" n'ont pas d'effet sur la difficulté des problèmes. Une explication possible à ce constat vient peut-être du fait que dans leurs expériences, la structure de l'atelier (nombre de machines à chaque étage) et d'autres contraintes influençant le nombre de solutions réalisables,

telles que la compatibilité des machines et des opérations, sont également prises en compte, et que leur contribution à la difficulté des problèmes masque celle, moins importante, des autres paramètres.

4.4.7 Conclusions pour le problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$

Le problème de minimisation du makespan dans un flowshop de permutation à m machines, en présence de time lags minimaux et maximaux, qui fait l'objet de cette section, est situé au coeur de notre étude. Nous avons développé une Procédure par Séparation et Evaluation pour sa résolution exacte. Afin de disposer d'une méthode la plus efficace possible, nous avons testé de nombreux éléments à incorporer, susceptibles de faciliter la recherche de la solution optimale. Les expériences numériques menées nous ont conduit à estimer l'apport de chacun de ces éléments. Ainsi, la version retenue consiste à utiliser une stratégie de type MFP, assez proche de la stratégie traditionnelle en profondeur d'abord, et à employer la borne inférieure LB_2^1, qui tasse les travaux non placés en traitant à part celui qui occupe la dernière position. On peut constater qu'un certain nombre de techniques plus sophistiquées (telles que les bornes inférieures LB_3^1 et LB_4^1, la stratégie PFP ou la règle de dominance), bien qu'elles permettent de réduire le nombre de noeuds examinés, n'apportent pas un gain suffisant pour compenser leur coût supplémentaire.

Pour une taille de problème fixée, nous avons aussi pu, grâce aux familles d'instances générées, identifié les facteurs qui ont une influence sur la difficulté des problèmes, notamment en termes de temps de calculs. En particulier la présence de time lags et la "force" de ces contraintes, l'hétérogénéité des durées opératoires des travaux et l'homogénéité des charges des machines tendent à rendre les problèmes plus difficiles à résoudre.

Les expériences réalisées portent sur des problèmes à 12 travaux. Etant donné le schéma de séparation employé, qui repose sur la construction progressive de la permutation des travaux, et la complexité de l'algorithme de construction d'un ordonnancement, c'est ce paramètre en premier lieu qui contribue à la durée d'exécution de la PSE. Les programmes que nous avons développés nous permettent de traiter des problèmes allant jusqu'à 18 travaux, en un temps raisonnable. Néanmoins, dans certaines situations particulières, par exemple lorsque les charges des machines vont du simple au double, il est possible de résoudre optimalement des instances comportant 100 travaux, en moins d'une seconde. Il est donc fondamental de préciser les paramètres pris en compte dans la génération des jeux d'essais.

4.5 Problème de minimisation de la somme pondérée des dates de fin des machines

Le problème que nous considérons dans cette section est celui de la minimisation de la somme pondérée des dates de fin des machines, dans un flowshop de permutation, en présence de contraintes de time lags minimaux et maximaux $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|\sum w_k MC_k$. La Procédure par Séparation et Evaluation que nous proposons est très proche de celle que nous avons présentée pour la minimisation du makespan (voir la section précédente). La seule différence concerne la définition d'une borne inférieure, et la génération des jeux d'essais.

4.5.1 Borne inférieure utilisée

Les résultats expérimentaux obtenus pour le problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, qui fait l'objet de la section précédente, indiquent que parmi les bornes inférieures testées, LB_2^1 conduit à la PSE la plus performante. Nous proposons donc d'adapter cette borne au nouveau critère que nous considérons $\sum w_k MC_k$. En fait, la nouvelle borne LB^2 procède exactement de la même manière que LB_2^1 : soit σ la séquence courante, comportant i travaux. Pour chaque travail $j^\star$ parmi les $n-i$ travaux non encore placés, la date de début au plus tôt sur la machine k de $j^\star$, que l'on suppose placé en dernière position, est donnée par $C_{\sigma(i),k} + \sum_{j\in\bar{\sigma}\setminus\{j^\star\}} p_{j,k}$ où $\bar{\sigma}$ désigne l'ensemble des travaux non encore placés. Cette situation revient à tasser sur chaque machine les opérations des travaux non encore placés, à l'exception de $j^\star$, sans temps mort. $j^\star$ est alors calé au plus tôt compte tenu de ces dates de début au plus tôt. La valeur $\sum_k w_k C_{j^\star,k}$ représente une borne inférieure pour les ordonnancements commençant par σ et se terminant par $j^\star$. En conservant la plus petite valeur obtenue pour tous les $j^\star$ possibles dans $\bar{\sigma}$, on dispose d'une borne inférieure pour les solutions commençant par σ. De même que pour le makespan, le fait de prendre en compte les contraintes de time lags pour le travail placé en dernière position permet d'accroître la borne inférieure, c'est-à-dire d'améliorer sa précision. C'est d'autant plus vrai ici que la date de fin sur chaque machine peut contribuer directement au calcul du critère.

4.5.2 Expériences numériques menées

Nous avons effectué une série d'expériences numériques dans le but d'évaluer la performance de notre PSE. L'algorithme a été implémanté en langage C et le programme a été exécuté sur un PC équipé d'un processeur à 1,2 GHz.

Nous avons généré trois grands groupes de jeux d'essais : le premier correspond à des problèmes avec time lags minimaux et maximaux, le second à des problèmes classiques, sans time lags ($\theta_{j,k}^{min} = 0, \theta_{j,k}^{max} = +\infty$), et le troisième à des problèmes sans attente ($\theta_{j,k}^{min} = \theta_{j,k}^{max} = 0$). Le nombre m de machines a été fixé à 5, 10 ou 15, et le nombre n de travaux à 5, 10 ou 13. Pour chaque combinaison (m, n), nous avons généré 25 instances. Les durées opératoires ont été obtenues selon une distribution uniforme entre 10 et 100, et pour le premier groupe, les time lags minimaux et maximaux ont été tirés aléatoirement entre 0 et 200 et entre 0 et 300 respectivement. Toutes les données sont des entiers. A noter que si un time lag minimal $\theta_{j,k}^{min}$ est plus grand que le time lag maximal correspondant $\theta_{j,k}^{max}$, alors on fixe les deux comme étant égaux à la plus petite valeur ($\theta_{j,k}^{max}$). Pour chaque instance, la procédure a été exécutée deux fois, pour résoudre le problème pondéré d'une part et le problème non pondéré d'autre part. Dans le cas pondéré, les poids ont été générés aléatoirement entre 1 et 5.

Le tableau 4.10 fournit les temps d'exécution moyens en millisecondes (t_{wMC} et t_{MC}), ainsi que le nombre de noeuds pour lesquels la borne inférieure a été calculée (N_{wMC} et N_{MC}). Les indices wMC et MC sont associés aux cas pondéré et non pondéré respectivement. D'après le tableau 4.10, les instances du groupe 2 demandent des temps de calculs moins élevés que celles des groupes 1 et 3, pour lesquels les temps sont du même ordre. Ainsi, les problèmes sans time lags sont plus faciles à résoudre que les problèmes sans attente ou les problèmes avec time lags. De plus, on peut constater que pour chaque classe d'instances, les valeurs dans le cas pondéré sont très proches de celles qu'on obtient dans le cas non pondéré. Il semblerait donc que la prise en compte de poids différent pour les machines ne rende pas les problèmes particulièrement plus difficiles à résoudre. Par ailleurs, on peut noter l'influence importante du nombre de machines m

Classe	m	n	Groupe	t_{wMC}	N_{wMC}	t_{MC}	N_{MC}
1	5	5	1	0	72	0	75
2	5	10	1	138	53722	143	53089
3	5	13	1	15452	5309537	15581	5328706
4	10	5	1	1	93	1	91
5	10	10	1	581	124208	577	122628
6	10	13	1	91615	16806744	96339	17834902
7	15	5	1	1	107	1	106
8	15	10	1	1137	172708	1177	179478
9	15	13	1	269451	35210041	276931	36402791
10	5	5	2	0	43	0	42
11	5	10	2	21	9494	17	7976
12	5	13	2	547	239429	597	257369
13	10	5	2	0	52	0	51
14	10	10	2	93	23143	89	22082
15	10	13	2	6335	1425351	5866	1314715
16	15	5	2	0	69	0	69
17	15	10	2	177	29995	187	31831
18	15	13	2	5515	807858	6269	920433
19	5	5	3	0	63	0	65
20	5	10	3	122	45875	133	50091
21	5	13	3	12316	4047957	12890	4236970
22	10	5	3	0	78	0	80
23	10	10	3	573	118696	558	115253
24	10	13	3	95509	17397392	93604	16997714
25	15	5	3	0	95	0	95
26	15	10	3	1161	173112	1166	175036
27	15	13	3	182893	23157905	189618	23878324

TAB. 4.10 – Résultats expérimentaux pour le critère $\sum w_k MC_k$

sur la difficulté des problèmes. Par exemple, le passage de 5 à 10 machines s'accompagne d'une multiplication des durées moyennes d'exécution par un facteur d'au moins 4. Ceci s'explique en partie par la structure du critère considéré, qui s'exprime comme une somme de termes relatifs aux machines.

4.6 Minimisation du plus grand retard, sans attente, avec temps de montage et démontage

Cette section concerne le problème $F2|no-wait, NSDST, NSDRT|L_{max}$ dont l'objectif est de minimiser le plus grand retard sur un flowshop à deux machines, avec des temps de montage et de démontage indépendants de la séquence et un traitement sans attente entre les deux opérations de chaque travail. Nous avons été amenés à travailler sur ce sujet à la suite de la visite d'une semaine du Professeur Allahverdi dans notre équipe de recherche, au mois d'août 2004. Etant donné que les temps de montage et démontage indépendants de la séquence, sur lesquels il a publié de nombreuses études, peuvent être représentés par des time lags, nous avons cherché à utiliser l'approche de résolution, développée dans un premier temps pour le problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, pour résoudre le problème plus particulier $F2|no-wait, NSDST, NSDRT|L_{max}$. Pour ce der-

nier problème, seuls les ordonnancements de permutation sont réalisables. Il convient de noter qu'une généralisation de ce problème, étudié dans le cadre des flowshops avec time lags, fait l'objet de la section suivante.

Dans un premier temps, nous rappelons comment procéder pour transformer les temps de montage et démontage, indépendants de la séquence et anticipatoires, en time lags.

Considérons un travail j qui doit d'abord être traité sur une machine 1 puis sur une machine 2, pendant respectivement $P_{j,1}$ et $P_{j,2}$ unités de temps, et supposons que sur chaque machine k, le traitement nécessite un temps de montage $S_{j,k}$ et un temps de démontage $R_{j,k}$. On se place dans le cas où ces temps sont anticipatoires et indépendants de la séquence. Ainsi, on ne se préoccupe pas des prédécesseurs de j sur les machines 1 et 2, mais on se contente de désigner les dates de disponibilité de ces machines par Δ_1 et Δ_2, respectivement. L'ordonnancement de j au plus tôt sur chacune des deux machines conduit aux égalités suivantes :

$C_{j,1} = \Delta_1 + S_{j,1} + P_{j,1}$
$C_{j,2} = max\{C_{j,1}, \Delta_2 + S_{j,2}\} + P_{j,2}$

Et les nouvelles dates de disponibilité des machines, après traitement du travail j, sont respectivement $C_{j,1} + R_{j,1}$ et $C_{j,2} + R_{j,2}$.

Considérons maintenant un travail i, qui doit être traité sur les machines 1 puis 2, pendant $p_{i,1} = S_{j,1} + P_{j,1} + R_{j,1}$ et $p_{i,2} = S_{j,2} + P_{j,2} + R_{j,2}$ unités de temps, tout en respectant un time lag minimal entre les deux opérations de $\theta_i^{min} = -R_{j,1} - S_{j,2}$ unités de temps. De la même manière que précédemment, les machines sont disponibles pour le traitement du travail i à partir de Δ_1 et Δ_2, respectivement. L'ordonnancement de i au plus tôt sur chacune des deux machines conduit aux égalités suivantes :

$C_{i,1} = \Delta_1 + p_{i,1} = C_{j,1} + R_{j,1}$
$C_{i,2} = max\{C_{i,1} + \theta_i^{min}, \Delta_2\} + p_{i,2} = C_{j,2} + R_{j,2}$

Et les nouvelles dates de disponibilité des machines, après traitement de i, sont respectivement $C_{i,1}$ et $C_{i,2}$.

On vérifie aisément que les deux modèles sont absolument équivalents, à la définition des dates de fin près, qui peut être différente selon que l'on considère un critère orienté travaux ou un critère orienté machines. Plus concrètement, cette transformation consiste à incorporer les temps de montage et de démontage dans la durée opératoire et à définir correctement le time lag minimal autorisant le recouvrement partiel afin de garantir la précédence entre les parties correspondant au traitement réel. Il faut de plus redéfinir les dates de fin des travaux en tenant compte des temps de démontage. Les deux modèles sont représentés sur la figure 4.10.

On retrouve une telle transformation dans plusieurs articles (voir, par exemple, [Szwarc,83], [Baker,90], [Botta et Guinet,96], [Espinouse et al.,00]).

Toutefois, les problèmes d'ordonnancement avec temps de montage et démontage sont généralement étudiés indépendemment des time lags, bien qu'ils fassent l'objet de nombreux travaux dans la littérature. Pour un aperçu de l'état de l'art sur ces problèmes, nous renvoyons le lecteur à l'article d'Allahverdi et al. [Allahverdi et al.,99]. Aldowaisan et Allahverdi [Aldowaisan et Allahverdi,98] considèrent un problème assez proche de celui qui nous intéresse :

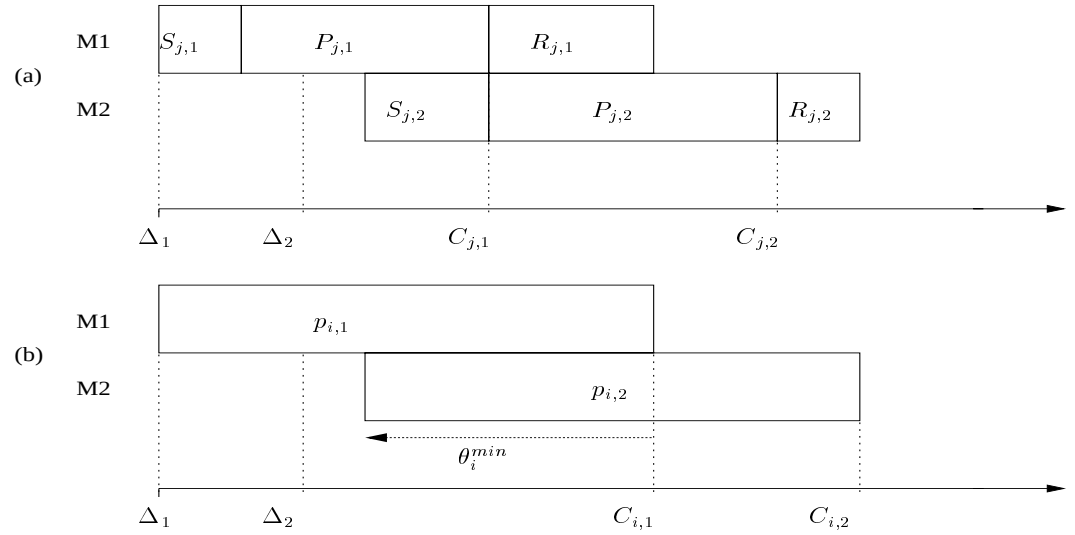

FIG. 4.10 – Représentation des temps de montage et démontage à l'aide d'un time lag minimal

celui de la minimisation de la somme des dates de fin des travaux dans un flowshop à deux machines sans attente avec temps de montage indépendant de la séquence. Ils présentent des cas particuliers polynomiaux et une relation de dominance "locale", c'est-à-dire une condition sur les couples de travaux placés consécutivement. Une heuristique est également décrite. Dileepan [Dileepan,04] s'intéresse au même problème mais avec pour objectif de minimiser le plus grand retard et établit des relations de dominance. D'autres extensions au travail sont proposées par Allahverdi et Aldowaisan [Allahverdi et Aldowaisan,00], [Allahverdi et Aldowaisan,01] : problème à trois machines, prise en compte de temps de montage dépendant de la séquence.

Puisque le problème sans temps de montage et démontage $F2|no-wait|L_{max}$ estNP-difficile au sens fort [Roeck,84b], notre problème l'est également. Avant de décrire l'approche de résolution adoptée, nous présentons deux cas particuliers polynomiaux, puis nous détaillons les caractéristiques de la Procédure par Séparation et Evaluation que nous avons développée. La section se termine avec les résultats obtenus suite aux expériences que nous avons menées.

4.6.1 Cas particuliers polynomiaux

Soient $S_{j,k}$, $R_{j,k}$ et $P_{j,k}$ les temps de montage, de démontage et de traitement du travail j sur la machine k, et soient C_j, D_j et L_j la date de fin sur la deuxième machine, la date de fin souhaitée et le retard de ce travail.

Théorème. 20 *Soit une instance du problème* $F2|no-wait, NSDST, NSDRT|L_{max}$ *telle que* $\forall j, R_{j,2} \leq R_{j,1} - P_{j,2}$ *et* $S_{j,2} \leq S_{j,1} + P_{j,1}$. *Une séquence optimale est obtenue en traitant les travaux dans l'ordre croissant de* $R_{j,1} - P_{j,2} + D_j$.

Preuve. D'après Allahverdi et Aldowaisan [Allahverdi et Aldowaisan,01], la date de fin du travail placé en i-ième position dans la séquence de traitement π s'exprime :

$C_{\pi(i)} = \sum_{1\leq h\leq i} max\{R_{\pi(h-1),2}+S_{\pi(h),2}, R_{\pi(h-1),1}+S_{\pi(h),1}+P_{\pi(h),1}-P_{\pi(h-1),2}\}+\sum_{1\leq h\leq i} P_{\pi(h),2}$, avec $P_{\pi(0),2} = R_{\pi(0),1} = R_{\pi(0),2} = 0$.

D'après les hypothèses du théorème, $C_{\pi(i)} = \sum_{1\leq h\leq i}(R_{\pi(h-1),1}+S_{\pi(h),1}+P_{\pi(h),1}-P_{\pi(h-1),2}\})+\sum_{1\leq h\leq i} P_{\pi(h),2}$.

D'autre part, $L_{\pi(i)} = C_{\pi(i)} - D_{\pi(i)}$.

Considérons deux séquences π_1 et π_2 telles que π_1 et π_2 sont identiques sauf aux positions τ et $\tau+1$. Plus formellement, $\forall h \notin \tau, \tau+1, \pi_1(h) = \pi_2(h)$, $\pi_1(\tau) = i = \pi_2(\tau+1)$ et $\pi_1(\tau+1) = j = \pi_2(\tau)$. Supposons que $R_{j,1} - P_{j,2} + D_j \leq R_{i,1} - P_{i,2} + D_i$.

Puisque les séquences sont identiques jusqu'à la position τ, on a :

$\forall h < \tau, L_{\pi_1(h)}(\pi_1) = L_{\pi_2(h)}(\pi_2)$.

Soit $A = \sum_{1\leq h\leq \tau-1}(R_{\pi_1(h-1),1} + S_{\pi_1(h),1} + P_{\pi_1(h),1} - P_{\pi_1(h-1),2}) + \sum_{1\leq h\leq \tau-1} P_{\pi(h),2}$.

$L_{\pi_2(\tau)}(\pi_2) = L_j(\pi_2) = A + R_{\pi_2(\tau-1),1} + S_{j,1} + P_{j,1} - P_{\pi_2(\tau-1),2} + P_{j,2} - D_j$.

$L_{\pi_1(\tau+1)}(\pi_1) = L_j(\pi_1) = A + R_{\pi_1(\tau-1),1} + S_{i,1} + P_{i,1} - P_{\pi_1(\tau-1),2} + R_{i,1} + S_{j,1} + P_{j,1} - P_{i,2} + P_{i,2} + P_{j,2} - D_j$.

D'où : $L_{\pi_1(\tau+1)}(\pi_1) - L_{\pi_2(\tau)}(\pi_2) = R_{\pi_1(\tau-1),1} + S_{i,1} + P_{i,1} - P_{\pi_1(\tau-1),2} + R_{i,1} + S_{j,1} + P_{j,1} - P_{i,2} - (R_{\pi_2(\tau-1),1} + S_{j,1} + P_{j,1} - P_{\pi_2(\tau-1),2}) + P_{i,2} = S_{i,1} + P_{i,1} + R_{i,1} \geq 0$.

D'autre part, $L_{\pi_2(\tau+1)}(\pi_2) = L_i(\pi_2) = A + R_{\pi_2(\tau-1),1} + S_{j,1} + P_{j,1} - P_{\pi_2(\tau-1),2} + R_{j,1} + S_{i,1} + P_{i,1} - P_{j,2} + P_{j,2} + P_{i,2} - D_i$.

D'où : $L_{\pi_1(\tau+1)}(\pi_1) - L_{\pi_2(\tau+1)}(\pi_2) = R_{\pi_1(\tau-1),1} + S_{i,1} + P_{i,1} - P_{\pi_1(\tau-1),2} + R_{i,1} + S_{j,1} + P_{j,1} - P_{i,2} - (R_{\pi_2(\tau-1),1} + S_{j,1} + P_{j,1} - P_{\pi_2(\tau-1),2}) - (R_{j,1} + S_{i,1} + P_{i,1} - P_{j,2}) + D_i - D_j = R_{i,1} - P_{i,2} - R_{j,1} + P_{j,2} + D_i - D_j \geq 0$ en raison de l'hypothèse faite sur i et j.

Enfin, pour $h \geq \tau+1$, on a : $L_{\pi_1(h)}(\pi_1) = A + R_{\pi_1(\tau-1),1} + S_{i,1} + P_{i,1} - P_{\pi_1(\tau-1),2} + R_{i,1} + S_{j,1} + P_{j,1} - P_{i,2} + R_{j,1} + S_{\pi_1(\tau+2),1} + P_{\pi_1(\tau+2),1} - P_{j,2} + \sum_{\tau+3\leq g\leq h}(R_{\pi_1(g-1),1} + S_{\pi_1(g),1} + P_{\pi_1(g),1} - P_{\pi_1(g-1),2}) + P_{i,2} + P_{j,2} + \sum_{\tau+3\leq g\leq h} P_{\pi_1(g),2} - D_{\pi_1(h)}$.

De même, $L_{\pi_2(h)}(\pi_2) = A + R_{\pi_2(\tau-1),1} + S_{j,1} + P_{j,1} - P_{\pi_2(\tau-1),2} + R_{j,1} + S_{i,1} + P_{i,1} - P_{j,2} + R_{i,1} + S_{\pi_2(\tau+2),1} + P_{\pi_2(\tau+2),1} - P_{i,2} + \sum_{\tau+3\leq g\leq h}(R_{\pi_2(g-1),1} + S_{\pi_2(g),1} + P_{\pi_2(g),1} - P_{\pi_2(g-1),2}) + P_{j,2} + P_{i,2} + \sum_{\tau+3\leq g\leq h} P_{\pi_2(g),2} - D_{\pi_2(h)}$.

Puisque π_1 et π_2 sont identiques sauf pour les positions τ et $\tau+1$, $L_{\pi_1(h)}(\pi_1) = L_{\pi_2(h)}(\pi_2)$.

On déduit de tout ce qui précède que $L_{max}(\pi_2) \leq L_{max}(\pi_1)$. On peut donc utiliser un argument d'échange de travaux consécutifs pour prouver que l'ordre croissant de $R_{j,1} - P_{j,2} + D_j$ conduit à une solution optimale.□

.

La situation considérée dans le théorème précédent correspond au cas où la deuxième machine est en quelque sorte "dominée" par la première : pour chaque travail j, le temps de démontage $R_{j,1}$ sur la première machine est plus long que le temps de traitement plus le temps de démontage

sur la seconde $P_{j,2} + R_{j,2}$, et le temps de montage $S_{j,2}$ sur la deuxième machine est plus court que le temps de montage plus le temps de traitement sur la première machine $S_{j,1} + P_{j,1}$. En considérant la situation inverse, on obtient le théorème suivant :

Théorème. 21 *Soit une instance du problème* $F2|no-wait, NSDST, NSDRT|L_{max}$ *telle que* $\forall j, R_{j,2} \geq R_{j,1} - P_{j,2}$ *et* $S_{j,2} \geq S_{j,1} + P_{j,1}$. *Une séquence optimale est obtenue en traitant les travaux dans l'ordre croissant de* $R_{j,2} + D_j$.

Preuve. La preuve est similaire à celle du théorème précédent.□

4.6.2 Procédure par Séparation et Evaluation pour le cas général

Comme nous l'avons montré dans la section précédente, les contraintes de temps de montage et de démontage peuvent être modélisées à l'aide de time lags. On peut facilement vérifier que ceci reste vrai si les opérations des travaux doivent être réalisées sans attente. D'autre part, du fait de cette contrainte sans attente, seuls les ordonnancements de permutation sont réalisables. Le fait de limiter la recherche à ces ordonnancements n'est donc pas restrictif. Il s'en suit que le schéma de séparation générique que nous proposons pour les problèmes du type $Fm_{\pi}|\theta_{j,k}^{min}, \theta_{j,k}^{max}|\gamma$, où γ est un critère régulier, reste valable pour le problème $F2|no-wait, NSDST, NSDRT|L_{max}$ qui nous intéresse ici. On peut également s'assurer que ce schéma, basé sur la construction progressive de la séquence de traitement par ajout de travaux en fin de séquence partielle, peut être utilisé en remarquant que la date de fin $C_{\pi(i)}$ et le retard $L_{\pi(i)}$ du travail placé en position i dans la séquence ne dépend que des travaux précédents. La stratégie d'exploration est elle aussi de type MFP (voir la section 4.3.3).

En ce qui concerne la borne inférieure, plutôt que d'essayer de se ramener à un problème avec time lags et minimisation du makespan, nous employons une borne spécifique notée LB_1^3. Supposons que π_i désigne une séquence partielle de i travaux ($1 \leq i \leq n-2$). Pour ces travaux déjà placés, les dates de fin et les retards sont connus et peuvent être déterminés grâce à la formule d'Allahverdi et Aldowaisan [Allahverdi et Aldowaisan,01]. Soit π une permutation complète des n travaux commençant par π_i. On a :

$C_{\pi(h)} = \sum_{1\leq g\leq h} max\{0, R_{\pi(g-1),1} + S_{\pi(g),1} + P_{\pi(g),1} - P_{\pi(g-1),2} - R_{\pi(g-1),2} - S_{\pi(g),2}\} + \sum_{1\leq g\leq h} S_{\pi(g),2} + \sum_{1\leq g\leq h} P_{\pi(g),2} + \sum_{1\leq g\leq h} R_{\pi(g-1),2}$.

Pour $h > i$, on peut exprimer la date de fin de la manière suivante :

$C_{\pi(h)} = C_{\pi(i)} + R_{\pi(i),2} + \sum_{i+1\leq g\leq h} max\{0, R_{\pi(g-1),1} + S_{\pi(g),1} + P_{\pi(g),1} - P_{\pi(g-1),2} - R_{\pi(g-1),2} - S_{\pi(g),2}\} + \sum_{i+1\leq g\leq h} S_{\pi(g),2} + \sum_{i+1\leq g\leq h} P_{\pi(g),2} + \sum_{i+1\leq g\leq h} R_{\pi(g),2} - R_{\pi(h),2}$.

Si C'_j désigne la date à laquelle se termine le temps de démontage du travail j (date de fin du travail en incorporant le temps de démontage dans le temps de traitement), on obtient :

$C'_{\pi(h)} \geq C'_{\pi(i)} + \sum_{i+1\leq g\leq h} S_{\pi(g),2} + \sum_{i+1\leq g\leq h} P_{\pi(g),2} + \sum_{i+1\leq g\leq h} R_{\pi(g),2}$.

Une borne inférieure sur $C'_{\pi(h)}$, notée $BI(h)$, peut être calculée en ordonnant les travaux qui n'ont pas encore été placés dans l'ordre croissant de la somme des durées de montage, de traitement et de démontage sur la deuxième machine $S_{j,2} + P_{j,2} + R_{j,2}$. Par ailleurs, puisque

$L_{\pi(h)} = C'_{\pi(h)} - R_{\pi(h),2} - D_{\pi(h)}$, le plus grand retard vérifie $L_{max} \geq max\{max\{L_{\pi_i(f)}/1 \leq f \leq i\}, max\{BI(h) - R_{\pi(h),2} - D_{\pi(h)}/i+1 \leq h \leq n\}\}$.

Soit ϕ une séquence des $n-i$ travaux non placés telle que $R_{\phi(1),2} + D_{\phi(1)} \leq R_{\phi(2),2} + D_{\phi(2)} \leq \ldots \leq R_{\phi(n-i),2} + D_{\phi(n-i)}$. Etant donné que $BI(i+1) \leq BI(i+2) \leq \ldots \leq BI(n)$, une borne inférieure pour le plus grand retard est obtenue en associant à chaque valeur $BI(h)$ la valeur $R_{\phi(h-i),2} + D_{\phi(h-i)}$ et en gardant le maximum :

$L_{max} \geq LB_1^3 = max\{max\{L_{\pi_i(f)}/1 \leq f \leq i\}, max\{BI(h) - R_{\phi(h-i),2} + D_{\phi(h-i)}/i+1 \leq h \leq n\}\}$.

Nous avons pris en compte une relation de dominance additionnelle, pour les problèmes sans temps de démontage, fournie par Dileepan [Dileepan,04]. Celle-ci fonctionne de la façon suivante. Soit O_1 un ordonnancement dans lequel le travail j est le prédécesseur immédiat du travail i, et soit O_2 l'ordonnancement obtenu à partir de O_1 lorsque l'on intervertit les travaux i et j. Si l'une des deux conditions suivantes est satisfaite, alors $L_{max}(O_2) \leq L_{max}(O_1)$:

Premier cas $S_{i,2} \geq S_{i,1} + P_{i,1} - min\{P_{u,2}\}$, $S_{j,2} \geq S_{j,1} + P_{j,1} - min\{P_{u,2}\}$, $D_i \leq D_j$ et $P_{i,2} \leq P_{j,2}$

Deuxième cas $S_{i,1} + P_{i,1} - S_{i,2} \geq max\{P_{u,2}\}$, $S_{j,1} + P_{j,1} - S_{j,2} \geq max\{P_{u,2}\}$, $D_i - P_{i,2} \leq D_j - P_{j,2}$ et $P_{j,2} \leq P_{i,2}$

Le principal avantage de ces ensembles de conditions provient du fait qu'elles ne dépendent que des travaux i et j, et pas du reste de l'ordonnancement. Ainsi, les tests peuvent être effectués avant l'exécution de la PSE, dans une phase de pré-traitement. En outre, de telles conditions peuvent tout-à-fait correspondre à des situations rencontrées dans la pratique.

Nous terminons la présentation de notre PSE par la description des heuristiques utilisées en amont afin d'obtenir des bornes supérieures de bonne qualité. Il s'agit de trois méthodes issues de NEH [Nawaz et al.,83] (voir Section 4.3.5). Elles sont désignées par $NEH^3_{\sum(S+P+R)}$, $NEH^3_{\sum(S+P+R)}[5]$ et $NEH^3_{R_2+D}[5]$. La première ne fait appel qu'à une itération, alors que les deux autres utilisent 5 itérations (indiqué par le nombre entre crochets). Les listes initiales sont obtenues en classant les travaux dans l'ordre décroissant de leur durée de traitement totale, en y incluant les temps de montage et de démontage ($S_{j,1}+P_{j,1}+R_{j,1}+S_{j,2}+P_{j,2}+R_{j,2}$) pour les deux premières et de leur date de fin souhaitée, en y incluant le temps de démontage ($R_{j,2}+D_j$) pour la dernière. Evidemment, $NEH^3_{\sum(S+P+R)}[5]$ domine $NEH^3_{\sum(S+P+R)}$ puisque la solution fournie par cette dernière est considérée dans $NEH^3_{\sum(S+P+R)}[5]$. Cependant, il peut être intéressant de comparer la meilleure solution retenue par $NEH^3_{\sum(S+P+R)}[5]$ au bout de 5 itérations avec celle de $NEH^3_{\sum(S+P+R)}$, obtenue en une seule itération.

4.6.3 Expériences numériques menées

Nous avons généré plusieurs classes de jeux d'essais sur lesquels notre PSE a été testée. Les paramètres retenus dans la génération sont les suivants :

- n : le nombre de travaux
- pg : le pourcentage de travaux appartenant au premier groupe. Les travaux sont divisés en deux groupes. Ceux qui appartiennent au premier ont des durées opératoires $P_{j,k}$ dans l'intervalle [1, 100] tandis que les autres ont des durées opératoires comprises entre 40 et 60

- r_S : l'amplitude des temps de montage. Les temps de montage sont compris entre 0 et $r_S \times P_{max}$, où P_{max} désigne la durée maximale (100 pour le premier groupe, 60 pour le second)
- r_R : l'amplitude des temps de démontage, dont la définition est analogue à celle de r_S
- R : l'amplitude des dates de fin souhaitées
- T : le facteur de retard

Les dates de fin souhaitées sont générées dans l'intervalle $[B \times x, B \times y]$ en suivant la méthode de Potts et Van Wassenhove [Potts et Van Wassenhove,82], où B est une borne inférieure pour le makespan. Les paramètres x et y sont définis par : $x = (1 - T - R/2)$ et $y = (1 - T + R/2)$. Une telle méthode pour la génération de dates de fin souhaitées est couramment utilisée dans la littérature (voir par exemple Volgenant et Teerhuis [Volgenant et Teerhuis,99], Szwarc et al. [Szwarc et al,99], Kim [Kim,93] [Kim,95], et Chu [Chu, 92]). Toutes les données sont des entiers générés selon des distributions uniformes discrètes.

Le tableau 4.11 présente les valeurs des paramètres utilisés pour générer les 9 classes d'instances testées. Les algorithmes ont été implémentés en langage C et les programmes ont été exécutés sur un PC muni d'un Pentium à 1,2 GHz.

Classe	n	pg	r_S	r_R	T	R
1	16	50%	0.5	0.5	0.6	0.75
2	16	50%	0.5	0.5	0.6	0.6
3	16	50%	2	2	0.6	0.75
4	16	50%	2	0.5	0.6	0.75
5	16	0%	0.5	0.5	0.6	0.75
6	16	100%	0.5	0.5	0.6	0.75
7	18	50%	0.5	0.5	0.6	0.75
8	16	50%	0.5	0	0.6	0.75
9	16	50%	2	0	0.6	0.75

TAB. 4.11 – Caractérisation des classes d'instances utilisées

Pour chaque classe, nous avons généré 10 instances et nous avons comparé le plus grand retard pour les solutions fournies par les heuristiques et la PSE. Le tableau 4.12 montre l'écart relatif moyen entre la solution de chaque heuristique et l'optimum.

Classe	$NEH^3_{\sum(S+P+R)}$	$NEH^3_{\sum(S+P+R)}$[5]	$NEH^3_{R_2+D}$[5]
1	13.5%	8.6%	11.7%
2	9.4%	6.3%	9.2%
3	11.0%	2.2%	5.6%
4	15.5%	5.1%	5.2%
5	4.1%	3.3%	5.0%
6	15.0%	10.9%	11.5%
7	24.8%	12.9%	9.4%
8	11.9%	8.0%	11.5%
9	17.5%	3.3%	4.2%

TAB. 4.12 – Ecart relatif moyen entre la valeur fournie par chaque heuristique et l'optimum

Comme on peut le constater, $NEH^3_{\sum(S+P+R)}[5]$ est plus performante que les deux autres heuristiques en moyenne, pour toutes les classes sauf la classe 7. Plus précisément, en comparant $NEH^3_{\sum(S+P+R)}$ et $NEH^3_{\sum(S+P+R)}[5]$, on peut remarquer qu'une utilisation itérative de NEH peut améliorer la qualité de la solution de façon significative. Etant donné que les temps d'exécution pour ces heuristiques sur les instances générées sont très courts, nous ne les présentons pas.

En ce qui concerne l'évaluation de la PSE proposée, nous avons fixé une limite de temps de 1200 secondes. Le tableau 4.13 fournit les résultats suivants pour chaque classe de jeux d'essais :

- le nombre N de problèmes pour lesquels la PSE a atteint la limite de temps
- le temps de calcul moyen t_{opt} (en secondes) pour les problèmes résolus optimalement avant la limite
- l'écart relatif moyen ϵ entre la meilleure valeur obtenue dans la limite de temps et l'optimum (obtenu sans limite de temps), dans le cas où $N \neq 0$; si $N = 0$, la colonne est sans objet, ce qui est représenté par le symbole "/
- le temps de calcul moyen $t_{\bar{o}pt}$ (en secondes) pour les problèmes qui n'ont pas pu être résolus dans la limite de temps, dans le cas où $N \neq 0$

Classe	N	t_{opt}	ϵ	$t_{\bar{o}pt}$
1	1	107.1	0%	1704
2	1	171.6	0%	2373
3	2	121.6	0.51%	2633
4	4	251.7	0%	2123
5	1	59.7	0%	1856
6	0	99.7	/	/
7	1	88.2	0%	4610
8	0	166.4	/	/
9	3	182.5	0%	2837

TAB. 4.13 – Performance de la PSE

D'après le tableau 4.13, la plupart des classes d'instances sont traitées de manière satisfaisante par la PSE : mis à part les classes 4 et 9, le nombre de problèmes pour lesquels l'algorithme nécessite plus de 1200 secondes ne dépasse pas 2, et le temps d'exécution moyen est inférieur à 180 secondes. Les deux dernières colonnes indiquent que pour les problèmes qui ne peuvent pas être résolus dans la limite de temps, la solution optimale est en fait atteinte avant cette limite et le temps additionnel, qui est parfois considérable, sert uniquement à prouver l'optimalité. Ainsi, il peut être très utile de fixer une limite de temps pour notre PSE.

Comme nous l'avons mentionné auparavant, les instances issues des classes 4 et 9 sont relativement plus difficiles à résoudre que les autres. Ceci est peut-être dû à la taille des temps de montage par rapport à celle des durées opératoires et des temps de démontage ($r_S = 2$ et $r_R = 0,5$ ou 0).

Pour les classes 8 et 9, tous les temps de démontage sont nuls. Il s'agit donc de problèmes du type $F2|no-wait, NSDST|L_{max}$, pour lesquels on peut faire appel aux relations de dominance de Dileepan [Dileepan,04]. Le tableau 4.14 fournit les valeurs de t_{opt} (en secondes), défini précédemment, lorsque les tests de dominance sont effectués ou pas, ainsi que le gain moyen en temps d'exécution G_t et le nombre moyen J de couples de travaux pour lesquels la relation de

dominance est vérifiée.

Classe	t_{opt} sans dominance	t_{opt} avec dominance	G_t	J
8	166.4	153.6	10%	3.4
9	182.5	139.7	14%	7.7

TAB. 4.14 – Gain obtenu grâce aux relations de dominance

Comme l'illustre le tableau 4.14, les relations de dominance de Dileepan [Dileepan,04] sont relativement efficaces puisqu'elles conduisent à une réduction significative du temps d'exécution de la PSE (10% pour la classe 8 et 14% pour la classe 9). Cette réduction peut atteindre près de 50% sur certaines instances.

4.7 Problème de minimisation du plus grand retard avec time lags exacts

Le problème étudié dans cette section est celui de la minimisation du plus grand retard dans un flowshop de permutation à m machines avec des time lags exacts, c'est-à-dire tels qu'entre chaque couple d'opérations consécutives au sein des travaux, le time lag minimal est égal au time lag maximal. Ce problème, noté $Fm_\pi|\theta_{j,k}^{min} = \theta_{j,k}^{max}|L_{max}$ généralise plusieurs problèmes :

1- le problème sans attente $Fm|no-wait|L_{max}$, pour lequel les time lags sont tous nuls (dans ce cas, les ordonnancements de permutation sont dominants)

2- le problème à deux machines sans attente avec temps de montage et démontage indépendants de la séquence $F2|no-wait, NSDST, NSDRT|L_{max}$ qui fait l'objet de la section précédente. On rappelle que toute instance de ce dernier problème peut être transformée en une instance du problème avec time lags exacts en définissant pour chaque travail j la durée opératoire sur la machine k, le time lag entre les machines k et $k+1$ et la date de fin souhaitée de la manière suivante : $p_{j,k} = S_{j,k} + P_{j,k} + R_{j,k}$, $\theta_{j,k} = -R_{j,k} - S_{j,k+1}$ et $d_j = D_j + R_{j,m}$ respectivement, où $S_{j,k}, P_{j,k}, R_{j,k}$ et D_j sont les données du problème $F2|no-wait, NSDST, NSDRT|L_{max}$ telles qu'elles sont définies dans la section précédente.

Il convient de noter que dès qu'il existe au moins un time lag positif, les ordonnancements de permutation ne sont plus dominants, même pour deux machines (voir la section 2.6). Néanmoins, nous nous limitons à ces ordonnancements, comme nous l'avons fait jusqu'à maintenant. Dans cette section, nous généralisons l'ensemble des résultats et des méthodes de résolution présentés précédemment, en nous plaçant dans le cadre des problèmes avec time lags. A notre connaissance, aucune approche de résolution n'a été proposée jusqu'à maintenant pour le problème général que nous considérons ici.

4.7.1 Définitions

Avant toute chose, nous introduisons dans cette partie des définitions qui nous permettent de présenter un certain nombre de résultats plus facilement.

Definition

Le retard L_j de chaque travail j peut s'exprimer en fonction de la date de fin $C_{j,k}$ sur n'importe quelle machine k de la manière suivante : $L_j = C_{j,k} - d'_{j,k}$, où $d'_{j,k} = d_j - \sum_{k \leq q \leq m-1}(\theta_{j,q} + p_{j,q+1})$

est la date de fin souhaitée pour le travail j sur la machine k.

Preuve. Le retard du travail j est défini par $L_j = C_{j,m} - d_j$. En raison des contraintes de time lags exacts, la date de fin de j sur toute machine q peut être déterminée à partir de la date de fin sur la machine suivante : $C_{j,q} = C_{j,q+1} - p_{j,q+1} - \theta_{j,q}$. Par récurrence, on obtient $C_{j,k} = C_{j,m} - \sum_{k \leq q \leq m-1}(\theta_{j,q} + p_{j,q+1})$, ce qui conduit à la formule fournie (voir figure 4.11).□

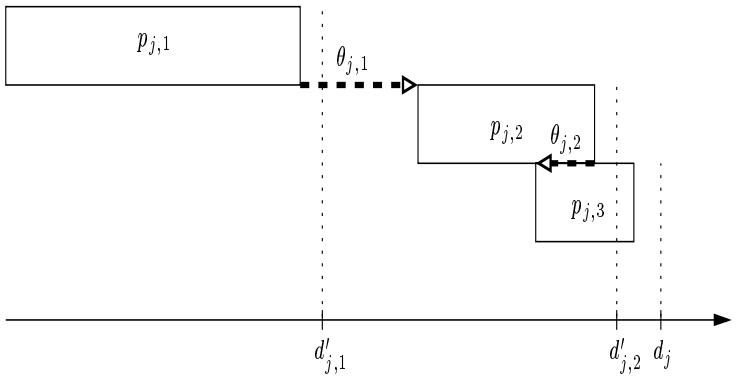

FIG. 4.11 – Dates de fin souhaitées sur les machines

En fonction de la valeur de chaque time lag, nous distinguons trois types de travaux :

– les travaux *totalement couvrants*, pour lesquels il existe une machine k telle que la période de traitement sur n'importe quelle autre machine est incluse dans la période de traitement sur la machine k : $\forall 1 \leq q \leq m, C_{j,q} - p_{j,q} \geq C_{j,k} - p_{j,k}$ et $C_{j,q} \leq C_{j,k}$ (voir figure 4.12). Suivant l'indice k de la machine, un tel travail est appelé k-totalement couvrant

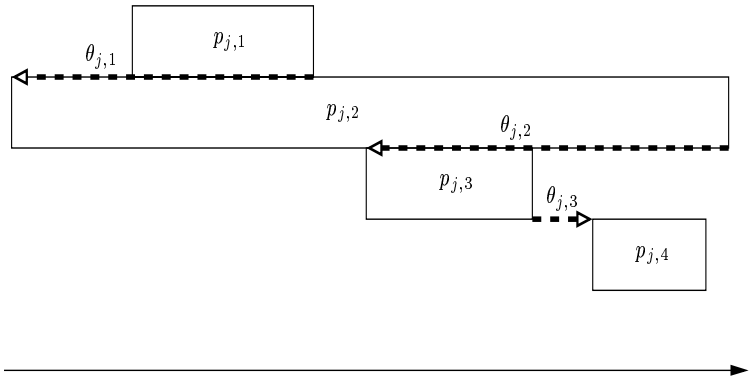

FIG. 4.12 – Travail totalement couvrant

– les travaux *non couvrants*, pour lesquels les périodes de traitement sur les machines sont disjointes deux à deux. Cette situation correspond au cas où les time lags sont tous positifs ou nuls (pas de recouvrement) : $\forall 1 \leq q \leq m-1, \theta_{j,q} \geq 0$ (voir figure 4.13)
– les travaux *partiellement couvrants*, qui n'appartiennent pas aux deux catégories précédentes (voir figure 4.14)

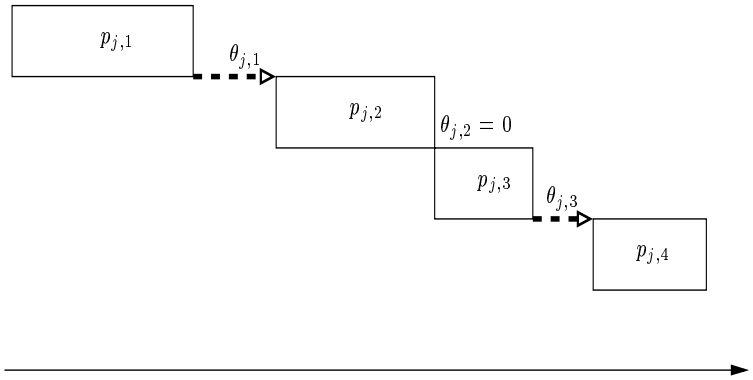

FIG. 4.13 – Travail non couvrant

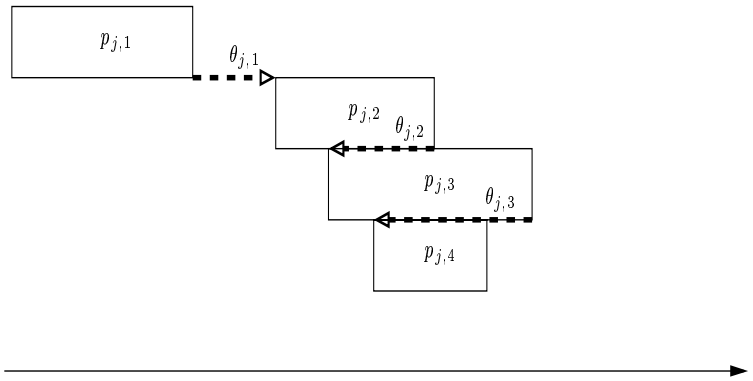

FIG. 4.14 – Travail partiellement couvrant

4.7.2 Cas particuliers polynomiaux

Le théorème suivant généralise les cas particuliers correspondant aux hypothèses des théorèmes 20 et 21 de la section précédente.

Théorème. 22 *Si, pour une machine donnée k, tous les travaux sont k-totalement couvrants, alors un ordonnancement optimal est obtenu en appliquant la règle EDD sur les dates de fin souhaitées $d'_{j,k}$ sur la machine k.*

Preuve. Supposons que tous les travaux sont k-totalement couvrants et considérons un ordonnancement arbitraire où la séquence de traitement des travaux sur la machine k est donnée par $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. Par définition des travaux k-totalement couvrants, la plus petite date de début pour le premier travail placé $\pi(1)$ correspond à la machine k, de même que la plus grande date de fin. Ainsi, $\pi(1)$ commence sur la machine k à la date 0 et se termine sur cette machine à la date $C_{\pi(1),k} = p_{\pi(1),k}$. Le traitement sur les autres machines est imposé par les time lags exacts. De manière analogue, le deuxième travail $\pi(2)$ est traité sur la machine k entre $C_{\pi(1),k}$ et $C_{\pi(1),k} + p_{\pi(2),k}$, tandis que les périodes de traitement sur les autres machines sont incluses dans cet intervalle. Plus généralement, le i-ième travail $\pi(i)$ est ordonnancé sur la machine k entre $\sum_{1 \leq h \leq i-1} p_{\pi(h),k}$ et $\sum_{1 \leq h \leq i} p_{\pi(h),k}$. Par conséquent, le problème est équivalent au problème à une machine $1||L_{max}$ avec des durées opératoires $p_{j,k}$ et des dates de fin souhaitées $d'_{j,k}$. Il est connu que pour ce problème, la règle EDD fournit une solution optimale [Jackson,55].□

En utilisant la transformation du problème $F2|no-wait, NSDST, NSDRT|L_{max}$ en problème avec time lags, on peut constater que les théorèmes 20 et 21 sont des cas particuliers du théorème précédent lorsque $m = 2$. Ils correspondent aux cas où tous les travaux sont 1-totalement couvrants et 2-totalement couvrants respectivement.

4.7.3 Relations de dominance pour les problèmes à deux machines

Nous proposons une extension des relations de dominance de Dileepan [Dileepan,04] prévues initialement pour le problème $F2|no-wait, NSDST|L_{max}$ (voir la section 4.6.2). Soient une séquence $\alpha = (\sigma_1, i, j, \sigma_2)$ dans laquelle le travail i précède directement le travail j, et une séquence $\beta = (\sigma_1, j, i, \sigma_2)$ identique à α, mis à part que j précède directement i (σ_1, σ_2 désignent des séquences partielles). En suivant les notations pour les problèmes avec time lags, les conditions de Dileepan [Dileepan,04] s'expriment de la façon suivante :

Propriété. 10 *[Dileepan,04]*

- *Cas A : si*
  - $p_{i,1} + \theta_{i,1} \leq min_{1 \leq u \leq n}\{p_{u,2} + \theta_{u,1}\}$,
  - $p_{j,1} + \theta_{j,1} \leq min_{1 \leq u \leq n}\{p_{u,2} + \theta_{u,1}\}$,
  - $p_{i,2} + \theta_{i,1} \leq p_{j,2} + \theta_{j,1}$,
  - *et* $d_i \leq d_j$,

  alors la solution α *domine la solution* β.
- *Cas B : si*
  - $p_{i,1} + \theta_{i,1} \geq max_{1 \leq u \leq n}\{p_{u,2} + \theta_{u,1}\}$,
  - $p_{j,1} + \theta_{j,1} \geq max_{1 \leq u \leq n}\{p_{u,2} + \theta_{u,1}\}$,
  - $p_{j,2} + \theta_{j,1} \leq p_{i,2} + \theta_{i,1}$,
  - *et* $d'_{i,1} \leq d'_{j,1}$,

  alors la solution α *domine la solution* β.

Les idées générales utilisées pour établir ce résultat peuvent s'exprimer ainsi :

- Cas A : si
  - dans la solution α il n'y a pas de temps mort sur la machine deux entre la fin de σ_1 et la fin de j,
  - dans la solution β il n'y a pas de temps mort sur la machine deux entre la fin de σ_1 et la fin de i,
  - la machine un est disponible plus tôt après j dans la solution α qu'après i dans la solution β,
  - et i a une plus petite date de fin souhaitée que j sur la machine deux,

  alors la solution α domine la solution β.
- Cas B : si
  - dans la solution α il n'y a pas de temps mort sur la machine un entre la fin de σ_1 et la fin de j,
  - dans la solution β il n'y a pas de temps mort sur la machine un entre la fin de σ_1 et la fin de i,
  - la machine deux est disponible plus tôt après j dans la solution α qu'après i dans la solution β,
  - et i a une plus petite date de fin souhaitée que j sur la machine un,

  alors la solution α domine la solution β.

Dans chaque cas de la propriété 10, les deux premières conditions sont suffisantes pour éviter les temps morts. Néanmoins, il est possible de fournir des conditions moins restrictives

pour lesquelles le résultat de dominance est toujours valable. Ces conditions sont vérifiées par un plus grand nombre d'instances que les précédentes. Soit x le dernier travail de la séquence partielle σ_1 (si σ_1 est vide, on pose $p_{x,1} = p_{x,2} = \theta_{x,1} = 0$). Les nouvelles conditions sont les suivantes :

Propriété. 11 – *Cas A' : si*
- $p_{i,1} + \theta_{i,1} \leq min\{p_{x,2} + \theta_{x,1}, p_{j,2} + \theta_{j,1}\}$,
- $p_{j,1} + \theta_{j,1} \leq min\{p_{x,2} + \theta_{x,1}, p_{i,2} + \theta_{i,1}\}$,
- $p_{i,2} + \theta_{i,1} \leq p_{j,2} + \theta_{j,1}$,
- *et* $d_i \leq d_j$,

alors la solution α domine la solution β.

– *Cas B' : si*
- $p_{i,1} + \theta_{i,1} \geq max\{p_{x,2} + \theta_{x,1}, p_{j,2} + \theta_{j,1}\}$,
- $p_{j,1} + \theta_{j,1} \geq max\{p_{x,2} + \theta_{x,1}, p_{i,2} + \theta_{i,1}\}$,
- $p_{j,2} + \theta_{j,1} \leq p_{i,2} + \theta_{i,1}$,
- *et* $d'_{i,1} \leq d'_{j,1}$,

alors la solution α domine la solution β.

Une preuve similaire à celle qui est utilisée par Dileepan [Dileepan,04] permet de démontrer que dans le cas A' ou dans le cas B', la solution α domine la solution β.

Il est possible de généraliser ce résultat à des problèmes comptant un nombre arbitraire m de machines, mais lorsque m augmente, les conditions deviennent de plus en plus complexes et restrictives.

4.7.4 Approche de résolution proposée

Nous avons développé une Procédure par Séparation et Evaluation pour résoudre le problème considéré de manière optimale. Nous employons le schéma de séparation générique issu d'Ignall et Schrage [Ignall et Schrage,65] et la stratégie d'exploration MFP (voir section 4.3.3).

La borne inférieure est calculée à partir de la règle EDD. Supposons que la séquence partielle initiale soit σ_i et comporte i travaux. De la même manière que pour le problème précédent, les dates de fin et les retards de ces travaux sont déterminés. Nous définissons dans un premier temps m bornes inférieures $LB_1^4, LB_2^4, \ldots, LB_m^4$ où LB_k^4 procède ainsi : pour les travaux qui n'ont pas encore été placés, on ne tient compte que du traitement sur la machine k, en relaxant les contraintes de capacité sur toutes les autres machines et en acceptant éventuellement que les opérations sur ces machines débutent avant la date 0. Comme nous l'avons vu dans la section 4.7.1, le retard L_j de chaque travail j peut être calculé à partir de la date de fin sur la machine k et de la date de fin souhaitée sur cette machine : $L_j = C_{j,k} - d'_{j,k}$. En utilisant un argument similaire à celui de la preuve du théorème 22, on peut montrer que le problème relaxé est équivalent au problème à une machine $1||L_{max}$ pour les travaux restants, avec des durées opératoires $p_{j,k}$ et des dates de fin souhaitées $d'_{j,k}$, et pour lequel la machine n'est disponible qu'à partir de la date $C_{\sigma_i(i),k}$ ($\sigma_i(i)$ désigne le dernier travail placé dans la séquence partielle). Une solution optimale à ce problème est fournie par la règle EDD. Soit L_k^{EDD} la valeur du plus grand retard correspondante. La borne inférieure LB_k^4 est donnée par $LB_k^4 = max(L_{\sigma_i}, L_k^{EDD})$, avec $L_{\sigma_i} = max_{1 \leq h \leq i}\{L_{\sigma_i(h)}\}$ (plus grand retard pour l'ordonnancement partiel associé à σ_i). La borne inférieure globale est alors définie par $LB^4 = max_{1 \leq k \leq m}\{LB_k^4\}$, et est plus précise que celle

qui a été employée dans le cadre du problème précédent $F2|no-wait, NSDST, NSDRT|L_{max}$.

En ce qui concerne les heuristiques chargées de fournir une borne supérieure initiale de bonne qualité, nous proposons des extensions de méthodes classiques. Pour $1 \leq k \leq m$, nous définissons l'heuristique H_k^4 de la manière suivante :

Algorithme. H_k^4.
- Appliquer la règle EDD sur les dates de fin souhaitées $d'_{j,k}$, sur la machine k.
- Construire l'ordonnancement correspondant à la séquence déterminée à l'étape précédente.

La meilleure valeur parmi les m valeurs obtenues peut être utilisée comme une borne supérieure intiale. Nous notons H_{EDD}^4 la solution correspondante.

Nous proposons également de faire appel à des heuristiques basées sur la méthode NEH [Nawaz et al.,83] (voir Section 4.3.5). Elles sont désignées par $NEH^4_{\sum p}[5]$, $NEH^4_{\sum(p+\theta)}[5]$ et $NEH^4_{H^4_{EDD}}[5]$ et réalisent 5 itérations (ce qui est indiqué par le nombre entre crochets). Les listes initiales sont obtenues en classant les travaux respectivement dans l'ordre décroissant de leur durée de traitement totale ($\sum_{1\leq k\leq m} p_{j,k}$), dans l'ordre décroissant de leur longueur totale ($\sum_{1\leq k\leq m-1}(p_{j,k}+\theta_{j,k})+p_{j,m}$) et dans l'ordre correspondant à la solution de l'heuristique H_{EDD}^4.

4.7.5 Expériences numériques menées

Nous avons testé notre approche de résolution sur 13 classes comportant chacune 10 instances. Les programmes, écrits en langage C, ont été exécutés sur un PC muni d'un Pentium à 1,2 GHz.

Les 9 premières classes sont celles qui ont été générées pour le problème précédent $F2|no-wait, NSDST, NSDRT|L_{max}$ (voir le tableau 4.11), qui est un cas particulier du problème considéré ici. Il s'agit d'instances avec uniquement des travaux partiellement couvrants. Nous avons également créé des classes de jeux d'essais pour des problèmes à 5 machines et 16 travaux, sauf pour la classe 12 pour laquelle $n = 14$. Ces classes sont définies ainsi :

- La classe 10 correspond exclusivement à des travaux non couvrants, dont les durées opératoires sont tirées aléatoirement entre 20 et 50. Les time lags sont compris entre 0 et 100.
- La classe 11 correspond exclusivement à des travaux partiellement couvrants, dont les temps de traitement sont tirés aléatoirement entre 20 et 50. $\theta_{j,k}$ est généré entre $-p_{j,k+1}$ et 0, de sorte qu'il y ait recouvrement partiel entre tout couple d'opérations successives au sein des travaux.
- Les classes 12 et 13 correspondent exclusivement à des travaux totalement couvrants. Pour chaque travail j, un entier est tiré aléatoirement entre 1 et 5 pour déterminer la machine k_j telle que j soit k_j totalement couvrant. Les durées opératoires sur toutes les machines sauf k_j sont générées entre 20 et 50, et les time lags qui ne sont pas liés à la machine k_j sont compris entre -20 et 20. $p_{j,k_j}, \theta_{j,k_j-1}$ et θ_{j,k_j} sont déterminés de manière à ce que j soit effectivement k_j totalement couvrant. Bien que de telles instances ne correspondent pas à des situations réelles, il peut être intéressant d'exécuter notre méthode de résolution sur ces instances afin d'évaluer sa performance dans de tels cas.

Les dates de fin souhaitées sont générées de la même manière que pour le problème $F2|no-wait, NSDST, NSDRT|L_{max}$ (voir la section 4.6.3), en suivant la méthode de Potts et Van

wassenhove [Potts et Van Wassenhove,82]. Nous avons fixé $T = 0.6$ et $R = 0.75$.

Nous comparons dans un premier temps les performances des différentes heuristiques. Pour chaque instance, nous calculons l'écart relatif (en %) entre la solution fournie par l'heuristique considérée et la solution optimale, obtenue sans fixer de limite de temps à la PSE. Les valeurs moyennes pour chaque classe apparaissent dans le tableau 4.15. Etant donné que les temps d'exécution sont très faibles (moins de 0,1 seconde), ils ne figurent pas dans ce tableau.

Classe	H^4_{EDD}	$NEH^4_{H^4_{EDD}}[5]$	$NEH^4_{\sum p}[5]$	$NEH^4_{\sum(p+\theta)}[5]$
1	22.1	11.9	8.6	8.9
2	24.3	9.1	6.3	7.3
3	17.7	4.1	2.4	2.9
4	12.9	4.9	5.1	4.2
5	9.8	4.8	3.3	4.8
6	32.9	12.5	10.9	10.9
7	25.9	10.3	12.9	8.6
8	29.1	13.7	8.0	8.4
9	14.8	4.1	3.3	5.3
10	38.9	5.6	5.8	5.3
11	33.5	11.5	7.4	9.4
12	26.2	7.1	5.5	6.0
13	32.5	9.1	6.4	8.0
Average	24.7	8.4	6.6	6.9

TAB. 4.15 – Ecart relatif moyen (en %) entre les solutions fournies par les heuristiques et l'optimum

Comme on peut le remarquer, H^4_{EDD} est moins performante que les méthodes de type NEH. Ceci est d'ailleurs vérifié non seulement en moyenne, mais également pour chaque instance. $NEH^4_{H^4_{EDD}}[5]$ est légèrement moins précise que $NEH^4_{\sum p}[5]$ et $NEH^4_{\sum(p+\theta)}[5]$, dont les écarts relatifs sont assez proches et n'augmentent pas lorsque le nombre de machines croît. De plus, pour chacune de ces trois heuristiques, il existe au moins une instance dans chaque classe pour laquelle elle domine les deux autres.

Afin d'évaluer la qualité de notre PSE, nous l'avons exécutée sur chaque instance avec une limite de temps de 1200 secondes. Pour chaque classe, nous présentons dans le tableau 4.16 le nombre N de problèmes pour lesquels la procédure atteint la limite de temps, le temps moyen de calcul t (en secondes) pour les problèmes résolus optimalement avant la limite, et le temps d'exécution moyen t_{dom} (en secondes) lorsque les conditions de dominance décrites dans la section 4.7.3 sont incorporées (seulement pour les problèmes à deux machines). Les valeurs correspondantes obtenues avec la méthode proposée en 4.6.2 pour le problème $F2|no-wait, NSDST, NSDRT|L_{max}$sont indiquées avec un symbole "prime" ($'$). Nous rappelons que les tests de dominance utilisés pour la méthode de la section 4.6.2 sont plus restrictifs que ceux que nous employons ici et ne concernent que les classes 8 et 9.

Si l'on compare la performance de notre nouvelle PSE et celle qui a été spécialement développée pour le problème $F2|no-wait, NSDST, NSDRT|L_{max}$, on peut noter une amélioration significative pour les temps d'exécution : tous les problèmes à deux machines sauf un sont résolus de façon optimale par la nouvelle procédure et le temps de calcul moyen est divisé par

Classe	N	t	t_{dom}	N'	t'	t'_{dom}
1	0	13.8	9.2	1	107.1	/
2	0	13.2	10.7	1	171.6	/
3	0	22.4	19.4	2	121.6	/
4	0	13.6	8.5	4	251.7	/
5	0	35.5	14.2	1	59.7	/
6	0	0.4	0.3	0	99.7	/
7	0	3.7	2.9	1	88.2	/
8	0	8.9	6.5	0	166.4	153.6
9	1	10.2	5.7	3	182.5	139.7
10	0	155.3	/	/	/	/
11	0	115.7	/	/	/	/
12	0	71.2	/	/	/	/
13	3	209.4	/	/	/	/

TAB. 4.16 – Performances de la PSE sans et avec les tests de dominance

un facteur variant entre 5 et 250, sauf pour la classe 5. Il peut paraître surprenant qu'une méthode développée pour un problème plus général soit plus efficace qu'une méthode conçue pour un cas particulier. Un tel gain est dû en partie à l'amélioration de la borne inférieure, ce qui a été rendu possible grâce à la formulation plus générale du problème. D'autre part, les relations de dominance, qui s'appliquent plus fréquemment, se révèlent assez efficaces puisqu'il en résulte une réduction de 25% du temps d'exécution en moyenne. En ce qui concerne les problèmes à 5 machines, il semble que ceux qui impliquent exclusivement des travaux totalement couvrants sont plus difficiles à résoudre que les autres.

Nous avons également réalisé une autre série d'expériences afin d'évaluer l'influence du nombre de machines m sur la performance de la PSE. Trois nouvelles classes notées 11A, 11B et 11C ont été générées de manière similaire à la classe 11 (comportant exclusivement des travaux partiellement couvrants), avec m égal à 2, 10 et 15 respectivement. Le tableau 4.17 présente le nombre de problèmes N (sur 10) pour lesquels la procédure atteint la limite de temps de 1200 secondes, et le temps d'exécution moyen τ (en secondes) lorsqu'on ne fixe pas de limite. Nous faisons également figurer le nombre Q de noeuds évalués (c'est-à-dire pour lesquels la borne inférieure est calculée), $\rho = \tau/Q$ et $\lambda = \rho/m$. Ainsi, ρ représente le temps moyen pour évaluer un noeud (en secondes). Sans limite de temps, la plus longue durée d'exécution est de 3720 secondes, pour une instance à 15 machines.

Classe	m	N	τ	Q	$\rho = \tau/Q$	$\lambda = \rho/m$
11A	2	0	5.6	5.086×10^6	1.10×10^{-6}	0.55×10^{-6}
11	5	0	115.7	47.50×10^6	2.44×10^{-6}	0.48×10^{-6}
11B	10	1	500.9	111.85×10^6	4.48×10^{-6}	0.45×10^{-6}
11C	15	5	1449.9	223.35×10^6	6.49×10^{-6}	0.43×10^{-6}

TAB. 4.17 – Influence du nombre de machines sur le temps d'exécution

On peut considérer la valeur de λ constante puisqu'elle varie entre $0,43 \times 10^{-6}$ et $0,55 \times 10^{-6}$. Ceci est vrai non seulement en moyenne, mais aussi pour chaque instance prise individuellement. Par définition, cela signifie que le temps moyen mis par la PSE pour évaluer un noeud augmente

linéairement avec le nombre de machines, ce qui concorde avec la complexité de la borne inférieure en $O(m)$. De plus, le nombre de noeuds semble grossièrement varier de manière linéaire en m. Cette constatation empirique nécessite cependant d'autres expériences pour être confirmée ou infirmée. Une telle augmentation pourrait être expliquée par le fait que la borne inférieure est de moins en moins précise au fur et à mesure que le nombre de machines croît.

4.8 Extensions à de nouvelles contraintes

Dans cette section, nous nous intéressons à de nouvelles contraintes et nous cherchons à étendre les approches de résolution précédentes lorsqu'elles sont prises en compte. Nous considérons dans un premier temps des dates de disponibilités des travaux et des durées de latence (ou queues), puis nous étudions les modifications à apporter à nos méthodes de résolution en présence de time lags généralisés. Enfin, nous nous interrogeons sur la possibilité de traiter avec une approche similaire les problèmes de flowshop généraux, c'est-à-dire pour lesquels on ne se limite pas aux ordonnancements de permutation.

4.8.1 Dates de disponibilités et durées de latence des travaux

Nous avons supposé jusqu'à maintenant que l'ensemble des travaux est disponible pour le traitement dès l'instant 0. Ceci n'est pas toujours le cas dans des cas réels, lorsque l'atelier est approvisionné par un autre atelier en amont et que les produits arrivent progressivement. D'une manière générale, les dates de disponibilité, notées r_j, peuvent être assimilées à des dates de début au plus tôt. Elles impliquent en effet que l'exécution d'un travail ne peut démarrer avant sa date de disponibilité : $t_{j,1} \geq r_j$. En prenant en compte ces nouvelles contraintes, il ne s'agit pas de résoudre un problème dynamique dans lequel l'ordonnancement serait déterminé "en temps réel", à mesure que les produits arrivent dans l'atelier. On suppose en effet que ces dates de disponibilité sont fixes et connues à l'avance.

Il est possible de représenter ces contraintes à l'aide de time lags minimaux : il suffit en effet de rajouter une machine fictive 0 située en amont des autres machines, sur laquelle les durées opératoires sont toutes nulles, et telle que, pour chaque travail, le time lag minimal entre cette machine et la première machine réelle soit égal à la date de disponibilité du travail : $\forall j, p_{j,0} = 0$ et $\theta_{j,0}^{min} = r_j$. Dans tout ordonnancement semi-actif, les travaux sont tous exécutés à la date 0 sur la machine fictive (il n'y a pas de time lag maximal associé à cette machine) et la contrainte de time lag minimal s'exprime alors : $t_{j,1} \geq C_{j,0} + \theta_{j,0}^{min} = r_j$. On retrouve donc bien les contraintes définies par les dates de disponibilité. Il faut également remarquer que la restriction aux solutions de permutation ne pose aucun problème ici : puisque tous les travaux sont exécutés "simultanément" à l'instant 0 sur la machine fictive, toutes les séquences de traitement sont équivalentes pour cette machine. La figure 4.15 illustre l'équivalence entre les deux situations (avec dates de disponibilités en 4.15(a) et avec une machine fictive M0 en 4.15(b)).

De manière symétrique, il est possible d'introduire des durées de latence pour les travaux. Celles-ci, notées q_j, sont parfois désignées par le terme "queues", et correspondent à un traitement ou une manipulation du travail effectué en-dehors de la ligne de production (livraison par exemple). La ressource associée à cette opération est supposée de capacité suffisante de manière à ce qu'il n'y ait pas de conflit entre les travaux. De telles contraintes modifient simplement la définition de la date de fin d'exécution du travail : $C_j = C_{j,m} + q_j$, où m est le

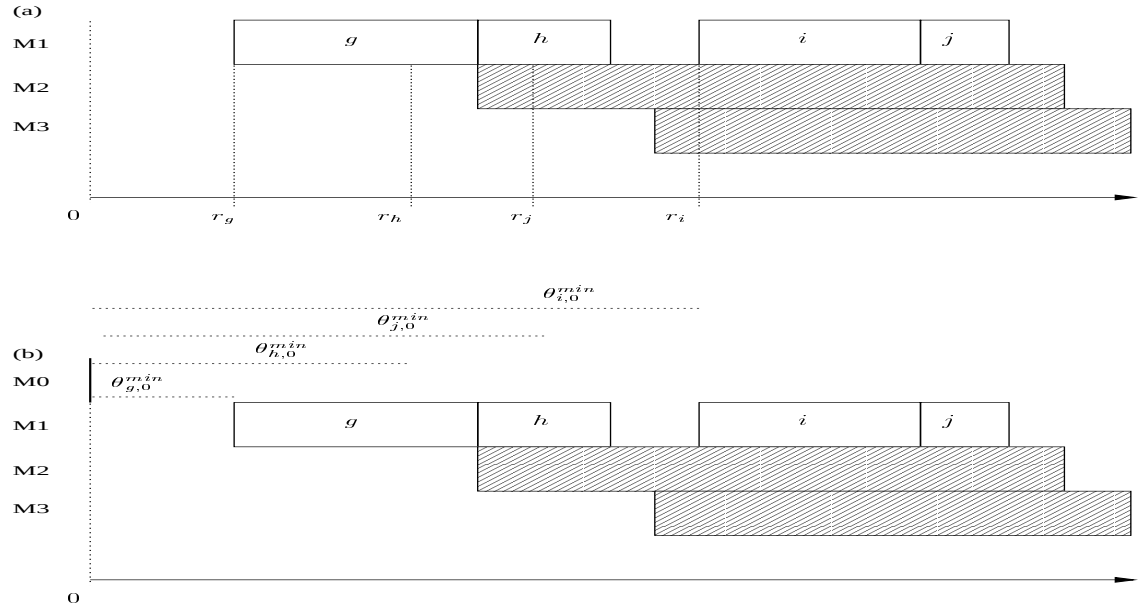

FIG. 4.15 – Représentation des dates de disponibilités à l'aide de time lags minimaux

nombre de machines (réelles) de l'atelier. De la même manière que précédemment, on peut représenter les durées de latence par des time lags minimaux, en ajoutant une machine fictive $m+1$ en aval de toutes les autres machines, sur laquelle les durées opératoires sont nulles et telle que pour chaque travail, le time lag minimal entre la dernière machine réelle et cette machine soit égal à la durée de latence du travail. Contrairement au cas des dates de disponibilité, la restriction aux ordonnancements de permutation a ici un impact sur les dates de fin : du fait des contraintes de précédence sur la machine fictive, on a $C_{\pi(i),m+1} = t_{\pi(i),m+1} = max\{C_{\pi(i-1),m+1}, C_{\pi(i),m} + \theta^{min}_{j,m+1}\} = max\{C_{\pi(i-1),m+1}, C_{\pi(i),m} + q_{\pi(i)}\}$. La date de fin d'un travail j sur la machine fictive $m+1$ ne correspond donc pas à sa date de fin réelle $C_{j,m}+q_j$ compte tenu de sa durée de latence. Par récurrence, on peut montrer facilement que pour le travail $\pi(i)$ placé en i-ième position, on a $C_{\pi(i),m+1} = max_{1\leq h\leq i}\{C_{\pi(h),m}+q_{\pi(h)}\}$. En particulier, le makespan obtenu vérifie $C_{max} = C_{\pi(n),m+1} = max_{1\leq h\leq n}\{C_{\pi(h),m}+q_{\pi(h)}\}$. Ce résultat correspond bien à la définition du makespan en présence de durées de latence. Plus généralement, la transformation du problème impliquant une machine fictive ajoutée en dernière position ne modifie pas la valeur du critère lorsque celui-ci est sous la forme d'un maximum de fonctions des dates de fin des travaux, alors qu'elle peut la modifier lorsqu'il s'agit d'une somme de fonctions des dates de fin des travaux. Dans ce dernier cas, il est toujours possible de considérer le problème sous sa forme initiale, c'est-à-dire de travailler uniquement avec les machines réelles, et de n'ajouter les durées de latence que pour le calcul des dates de fin des travaux C_j. La figure 4.16 représente les deux situations (avec durées de latences en 4.16(a) et avec une machine fictive Mm+1 en 4.16(b)). On peut noter que la date de fin C_i du travail i n'est pas la même d'une situation à l'autre, pour les raisons que nous venons d'expliquer. Par contre, le makespan, donné par C_j, est bien le même.

Un résultat classique en ordonnancement est l'équivalence entre les problèmes de minimisation du makespan C_{max} en présence de durées de latence q_j et les problèmes de minimisation du plus grand retard L_{max} en présence de dates de fin souhaitées d_j, avec $q_j = -d_j$ (si l'on désire travailler uniquement avec des durées de latence positives, il suffit de fixer $q_j = max_i\{d_i\} - d_j$). Ainsi, l'ap-

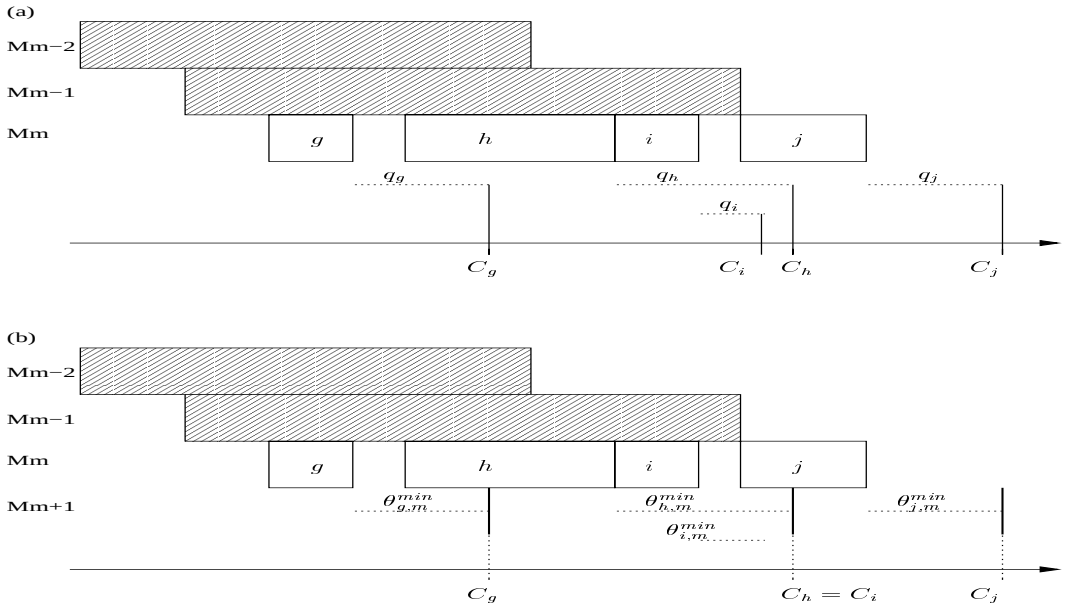

FIG. 4.16 – Représentation des durées de latences à l'aide de time lags minimaux

proche présentée en section 4.4 développée initialement pour le problème $Fm_\pi|\theta^{min}_{j,k}, \theta^{max}_{j,k}|C_{max}$ peut être utilisée directement pour résoudre un problème de type $Fm_\pi|r_j, \theta^{min}_{j,k}, \theta^{max}_{j,k}|L_{max}$, uniquement en adaptant l'instance fournie en entrée, selon les modifications indiquées dans les paragraphes précédents. Ces résultats ne sont pas nouveaux et figurent dans de nombreux travaux (par exemple : [Bartusch et al.,88][Munier et al.,98][Fest et al.,99][Bouquard,04]).

On peut cependant évoquer une légère restriction à cette extension. Supposons que l'on dispose d'une méthode développée spécifiquement pour un cas particulier de flowshop avec time lags, comme le cas des time lags exacts (chaque time lag minimal a la même valeur que le time lag maximal correspondant). Comme nous l'avons vu, l'introduction des machines fictives s'accompagne de time lags minimaux dont les valeurs dépendent des dates de disponibilité et des durées de latence, et de time lags maximaux infinis (pour ne pas rajouter de contrainte). On perd alors les spécificités des time lags exacts, ce qui oblige à adapter l'approche de résolution.

4.8.2 Prise en compte de time lags généralisés

Jusqu'à maintenant, nous avons toujours supposé que les time lags sont définis exclusivement entre les couples d'opérations successives au sein des travaux. Dans la pratique, il est pourtant courant de rencontrer des situations où ce type de contraintes relie des opérations non consécutives (voir les exemples cités dans [Deppner,04]). Pour représenter ces time lags, nous utilisons la même notation que pour des time lags entre opérations consécutives, en rajoutant un indice : par exemple, $\theta^{max}_{j,(k_1,k_2)}$ désigne le time lag maximal entre les opérations k_1 et k_2 du travail j. De nouveau, nous supposons que les time lags sont définis entre tout couple d'opérations quelconques au sein des travaux, quitte à fixer éventuellement les valeurs à 0 pour les time lags minimaux et $+\infty$ pour les time lags maximaux en l'absence explicite de contrainte. On peut alors s'interroger sur les modifications à apporter à l'approche de résolution que nous avons proposée, afin de prendre en compte ces contraintes plus générales. Comme nous l'avons constaté, notre approche repose

sur la propriété selon laquelle le problème restreint peut être résolu polynomialement. On peut facilement montrer que cette propriété reste valable en présence de time lags généralisés, et que l'algorithme correspondant peut être modifié de la manière suivante :

Algorithme. A3.

1 *Placer le premier travail* $\pi(1)$ *au plus tôt*
  1-1 $C_{\pi(1),1} = p_{\pi(1),1}$
  1-2 $k_2 = 2$. Tant que $k_2 \leq m$ répéter
    1-2-1 $C_{\pi(1),k_2} = C_{\pi(1),1} + \theta^{min}_{\pi(1),(1,k_2)} + p_{\pi(1),k_2}$.
    1-2-2 $k_1 = 2$. Tant que $k_1 < k_2$ répéter
      – Si $C_{\pi(1),k_2} < C_{\pi(1),k_1} + \theta^{min}_{\pi(1),(k_1,k_2)} + p_{\pi(1),k_2}$ alors $C_{\pi(1),k_2} = C_{\pi(1),k_1} + \theta^{min}_{\pi(1),(k_1,k_2)} + p_{\pi(1),k_2}$
      – $k_1 = k_1 + 1$
    1-2-3 $k_2 = k_2 + 1$
2 *Placer les autres travaux au plus tôt, dans l'ordre de la séquence.* $j = 2$. Tant que $j \leq n$ répéter
  2-1 *Placer la première opération au plus tôt sur la première machine.* $C_{\pi(j),1} = C_{\pi(j-1),1} + p_{\pi(j),1}$
  2-2 *Placer successivement les autres opérations au plus tôt, en ne tenant compte que des contraintes de précédence et des time lags minimaux.* $k_2 = 2$. Tant que $k_2 \leq m$ répéter
  2-2-1 $C_{\pi(j),k_2} = max\{C_{\pi(j),1} + \theta^{min}_{\pi(j),(1,k_2)}, C_{\pi(j-1),k_2}\} + p_{\pi(j),k_2}$
  2-2-2 $k_1 = 2$. Tant que $k_1 < k_2$ répéter
    – Si $C_{\pi(j),k_2} < C_{\pi(j),k_1} + \theta^{min}_{\pi(j),(k_1,k_2)} + p_{\pi(j),k_2}$ alors $C_{\pi(j),k_2} = C_{\pi(j),k_1} + \theta^{min}_{\pi(j),(k_1,k_2)} + p_{\pi(j),k_2}$
    – $k_1 = k_1 + 1$
  2-2-3 $k_2 = k_2 + 1$
  2-3 *Vérifier si les contraintes de time lags maximaux sont satisfaites, en commençant par la dernière machine.*
  Tant que $k_2 > 1$ répéter
  2-3-1 $k_1 = k_2 - 1$. Tant que $k_1 \geq 1$
    – Si $C_{\pi(j),k_2} - C_{\pi(j),k_1} > \theta^{max}_{\pi(j),(k_1,k_2)} + p_{\pi(j),k_2}$
    – *Retarder l'exécution de la première opération du couple concerné afin de respecter le time lag maximal* $C_{\pi(j),k_1} = C_{\pi(j),k_2} - p_{\pi(j),k_2} - \theta^{max}_{\pi(j),(k_1,k_2)}$
    – $k_1 = k_1 - 1$
  2-3-2 $k_2 = k_2 - 1$

Théorème. 23 *L'algorithme précédent (A3) fournit une solution optimale pour le problème restreint en* $O(n.m^2)$.

Preuve. La preuve est en tout point identique à celle du théorème 19.□

Ce résultat nous assure la même propriété que pour le cas particulier des time lags définis entre opérations consécutives : à partir d'une permutation des travaux, l'algorithme précédent construit l'ordonnancement semi-actif associé en $O(n.m^2)$. Il suffit donc d'employer des méthodes de résolution développées pour les problèmes "de permutation". En particulier, l'approche de résolution générique que nous avons présentée est toujours valable.

4.8.3 Cas du flowshop général, où l'on ne se restreint pas aux solutions de permutation

Dans cette section, nous nous interrogeons sur la possible application de notre approche de résolution dans le cas où l'on ne se limite plus aux ordonnancements de permutation. Comme nous l'avons vu précédemment, ceux-ci ne sont pas dominants en présence de time lags, même pour deux machines. Même si la restriction aux solutions de permutation peut se trouver justifier dans la pratique par des considérations techniques (utilisation d'un système de transport filo-guidé empêchant tout dépassement de travaux entre les machines) ou organisationnelles (simplification du problème et réduction du nombre de solutions à considérer), il peut être intéressant de rechercher l'écart entre la solution optimale générale et la meilleure solution de permutation. Deppner fournit un résultat théorique à ce sujet : pour le problème $Fm|\theta_j^{min}|C_{max}$, si l'on suppose les durées opératoires et les time lags minimaux bornés, le rapport entre la valeur du meilleur ordonnancement de permutation et l'optimum général tend vers 1 lorsque le nombre de travaux n tend vers l'infini. Sur le plan pratique, une manière envisageable pour obtenir l'ordonnancement optimal général consisterait à étendre notre Procédure par Séparation et Evaluation de sorte à considérer également les solutions pour lesquelles les séquences de traitement des travaux varient d'une machine à l'autre. Pour étudier cette question, il convient d'abord de redéfinir rigoureusement le problème restreint :

Problème restreint.
Pour une permutation donnée $\pi = (\pi(1), \pi(2), \ldots \pi(n))$ des travaux, calculer les dates d'exécution de toutes les opérations de façon à obtenir un ordonnancement qui respecte à la fois la séquence de traitement π sur la première machine et les contraintes du problème, et qui optimise la fonction objectif.

Nous supposons ainsi que l'entrée du problème est la séquence d'exécution des travaux sur la première machine uniquement, et nous nous intéressons seulement à la minimisation du makespan. D'autre part, on exclut bien entendu les situations particulières pour lesquelles les solutions de permutation restent dominantes, auquel cas il suffirait d'appliquer l'approche de résolution développée pour le flowshop de permutation. Dans la suite, nous sommes amenés à considérer trois cas distincts.

a) Deux machines et time lags minimaux uniquement

Du fait de l'absence des time lags maximaux, tout ordonnancement semi-actif exécute les travaux dans l'ordre π sur la première machine sans interruption, de l'instant 0 à l'instant $\sum_{1\leq j\leq n} p_{j,1}$. Chaque travail j, situé à la position $\pi^{-1}(j)$ dans la séquence, est traité entre la date $t_{j,1} = \sum_{1\leq i\leq \pi^{-1}(j)-1} p_{\pi(i),1}$ et la date $C_{j,1} = \sum_{1\leq i\leq \pi^{-1}(j)} p_{\pi(i),1}$ sur la première machine. Il est donc disponible sur la deuxième machine à partir de la date $C_{j,1} + \theta_j^{min}$. On peut alors facilement montrer que le problème est équivalent à minimiser le makespan sur une machine, en présence de dates de disponibilités des travaux $r_j = \sum_{1\leq i\leq \pi^{-1}(j)} p_{\pi(i),1} + \theta_j^{min}$ (voir figure 4.17). Une séquence optimale est obtenue en traitant les travaux dans l'ordre croissant des dates de disponibilités. Ainsi, le problème restreint est toujours polynomial dans ce cas. On retrouve ce résultat dans [Dell'Amico,96].

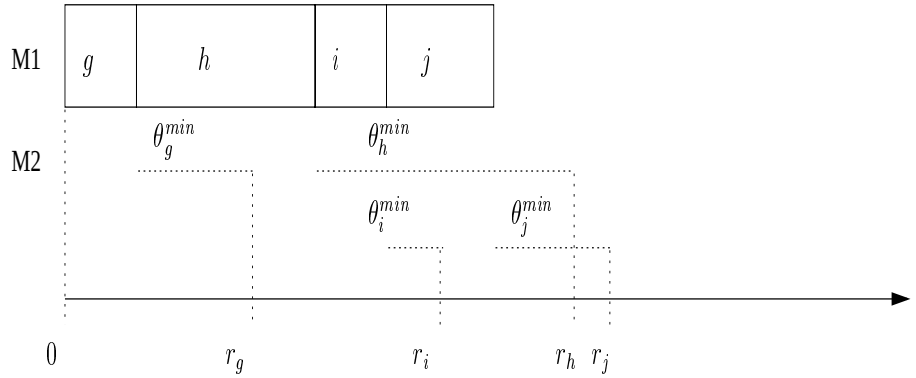

FIG. 4.17 – Cas $m = 2$ sans time lags maximaux

b) Trois machines sans time lags

De la même manière que précédemment, il suffit de considérer les solutions dans lesquelles les travaux sont traités sur la première machine dans l'ordre π sans interruption. Il nous reste donc à ordonnancer les travaux sur les deux dernières machines, en tenant compte des dates de fin sur la première machine qui conduisent à des dates de disponibilités. Ainsi, on est ramené à résoudre un problème du type $F2|r_j|C_{max}$ (voir figure 4.18) qui est NP-difficile au sens fort [Lenstra et al.,77].

De manière plus rigoureuse, on montre que :

Théorème. 24 *Le problème restreint à 3 machines sans time lags est NP-difficile au sens fort.*

Preuve. On peut employer une réduction de $F2|r_j|C_{max}$ qui est NP-difficile au sens fort [Lenstra et al.,77]. Soit une instance I de $F2|r_j|C_{max}$ donnée par n le nombre de travaux, pour chaque travail j, ses durées d'exécution $p_{j,1}$ et $p_{j,2}$ sur la première et la deuxième machines respectivement, et sa date de disponibilité r_j, et un entier Z. La question est de savoir s'il existe une solution S telle que $C_{max}(S) \leq Z$. Sans perte de généralité, on peut supposer que les travaux sont numérotés dans l'ordre croissant de leur date de disponibilité ($r_1 \leq r_2 \leq \ldots \leq r_n$). Etant donnée l'instance I, on construit une instance I' pour notre problème restreint de la manière

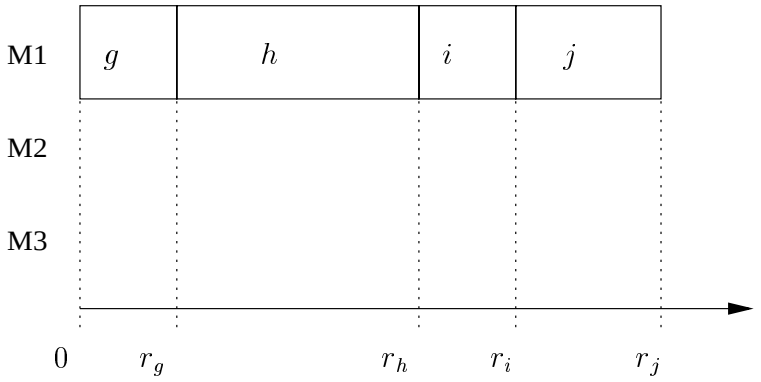

FIG. 4.18 – Cas $m = 3$ sans time lags

suivante : le nombre de travaux à ordonnancer est $n' = n$, et pour chaque travail j, les durées opératoires sont respectivement $p'_{j,1} = r_j - r_{j-1}$ (avec $r_0 = 0$) sur la première machine, $p'_{j,2} = p_{j,1}$ sur la deuxième machine et $p'_{j,3} = p_{j,2}$ sur la troisième machine. La séquence de travaux à traiter pour la première machine est $\pi' = (1, 2, \ldots, n)$ et l'on cherche s'il existe une solution S' pour le problème restreint telle que $C_{max}(S') \leq Z$.

Supposons qu'il existe une solution S au problème $F2|r_j|C_{max}$ pour l'instance I. S' est obtenue en fixant pour chaque travail j : $C'_{j,1} = \sum_{1 \leq i \leq j} p'_{i,1} = r_j$, $C'_{j,2} = C_{j,1}$ et $C'_{j,3} = C_{j,2}$. Autrement dit, la solution S' sur les machines deux et trois est identique à S sur les deux machines. On peut facilement vérifier que S' respecte la séquence π' sur la première machine, ainsi que toutes les contraintes de précédence classiques. D'autre part, $C_{max}(S') = C_{max}(S) \leq Z$.

Réciproquement, si on dispose d'une solution S' pour le problème restreint avec $C_{max}(S') \leq Z$, on construit S en appliquant la transformation réciproque, c'est-à-dire en conservant l'ordonnancement sur les deux dernières machines. Puisque dans S', les travaux sont traités dans l'ordre π' sur la première machine, on a $\forall j, t'_{j,2} \geq C'_{j,1} \geq \sum_{1 \leq i \leq j} p'_{i,1}$. On en déduit que dans S, chaque travail j vérifie $t_{j,1} = t'_{j,2} \geq r_j$. Donc les contraintes liées aux dates de disponibilité sont satisfaites dans S. On vérifie aisément qu'il en est de même pour les contraintes de précédence, et que $C_{max}(S) = C_{max}(S') \leq Z$. L'ordonnancement correspondant à l'instance I est représenté sur la figure 4.19(a), alors que celui pour l'instance I' apparaît sur la figure 4.19(b).□

Ce résultat peut à première vue sembler surprenant. En effet, d'après un résultat classique en ordonnancement, que nous avons déjà mentionné en section 2.3.1, les solutions de permutation sont dominantes pour le problème $F3||C_{max}$. On pourrait donc croire que, étant donnée la séquence de traitement sur la première machine, il suffit d'exécuter les travaux selon cette séquence sur les deux autres machines pour obtenir le meilleur ordonnancement. On aurait ainsi un algorithme polynomial pour le problème restreint à 3 machines et sans time lags. Ce raisonnement est erronné, au niveau de l'utilisation de la propriété de dominance : celle-ci implique qu'il existe un ordonnancement optimal parmi les ordonnancements de permutation. Cependant, ce résultat ne signifie pas pour autant que le meilleur ordonnancement respectant une séquence donnée sur la première machine fait partie des ordonnancements de permutation. Pour s'en convaincre, il suffit de considérer l'exemple suivant.

Exemple. Soient $n = 3$ travaux dont les durées sont données dans le tableau 4.18. On suppose que la séquence de traitement imposée sur la première machine est $(1, 2, 3)$. L'ordon-

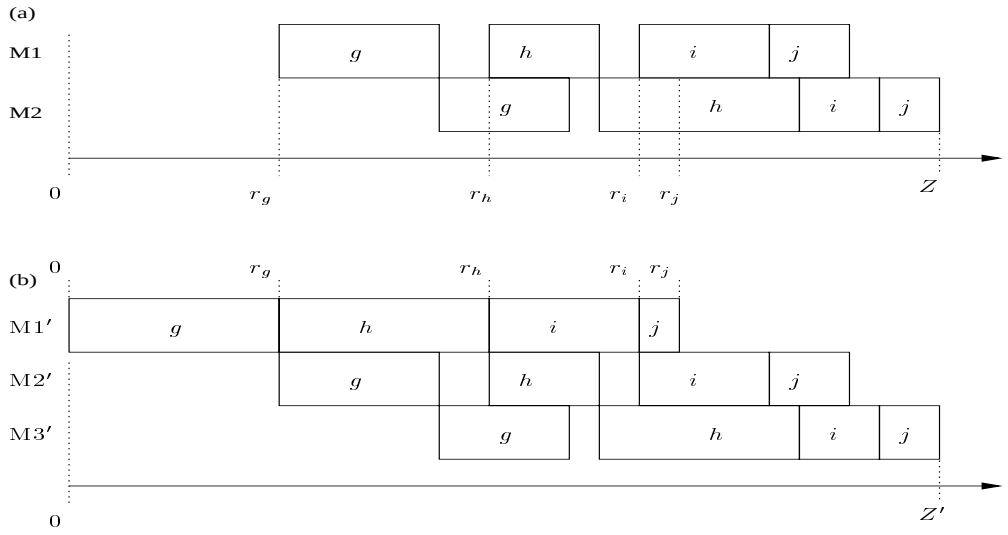

FIG. 4.19 – Problème restreint avec 3 machines et time lags minimaux entre les deux premières machines

nancement O qui consiste à traiter les travaux dans l'ordre $(1, 3, 2)$ sur la seconde et la troisième machines conduit à un makespan de 26. Cette valeur est optimale, puisqu'elle est égale à la borne inférieure $min_{1\leq j\leq 3}\{p_{j,1}\} + \sum_{1\leq i\leq 3} p_{i,2} + min_{1\leq h\leq 3}\{p_{h,3}\}$. Si l'on considère l'ordonnancement O' de permutation, qui correspond à la séquence $(1, 2, 3)$ fixée, le makespan est de 35. Celui-ci n'est donc pas optimal, ce qui prouve que les solutions de permutation ne sont pas dominantes pour le problème restreint considéré. Les ordonnancements O et O' sont représentés sur les figures 4.20a et 4.20b.

Travail j	$p_{j,1}$	$p_{j,2}$	$p_{j,3}$
1	1	4	10
2	2	10	1
3	2	10	10

TAB. 4.18 – Durées opératoires de l'exemple

c) Deux machines et time lags maximaux uniquement

Contrairement aux deux cas précédents, rien a priori ne garantit que les travaux sont exécutés sans interruption sur la première machine. Néanmoins, si l'on suppose qu'il est possible de fixer les dates d'exécution de ces opérations, on remarque intuitivement qu'on est ramené à un problème à une machine avec dates de disponibilités et échéances strictes des travaux, de type $1|r_j, \tilde{d}_j|$. Les dates de disponibilités r_j sont dues au traitement sur la première machine et les échéances strictes $\tilde{d}_j$ sont dues aux contraintes de time lags maximaux (voir figure 4.21).

Plus rigoureusement, on peut démontrer que :

Théorème. 25 *Le problème restreint à 2 machines sans time lags minimaux est NP-difficile au sens fort.*

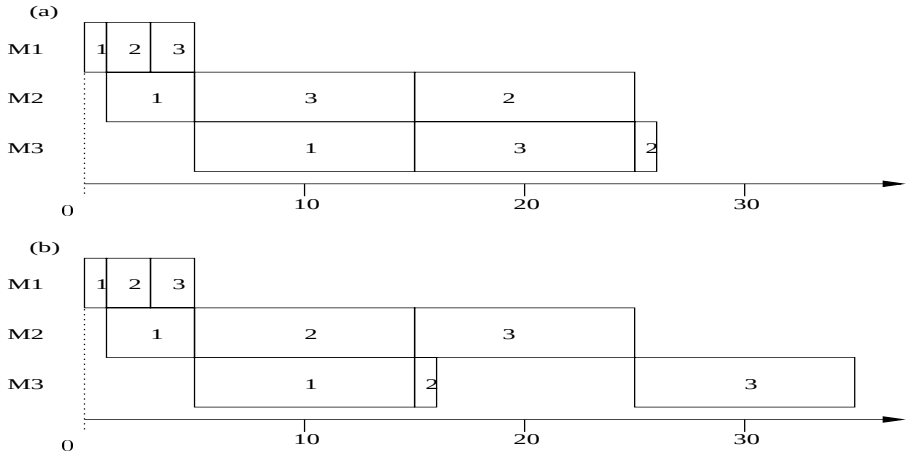

FIG. 4.20 – Non dominance des solutions de permutation pour le problème restreint à 3 machines

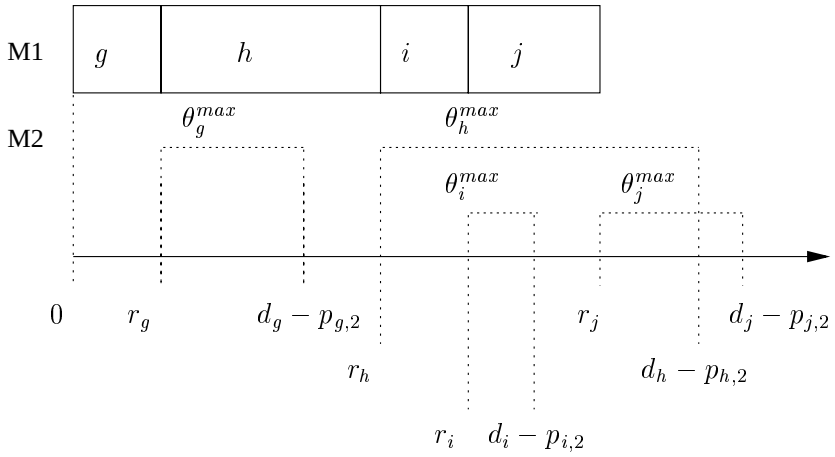

FIG. 4.21 – Cas $m = 2$ sans time lags minimaux

Preuve. On peut employer une réduction du problème $1|r_j, \tilde{d}_j|$ qui consiste à ordonnancer des travaux sur une machine entre leurs date de disponibilité et leur échéance, et qui est NP-difficile au sens fort [Lenstra et al.,77]. Soit une instance I de $1|r_j, \tilde{d}_j|$ donnée par n le nombre de travaux, pour chaque travail j, sa durée d'exécution p_j, sa date de disponibilité r_j et son échéance $\tilde{d}_j \geq r_j$. La question est de savoir s'il existe une solution S telle que $\forall j, t_j \geq r_j$ et $C_j \leq \tilde{d}_j$. Sans perte de généralité, on peut supposer que les travaux sont numérotés dans l'ordre croissant de leur date de disponibilité ($r_1 \leq r_2 \leq \ldots \leq r_n$). Etant donnée l'instance I, on construit une instance I' pour notre problème restreint de la manière suivante : le nombre de travaux à ordonnancer est $n' = n + 1$, et pour chaque travail $j \leq n$, les durées opératoires sont respectivement $p'_{j,1} = r_j - r_{j-1}$ (avec $r_0 = 0$) sur la première machine et $p'_{j,2} = p_j$ sur la deuxième machine, et le time lag maximal est $\theta_j^{max\prime} = \tilde{d}_j - p_j - r_j$. Le travail artificiel $n+1$ est décrit par $p'_{n+1,1} = max_{1 \leq j \leq n}\{\tilde{d}_j\} - r_n$, $p'_{n+1,2} = 1$ et $\theta_{n+1}^{max\prime} = 0$. La séquence de travaux à traiter pour la première machine est $\pi' = (1, 2, \ldots, n, n+1)$ et l'on cherche s'il existe une solution S' pour le problème restreint telle que $C_{max}(S') \leq Z' = max_{1 \leq j \leq n}\{\tilde{d}_j\} + 1$.

Supposons qu'il existe une solution S au problème $1|r_j, \tilde{d}_j|$ pour l'instance I. S' est obtenue en fixant pour chaque travail $j \leq n$: $C'_{j,1} = \sum_{1 \leq i \leq j} p'_{i,1} = r_j$, $C'_{j,2} = C_j$, et pour le travail $n+1$, $C'_{n+1,1} = max_{1 \leq j \leq n}\{\tilde{d}_j\}$ et $C'_{n+1,2} = max_{1 \leq j \leq n}\{\tilde{d}_j\} + 1$. Autrement dit, la solution S' sur la machine deux est identique à S, plus le travail $n+1$ placé à la fin. On peut facilement vérifier que S' respecte la séquence π' sur la première machine, et qu'il n'y a pas de conflit entre opérations. Soit un travail $j \leq n$. Sa date de début sur la deuxième machine et sa date de fin sur la première machine vérifie $t'_{j,2} - C'_{j,1} = C_j - p_j - r_j$. Puisque S est solution de I, on a : $0 \leq C_j - p_j - r_j \leq \tilde{d}_j - p_j - r_j = \theta_j^{max\prime}$. Donc les contraintes de time lags maximaux sont satisfaites pour $j \leq n$. Il en est de même pour le travail $n+1$ puisqu'on a : $t'_{n+1,2} - C'_{n+1,1} = 0 = \theta_{n+1}^{max\prime}$. D'autre part, $C_{max}(S') = C'_{n+1,2} = Z'$.

Réciproquement, supposons que l'on dispose d'une solution S' pour le problème restreint telle que $C_{max}(S') \leq Z'$. Puisque dans S', les travaux sont traités dans l'ordre donné par π' sur la première machine, on a : $C_{max}(S') \geq \sum_{1 \leq j \leq n+1} p'_{j,1} + p'_{n+1,2} = max_{1 \leq j \leq n}\{\tilde{d}_j\} + 1 = Z'$. On en déduit que les travaux sont exécutés sans interruption à partir de l'instant 0 sur la première machine. En particulier, on a : $\forall j \leq n, C'_{j,1} = \sum_{1 \leq i \leq j} p'_{i,1} = r_j$. On construit la solution S en appliquant la transformation réciproque, c'est-à-dire en conservant l'ordonnancement sur la deuxième machine. D'après ce qui précède, chaque travail $j \leq n$ dans S vérifie $t_{j,1} = t'_{j,2} \geq C'_{j,1} = r_j$. De plus, puisque S' est réalisable, les contraintes de time lags maximaux sont vérifiées, c'est-à-dire $\forall j \leq t'_{j,2} - C'_{j,1} \leq \theta_j^{max\prime} = \tilde{d}_j - p_j - r_j$, soit $C'_{j,2} \leq \tilde{d}_j$. Il s'en suit que pour tout travail j dans S, $C_j = C'_{j,2} \leq \tilde{d}_j$. Par conséquent, chaque travail respecte son échéance et S est bien une solution du problème $1|r_j, \tilde{d}_j|$ pour l'instance I. L'ordonnancement correspondant à l'instance I est représenté sur la figure 4.22(a), alors que celui pour l'instance I' apparaît sur la figure 4.22(b).□

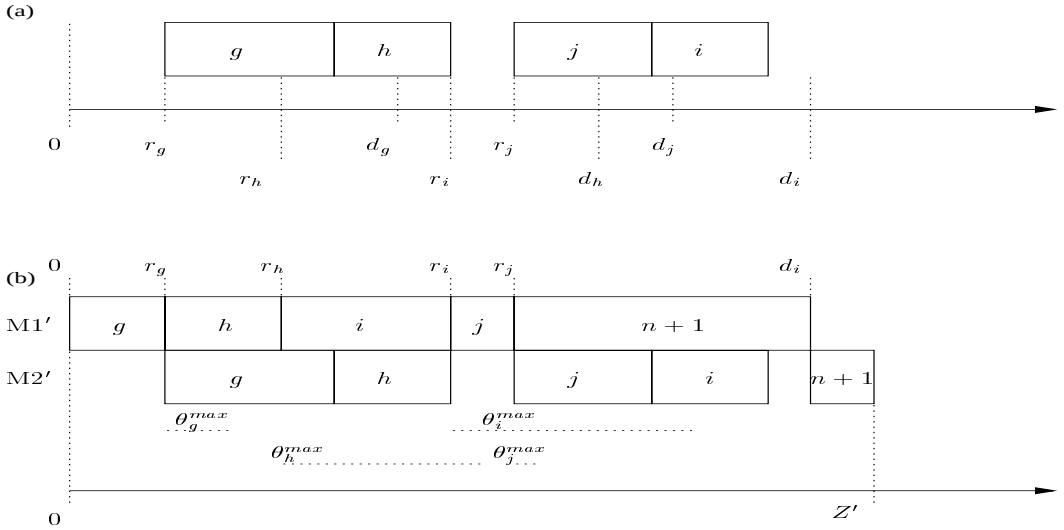

FIG. 4.22 – Problème restreint avec 2 machines et sans time lags minimaux

d) Bilan des trois cas considérés

Il résulte de cette analyse que le schéma de séparation développé pour les problèmes de flow-shop de permutation, en présence de time lags, ainsi que toutes les techniques qui reposent sur la correspondance entre séquence de traitement et ordonnancement semi-actif, ne sont valables

que pour le cas $m = 2$ et en l'absence de time lags maximaux, lorsque l'on considère aussi les solutions qui ne sont pas de permutation et que la séquence de traitement n'est donnée que pour la première machine. Dans ce cas particulier, issu du problème $F2|\theta_j^{min}|\gamma$ (γ étant un critère régulier), il est tout-à-fait envisageable de concevoir une Procédure par Séparation et Evaluation basée sur le schéma générique décrit dans la section 4.3.

On est alors ramené à chercher la permutation conduisant à un ordonnancement minimisant la fonction γ. L'algorithme pour le problème restreint, permettant de passer d'une permutation π à l'ordonnancement correspondant, devient le suivant :

Algorithme. A4.
1 *Placer chaque travail au plus tôt sur la première machine sans interruption*
– $dispo_{m1} = 0$. Pour tout j entre 1 et n faire :
  – $C_{\pi(j),1} = dispo_{m1} + p_{\pi(j),1}$
2 *Calculer pour chaque travail j sa date de début au plus tôt r_j sur la deuxième machine*
– Pour tout j entre 1 et n faire :
  – $r_j = C_{\pi(j),1} + \theta_j^{min}$
3 *Trier les travaux dans l'ordre croissant de leur date de début au plus tôt sur la deuxième machine, défini par la permutation ϕ*
– Déterminer $\phi = (\phi(1), \phi(2), \ldots, \phi(n))$ telle que $r_{\phi(1)} \leq r_{\phi(2)} \leq \ldots \leq r_{\phi(n)}$
4 *Exécuter les travaux sur la deuxième machine dans l'ordre donné par ϕ, en les calant au plus tôt*
– $C_{\phi(1),2} = r_{\phi(1)} + p_{\phi(1),2}$
– Pour tout j entre 2 et n faire :
  – $C_{\phi(j),2} = max\{C_{\phi(j-1),2}, r_{\phi(j)}\} + p_{\phi(j),2}$

Remarques :
1- L'algorithme que nous avons présenté travaille avec les dates de fin des opérations $C_{j,k}$. De manière équivalente, on aurait pu procéder avec les dates de début $t_{j,k}$. Etant donné que le problème est non-préemptif, on peut passer d'un type de variables à l'autre grâce aux relations $\forall j, k C_{j,k} = t_{j,k} + p_{j,k}$.
2- L'étape [3] de tri n'est pas détaillée ici. Cependant, c'est cette étape qui contribue à la complexité de l'algorithme, puisqu'elle est réalisée en $O(n.log(n))$, tandis que toutes les autres étapes nécessitent $O(n)$ calculs.
3- Contrairement au cas rencontré pour les problèmes de flowshop de permutation, l'ajout d'un travail en fin de séquence partielle sur la première machine est susceptible de remettre en cause l'ordonnancement des travaux précédents, du moins sur la deuxième machine. Après avoir déterminé la date de début au plus tôt sur la deuxième machine pour ce travail, on est contraint de l'insérer dans l'ordonnancement partiel sur la seconde machine, ce qui peut conduire à décaler l'exécution de travaux précédemment placés.

e) Propositions de bornes inférieures pour le problème $F2|\theta_j^{min}|C_{max}$

Comme nous venons de le voir, le problème restreint pour la minimisation du makespan dans un flowshop à deux machines avec time lags minimaux peut être traité avec un algorithme polynomial, par exemple celui que nous proposons. Il est alors possible de construire une PSE pour la résolution du problème $F2|\theta_j^{min}|C_{max}$, avec un schéma de séparation analogue au schéma générique que nous employons pour les flowshops de permutation. Néanmoins, contrairement à ce

qui se passe lorsqu'on se limite aux ordonnancements de permutation, les opérations des travaux déjà placés, ne forment pas sur la seconde machine un profil qu'il suffit de compléter à partir de sa date de fin. En réalité, on ne connaît pour ces travaux que leur date de début au plus tôt sur la seconde machine. Leur exécution sur la seconde machine risque d'être retardée par des travaux qui n'ont pas encore été placés, mais qui peuvent être disponibles plus tôt en raison de faibles time lags. Par ailleurs, le développement d'une PSE nécessite en plus la conception de bornes inférieures spécifiques. Nous nous limitons pour l'instant au cas où les time lags sont positifs ou nuls.

Le problème $F2|\theta_j^{min}|C_{max}$ est abordé par Dell'Amico [Dell'Amico,96]. L'auteur propose des bornes inférieures globales, que nous notons avec un exposant D, ainsi que des heuristiques par construction et un algorithme de recherche tabou. Toutes ces méthodes représentent les solutions par la séquence de traitement des travaux sur la première machine, ce qui est possible d'après la propriété du problème restreint. Afin d'obtenir la solution optimale, Dell'Amico propose de traiter le problème comme un jobshop à n travaux à 3 opérations et $n+2$ machines : les première et troisième opération de chaque travail j correspondent aux opérations réelles de durée $p_{j,1}$ et $p_{j,2}$ et sont réalisées sur les machines 1 et $n+2$. Chaque machine k ($2 \leq k \leq n+1$) est utilisée pour exécuter uniquement la seconde opération du travail $k-1$, de durée θ_{k-1}^{min}, qui représente le time lag dans le problème initial. La méthode utilisée pour résoudre de manière exacte ce problème est la PSE de Brucker et al. [Brucker et Jurisch,92].

Les différentes bornes inférieures proposées sont les suivantes :

- LB_0^D : borne issue de la charge des machines. $LB_0^D = max\{\sum_{1\leq j\leq n} p_{j,1}, \sum_{1\leq j\leq n} p_{j,2}\}$
- LB_1^D : borne précédente et prise en compte de la longueur des travaux. $LB_1^D = max\{LB_0^D, max_{1\leq j\leq n}\{p_{j,1} + \theta_j^{min} + p_{j,2}\}\}$
- LB_R^D : borne précédente et relaxation de la contrainte de capacité sur la première machine (qui conduit au problème $R = 1|r_j|C_{max}$). $LB_R^D = max\{LB_1^D, C_{max}^\star(R)\}$, où $C_{max}^\star$ désigne le makespan optimal.
- LB_Q^D : borne LB_1^D et relaxation de la contrainte de capacité sur la seconde machine (qui conduit au problème $Q = 1|q_j|C_{max}$). $LB_Q^D = max\{LB_1^D, C_{max}^\star(Q)\}$
- LB_T^D : borne LB_1^D et relaxation des contraintes de time lags. On remplace dans l'instance le time lag θ_j^{min} de tout travail j par $min\{\theta_j^{min}, min_{1\leq i\leq n}\{p_{i,1} + \theta_i^{min}\}\}$. Dans ce cas, le problème (noté T) est un cas particulier pour lequel les ordonnancements de permutation sont dominants. Il peut donc être résolu avec l'algorithme de Mitten-Johnson [Mitten,59]. $LB_T^D = max\{LB_1^D, C_{max}^\star(T)\}$.

Dell'Amico fournit des garanties de performance théoriques pour ces bornes inférieures : pour toute instance I, et toute borne inférieure $L \in \{1, R, Q, T\}$, $LB_L^D(I)/C_{max}^\star(I) \geq 1/2$ et ce ratio est atteint. Autrement dit, l'optimum est au plus égal au double d'une borne inférieure.

D'après les expériences numériques qu'il a menées, pour des problèmes comportant jusqu'à 200 travaux, Dell'Amico constate que la combinaison de toutes ces bornes (qui consiste à calculer chacune d'entre elles et à conserver la plus grande valeur), fournit l'optimum dans 86 % des cas, avec un temps de calcul très faible. Il pourrait donc être intéressant de chercher si ces bornes inférieures globales peuvent être adaptées de manière à obtenir des bornes inférieures à appliquer à des ordonnancements partiels, au cours de l'exploration dans une PSE. On disposerait alors d'une méthode de résolution exacte qui pourrait être comparée à la PSE de Brucker et al. [Brucker et Jurisch,92].

D'autres bornes inférieures globales pour le problème $F2|\theta_j^{min}|C_{max}$ sont proposées par Yu [Yu,96]. Nous les présentons maintenant, en faisant figurer un exposant Y :

- LB_1^Y : borne issue de la charge des machines. $LB_1^Y = max\{\sum_{1\leq j\leq n} p_{j,1} + min_{1\leq i\leq n}\{\theta_i^{min} + p_{i,2}\}, min_{1\leq i\leq n}\{p_{i,1} + \theta_i^{min}\} + \sum_{1\leq j\leq n} p_{j,2}\}$. On peut remarquer que cette borne est plus précise que LB_0^D
- LB_2^Y : borne avec prise en compte de la longueur des travaux. $LB_2^Y = max_{1\leq j\leq n}\{p_{j,1} + \theta_j^{min} + p_{j,2}\}$
- LB_3^Y : borne avec découpage des travaux et time lags moyens. $LB_3^Y = \lceil(\sum_{1\leq j\leq n} u_j(\theta_j^{min} + v_j - 1))/\sum_{1\leq j\leq n} u_j\rceil + 1 + \sum_{1\leq j\leq n} u_j$, avec $u_j = min\{p_{j,1}, p_{j,2}\}$ et $v_j = max\{p_{j,1}, p_{j,2}\}$.
- LB_4^Y : autre borne avec découpage des travaux et time lags moyens. $LB_4^Y = \lceil(\sum_{1\leq j\leq n} \theta_j^{min} + \sum_{1\leq j\leq n} a_j + \sum_{1\leq j\leq n} b_j)/n\rceil$, où a_j (b_j) désigne la somme des j plus petites valeurs parmi les durées opératoires sur la première (seconde) machine

Les deux dernières bornes sont en fait obtenues grâce à des résultats sur le problème à durées unitaires $F2|p_{j,k} = 1, \theta_j^{min}|C_{max}$. Yu suggère également d'appliquer ces bornes à des sous-ensembles de travaux, ce qui peut conduire à une évaluation plus précise.

Toutes les bornes inférieures globales que nous venons de présenter peuvent être calculées à la racine de l'arbre de recherche dans une PSE, puisqu'elles sont valables pour tous les ordonnancements réalisables. Par contre, elles ne présentent pas d'intérêt direct dans l'évaluation d'un noeud au cours de l'exploration, dans la mesure où leur valeur ne prend pas en compte l'ordonnancement partiel correspondant à ce noeud. Nous proposons, dans un premier temps, trois bornes inférieures que l'on peut appliquer pour compléter un ordonnancement partiel, associé à la séquence de traitement σ de longueur i, sur la première machine :

- LB_1^F : borne avec relaxation de la contrainte de capacité sur la première machine. Pour tous les travaux figurant dans la séquence σ, on connaît la date de début au plus tôt sur la seconde machine, qui peut être assimilée à une date de disponibilité sur cette machine : $r_{\sigma(h),2} = C_{\sigma(h),1} + \theta_{\sigma(h)}^{min} = \sum_{1\leq g\leq h} p_{\sigma(g),1} + \theta_{\sigma(h)}^{min}$, pour $h \leq i$. Soit un travail s n'appartenant pas à σ (travail non encore placé). Sa date de début au plus tôt sur la première machine correspond à la date de fin du dernier travail de σ sur cette machine, c'est-à-dire $C_{\sigma(i),1} = \sum_{1\leq g\leq i} p_{\sigma(g),1}$. Par conséquent, sa date de disponibilité sur la seconde machine est $r_{s,2} = \sum_{1\leq g\leq i} p_{\sigma(g),1} + p_{s,1} + \theta_s^{min}$. On est ainsi ramené à résoudre un problème à une machine avec dates de disponibilité ($1|r_j|C_{max}$), dont une solution optimale est obtenue en traitant les travaux dans l'ordre croissant des dates de disponbilités. Cette borne inférieure repose sur le même principe que la borne globale LB_R^D
- LB_2^F : borne avec relaxation des time lags. Soit une permutation complète des travaux π, construite à partir de σ ($\forall h \leq i, \pi(h) = \sigma(h)$). Les dates de fin sur la première machine et les dates de début au plus tôt sur la seconde machine sont connues pour les travaux déjà placés : si $1 \leq h \leq i$, $C_{\pi(h),1} = \sum_{1\leq g\leq h} p_{\sigma(g),1}$ et $r_{\pi(h),2} = \sum_{1\leq g\leq h} p_{\sigma(g),1} + \theta_{\pi(h)}^{min}$. On détermine une borne inférieure pour la date de fin du travail placé en position $t > i$ en classant les travaux non encore placés dans l'ordre croissant des durées opératoires sur la première machine. Si ϕ désigne l'ordre correspondant, on a : $C_{\pi(t),1} \geq C_{\sigma(i),1} + \sum_{1\leq u\leq t-i} p_{\phi(u),1}$. Les dates de début au plus tôt sur la seconde machine sont alors obtenues en ajoutant aux valeurs précédentes le plus petit time lag minimal θ parmi ceux des travaux non placés : $r_{\pi(t),2} \geq C_{\sigma(i),1} + \sum_{1\leq u\leq t-i} p_{\phi_1(u),1} + \theta$. Le problème qui se pose alors est de savoir comment affecter ces dates de début au plus tôt aux travaux non encore placés. On peut facilement montrer que le makespan obtenu est minimisé si l'on place les travaux dans l'ordre décroissant de leur durée sur la seconde machine. Si ψ désigne cet ordre, on a :

$p_{\pi(t),2} = p_{\psi(t-i),2}$. La borne inférieure est alors déterminée en résolvant le problème à une machine $1|r_j|C_{max}$ avec les dates de disponibilité $r_{\pi(j),2}$ et les durées $p_{\pi(j),2}$, avec $1 \leq j \leq n$.

- LB_3^F : borne avec relaxation de la contrainte de capacité sur la seconde machine. Etant donné le choix que nous avons fait de travailler à partir de la séquence de traitement des travaux sur la première machine, il ne nous est pas possible de procéder de manière exactement symétrique à la borne LB_1^F. En effet, on ne connaît pas la position exacte sur la seconde machine des travaux déjà placés, on dispose seulement de leur date de début au plus tôt. Cependant, on peut tout de même obtenir une borne inférieure en relaxant la contrainte de capacité sur la seconde machine. Pour les travaux $\sigma(h)$ déjà placés (avec $1 \leq h \leq i$), la date de fin sur la seconde machine vérifie $C_{\sigma(h),2} \geq C_{\sigma(h),1} + \theta^{min}_{\sigma(h)} + p_{\sigma(h),2} = \sum_{1 \leq g \leq h} p_{\sigma(g),1} + \theta^{min}_{\sigma(h)} + p_{\sigma(h),2}$. Pour les travaux s non encore placés, on se ramène au problème $1|q_j|C_{max}$ pour lequel une solution optimale est obtenue en classant les travaux dans l'ordre décroissant des durées de latence. Ici, la durée de latence correspond à la somme du time lag minimal et de la durée opératoire sur la seconde machine : $q_{s,1} = \theta^{min}_s + p_{s,2}$. Ainsi, on traite les travaux non encore placés sur la première machine dans l'ordre décroissant des durées de latence et la plus grande date de fin obtenue (en prenant en compte également les travaux déjà placés) constitue la borne inférieure recherchée. Celle-ci repose sur le même principe que la borne globale LB_Q^D

On peut facilement vérifier que ces bornes, qui s'appliquent à des ordonnancements partiels, intègrent certaines des bornes globales présentées auparavant. En particulier, quel que soit la séquence partielle σ considérée, on a :

- $LB_2^F(\sigma) \geq LB_0^D$ et $max\{LB_1^F(\sigma), LB_3^F(\sigma)\} \geq LB_0^D$
- $max\{LB_1^F(\sigma), LB_2^F(\sigma)\} \geq LB_1^D$, $max\{LB_1^F(\sigma), LB_3^F(\sigma)\} \geq LB_1^D$ et $max\{LB_2^F(\sigma), LB_3^F(\sigma)\} \geq LB_1^D$
- $max\{LB_1^F(\sigma), LB_3^F(\sigma)\} \geq LB_R^D$ et $max\{LB_1^F(\sigma), LB_2^F(\sigma)\} \geq LB_R^D$
- $max\{LB_1^F(\sigma), LB_3^F(\sigma)\} \geq LB_Q^D$ et $max\{LB_2^F(\sigma), LB_3^F(\sigma)\} \geq LB_Q^D$
- $max\{LB_1^F(\sigma), LB_3^F(\sigma)\} \geq LB_1^Y$
- $LB_1^F \geq LB_2^Y$ et $LB_3^F \geq LB_2^Y$

Les bornes LB_3^Y et LB_4^Y proposées par Yu [Yu,96] et basées sur un découpage des travaux en durées unitaires semblent très intéressantes, mais leur adaptation de manière à ce qu'elles prennent en compte une séquence partielle de travaux déjà placés pose problème. En effet, contrairement à ce qui se passe dans le cas du flowshop de permutation, les opérations des travaux déjà placés sur la seconde machine ne forment pas un profil qu'il suffit de compléter à partir de sa date de fin. Le fait d'autoriser les solutions qui ne sont pas de permutation peut nous amener à utiliser des temps morts entre les opérations déjà placées sur la seconde machine, voire à retarder ces opérations.

Les bornes inférieures que nous avons proposées pourraient être incorporées dans une Procédure par Séparation et Evaluation suivant le même schéma de branchement que le schéma générique que nous utilisons pour les problèmes de flowshop de permutation, mais avec un recours à l'algorithme de la section 4.8.3 pour la construction de l'ordonnancement à partir de la séquence de traitement des travaux sur la première machine. En générant différentes classes de jeux d'essais, on pourrait ainsi comparer les performances de chaque borne et éventuellement identifier les situations pour lesquelles elles sont les plus efficaces.

4.9 Conclusion du chapitre

Ainsi, l'analyse du problème restreint, qui consiste à déterminer le meilleur ordonnancement respectant une séquence de traitement donnée, nous a permis de mettre en place une approche de résolution générique pour les problèmes de flowshop de permutation avec time lags. Celle-ci repose sur une Procédure par Séparation et Evaluation dont le schéma de branchement est similaire à celui qu'Ignall et Schrage [Ignall et Schrage,65] ont proposé pour le flowshop de permutation classique, et qui consiste à construire progressivement la séquence de traitement par ajout de travail en fin de séquence partielle. Nous avons décliné cette méthode afin de résoudre plusieurs problèmes ($Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|\sum w_k MC_k$, $F2|no-wait, NSDST, NSDRT|L_{max}$ et $Fm_\pi|\theta_{j,k}^{min} = \theta_{j,k}^{max}|L_{max}$), pour lesquels nous avons proposé et testé des bornes inférieures spécifiques, ainsi que des conditions de dominance dans certains cas. L'efficacité de ces méthodes a été testée grâce à des séries d'expériences numériques, réalisées sur plusieurs familles de jeux d'essais. Les différents paramètres pris en compte pour la génération de celles-ci, que nous avons sélectionnés de manière à représenter des situations réalistes, nous ont également conduit à distinguer les cas en fonction de leur difficulté.

Comme l'a montré l'étude bibliographique exposée dans le chapitre 3, aucun algorithme de résolution exacte n'avait été proposé jusqu'à maintenant pour ces problèmes spécifiques, qui sont pourtant susceptibles d'être rencontrés dans la pratique. Nous avons volontairement privilégié une approche de résolution exacte, qui, étant donné que ces problèmes sont tous NP-difficiles au sens fort, ne permet de traiter que des instances de taille réduite. Néanmoins, les procédures que nous avons développées constituent une base de travail intéressante, étant donné qu'elles permettent d'évaluer, sur de petits jeux d'essais, la qualité des solutions fournies par des heuristiques, applicables pour des instances de taille industrielle.

En ce qui concerne les méthodes de résolution approchées, outre les heuristiques simples par construction que nous employons pour obtenir des bornes supérieures initiales de bonne qualité, on pourrait envisager d'adopter des métaheuristiques conçues pour des problèmes où la solution est représentée par une permutation, qu'il s'agisse d'algorithmes génétiques, de recherche tabou ou de recuit simulé. Avant de débuter cette thèse, nous avions programmé un algorithme génétique simpliste pour le problème de flowshop de permutation avec time lags maximaux uniquement. Ce travail n'a pas été poursuivi, en raison des recherches développées parallèlement dans l'équipe, pour des problèmes plus généraux, et faisant appel à des méthodes évolutionnistes plus sophistiquées (voir [Deppner,04]).

Par ailleurs, nous avons présenté plusieurs extensions possibles de ce travail : la prise en compte de contraintes additionnelles de type dates de disponibilités et durées de latence ne pose aucun problème, dans la mesure où celles-ci peuvent être modélisées à l'aide de time lags. D'autre part, une simple modification de l'algorithme de traitement du problème restreint permet de considérer des time lags généralisés, définis entre des couples d'opérations quelconques au sein des travaux. Enfin, il est également possible d'utiliser une approche de résolution analogue pour le problème $F2|\theta_j^{min}|C_{max}$, en travaillant sur la séquence de traitement des travaux sur la première machine. Néanmoins, il faut dans ce cas développer un nouvel algorithme pour le problème restreint ainsi que des bornes inférieures spécifiques. Nous proposons tous ces éléments, sans avoir pu pour autant les implémenter et les évaluer sur des jeux d'essais.

Les autres problèmes de flowshop général, où l'on ne se limite pas aux solutions de permu-

tation ($m \geq 3$ ou présence de time lags maximaux), ne peuvent pas être résolus avec une telle approche. La construction d'une Procédure par Séparation et Evaluation pour ces problèmes nécessite un schéma de séparation différent : plutôt que de représenter l'ensemble des séquences de traitement sur toutes les machines, il semble plus judicieux d'employer un schéma reposant sur le modèle du graphe disjonctif, comme c'est le cas pour les problèmes de jobshop classiques ou pour les problèmes à une machine avec time lags (voir la section 3.2.1).

Chapitre 5

Bilan et perspectives

Résumé

Dans ce chapitre, nous proposons un bilan de notre travail, qui récapitule les points principaux que nous avons mis en évidence. Puis plusieurs perspectives de poursuite de ce travail sont évoquées, à la fois sur un plan théorique et sur un plan pratique.

Sommaire

5.1 Bilan de l'étude

Le travail que nous venons de présenter nous permet de mieux appréhender les problèmes d'ordonnancement de type flowshop de permutation, en présence de contraintes de time lags minimaux et maximaux. Ce type de contraintes, qui généralisent les contraintes de précédence classiques entre opérations d'un même travail, bien qu'introduites depuis une cinquantaine d'années, n'a pas fait l'objet de beaucoup d'attention par le passé, malgré toutes les possibilités qu'elles offrent au niveau de la modélisation. Ce n'est que depuis une dizaine d'années que les publications sur le sujet ont commencé à se développer, dans le cadre de l'ordonnancement de projet dans un premier temps, puis dans le domaine de l'ordonnancement d'atelier (voir chapitre 3).

Ces études répondent avant tout à un besoin issu du monde industriel, pour lequel les modèles classiques en ordonnancement ne sont pas satisfaisants. Comme le montre le large panel d'applications pratiques mentionnées dans la section 1.5, de nombreux processus industriels font intervenir des contraintes d'écarts, que ce soit dans l'industrie chimique et pharmaceutique avec l'utilisation de composés réactionnels instables, dans l'agro-alimentaire avec toutes les normes de sécurité concernant la conservation des produits périssables, dans la sidérurgie avec les temps de refroidissement ou de réchauffement du métal entre les différents traitements qu'il subit ou dans bien d'autres cas encore. Par ailleurs, le concept de time lags permet également de représenter d'autres contraintes couramment rencontrées en ordonnancement, conduisant ainsi à un modèle unifié : dates de disponibilité des travaux, lorsque le système de production est approvisionné par un atelier situé en amont, durées de latence, dues à des livraisons par exemple, ou temps de montage et démontage qui utilisent la machine sans que le produit soit présent.

Face aux nombreuses configurations possibles de l'atelier, nous nous sommes volontairement limités au cas du flowshop de permutation, qui présente l'avantage d'être à la fois relativement simple sur le plan structurel (tous les travaux suivent la même gamme opératoire et toutes les machines traitent les travaux dans le même ordre) et assez courant dans la pratique (production en ligne, sans dépassement autorisé des travaux entre les machines). Nous avons également restreint la définition des time lags en ne considérant que des contraintes définies entre les opérations consécutives au sein des travaux. Pourtant, l'analyse de la complexité de ce type de problèmes d'ordonnancement en présence de time lags nous indique que la plupart d'entre eux sont NP-difficiles au sens fort et nécessitent des méthodes génériques ou spécifiques relativement sophistiquées si l'on souhaite les résoudre efficacement. On peut regretter le fait de s'être limité à des solutions de permutation, ce qui ne garantit plus l'optimalité de la solution, si l'on se place dans l'ensemble de toutes les solutions, et ceci même pour deux machines. Néanmoins, des expériences, non présentées dans ce mémoire, de comparaison entre les algorithmes génétiques développés par Deppner [Deppner,04], sur des flowshops généraux, et nos méthodes exactes, qui n'autorisent que des solutions de permutation, ont montré des gains relatifs faibles en faveur des ordonnancements pour lesquels on accepte le dépassement des travaux. Mais nous ne pouvons pas réellement en tirer une conclusion, puisque même si les algorithmes génétiques mentionnés paraissent efficaces, ils ne garantissent pas l'optimalité de leurs solutions.

Il ressort de l'étude approfondie de la littérature concernant les problèmes d'ordonnancement impliquant des time lags, que nous avons menée, que l'essentiel des approches de résolution consacrées à ces problèmes reposent sur des méthodes heuristiques, c'est-à-dire pour lesquelles on n'a pas de garantie que le résultat final soit optimal. Si, du point de vue pratique, on peut

comprendre que la recherche de l'optimum ne présente pas un intérêt énorme, d'autant plus lorsque celle-ci s'accompagne d'un temps de calcul extrêmement long, la conception de méthodes de résolution exactes peut se révéler utile, dans la mesure où celles-ci permettent d'évaluer précisément les performances des heuristiques mentionnées précédemment. Les seuls travaux portant sur des méthodes exactes pour les problèmes d'ordonnancement avec time lags sont ceux de Brucker et al. [Brucker et al.,99b] pour les problèmes à une machine, et les articles consacrés à l'ordonnancement de projet (voir section 3.6). Bien qu'elles puissent être utilisés pour des problèmes de flowshop, les quelques tentatives menées dans ce sens indiquent que ces méthodes ne sont pas adaptées aux spécificités des problèmes d'ordonnancement d'atelier à plusieurs machines [Brucker et al.,99b][Bartusch et al.,88]. Pour toutes ces raisons, nous nous sommes tournés vers une approche de résolution exacte, basée sur une Procédure par Séparation et Evaluation.

Nous avons utilisé un schéma générique, valable pour tous les problèmes de flowshop de permutation avec time lags, à partir du moment où le critère à optimiser est régulier, c'est-à-dire croissant selon chaque date de fin des travaux. L'analyse du problème restreint, pour lequel on recherche un ordonnancement optimal respectant une séquence de traitement donnée, nous a permis de travailler avec les permutations de travaux en lieu et place des ordonnancements eux-mêmes. Ce résultat conduit directement à utiliser le schéma de séparation classique pour le flowshop, proposé par Ignall et Schrage [Ignall et Schrage,65], ainsi que des heuristiques qui s'appliquent à des permutations. Nous avons en particulier adapté la procédure NEH [Nawaz et al.,83] de manière à ce qu'elle prenne en compte les contraintes additionnelles. Parmi toutes les stratégies d'exploration envisageables, nous en avons privilégié une qui s'apparente à une recherche en profondeur d'abord, ce qui permet de parvenir rapidement à une solution complète.

Le schéma générique a été décliné afin de résoudre quatre problèmes distincts :
- minimisation du makespan avec time lags quelconques $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$
- minimisation de la somme pondérée des dates de fin sur les machines avec time lags quelconques $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|\sum w_k MC_k$ (il s'agit d'un critère non classique, lié à l'utilisation des ressources et pour lequel nous avons aussi fourni de nouveaux résultats de complexité)
- minimisation du plus grand retard sur deux machines, avec traitement sans attente et temps de montage et démontage séparés $F2|no-wait, NSDST, NSDRT|L_{max}$
- minimisation du plus grand retard avec time lags exacts $Fm_\pi|\theta_{j,k}^{min} = \theta_{j,k}^{max}|L_{max}$, qui généralise le problème précédent

Nous avons développé des bornes inférieures spécifiques ainsi que des relations de dominance pour certains d'entre eux. Tous les éléments des PSE obtenues ont fait l'objet d'études expérimentales, sur des jeux d'essais générés aléatoirement. Lors de cette génération, nous avons sélectionné un certain nombre de paramètres de manière à représenter des situations réalistes, correspondant à des familles d'instances.

5.2 Poursuite du travail

L'ensemble des travaux présentés dans ce mémoire ne doit pas être vu comme une fin en soi. Bien que les résultats qui y sont établis constituent une base importante pour l'étude des problèmes d'ordonnancement de type flowshop en présence de contraintes d'écarts, les réflexions que nous avons menées jusqu'à présent ont aussi fait émerger de nombreuses questions qui restent à traiter. Cette section est consacrée à la description de plusieurs pistes de recherche qui se situent

dans le prolongement de notre travail, que nous avons choisi de regrouper en trois catégories : celles qui concernent des résultats de complexité, celles qui portent sur l'approche de résolution employée et enfin celles qui correspondent à de nouveaux problèmes. Pour chacune d'elles, nous présentons brièvement le problème posé ainsi que les premiers éléments de réponse que nous pouvons apporter.

5.2.1 Perspectives concernant de nouveaux résultats de complexité

En ce qui concerne la complexité des problèmes de flowshop avec time lags, il reste de nombreux problèmes ouverts. Certains sont explicitement mentionnés dans les tableaux récapitulatifs à la fin du chapitre 2. En outre, les recherches concernant la prise en compte combinée de time lags et de machines à traitement par fournées méritent d'être approfondies. Nous avons jusqu'à maintenant étudié uniquement le cas à deux machines, dont la première est une machine à traitement parallèle et la seconde est classique, en présence de time lags minimaux uniquement ($F2|p-batch(1), \theta_j^{min}|C_{max}$). La complexité du problème où la capacité est illimitée reste ouverte dans le cas général (il existe un algorithme de programmation dynamique en $O(n^2)$ en l'absence de time lags [Potts et al.,01]), même si nous avons présenté deux cas particuliers polynomiaux.

Si l'on modifie les caractéristiques des machines, on se retrouve face à des problèmes assez différents, dont la complexité n'est pas nécessairement la même. La prise en compte de time lags maximaux est également envisageable, mais dans ce cas la définition de ces contraintes nécessitent une attention particulière, dans la mesure où les dates de début et de fin sur des machines à traitement par fournées sont les mêmes pour tous les travaux regroupés au sein d'une fournée. Etant donné que la plupart des problèmes de flowshop avec time lags et machines classiques sont déjà NP-difficiles au sens fort, il risque d'en être de même lorsqu'on considère des machines à traitement par fournées. Néanmoins, il peut être intéressant de rechercher des cas particuliers polynomiaux, qui pourraient ensuite conduire à des bornes inférieures ou à des méthodes approchées.

5.2.2 Perspectives concernant l'approche de résolution proposée

Comme nous pouvons le constater avec les résultats expérimentaux obtenus sur les différents problèmes que nous avons étudiés, le choix d'adopter une méthode de résolution exacte limite considérablement la taille des problèmes que l'on peut espérer traiter en un temps de calcul raisonnable. Etant donné la combinatoire du problème, qui, du fait de la restriction aux ordonnancements de permutation, est en $n!$, le nombre m de machines n'est pas spécialement critique (la complexité de l'algorithme pour la construction d'un ordonnancement semi-actif à partir d'une séquence de traitement est en $O(m.n)$). C'est essentiellement le nombre n de travaux qui détermine la taille maximale des instances que l'on peut résoudre. Le fait de recourir à des bornes inférieures plus sophistiquées (pour le problème $Fm_\pi|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$), ou à des règles de dominance (par exemple pour le problème $Fm_\pi|\theta_{j,k}^{min} = \theta_{j,k}^{max}|L_{max}$) n'apportent pas de gain suffisant pour permettre d'augmenter cette taille maximale. Nous envisageons d'analyser le comportement de notre approche de résolution sur des instances avec dates de disponibilité r_j et durées de latence q_j. Comme nous l'avons vu dans la section 4.8.1, les méthodes que nous avons développées sont capables de traiter ces instances sans qu'on y apporte la moindre modification. L'existence de dates de disponibilités et de durées de latence plus ou moins dispersées pourrait conduire à limiter le nombre de solutions potentiellement intéressantes et donc permettre de résoudre des instances de plus grande taille. De la même manière que pour d'autres travaux de

l'équipe MACSI, il serait également possible d'adopter une méthode approchée par décomposition temporelle, c'est-à-dire une méthode de plan glissant, où les travaux seraient regroupés en paquets. Au sein de chaque paquet, un ordonnancement optimal serait déterminé grâce à la PSE, et les ordonnancements partiels obtenus seraient ensuite juxtaposés afin d'obtenir une solution de bonne qualité. Il conviendrait alors de réfléchir sur la manière de répartir les travaux dans les paquets et sur la taille maximale de ceux-ci. Cette méthode de décomposition n'est qu'une heuristique, pour laquelle on perd l'assurance d'obtenir une solution optimale, mais on pourrait chercher à améliorer certains éléments de notre PSE, comme la borne inférieure, ou le schéma de séparation.

5.2.3 Perspectives concernant l'étude d'autres problèmes

a) Etudes d'autres critères

Les problèmes pour lesquels nous avons développé des méthodes de résolution sont tous des problèmes de flowshop de permutation, avec time lags minimaux et maximaux, et comme objectif le makespan (C_{max}), le plus grand retard (L_{max}) ou la somme pondérée des dates de fin sur les machines ($\sum w_k MC_k$). Il serait intéressant de chercher à adapter notre approche de résolution pour les mêmes problèmes impliquant d'autres critères, notamment des critères sous forme de somme (éventuellement pondérée) de fonctions des dates de fin des travaux : la somme des dates de fin des travaux ($\sum C_j$), la somme des travaux en retard ($\sum U_j$) ou la somme des retards ($\sum T_j$). Puisque ces critères sont réguliers, le problème restreint est polynomial, comme nous l'avons montré dans la section 4.3.1, et l'algorithme permettant de construire un ordonnancement semi-actif correspondant à une séquence de traitement des travaux reste valable. Le principal travail concerne le développement de bornes inférieures efficaces, et éventuellement de règles de dominance additionnelles. Pour cela, il convient de rechercher des cas particuliers polynomiaux, auxquels on est susceptible de se ramener par relaxation de contraintes.

b) Problèmes de flowshop généraux

La section 4.8.3 concerne l'extension possible de notre approche de résolution pour des problèmes de flowshop où l'on ne se limite plus aux ordonnancements de permutation. Comme nous l'avons vu, il est possible d'utiliser le même schéma de séparation, proposé par Ignall et Schrage [Ignall et Schrage,65], pour le problème à deux machines avec time lags minimaux uniquement ($F2|\theta_j^{min}|C_{max}$). Pour ce problème, nous avons proposé plusieurs bornes inférieures, qui généralisent les bornes globales de Dell'Amico [Dell'Amico,96] dans la mesure où elles peuvent être appliquées pour compléter une séquence partielle de travaux fixée sur la première machine. Nous envisageons de poursuivre ce travail en cherchant à généraliser également les bornes de Yu [Yu,96], et en intégrant toutes ces bornes inférieures à une PSE. Il serait alors intéressant de tester la taille maximale des instances que cette méthode pourrait résoudre optimalement, mais aussi de comparer les performances des différentes bornes, et de rechercher en particulier pour quel type de problèmes elles sont efficaces.

En présence de time lags maximaux, le problème restreint, tel que nous l'avons défini (à partir de la séquence de traitement sur la première machine), devient NP-difficile au sens fort, même pour deux machines. Cependant, si on prend en compte non seulement la séquence des travaux sur la première machine mais aussi celle sur la seconde machine, le nouveau problème restreint est polynomial. En effet, on dispose alors de toutes les relations de précédence et un

calcul de plus long chemin dans le graphe totalement arbitré permet d'obtenir l'ordonnancement au plus tôt, si la solution est réalisable. Dans le cas contraire, l'algorithme permet de détecter la présence d'un circuit de longueur strictement positive, ce qui traduit la non faisabilité de la solution [Bartusch et al.,88]. On peut alors concevoir, pour le problème $F2|\theta_j^{min}, \theta_j^{max}|C_{max}$, un schéma de séparation dans lequel chaque noeud de l'arborescence est associé à un couple de séquences partielles (σ_1, σ_2) contenant les travaux déjà placés sur la première et la seconde machine. En supposant que l'on n'ajoute à l'ordonnancement partiel que des opérations dont le prédécesseur éventuel est déjà placé, tous les éléments figurant dans σ_2 doivent aussi appartenir à σ_1. Bien que le nombre total de couples de permutation possibles soit très élevé $((n!)^2)$, un grand nombre d'entre eux correspondent probablement à des solutions non réalisables, du fait des contraintes de time lags maximaux. Un élément important de la PSE consisterait alors à détecter cette situation le plus tôt possible dans l'arborescence. Par exemple, lors de l'ajout de l'opération du travail j sur la seconde machine, si le time lag maximal θ_j^{max} est violé, et que toutes les opérations à partir de j sur la première machine sont tassées, c'est-à-dire exécutées sans temps mort, on peut prouver que les séquences partielles ne conduisent à aucune solution réalisable, et éliminer ainsi le sous-arbre issu du noeud courant. Par ailleurs, il convient aussi de réfléchir aux bornes inférieures à employer pour obtenir une PSE performante.

c) Problèmes avec machines à traitement par fournées

Jusqu'à maintenant, nous nous sommes contentés d'étudier la complexité de quelques problèmes de ce type. Mis à part certains cas très particuliers, la plupart de ces problèmes sont NP-difficiles. Il peut donc être intéressant de mettre au point des procédures de résolution capables de les traiter efficacement. Dans un premier temps, nous avons cherché à adapter la méthode NEH [Nawaz et al.,83] de manière à prendre en compte les contraintes de traitement par fournées, pour le problème $F2|p - batch(1), 1 < b < n, \theta_j^{min}|C_{max}$. Le travail devrait se poursuivre avec l'élaboration d'une méthode exacte de type PSE. Il est nécessaire que le schéma de séparation, ainsi que les bornes inférieures utilisées, intègrent judicieusement les spécificités du problème pour conduire à un algorithme performant.

d) Le problème du flowshop en anneau

Ce problème est issu du domaine des réseaux de communication. On considère un réseau optique organisé en anneau, auquel sont reliés m noeuds, qui cherchent à s'échanger des paquets de données. Ceux-ci sont acheminés sans interruption jusqu'à leurs destinations, via les slots de l'anneau. L'objectif est d'envoyer l'ensemble des paquets en un minimum de temps.

Amet et al. [Amet et al.,05a] modélisent ce problème et montrent qu'il est équivalent à la minimisation du makespan dans un atelier de type jobshop ayant une structure particulière, en présence de contraintes "sans attente avec recouvrement entre opérations consécutives". En réalité, la structure de l'atelier est celle d'un flowshop généralisé, dans lequel chaque travail j, associé à un paquet visite toutes les machines situées entre OR_j et $DEST_j - 1$ ou $DEST_j$ (selon que le noeud qui reçoit le paquet puisse ou non émettre un autre paquet en parallèle dans le même slot), où OR_j et $DEST_j$ désignent respectivement les indices des noeuds origine et destination du paquet j. La configuration en anneau permet de passer de la machine m à la machine 1 (ce qui est le cas des paquets j pour lesquels $OR_j > DEST_j$). Les contraintes additionnelles proviennent de l'acheminement sans interruption des paquets et correspondent à des time lags exacts, c'est-à-dire des time lags minimaux et maximaux égaux. La valeur du time lag de début à début entre

deux machines dépend uniquement du nombre de slots séparant les noeuds correspondant. Par conséquent, elle est indépendante du paquet. Enfin, la durée de traitement d'un travail sur une machine représente la taille du paquet en nombre de slots et ne dépend donc que du travail, et pas de la machine. La figure 5.1 illustre un exemple d'ordonnancement avec 4 travaux sur un flowshop en anneau à 4 machines, où ω_k représente le time lag de début à début (*start-start*) entre la machine k et la machine $k+1$ (modulo m).

Il est possible d'ajouter des dates de disponibilités r_j et des durées de latence q_j, ce qui rend compte de l'arrivée progressive de travaux dans le système et de délais supplémentaires à ajouter en fin de traitement (en particulier pour représenter la réception d'un paquet lorsqu'elle effectuée en parallèle avec une émission d'un autre paquet).

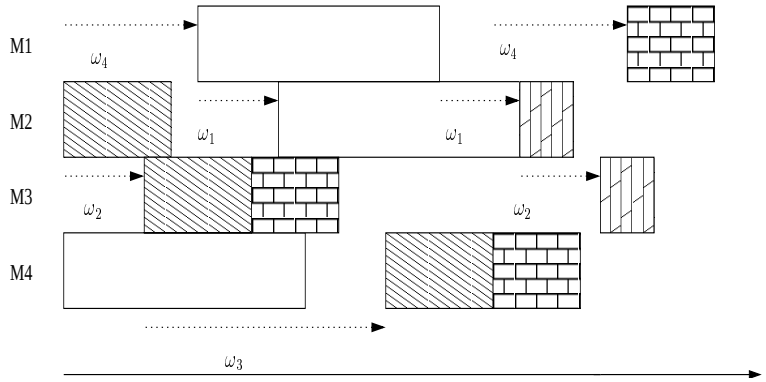

FIG. 5.1 – Exemple d'ordonnancement sur un flowshop en anneau

Plusieurs approches de résolution ont déjà été proposées par des membres de l'équipe MACSI [Amet et al.,05b]. La résolution d'un programme linéaire en nombres entiers avec un solveur ne permet pas de traiter en un temps raisonnable des problèmes comportant plus de 10 à 12 travaux. Le nombre de machines n'a quant à lui qu'une influence linéaire sur la durée d'exécution. Le programme linéaire a été intégré au sein d'une méthode par décomposition temporelle, avec une technique de plan glissant. Une métaheuristique basée sur un algorithme génétique a également été testée.

Nous envisageons d'étendre notre schéma de PSE à ce problème, en suivant les idées préliminaires que voici. Un pré-traitement du problème peut consister à renuméroter les machines, de manière à minimiser le nombre de travaux "en deux parties", c'est-à-dire passant de la machine m à la machine 1. Etant donné qu'il y a m numérotations possibles, cette étape n'est pas critique en termes de complexité. Si le problème qui en résulte est tel qu'aucun travail ne va d'une machine k_1 jusqu'à une machine $k_2 < k_1$ d'indice plus petit, alors on est ramené à un flowshop classique, avec opérations manquantes et time lags exacts, que notre PSE est capable de résoudre. Si ce n'est pas le cas, nous envisageons de traiter de manière spécifique les travaux "en deux parties" : en relaxant la contrainte de time lag entre les deux parties (opérations de la machine k_1 à la machine m, et opérations de la machine 1 à la machine k_2), on obtient deux travaux distincts j_1 et j_2, liés par une contrainte de précédence. On impose alors que le travail j_2 correspondant à la seconde partie ne soit pas placé avant le travail j_1 correspondant à la première partie. D'autre part, on tient compte de l'écart exact entre j_1 et j_2 au moment de placer ce dernier. Cette technique est proche de celle qu'emploie Guinet pour transformer un problème de jobshop classique

en flowshop avec contraintes de précédence entre travaux [Guinet et Legrand,98], [Guinet,00].

La question que l'on est amené à se poser concerne la possible adaptation de notre schéma générique de résolution à ce problème particulier. Sans la structure particulière en anneau, seules des solutions "de permutation" seraient réalisables, du fait des contraintes de time lags spécifiques à ce problème. Par solution de permutation, on entend solutions dans lesquelles l'ordre relatif des travaux les uns par rapport aux autres est le même d'une machine à l'autre : si i précède j sur une machine k, alors il en est de même pour toutes les machines sur lesquelles i et j sont traités. Mais à cause de la structure en anneau, il est possible d'avoir des solutions pour lesquelles cette propriété n'est pas vérifiée, comme on peut le voir sur la figure 5.1 (avec par exemple, le travail blanc et le travail avec des hachures obliques). Les séquences de traitement n'étant pas les mêmes sur les différentes machines, il n'est donc pas possible d'utiliser le schéma de séparation générique tel que nous l'employons pour le flowshop de permutation. Il faut alors s'interroger sur la manière de représenter l'ensemble des solutions, ou du moins, un ensemble dominant, et sur le schéma de branchement correspondant, de telle sorte que la recherche aboutisse bien à une solution optimale.

Une fois la PSE entièrement développée, il serait intéressant de tester son efficacité, en fonction du type de problèmes considérés, et de comparer ses performances à celles du programme linéaire déjà élaboré. Comme nous l'avons fait pour le problème $Fm_{\pi}|\theta_{j,k}^{min}, \theta_{j,k}^{max}|C_{max}$, on pourrait aussi concevoir des méthodes approchées, de type approximation à paramètre ϵ et beam search, à partir de la PSE.

Une autre application possible, pour ce problème, provient des ateliers flexibles de production. On peut en effet imaginer un tel atelier automatisé, au centre duquel est installé un convoyeur en anneau permettant d'alimenter les différentes stations de travail. Chaque travail dans le modèle précédent correspondrait alors à un déplacement d'un produit sur le convoyeur, et le nombre d'opérations dans la gamme multiplierait d'autant le nombre de travaux à considérer. Ainsi, ce modèle conduirait vraisemblablement à des instances de très grande taille, par rapport aux modèles existant pour ce type de problèmes dans la littérature (jobshop avec système de transport).

5.2.4 Bilan des perspectives

Les perspectives que nous avons exposées précédemment sont motivées par la volonté de se rapprocher le plus possible des problèmes rencontrés dans le monde industriel, en restant dans le domaine de l'ordonnancement d'atelier. D'une manière générale, on peut constater une tendance à incorporer d'avantage de contraintes réalistes dans les travaux de recherche actuels dans ce domaine ([Deppner,04], [Ruiz et Serifoglu,05a], [Ruiz et Serifoglu,05b]). Nous rappelons que c'est déjà cette idée qui est à l'origine du travail mené dans l'équipe MACSI sur les problèmes d'ordonnancement en présence de contraintes d'écarts temporels, dans un premier temps sous la forme d'une thèse CIFRE en entreprise. Les principes qui ont guidé la génération des jeux d'essais que nous avons employés s'inscrivent également dans ce cadre. De quoi rapprocher ce domaine de son but initial : aider les entreprises à gérer plus efficacement leur système de production, au niveau opérationnel.

Bibliographie

[Achugbue et Chin,82] J.O. Achugbue et F.Y. Chin. Complexity and solution of some three-stage flow-shop scheduling problem. *Mathematics of Operations Research* 7 : 532-544, 1982.

[Adams et al.,88] J. Adams, E. Balas et D. Zawack. The Shifting Bottleneck Procedure for job shop scheduling. *Management Science* 34(3) : 391-401, 1988.

[Ahmadi et al,92] J.H. Ahmadi, R.H. Ahmadi, S. Dasu et C.S. Tang. Batching and scheduling jobs on batch and discrete processors. *Operations Research* 39(4) : 750-763,1992.

[Ahr et al.,04] D. Ahr, J. Bekesi, G. Galambos, M. Oswald et G. Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research* 59 : 193-203, 2004.

[Aldowaisan et Allahverdi,98] T. Aldowaisan et A. Allahverdi. Total flowtime in no-wait flowshops with separated setup times. *Computers and Operations Research* 25 : 757-765, 1998.

[Allahverdi et Aldowaisan,00] A. Allahverdi et T. Aldowaisan. No-wait and separate setup three-machine flowshop with total completion time criterion. *International Transactions in Operational Research* 7 : 245-264, 2000.

[Allahverdi et Aldowaisan,01] A. Allahverdi et T. Aldowaisan. Minimizing total completion time in a no-wait flowshop with sequence-dependent additive changeover times. *Journal of the Operational Research Society* 52 : 449-462, 2001.

[Allahverdi et al.,99] A. Allahverdi, J.N.D. Gupta et T. Aldowaisan. A review of scheduling research involving setup considerations. *OMEGA* 27 : 219-239, 1999.

[Amet et al.,05a] H. Amet, J. Cohen, F. Deppner, M.-C. Portmann et S. Rousseau. Un problème d'ordonnancement de messages : Partie 1 - Modélisations. *Sixième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* 54-55, 2005.

[Amet et al.,05b] H. Amet, J. Cohen, F. Deppner, M.-C. Portmann et S. Rousseau. Un problème d'ordonnancement de messages : Partie 2 - Approches de résolution. *Sixième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* 56-57, 2005.

[Baker,74] K.R. Baker. *Introduction to sequencing and scheduling.* John Willey and sons, 1974.

[Baker,90] K.R. Baker. Scheduling groups of jobs in the two-machine flow shop. *Mathematical and Computer Modelling* 13(3) : 29-36, 1990.

[Balas et al.,95] E. Balas, J.K. Lenstra et A. Vazacopoulos. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science* 41(1) : 94-109, 1995.

[Bartusch et al.,88] M. Bartusch, R.H. Mohring et F.J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research* 16 : 201-240, 1988.

[Bellmann et al.,82] R. Bellmann, A.O. Esogbue et I. Nabeshima. *Mathematical aspects of scheduling and applications*. Pergamon Press, 1982.

[Bertolissi,00] E. Bertolissi. Heuristic algorithm for scheduling in the no-wait flow-shop. *Journal of Materials Processing Technology* 107 : 459-465, 2000.

[Bierwirth,95] C. Bierwirth. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spektrum* 17 : 87-92, 1995.

[Blazewicz et al.,92] J. Blazewicz, P. Dell'Olmo, M. Drozdowski et M.G. Speranza. Scheduling multiprocessor tasks on three dedicated processors. *Information Processing Letters* 41 : 275-280, 1992.

[Blazewicz et al.,01] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt et J. Weglarz. *Scheduling Computer and Manufacturing Processes. Second Edition*. Springer Verlag, 2001.

[Botta-Genoulaz,00] V. Botta-Genoulaz. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *International Journal of Production Economics* 64 : 101-111, 2000.

[Botta et Guinet,96] V. Botta et A. Guinet. Scheduling flowshops with precedence constraints and time lags. *Workshop on Production Planning and Control* 16-19, 1996.

[Bouquard,04] J.-L. Bouquard. Problèmes d'ordonnancement avec périodes d'indisponibilité et application à l'ordonnancement de l'algèbre Max-Plus. Habilitation à Diriger des Recherches. Université François Rabelais, Tours. 2004.

[Brucker et al. 99a] P. Brucker, A. Drexl, R. Mohring, K. Neumann et E. Pesch. Rescource-constrained project scheduling : notation, classification, models and methods. *European Journal of Operational Research* 112 : 3-41, 1999.

[Brucker et al.,98] P. Brucker, A. Gladki, H. Hoogeveen, M.Y. Kovalyov, C. Potts, T. Tautenhahn et S.L. Van De Velde. Scheduling a batching machine. *Journal of Scheduling* 1 : 31-54, 1998.

[Brucker et al.,99b] P. Brucker, T. Hilbig et J. Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time lags. *Discrete Applied Mathematics* 94 : 77-99, 1999.

[Brucker et Jurisch,92] P. Brucker et B. Jurisch. Job-shop (C codes). *European Journal of Operational Research* 57 : 132-133, 1992.

[Brucker et al.,94] P. Brucker, B. Jurisch et B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49 : 107-127, 1994.

[Brucker et Knust] P. Brucker et S. Knust. Complexity results for scheduling problems. *page web* http ://www.mathematik.uni-osnabrueck.de/research/OR/class/

[Brucker et Knust,99] P. Brucker et S. Knust. Complexity results for single-machine problems with positive finish-start time-lags. *Computing* 63 : 299-316, 1999.

[Brucker et Knust,03] P. Brucker et S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research* 149 : 302-313, 2003.

[Brucker et al.,04a] P. Brucker, S. Knust, T.C.E. Cheng et N.V. Shakhlevich. Complexity results for flow-shop and open-shop scheduling problems with transportation delays. *Annals of Operations Research* 129 : 81-106, 2004.

[Brucker et al.,03] P. Brucker, S. Knust et C. Oguz. Scheduling chains with identical jobs and constant delays on a single machine. *Osnabrucker Schriften zur Mathematik* 250, 2003.

[Brucker et al.,04b] P. Brucker, S. Knust et C. Oguz. Scheduling chains with identical jobs and constant delays on a single machine. *Ninth International Workshop on Project Management and Scheduling* Nancy, 119-122, 2004.

[Brucker et Schlie,90] P. Brucker et R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing* 45 : 369-375, 1990.

[Burns et Rooker,75] F. Burns et J. Rooker. A special case of the $3 \times n$ flow-shop problem. *Naval Research Logistics Quarterly* 22 : 811-817, 1975.

[Burns et Rooker,76] F. Burns et J. Rooker. Johnson's three machine flow-shop conjecture. *Operations Research* 24(3) : 578-580, 1976.

[Burns et Rooker,78] F. Burns et J. Rooker. Three-stage flow shop with recessive second stage. *Operations Research* 26(1) : 207-208, 1978.

[Campbell et al.,70] H.G. Campbell, R.A. Dudek et M.L. Smith. A heuristic algorithm for the n-job, m-machine sequencing problem. *Management Science* 16 : B630-B637, 1970.

[Carlier,82] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research* 11 : 42-47, 1982.

[Carlier et Chretienne,88] J. Carlier et P. Chretienne. *Problèmes d'ordonnancement (modélisation / complexité / algorithmes)*. Collection Etudes et Recherches en Informatique, Masson éditions, 1988.

[Caumond et al.,04a] A. Caumond, M. Gourgand, P. Lacomme et N. Tchernev. Taboo algorithm for the jobshop problem with time lags : $Jm|l_{i,SJ(i)}|C_{max}$. *Annual conference of the Italian Operations Research Society* 97-98, 2004.

[Caumond et al.,04b] A. Caumond, M. Gourgand, P. Lacomme et N. Tchernev. Métaheuristiques pour le problème de jobshop avec time lags : $Jm|l_{i,SJ(i)}|C_{max}$. *Cinquième conférence francophone de MOdélisation et SIMulation* 939-946, 2004.

[Caumond et al.,05a] A. Caumond, P. Lacomme et N. Tchernev. Proposition d'un algorithme génétique pour le job-shop avec time lags. *Sixième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* 183-200, 2005.

[Caumond et al.,05b] A. Caumond, P. Lacomme et N. Tchernev. Feasible schedules generation with an extension of the Giffler and Thompson's heuristic for the jobshop problem with time lags. *International conference on Industrial Engineering and Systems Management* 489-499, 2005.

[Chrétienne et al.,95] P. Chrétienne, E.G. Coffman Jr, J.K. Lenstra et Z. Liu. *Scheduling Theory and its Applications* John Wiley & Sons, 1995.

[Chrétienne et Picouleau,95] P. Chrétienne et C. Picouleau. Scheduling with communication delays : a survey. Dans P. Chrétienne, E.G. Coffman Jr, J.K. Lenstra et Z. Liu. *Scheduling Theory and its Applications* John Wiley & Sons, 1995.

[Chu, 92] C. Chu. A branch and bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics* 39 : 265-283, 1992.

[Chu et Proth,96] C. Chu et J.-M. Proth. Single machine scheduling with chain structured precedence constraints and separation time windows. *IEEE Transactions on robotics and automation* 12(6) : 835-844, 1996.

[Coffman et Graham,72] E.G. Coffman Jr. et R.L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica* 1 : 200-213, 1972.

[Colorni et al,91] A. Colorni, M. Dorigo et V. Maniezzo. Distributed optimization by ant colonies. *First European Conference on Artificial Life* 134-142, 1991.

[Dauzère-Peres et Lasserre,93] S. Dauzère-Peres et J.-B. Lasserre. A modified Shifting Bottleneck Procedure for job-shop scheduling. *International Journal of Production Research* 31 : 923-932, 1993.

[De Reyck,98] B. De Reyck. Scheduling projects with generalized precedence relations : exact and heuristic procedures. Dissertation, Katholieke Universiteit Leuven. 1998.

[De Reyck et Herroelen,98] B. De Reyck et W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research* 111 : 152-174, 1998.

[Dell'Amico,96] M. Dell'Amico. Shop problems with two machines and time lags. *Operations Research* 44(5) : 777-787, 1996.

[Della Croce et T'kindt,02] F. Della Croce et V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society* 53(11) : 1275-1280, 2002.

[Deppner,03] F. Deppner. Ordonnancement d'atelier avec contraintes d'écart minimal et maximal entre opérations. *Quatrième conférence francophone de MOdélisation et SIMulation* 430-436, 2003.

[Deppner,04] F. Deppner. Ordonnancement d'atelier avec contraintes temporelles entre opérations. Thèse de Doctorat. Institut National Polytechnique de Lorraine. 2004.

[Dileepan,04] P. Dileepan. A note on minimizing maximum lateness in a two-machine no-wait flowshop. *Computers and Operations Research* 31 : 2111-2115, 2004.

[Dorndorf et al.,00] U. Dorndorf, E. Pesch et T. Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science* 46(10) : 1365-1384, 2000.

[Erschler,76] J. Erschler. Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement. Thèse de doctorat d'Etat. Université Paul Sabatier, Toulouse. 1976.

[Espinouse et al.,00] M.-L. Espinouse, M.-C. Portmann et G. Finke. Flowshop avec chevauchements, délais d'attente, temps de préparation et temps de remise en état. *Troisième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* 76-77, 2000.

[Fest et al.,99] A. Fest, R.H. Mohring, F. Stork et M. Uetz. Resource-constrained project scheduling with time windows : a branching scheme based on dynamic release dates. *Technical Report 596, Technische Universitat Berlin* 1999.

[Finke et al.,02] G. Finke, M.-L. Espinouse et H. Jiang. General flowshop models : job dependent capacities, job overlapping and deterioration. *International Transactions in Operational Research* 9 : 399-414, 2002.

[Finta et Liu,94] L. Finta et Z. Liu. Scheduling of parallel programs in single-bus multiprocessor systems. Rapport de Recherche INRIA, 2302, 1994.

[Finta et Liu,96] L. Finta et Z. Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics* 70 : 247-266, 1996.

[Fisher et Thompson,63] H. Fisher et G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. J.F. Muth et G.L. Thompson (Editeurs). *Industrial Scheduling.* Prentice-Hall, 1963.

[Fondrevelle,02a] J. Fondrevelle. Ordonnancement de produits périssables. *Colloque Confere* 2002.

[Fondrevelle,02b] J. Fondrevelle. Aspects pratiques concernant l'ordonnancement de produits périssables : application au cas du flowshop de permutation. Mémoire de DEA Génie des Systèmes Industriels. Institut National Polytechnique de Lorraine. 2002.

[Fondrevelle,03a] J. Fondrevelle. Résolution exacte d'un problème d'ordonnancement en présence de contraintes d'attentes maximales entre opérations d'un même job : cas du flowshop de permutation. *Quatrième conférence francophone de MOdélisation et SIMulation* 437-444, 2003.

[Fondrevelle,03b] J. Fondrevelle. Un problème de flowshop à deux machines avec contraintes d'attentes maximales. *Ecole d'Automne de Recherche Opérationnelle* 138-141, 2003.

[Fondrevelle et al.] J. Fondrevelle, A. Oulamara et M.-C. Portmann. Permutation flowshop scheduling problems with maximal and minimal time lags. *Computers and Operations Research* A paraître.

[Fondrevelle et al.,04a] J. Fondrevelle, A. Oulamara et M.-C. Portmann. Scheduling unit time operation permutation flowshop with minimal time lags. *Ninth international workshop on Project Management and Scheduling* 178-181, 2004.

[Fondrevelle et al.,04b] J. Fondrevelle, A. Allahverdi et A. Oulamara. Two-machine no-wait flowshop scheduling problem to minimize maximum lateness with separate setup and removal times. Rapport de Recherche du Loria. 2004.

[Fondrevelle et al.,05a] J. Fondrevelle, A. Oulamara et M.-C. Portmann. Approche de résolution pour les problèmes de flowshop avec time lags minimaux et maximaux. *Sixième congrès de la société française de Recherche Opérationnelle et d'Aide à la Décision* 179-180, 2005.

[Fondrevelle et al.,05b] J. Fondrevelle, A. Oulamara et M.-C. Portmann. Minimizing the weighted sum of machine completion times in flowshop with time lags. *International Conference on Industrial Engineering and Systems Management* 2005.

[Fondrevelle et al.,05c] J. Fondrevelle, A. Oulamara et M.-C. Portmann. Minimizing makespan in flowshop with time lags. *Seventh workshop on Models and Algorithms for Planning and Scheduling Problems* 138-139, 2005.

[Fondrevelle et al.,05d] J. Fondrevelle, A. Allahverdi, A. Oulamara et M.-C. Portmann. Permutation flowshop with exact time lags to minimize maximum lateness. Rapport de Recherche du Loria. 2005.

[Foulds et Wilson,05] L.R. Foulds et J.M. Wilson. Scheduling operations for the harvesting of renewable resources. *Journal of Food Engineering* 70 : 281-292, 2005.

[Framinan et al.,03] J.M. Framinan, R. Leisten et C. Rajendran. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research* 41(1) : 121-148, 2003.

[Franck et al.,01a] B. Franck, K. Neumann et C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum* 23 : 297-324, 2001.

[Franck et al.,01b] B. Franck, K. Neumann et C. Schwindt. Project scheduling with calendars. *OR Spektrum* 23 : 325-334, 2001.

[Garey et Johnson,79] M.R. Garey et D.S. Johnson. *Computers and Intractability : a guide to the theory of NP-Completeness.* W.H. Freemann, 1979.

[Garey et al.,76] M.R. Garey, D.S. Johnson et R. Sethi. The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research* 1 : 117-129, 1976.

[Gilmore et Gomory,64] P.C. Gilmore et R.E. Gomory. Sequencing a one state variable machine : a solvable case of the traveling salesman problem. *Operations Research* 12(5) : 655-679, 1964.

[Glover,90] F. Glover. Tabu search : a tutorial. *Interfaces* 20(4) : 74-94, 1990.

[Goldberg,89] D.E. Goldberg. *Genetic algorithms in search.* Optimization and Machine Learning. Addison Wesley, 1989.

[Gondran et Minoux,94] M. Gondran et M. Minoux. *Graphes et algorithmes.* Collection de la Direction des Etudes et Recherches d'Electricité de France, Eyrolles, 1994.

[Gonzalez et Sahni,76] T. Gonzalez et S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery* 23 : 665-679, 1976.

[GOThA] Groupe GOThA. Livres d'ordonnancement. *page web* http ://www-poleia.lip6.fr/ sourd/gotha/livres.html

[GOThA,93] Groupe GOThA. Les problèmes d'ordonnancement. *RAIRO-RO* 27(1) : 77-150, 1993.

[GOThA,04] Groupe GOThA. *Modèles et Algorithmes en Ordonnancement* Ellipses, 2004.

[Graham,66] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* 45 : 1563-1581, 1966.

[Graham et al.,79] R.L. Graham, E.L. Lawler, J.K. Lenstra et A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory : a survey. *Annals of Discrete Mathematics* 5 : 287-326, 1979.

[Guinet,96] A. Guinet. Integrated and sequential approaches to scheduling hybrid flowshop in make-to-stock environment. *Workshop on Production Planning and Control* 299-302, 1996.

[Guinet,00] A. Guinet. Efficiency of reductions of job-shop to flow-shop problems. *European Journal of Operational Research* 125 : 469-485, 2000.

[Guinet et Legrand,98] A. Guinet et M. Legrand. Reduction of job-shop problems to flow-shop problems with precedence constraints. *European Journal of Operational Research* 109 : 96-110, 1998.

[Gupta,96] J.N.D. Gupta. Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *Journal of Global Optimization* 9 : 239-253, 1996.

[Hall et al.,03] N.G. Hall, G. Laporte, E. Selvarajah et C. Sriskandarajah. Scheduling and lot streaming in flowshops with no-wait in process. *Journal of Scheduling* 6 : 339-354, 2003.

[Hall et Sriskandarajah,96] N.G. Hall et C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 44(3) : 510-525, 1996.

[Heilmann,03] R. Heilmann. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research* 144 : 348-365, 2003.

[Herroelen et al.,98] W. Herroelen, E. Demeulemeester et B. De Reyck. Resource-constrained project scheduling : a survey of recent developments. *Computers and Operations Research* 25 : 279-302, 1998.

[Ho et Gupta,95] J.C. Ho et J.N.D. Gupta. Flowshop scheduling with dominant machines. *Computers and Operations Research* 22(2) : 237-246, 1995.

[Hodson et al.,85] A. Hodson, A.P. Muhlemann et D.H.R. Price. A microcomputer based solution to a practical scheduling problem. *Journal of the Operational Research Society* 36(10) : 903-914, 1985.

[Holland,75] J.H. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1975.

[Hoogeven et al.,94] J.A. Hoogeveen, S.L. Van de Velde et B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocation. *Discrete Applied Mathematics* 55 : 259-272, 1994.

[Horn,74] W.A. Horn. Some scheduling algorithms. *Naval Research Logistics Quarterly* 21 : 177-185, 1974.

[Hurink et Keuchel,01] J. Hurink et J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics* 112 : 179-197, 2001.

[Ibaraki,87] T. Ibaraki. Enumerative approaches to combinatorial optimization : part I. *Annals of Operations Research* 10(1-4) : 1-340, 1987.

[Ignall et Schrage,65] E. Ignall et L. Schrage. Application of the branch-and-bound technique to some flow-shop scheduling problems. *Operations Research* 13 : 400-412, 1965.

[Jackson,55] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43, *Management Science Research Project*, University of California, 1955.

[Janczewski,99] R. Janczewski. On an interrelation between travelling salesman problem and T-coloring of graphs. *Proceedings of Advanced Computer Systems VI* Szczecin, 23-25, 1999.

[Janczewski et Kubale,01] R. Janczewski et M. Kubale. Scheduling unit execution time tasks with symmetric time-lags. *Journal of Applied Computer Science* 9(2) : 45-51, 2001.

[Johnson,54] S.M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly* 1(1) : 61-68, 1954.

[Khurana et Bagga,84] K. Khurana et P.C. Bagga. Minimizing the makespan in a 2-machine flowshop with time lags and setup conditions. *Zeitschrift fur Operations Research* 28 : 163-174, 1984.

[Kim,93] Y.-D. Kim. A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers and Operations Research* 20 : 391-401, 1993.

[Kim,95] Y.-D. Kim. Minimizing total tardiness in permutation flowshops. *European Journal of Operational Research* 85 : 541-555, 1995.

[Kim et al.,96] Y.-D. Kim, H.-G. Lim et M.-W. Park. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research* 91 : 124-143, 1996.

[Kirkpatrick et al.,83] S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi. Optimization by simulated annealing. *Sciences* 220 : 671-680, 1983.

[Kolisch,95] R. Kolisch. *Project scheduling under resource constraints : efficient heuristics for several problem classes.* Physica, 1995.

[Kolisch et al.,95] R. Kolisch, A. Sprecher et A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science* 41 : 1693-1703, 1995.

[Kruskal,56] J.B. Kruskal. On the shortest spanning sub-tree and the traveling salesman problem. *Proceedings of the American Mathematical Society* 7 : 48-50, 1956.

[Ladhari et Haouari,05] T. Ladhari et M. Haouari. A computational study of the permutation flow shop problem based on a tight lower bound. *Computers and Operations Research* 32 : 1831-1847, 2005.

[Lageweg et al.,78] B.J. Lageweg, J.K. Lenstra et A.H.G. Rinnooy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research* 26(1) : 53-67, 1978.

[Lawler,76] E.L. Lawler. *Combinatorial Optimization : Networks and Matroids*, Holt, Rinehart and Winston, 1976.

[Lawler et al.,85] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan et D.B. Shmoys. *The Traveling Salesman Problem.* Wiley, 1985.

[Lawler et al.,93] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan et D.B. Shmoys. Sequencing and scheduling : algorithms and complexity. In : S.C. Graves, P.H. Zipkin et A.H.G. Rinnooy Kan. *Handbooks in operations research and management science, vol. 4.* North-Holland : Amsterdam, 7-10, 1993.

[Lawler et Wood,66] E.L. Lawler et D.E. Wood. Branch-and-bound methods : a survey. *Operations Research* 14 : 699-719, 1966.

[Lawrence,84] S. Lawrence. Resource constrained project scheduling : an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie Mellon University, 1984.

[Lenstra et al.,77] J.K. Lenstra, A.H.G. Rinnooy Kan et P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1 : 343-362, 1977.

[Lin et Cheng,01] B.M.T. Lin et T.C.E. Cheng. Batch scheduling in the no-wait two-machine flowshop to minimize the makespan. *Computers and Operations Research* 28 : 613-624, 2001.

[Lin et Haley,94] C.K.Y. Lin et K.B. Haley. Scheduling two-phase jobs with arbitrary time lags in a single-server system. *IMA Journal of Mathematics Applied in Business and Industry* 5 : 143-161, 1994.

[Lopez,91] P. Lopez. Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources. Thèse de doctorat. Université Paul Sabatier, Toulouse. 1991.

[Lopez et Roubellat,01] P. Lopez et F. Roubellat. *Ordonnancement de la production.* IC2 Productique, Hermes Sciences Publications, 2001.

[Lourenço,98] H.R. Lourenço. A polynomial algorithm for special case of the one-machine scheduling problem with time-lags. Economics Working Papers Series, Department of Economics and Business, Universitat Pompeu Fabra, 339, 1998.

[MacCarthy et Liu,93] B.L. MacCarthy et J. Liu. Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research* 31(1) : 59-79, 1993.

[Maggu et al.,82] P.L. Maggu, M.L. Singhal, N. Mohammad et S.K. Yadav. On N-job, 2-machine flow-shop scheduling problem with arbitrary time lags and transportation times of jobs. *Journal of the Operations Research Society of Japan* 25(3) : 219-227, 1982.

[Manier et Bloch,03] M.-A. Manier et C. Bloch. A classification of hoist scheduling problems. *International Journal of Flexible Manufacturing Systems* 15(1) : 37-55, 2003.

[Manne,60] A.S. Manne. On the job-shop scheduling problem. *Operations Research* 8 : 219-226, 1960.

[Mastor,70] A.A. Mastor. An experimental and comparative evaluation of production line balancing techniques. *Management Science* 16 : 728-746, 1970.

[Mitten,59] L.G. Mitten. Sequencing n jobs on two machines with arbitrary time lags. *Management Science* 5 : 293-298, 1959.

[Monma et Potts,89] C.L. Monma et C.N. Potts. On the complexity of scheduling with batch setup times. *Operations Research* 37 : 798-804, 1989.

[Moore,68] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science* 15 : 102-109, 1968.

[Munier et al.,98] A. Munier, M. Queyranne et A.S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *Sixth International Integer Programming and Combinatorial Optimization Conference* 367-382, 1998.

[Munier et Sourd,03] A. Munier et F. Sourd. Scheduling chains on a single machine with non-negative time lags. *Mathematical Methods of Operations Research* 57 : 111-123, 2003.

[Nawaz et al.,83] M. Nawaz, E.E. Enscore Jr. et I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA* 11(1) : 91-95, 1983.

[Neumann et al.,02] K. Neumann, C. Schwindt et J. Zimmermann. Recent results on resource-constrained project scheduling with time windows : models, solution methods, and applications. *Central European Journal of Operations Research* 10 : 113-148, 2002.

[Neumann et Zhan,95] K. Neumann et J. Zhan. Heuristics for the minimum project-duration problem with minimal and maximal time-lags under fixed resource constraints. *Journal of Intelligent Manufacturing* 6 : 145-154, 1995.

[Nowicki et Smutnicki,96] E. Nowicki et C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science* 42(6) : 797-813, 1996.

[Orman et Potts,97] A.J. Orman et C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics* 72 : 141-154, 1997.

[Oulamara et Finke,01] A. Oulamara et G. Finke. Flowshop problems with batch processing machines. *International Journal of Mathematical Algorithms* 2 : 269-287, 2001.

[Oulamara et al.,05] A. Oulamara, M.Y. Kovalyov et G. Finke. Scheduling a no-wait flowshop containing unbounded batching machines. *IIE Transactions on Scheduling and Logistics* 37(8) : 689-696, 2005.

[Pascoe,66] T.L. Pascoe. Allocation of resources - CPM. *Revue Française de Recherche Opérationnelle* 38 : 31-38, 1966.

[Pinedo,02] M. Pinedo. *Scheduling : Theory, Algorithms, and Systems. Second Edition.* Prentise hall, 2002.

[Portmann,87] M.-C. Portmann. Méthodes de décomposition spatiales et temporelles en ordonnancement de la production. Thèse d'état. Université de Nancy 1. 1987.

[Portmann,88] M.-C. Portmann. Méthodes de décomposition spatiales et temporelles en ordonnancement de la production. *RAIRO-APII* 22(5) : 439-451, 1988.

[Portmann,97] M.-C. Portmann. Scheduling methodology : optimization and compu-search approaches I, in A. Artiba et S.E. Elmaghraby*The Planning and Scheduling of Production Systems.* Chapman et Hall edition, 271-300, 1997.

[Potts,80] C.N. Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research* 28(6) : 1436-1441, 1980.

[Potts et Kovalyov,00] C.N. Potts et M.Y. Kovalyov. Scheduling with batching : a review. *European Journal of Operational Research* 120 : 228-249, 2000.

[Potts et al.,01] C.N. Potts, V.A. Strusevich et T. Tautenhahn. Scheduling batches with simultaneous job processing for two-machine shop problems. *Journal of Scheduling* 4 : 25-51, 2001.

[Potts et Van Wassenhove,82] C.N. Potts et L.N. Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters* 1 : 177-181, 1982.

[Potts et Van Wassenhove,91] C.N. Potts et L.N. Van Wassenhove. Integrating scheduling with batching and lot-sizing : a review of algorithms and complexity. *Journal of the Operational Research Society* 46 : 395-406, 1991.

[Potts et Whitehead,05] C.N. Potts et J.D. Whitehead. Local search for a single machine coupled-operation scheduling problem. *Seventh workshop on Models and Algorithms for Planning and Scheduling Problems* 225-227, 2005.

[Prim,57] R.C. Prim. Shortest connection networks and some generalisations. *Bell System Technical Journal* 36 : 1389-1401, 1957.

[Rajendran,93] C. Rajendran. Heuristic algorithm for scheduling in a flowshop to minimise total flowtime. *International Journal of Production Economics* 29 : 65-73, 1993.

[Rayward-Smith et Rebaine,92] V.J. Rayward-Smith et D. Rebaine. Open-shop scheduling with delays. *Informatique Théorique et Applications* 26 : 439-448, 1992.

[Rayward-Smith et Rebaine,96] V.J. Rayward-Smith et D. Rebaine. UET flow shop scheduling with delays. *Informatique Théorique et Applications* 30(1) : 23-30, 1996.

[Rebaine et Strusevich,99] D. Rebaine et V.A. Strusevich. Two-machine open shop scheduling with special transportation times. *Journal of the Operational Research Society* 50 : 756-764, 1999.

[Riezebos et Gaalman,98] J. Riezebos et G.J.C. Gaalman. Time lag size in multiple operations flow shop scheduling heuristics. *European Journal of Operational Research* 105 : 72-90, 1998.

[Riezebos et al.,95] J. Riezebos, G.J.C. Gaalman et J.N.D. Gupta. Flow shop scheduling with multiple operations and time lags. *Journal of Intelligent Manufacturing* 6 : 105-115, 1995.

[Rios-Mercado et Bard,99] R.Z. Rios-Mercado et J.F. Bard. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions* 31 : 721-731, 1999.

[Roberts,91] F.S. Roberts. T-colorings of graphs : recent results and open problems. *Discrete Mathematics* 93 : 229-245, 1991.

[Roeck,84a] H. Roeck. The three-machine no-wait flow shop is NP-complete. *Journal of the Association for Computing Machinery* 31(2) : 336-345, 1984.

[Roeck,84b] H. Roeck. Some new results in flow shop scheduling. *Zeitschrift fur Operations Research* 28(1) : 1-16, 1984.

[Roy,59] B. Roy. Contribution de la théorie des graphes à l'étude de certains problèmes linéaires. *Comptes rendus de l'Académie des Sciences* 248, 1959.

[Roy et Sussman,64] B. Roy et B. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives. SEMA, Note DS No.9bis, 1964.

[Ruiz et Serifoglu,05a] R. Ruiz et F.S. Serifoglu. Solving realistic hybrid flexible flowshop scheduling problems. *International Conference on Industrial Engineering and Systems Management* 2005.

[Ruiz et Serifoglu,05b] R. Ruiz et F.S. Serifoglu. On solving realistic scheduling problems. *Seventh workshop on Models and Algorithms for Planning and Scheduling Problems* 235-238, 2005.

[Saadani et al.,03] N.E.H. Saadani, A. Guinet et M. Moalla. Three stage no-idle flow-shops. *Computers and Industrial Engineering* 44 : 425-434, 2003.

[Schuster et Framinan,03] C.J. Schuster et J.M. Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters* 31 : 308-318, 2003.

[Schwindt,96] C. Schwindt. Generation of resource-constrained project scheduling problems with minimal and maximal time lags. *Technical Report WIOR-489, Institut fur Wirtschaftstheorie und Operations Research, Universitat Karlsruhe* 1996.

[Schwindt et Trautmann,00] C. Schwindt et N. Trautmann. Batch scheduling in process industries : an application of resource-constrained project scheduling. *OR Spektrum* 22 : 501-524, 2000.

[Sekiguchi,82] Y. Sekiguchi. Inter- and intra- group-of-jobs schedule for minimizing makespan in a two-machine GT shop. *Control Science and Technology for the Progress of Society. Proceedings of the Eighth Triennial World Congress of the International Federation of Automatic Control* 4 : 2021-2026, 1982.

[Shapiro,81] R.D. Shapiro. Scheduling coupled-tasks. *Naval Research Logistics Quarterly* 28 : 489-497, 1981.

[Shyu et al.,04] S.J. Shyu, B.M.T. Lin et P.Y. Yin. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. *Computers and Industrial Engineering* 47 : 181-193, 2004.

[Sprecher et al.,95] A. Sprecher, R. Kolisch et A. Drexl. Semi-active, active, and non-delay schedules for the resource-constrained scheduling problem. *European Journal of Operational Research* 80 : 94-102, 1995.

[Sridhar et Rajendran,96] J. Sridhar et C. Rajendran. Scheduling in flowshop and cellular manufacturing systems with multiple objectives - a genetic algorithmic approach. *Production Planning and Control* 7(4) : 374-382, 1996.

[Stafford et al.,05] E.F. Stafford, F.T. Tseng et J.N.D. Gupta. Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society* 56 : 88-101, 2005.

[Strusevich,99] V.A. Strusevich. A heuristic for the two-machine open-shop scheduling problem with transportation times. *Discrete Applied Mathematics* 93 : 287-304, 1999.

[Strusevich et Hall,97] V.A. Strusevich et L.A. Hall. An open-shop scheduling problem with a non-bottleneck machine. *Operations Research Letters* 21 : 11-18, 1997.

[Sviridenko,03] M. Sviridenko. Makespan minimization in no-wait flow shops : a polynomial time approximation scheme. *SIAM Journal on Discrete Mathematics* 16(2) : 313-322, 2003.

[Szwarc,74] W. Szwarc. Mathematical aspects of the $3 \times n$ job-shop sequencing problem. *Naval Research Logistics Quarterly* 21 : 145-153 et 725-726, 1974.

[Szwarc,77] W. Szwarc. Optimal two-machine orderings in the $3 \times n$ flow-shop problem. *Operations Research* 25 : 70-77, 1977.

[Szwarc,83] W/ Szwarc. Flow shop problems with time lags. *Management Science* 29(4) : 477-481, 1983.

[Szwarc,86] W. Szwarc. The flow shop problem with time lags and separated setup times. *Zeitschrift für Operations Research* 30 : B15-B22, 1986.

[Szwarc et al,99] W. Szwarc, F. Della Croce et A. Grosso. Solution of the single machine total tardiness problem. *Journal of Scheduling* 2 : 55-71, 1999.

[Taillard,93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64 : 278-285, 1993.

[Tanaev et al.,94] V.S. Tanaev, Y.N. Sotskov et V.A. Strusevich. *Scheduling theory : multi-stage systems*. Kluwer, 1994.

[Townsend,77] W. Townsend. Sequencing n jobs on m machines to minimize maximum tardiness : A branch-and-bound solution. *Management Science* 23 : 1016-1019, 1977.

[Vairaktarakis,03] G.L. Vairaktarakis. Simple algorithms for Gilmore-Gomory's traveling salesman and related problems. *Journal of Scheduling* 6 : 499-520, 2003.

[Vignier et al.,99] A. Vignier, J.-C. Billaut et C. Proust. Les problèmes d'ordonnancement de type flow-shop hybride : état de l'art. *RAIRO Recherche Opérationnelle* 33(2) : 117-183, 1999.

[Volgenant et Teerhuis,99] A. Volgenant et E. Teerhuis. Improved heuristics for the n-job single machine weighted tardiness problem. *Computers and Operations Research* 26 : 35-44, 1999.

[Webster et Baker,95] S.T. Webster et K.R. Baker. Scheduling groups of jobs on a single machine. *Operations Research* 43 : 692-703, 1995.

[Wikum et al.,94] E.D. Wikum, D.C. Llewellyn et G.L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters* 16 : 87-99, 1994.

[Wismer,72] D.A. Wismer. Solution of the flowshop scheduling problem with no intermediate queues. *Operations Research* 20 : 689-697, 1972.

[Woo et Yim,98] D.S. Woo et H.S. Yim. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers and Operations Research* 25 : 175-182, 1998.

[Yang et Chern,95] D.L. Yang et M.S. Chern. A two-machine flowshop sequencing problem with limited waiting time constraints. *Computers and Industrial Engineering* 28(1) : 63-70, 1995.

[Yu,96] W. Yu. The two-machine flow shop problem with delays and the one-machine total tardiness problem. PhD Thesis. Technische Universiteit Eindhoven. 1996.

[Yu et al.,04] W. Yu, H. Hoogeveen et J.K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling* 7 : 333-348, 2004.

Glossaire

Nous reprenons ici l'ensemble des sigles et symboles utilisés à plusieurs reprises dans cette thèse, en indiquant pour chacun d'eux sa signification et la section dans laquelle il est introduit. Nous revenons en particulier sur les champs de la notation $\alpha|\beta|\gamma$ habituellement employée pour désigner les problèmes d'ordonnancement, sans être exhaustif.

Sigle		Signification	Section
$\alpha =$	1	Problème à une machine	1.2.1
$\alpha =$	P	Problème à machines parallèles	1.2.1
$\alpha =$	F	Problème de flowshop	1.2.1
$\alpha =$	Fm_π	Problème de flowshop de permutation à m machines	1.2.1
$\alpha =$	J	Problème de jobshop	1.2.1
$\alpha =$	O	Problème d'openshop	1.2.1
	$t_{j,k}$	Date de début du travail j sur la machine k	1.4
	$C_{j,k}$	Date de fin du travail j sur la machine k	1.4
	C_j	Date de fin du travail j	1.2.2
	$p_{j,k}$	Durée d'exécution du travail j sur la machine k	1.4
	$\theta_{j,k}^{min}$	Time lag minimal du travail j entre les machines k et $k+1$	1.4
	$\theta_{j,k}^{max}$	Time lag maximal du travail j entre les machines k et $k+1$	1.4
	r_j	Date de disponibilité du travail j	1.2.3
	q_j	Durée de latence du travail j	1.2.3
	d_j	Date de fin souhaitée du travail j	1.2.2
$\beta =$	$no-wait$	Traitement des travaux sans attente	2.5
$\beta =$	$NSDST$	Temps de montage indépendants de la séquence de traitement des travaux	1.5
$\beta =$	$NSDRT$	Temps de démontage indépendants de la séquence de traitement des travaux	1.5
$\beta =$	$p-batch(1)$	Machine à traitement parallèle par fournées	2.7
$\beta =$	$1 < b < n$	Capacité limitée des fournées	2.7
$\beta =$	$b \geq n$	Capacité illimitée des fournées	2.7
	U_j	Indicateur de retard du travail j	1.2.2
	w_j	Poids du travail j	1.2.2
	MC_k	Date de fin sur la machine k	2.2
$\gamma =$	C_{max}	Makespan / plus grande date de fin des travaux	1.2.2
$\gamma =$	$\sum C_j$	Somme des dates de fin des travaux	1.2.2
$\gamma =$	L_{max}	Plus grand retard des travaux	1.2.2
$\gamma =$	$\sum T_j$	Somme des retards des travaux	1.2.2
$\gamma =$	$\sum U_j$	Nombre de travaux en retard	1.2.2
$\gamma =$	$\sum w_k MC_k$	Somme pondérée des dates de fin sur les machines	2.2
	EDD	Ordre croissant des dates de fin souhaitées	2.4.2
	$RCPSP$	Problème d'ordonnancement de projet à contraintes de ressources	3.6
	PSE	Procédure par Séparation et Evaluation	4.2
	NEH	Heuristique de Nawaz et al. [Nawaz et al.,83]	4.3.5
	LB_v^u	Borne inférieure v pour le problème u	4.3.4
	H_v^u	Borne supérieure v pour le problème u	4.3.5