Université François Rabelais Tours

École Doctorale : Santé, Sciences et Technologies
Année Universitaire : 2004-2005

THÈSE POUR OBTENIR LE GRADE DE
DOCTEUR DE L'UNIVERSITÉ DE TOURS

Discipline : Informatique

présentée et soutenue publiquement

par :

Bertrand Esteve

le 22 juin 2005

Des Problèmes d'Ordonnancement Multicritères de type Juste-A-Temps : Modélisation et Résolution

---

Sous la direction de M. Jean-Charles Billaut
Co-encadré par M. Vincent T'kindt

Membres du jury :

M. Jean-Charles Billaut	*Professeur*	Université de Tours
Mme Clarisse Dhaenens	*Maître de Conférences*	Université de Lille 1
M. Federico DellaCroce	*Professeur associé*	Université de Turin (Italie)
M. Francis Sourd	*Chargé de Recherches CNRS*	Université de Paris 6
Mlle Marie-Claude Portmann	*Professeur*	Université de Nancy
M. Bernard Penz	*Professeur*	Université de Grenoble
M. Vincent T'kindt	*Maître de Conférences*	Université de Tours

Table des matières

Introduction

Durant les cinquante dernières années, les problèmes d'ordonnancement d'atelier ont pris une ampleur de plus en plus grande. L'évolution des algorithmes et des matériels permettent la résolution de problèmes de taille de plus en plus élevée et leur intégration de plus en plus élaborée dans le système que constitue l'entreprise industrielle. Le récent développement des études sur les problèmes d'ordonnancement multicritères est un élément de plus qui montre le souci réel de placer les études dans un contexte de plus en plus concret. Dans la présente étude, nous en tentons une application à ce que nous qualifions à juste titre de *Ordonnancement Multicritère de type Juste-A-Temps*.

Le **chapitre 1** constitue une brève introduction aux problèmes de la gestion de production. Nous cherchons à dégager la philosophie de la gestion en *Juste-A-Temps* de la production et mettons en évidence ceux des principes qui nous paraissent pertinents au niveau opérationnel et en tout cas ceux qui ont servi de fondement au présent travail dont le thème principal est l'ordonnancement.

Le **chapitre 2** commence par une description de l'ordonnancement, vu comme un ensemble de problèmes de nature combinatoire en recherche opérationnelle. Elle continue par une typologie et un état de l'art détaillé des problèmes d'ordonnancement que nous qualifierons de *problèmes de type Juste-A-Temps* dans la littérature consacrée aux problèmes d'ordonnancement. Une attention particulière est donnée aux modèles de type avance-retard à dates dues distinctes et à la résolution des problèmes correspondants.

Dans **le chapitre 3**, après un bref rappel des techniques de l'optimisation multicritère et de ses applications à l'ordonnancement, nous présentons dans un premier temps la modélisation choisie pour les *problèmes d'ordonnancement multicritères de type Juste-A-Temps*. Notre apport à l'approche par *fonctions de coûts* est alors développée. Nous montrons en quoi elle est cohérente avec les principes du chapitre 1 ; en quoi elle constitue une généralisation de la littérature étudiée au chapitre 2 ; en quoi elle permet une aide à la décision multicritère rappelée en début de chapitre.

Le **chapitre 4** et le **chapitre 5** présentent, respectivement, les algorithmes de résolution développés respectivement dans le cadre d'un problème à une machine et dans le cadre d'un problème de type flowshop à deux machines avec produits divisibles.Tout d'abord, nous présentons nos contributions. BLBLBL Ensuite, nous continuons par une étude expérimentale. BLBLBL

Enfin, le **chapitre 6** présente le prototype d'un outil logiciel mettant en oeuvre de façon interactive certains des principes et algorithmes exposés précédemment.

Nous présentons enfin une conclusion à notre travail ainsi que des directions de recherches futures.

Chapitre 1

Produire en Juste-A-Temps

1.1 Introduction

La gestion de la production manufacturière a beaucoup évolué depuis les trente dernières années. D'une part on est passé d'une production en volume gérée par l'offre, puisqu'il s'agissait d'équiper les ménages, à une production diversifiée gérée par la demande, puisque le rapport de force entre client et fournisseur s'est inversé dans le courant des années quatre-vingt-dix, avec d'une part la crise économique à l'issue de la fin de la guerre froide et d'autre part l'arrivée de nouveaux producteurs dans la compétition devenue désormais mondiale. Le besoin de rationalisation du fonctionnement des entreprises est donc devenu croissant et des techniques de plus en plus sophistiquées ont été mises en oeuvre. Ce fut possible grâce notamment aux avancées, tant théoriques qu'appliquées, de l'informatique pour l'industrie. Dans ce contexte, une méthode alternative à la méthode occidentale classique de flux poussés est apparue au Japon. Sa renommée, amplifiée par les succès de l'économie japonaise au milieu des années quatre-vingt, a poussé les entreprises tant européennes qu'américaines à s'y intéresser. Par commodité de langage cet ensemble de techniques fut appelé *Flux Tendus*, *Qualité Totale* ou encore *Juste-A-Temps (JAT)*.

De façon informelle, il existe pour les systèmes de production deux modèles dominants : le modèle classique des flux poussés et le modèle japonais des flux tendus. Dans le premier modèle, la production est organisée dans des ateliers spécialisés avec implantation fonctionnelle et la logique de production est le pilotage par l'amont, autrement dit la production se fait par anticipation des besoins de la clientèle, on produit donc sur des prévisions et on stocke. Dans le second modèle, la production est organisée en ateliers polyvalents avec implantation en cellule et la logique de production est le pilotage par l'aval, autrement dit la production se fait en fonction des commandes fermes de la clientèle. Le premier modèle est adapté à la production de masse caractérisée par des produits peu différenciés à long cycle de vie tandis que le second est concerné par des produits davantage différenciés dont la durée de vie est raccourcie. Le modèle classique se caractérise également par une main d'oeuvre peu qualifiée tandis que le modèle japonais nécessite une main d'oeuvre plus qualifiée. Le modèle en flux poussés a pour avantages de permettre la planification de la production et de faciliter les calculs de coût, il évite la rupture du stock et permet la prévision des capacités de production. Il a pour inconvénients le surstockage, le manque de flexibilité et un travail peu enrichissant pour le personnel. Le modèle en flux tendus a pour avantage une baisse des stocks et des coûts, une réduction des délais, des gains de productivité et de flexibilité, un enrichissement des tâches réalisées. Il a pour inconvénients un risque de rupture de stock

ainsi que l'obligation de réunir tout un ensemble de conditions, entre autres une coopération excellente aussi bien de la part des clients que des fournisseurs et une parfaite formation du personnel.

Le thème de ce chapitre est la gestion de production de type Juste-A-Temps (JAT). Après le rappel des principaux éléments de cette philosophie du management opérationnel des systèmes manufacturiers, nous mettons en évidence les points sur lesquels nous avons choisi de concentrer notre attention lors de ce travail de recherche en informatique portant sur *L'Ordonnancement Multicritère de type JAT*. Le propos de ce premier chapitre est donc dans un premier temps, de s'insérer dans le management global de l'entreprise, puis de présenter brièvement les méthodes devenues classiques de flux poussé et de flux tendu. Dans un deuxième temps, nous rappelons les principes et la philosophie des méthodes dites Juste-A-Temps. Dans un troisième temps, nous reprenons la liste des outils mis au point pour rendre possible autant que faire se peut la mise en oeuvre de ces principes. En conclusion nous insistons sur les outils qui sont concretement utilisables pour l'ordonnancement multicritère, tout particulièrement la gestion des délais, les coûts d'avance et de retard et les coûts de gestion des stocks intermédiaires.

Pour prospérer et même parfois tout simplement pour survivre les entreprises de production se doivent de fournir de la valeur au moins égale à celle de leurs concurrents sur le marché mondial. Aujourd'hui la compétition est telle que la concurrence peut venir de toutes les parties du monde libre. Même si le commerce international est une réalité qui ne date pas de maintenant, il faut bien reconnaître qu'il a littéralement explosé dans les dernières décennies. L'amplification de cette compétition est bien sûr le fait de l'accroissement de moyens de transport et de communication, cependant il ne faut pas oublier l'écrasante contribution de tous les processus visant à augmenter la productivité. Par productivité, il faut ici surtout entendre tout ce qui concoure à l'amélioration de la qualité des produits et à la diminution de coûts liés à la fabrication de ces derniers.

En effet, si le monde occidental dans son acception la plus large veut être dans la course mondiale du marché actuel aux dimensions les plus larges du monde dans sa globalité, il doit observer avec attention les caractéristiques opérationnelles des systèmes et structures qui tirent le plus admirablement bien leur épingle du jeu. Les caractéristiques essentielles de telles compagnies industrielles sont qu'ils produisent des biens de grande qualité à des coûts toujours plus attractifs, et cela en étant capable de répondre le plus immédiatement possible aux exigences du client ou de l'utilisateur final qui, lui, est constamment à la recherche de meilleurs conditions de livraison, de configuration ou de quantification. Faisons le pari qu'une entreprise qui est capable de relever tous ces défis simultanément sera capable de se battre contre n'importe quel compétiteur dans n'importe quelle catégorie, n'importe où dans le monde civilisé et même au delà. Il est donc nécessaire de bien prendre conscience que la qualité la plus excellente et le coût le moins élevé sont les deux faces d'une même pièce de monnaie. De même il est important de bien garder à l'esprit que l'excellence n'est qu'une valeur relative et que ce qui compte avant tout c'est tout l'effort surhumain visant au dépassement de l'existant dans un mouvement perpétuel, continu, et parfois insensible localement d'amélioration constante. Ce qui compte c'est la variation relative positive de qualité et la variation relative négative de coût total.

Pour y voir plus clair avant de commencer, rappelons quelques définitions issues de [Molet, 1989].

Approches japonaises : méthodologie et démarches ayant pour but d'atteindre une production sans gaspillage ; elles mettent l'accent sur la nécessité d'une approche collective de la gestion préventive des aléas.

Ordonnancement : méthodes pour fixer l'ordre de passage des opérations sur les différents moyens de production pour réaliser un objectif donné.

Gestion à flux tendus : Fonctionnement de l'atelier avec un stock minimum d'en cours entre chaque poste de travail.

Juste-A-temps : fonctionnement de l'atelier dans lequel on ne fabrique (ou approvisionne) que ce qui est nécessaire au moment où c'est nécessaire.

Dans un sens restreint, un système fonctionnant en JAT peut être décrit comme étant un système qui fabrique et livre des produits finis juste à temps pour être vendus, des sous-ensembles disponibles juste à temps pour être assemblés en produits finis et des matières juste à temps pour être transformées en composants. Cependant, dans un sens plus large, un système JAT est un système de gestion construit autour d'une philosophie simple sous-tendue par deux principes : la chasse au gaspillage et la mobilisation du personnel.

1.2 Une philosophie de la gestion de production

Pour reprendre la définition de l'*American Production and Inventory Control Society*, le Juste-A-Temps est une **philosophie des systèmes industriels ayant pour fondements l'élimination systématique des gaspillages et l'amélioration de la productivité en un mouvement continuel de perfectionnement**.

C'est une démarche englobante visant la réalisation sans faille de toutes les activités opérationnelles requises pour fabriquer le produit fini ; depuis l'ingénierie de conception jusqu'à la livraison définitive en passant par tous les stades de fabrication, y compris les approvisionnements. Une attention particulière est prétée, entre autre, au fait d'avoir en stock seulement ce dont on a besoin et seulement au moment où on en a besoin ; d'améliorer la qualité jusqu'à ce qu'il y ait zéro défaut ; de réduire les délais de fabrication en réduisant les temps de changement d'outils, la longueur des files d'attente ou encore la taille des lots ; de réviser périodiquement le processus de fabrication lui même ; et d'accomplir tout cela à moindres frais. En d'autres termes, le Juste-A-Temps est donc une philosophie de la gestion de production des entreprises manufacturières qui vise à éliminer, à la source, ce qui se perd inutilement au cours du processus de fabrication. Il tend à réduire ces gaspillages existants en organisant la production de sorte à fournir la quotité correcte aux stades adéquats au moment juste. Gaspillage est en fait le nom donné à une activité lorsque son résultat est d'augmenter le coût du produit sans augmenter sa valeur ajoutée ; c'est le cas, par exemple des activités de manutention et de stockage. Vu sous cet angle, le Juste-A-Temps, autrement appelé parfois dans la littérature anglo-saxonne *lean production* c'est-à-dire production *maigre* ou *dégraissée*, se veut capable d'améliorer les résultats financiers de l'entreprise ainsi que les retours sur investissement : cela peut être fait au moyen de la réduction des niveaux de stocks –en fait augmentation de la rotation de stock–, au moyen de l'augmentation de la qualité –en fait la diminution des rebuts à chaque stade–, en réduisant au strict nécessaire les délais de production et de livraison et de façon générale en réduisant autant que faire se peut les coûts tels que ceux liés aux pannes à répétition ou tout simplement au réglage systématique des machines.En fait, dans les systèmes ayant adopté le Juste-A-Temps, la tendance naturelle à utiliser les machines au maximum de leurs capacités en volume n'est pas la priorité, une certaine sous-capacité est même privilégiée pour faire face aux cas d'urgence qui sont réglés dans d'autres systèmes par une pléthore de stocks-tampons.

L'adoption du Juste-A-Temps prend du sens principalement dans les industries manufacturières répétives dans lesquelles les mêmes familles de produits et de composants sont

produits encore et toujours. L'idée générale, au plan de la mise en oeuvre, est d'établir un processus de flux ininterrompu en reliant idéalement les stades de fabrication, comme s'ils faisaient partie, pour un produit donné, d'une même ligne d'assemblage ou chaîne de fabrication ; c'est-à-dire en faisant en sorte que tout au long du processus de production d'un produit quel qu'il soit, les flux de matériaux soient les plus équilibrés et lissés possibles. Asymptotiquement, les files d'attentes au pied des machines seraient réduites à zéro et la taille des lots serait ramenée à l'unité.

1.3 Mise en oeuvre du Juste-A-Temps

[Monden, 1983] suivi de [Schonberger, 1986] comme texte fondateur propose une grille de lecture théorique et appliquée de la philosophie du Juste-A-Temps et de sa mise en oeuvre dans les usines *Toyota*. Ces travaux considèrent la réduction des coûts comme l'aspect primaire de cette philosophie et proposent trois objectifs pour réaliser ce but. Ils sont tous les trois indissociablement complémentaires et indispensables. Le premier, qui dans le cadre de notre étude reste malgré tout le principal, est lié à la maîtrise de la gestion des opérations à travers une panoplie assez hétéroclite d'outils dont un catalogue sera donné dans le paragraphe suivant. Le deuxième et le troisième de ces objectifs sont d'une part l'assurance qualité, autrement dit l'auto-inspection à cent pour cent, les inspections statistiques, les cercles de qualité et d'autre part le respect pour l'humanité autrement dit le fait de donner aux travailleurs des tâches valorisantes à effectuer, en développant la formation et la pluridisciplinarité, la communication ouverte suscite chez les employés le désir de soumettre des suggestions en vue de l'amélioration perpetuelle de l'entreprise et de ses salariés. Ce dernier point est contesté par certains auteurs qui y voient seulement une façon d'affaiblir la résistance au changement lors des opérations de restructuration.

Gardons constamment à l'esprit que le concept de Juste-A-Temps, dans son sens le plus vaste, est en fait une approche on ne peut plus éclectique. Il inclut des idées qui sont vieilles comme le monde et d'autres qui datent de la dernière pluie. Il repose sur des concepts pluridisciplinaires qui vont des statistiques inférentielles aux sciences du comportement cognitif en passant par le génie industriel et la gestion de production. Mais, d'abord, ne perdons jamis de vue que c'est une approche qui se veut avant tout empirique. La découverte de ce qui va marcher et comment cela peut marcher nécessite de mettre à plat et d'observer avec attention le fonctionnement de l'usine concernée. Cela nécessite de recueillir et d'analyser toutes les données pertinentes concernant l'entreprise, son organisation et ses performances. Il s'agit également de considérer l'entreprise dans son environnement de façon pragmatique, comme un véritable bouillon de culture, un laboratoire de recherche où on cherche à faire très bien la première fois et encore mieux les fois suivantes.

Ainsi, selon l'étude de [Golhar and Stamm, 1991] le succès dans la mise en oeuvre de la philosophie du Juste-A-Temps dépend de l'exécution de quatre points clés absolument incontournables :

1. l'élimination physique de tout ce qui n'est pas strictement utile au but recherché ;
2. l'adhésion spontanée des salariés de l'entreprise aux processus de décision ;
3. la participation volontaire des fournisseurs et des sous-traitants à l'opération ;
4. une maîtrise absolue de la qualité des produits.

Même si l'ensemble de ces quatre points est indispensable pour réussir complétement le passage au Juste-A-Temps, on peut néanmoins chercher à les étudier séparément. Avant de les développer un par un signalons que seul le premier point interesse directement la

suite de notre travail, le deuxième et le quatrième pointent des domaines qui sont plus éloignés, quand au troisième signalons seulement qu'il rejoint une notion qu'on rencontre fréquemment et que nous ne traiterons pas ici, à savoir le *supply chain management*. Mais reprenons donc de façon plus détaillée ce que [Golhar and Stamm, 1991] recense.

Tout d'abord, l'élimination des gaspillages. Dans ce même article, on met en évidence 11 points pour cette clef :

1. la mise en oeuvre d'un système d'étiquettes ou *kanban* ;
2. la chasse aux gaspillages ;
3. la réduction des temps de montage et de démontage ;
4. une production stable c'est-à-dire fluide et sans à-coups ;
5. des cellules de production en forme de U lorsque la configuration des ateliers le permet ;
6. la réduction de la taille des lots ;
7. la technologie de groupe c'est-à-dire la conception même du produit facilitant la mise en oeuvre ;
8. la réduction des délais de fabrication ;
9. l'automatisation ;
10. un flux de production pièce à pièce ;
11. les bons outils placés au bon endroit

En ce qui concerne la participation des employés au processus de décision, on peut distinguer 5 points importants :

1. la formation croisée des salariés ;
2. la prise de décision en groupe de travail ;
3. la résolution des problème en réseau ;
4. la possibilité pour les subordonnés d'émettre des suggestions ;
5. et non des moindres dans l'optique japonaise de fidélité des travailleurs, une forme d'emploi à vie au service de l'entreprise.

Quant à la contribution volontaire des sous-traitants, on peut noter :

1. des livraisons fréquentes et fiables ;
2. un engagement dans la qualité globale ;
3. une taille de lots plutôt petite ;
4. la création de réseaux de sous-traitants ;
5. la nécessité et l'importance de communiquer correctement et fréquemment avec ces fournisseurs ;
6. la réduction des délais ;
7. la proximité avec le client ;
8. ne pas multiplier sans raison le nombre de fournisseurs ;
9. privilégier avec les fournisseurs les contrats à long terme ;
10. former et informer les fournisseurs.

Pour ce qui est de la qualité totale on peut distinguer les 8 principes appliqués de mise en oeuvre :

1. le processus continu d'amélioration constante de la qualité ;
2. le fait que les opérateurs soient eux-mêmes responsables de l'inspection de leur propre travail ;
3. le droit pour quiconque de stopper complètement la production s'il estime que le résultat final pourrait être altéré c'est à dire qu'un problème doit impérativement être résolu avant de redémarrer ;
4. la maintenance préventive et prédictive remplaçant progressivement la maintenance curative ou réparation des pannes avant qu'elles ne surviennent ;
5. la prise d'autonomie ; item la visibilité des contrôles ;
6. la chasse à la variance ;
7. associée au statistic process control.

Tous ces points qui viennent d'être évoqués ne doivent malgré tout pas nous faire perdre de vue l'une des perspectives majeures qu'apporte le système Juste-A-Temps : il s'agit là d'un système tiré par l'aval. En effet c'est l'aval qui tire et non l'amont qui pousse à toutes les étapes de la fabrication, de la fin jusqu'au début des opérations, en d'autres termes tous les procédés antérieurs au dernier procédé (assemblage final) sont synchronisés au taux de production de ce dernier. [Nollet et al., 1994] compare ce système, toutes proportions gardées, à un système d'approvisionnement à double casier ; lorsque le premier casier est vidé de son stock, la commande est placée pour réactiver sa production, et il reste suffisamment de stocks dans le deuxième casier pour satisfaire la fabrication jusqu'à l'arrivée des composants. Un outil très simple bien qu'également très vulnérable pour réaliser la nécessaire synchronisation consiste en une circulation d'étiquettes manuelles sous formes de *cartes visibles* ou *kanban*. On peut trouver un exemple dans [Nollet et al., 1994] si nécessaire.

En résumé donc, rappelons nous que le système Juste-A-Temps est appelé parfois également système en flux tendu. N'oublions pas que la méthode kanban, mise en oeuvre avec succés dans les usines Toyota est une possibilité non des moindres, mais pas la seule de réaliser dans les faits le flux tendu. Certains parlent de façon systèmatique de flux tendus de type kanban. C'est bien évidemment un abus de langage tout à fait regrettable. Mais revenons un instant au flux tendu en tant que tel. Les objectifs sont : tout d'abord de synchroniser le mouvement des matériaux tout au long du système de fabrication et de distribution à la vitesse régulière nécessaire pour permettre une diffusion du flux dans tout le système ; il s'agit également de limiter les stocks vus comme des engorgements dans ce processus de diffusion ; le flux devient alors analysable, perfectible et la boucle est bouclée pour de nouveau progrès se renforçant mutuellement. A l'opposé, la méthode traditionnelle peut être traitée de flux poussés dans la mesure où il s'agit de garder la maîtrise des ordres de fabrication et des matériaux d'ateliers en ateliers, de stades de fabrication en stades de fabrication. Dès qu'un ordre de fabrication est terminé sur un stade, il est poussé vers le stade suivant dans la gamme quelle que soit la disponibilité du stade aval pour le traiter au moment venu.

1.4 Les avantages de l'adoption du Juste-A-Temps

L'implémentation des principes présentés dans la section précédente, à condition qu'elle soit réussie est censée offrir de substantiels avantages dans la compétition économique aux entreprises les ayant adoptés. N'oublions pas en effet que l'intense compétition sur les marchés actuels a contraint les firmes à remettre en cause la façon dont elles fonctionnent. Les

sociétés occidentales ont été obligées de faire face à des déficits chroniques et à la gestion de la sous-traitance alors qu'une nouvelle catégorie de concurrents arrivait sur un marché de plus en plus fluctuant et exigeant. Ces nouveaux concurrents avaient adopté quant à eux une forme de gestion s'inspirant des principes du Juste-A-Temps et de l'amélioration perpétuelle. Est ainsi née l'idée d'adapter certaines de ces techniques en espérant en tirer tous les bénéfices potentiels.

La réponse fut positive et l'expérience montra, voir [Fullerton and McWatters, 2001] et [Lea and Min, 2003], que ces principes lorqu'ils sont appliqués correctement conduisent à une production de meilleure qualité, à des niveaux d'inventaires faibles, à des temps de réaction rapides et à des réponses adaptées aux attentes des clients. Ainsi, dans cette ère nouvelle de compétition globale, on constate que la clé de la survie des entreprises passe par une remise en cause des anciennes façons de produire.

On trouve ainsi toute une série d'articles sur la philosophie empirique de la gestion de la production. Souvent fondées sur des enquêtes auprès d'entreprises, elles mettent en évidence statistiquement les degrés et les modalités d'application de certain principes ainsi que l'appréciation quantifiée ou subjective de leur contribution à l'amélioration des performance de l'entreprise.

Ainsi dans [Huson and Nanda, 1995] les auteurs mènent une enquête statistique auprès d'une centaine d'entreprises américaines bien portantes et ayant fait le choix du JaT pour des raisons de survie dans une monde changeant. Au moyen d'une régression par moindres carrés entre des ratios internes et des ratios de bilan, ils apportent la preuve de la corrélation intuitive entre la mise en oeuvre d'une gestion de la production de type JaT et la création de richesses dans l'entreprise – mesurée par la dernière ligne des bilans annuels avant et après la décision d'adoption. En cela ils innovent par rapport aux études statistiques montrant la corrélation entre les pratiques de type JaT et les effets de type JaT en gestion de production. En effet se contenter d'étudier la minimisation des stocks, la réduction des gaspillages, l'amélioration de la productivité, la "cure d'amaigrissement" des procédés de fabrication revient à étudier le degré d'implication de l'entreprise dans la philosophie du JaT, mais pas le gain financier en fin d'exercice qu'on peut en attendre. Intuitivement l'un des premiers effets est l'augmentation de la rotation des stocks permise par un contrôle plus précis des flux d'information. Cependant cela impose de plus petites séries et augmente donc les couts unitaires de production directs. L'avantage ne se verra donc qu'on moment de la dernière ligne du bilan lorsqu'on aura déduit les économies réalisées sur les coûts de stockage. Un autre effet intuitif est la diminution de la part du coût de main d'oeuvre dans le coût total de fabrication à cause des progrès de productivité d'une part et de l'automatisation des moyens de production d'autre part. Finalement, l'étude statistique confirme les intuitions précédentes : on a bien une augmentation des profits malgré une diminution des marges opérationnelles.

Signalons également que dans [Sale and Inman, 2003] l'auteur mene une enquête sur le degré de satisfaction des managers utilisant la méthode traditionnelle par équilibrage des lignes, ceux utilisant le JIT (caractérisé par une réduction des temps de cycle – sur un produit – une diminution des stocks d'en cours et une réduction des temps de lancements (durée entre deux campagnes) et ceux utilisant leur nouvelle méthode soi-disant révolutionnaire (*TOC, Theory of Constraints*). Il apparait que les praticiens du JIT sont les moins satisfaits des améliorations de leur performances, peut-être parce qu'ils les placaient trop haut ? A noter que le lien entre JIT et TQM (qualité) n'est pas pris en compte.

Cela doit cependant être nuancé ([Durden, 1999]) par le fait que tout cela doit s'accompagner d'une adaptation des critères d'évaluation des performances et de la façon de

calculer les coûts internes. En particulier, il n'est pas sûr que l'externalisation de certains coûts signifie leur diminution mais plutôt leur transfert sur les agents les plus faibles dont les effets induits ne se feront pas sentir immédiatement mais peut-être à long terme. Cela dit, quelque soit l'impact réel du JAT ou ses effets supposés, force est de constater que les réflexions et les recherches qu'il a suscitées dans le domaine de la gestion de production ont permis un grand bond en avant à la fois quantitatif et qualitatif.

Remarquons cependant que si la diminution des stocks n'est pas en soi le but du Juste-A-Temps il n'en demeure pas moins vrai qu'un gros effort doit être consenti dans la conversion du regard concernant la notion même de stocks. En effet, même si cela tend à disparaître dans le domaine de la production en général, on peut avancer en caricaturant à peine que dans les anciennes organisations traditionnelles les stocks sont chéris et adorés dans la mesure où ils sont considérés comme un trésor de guerre que l'on peut monnayer en cas de difficultés graves. Bien au contraire, même si dans d'autres systèmes les stocks ne sont pas non plus ni franchement souhaités ni aimés, on peut oser affirmer que, dans l'optique Juste-A-Temps, le fait de stocker ne donne aucune valeur ajoutée au produit. C'est même une opération coûteuse qui est un gaspillage en puissance. Gérer un stock peut alors être assimilé à faire un dépôt à la banque sans laisser courir ou réclamer aucun intérêt ou usure à celle-ci et même pire en la payant pour garder ce capital inutilisé. En fait, dans une optique traditionnelle, le fait de tenir un stock bien fourni est considéré comme beaucoup moins onéreux que de résoudre à la racine tous les problèmes que l'existence et l'entretien de stocks pléthoriques masquent, cachent ou dissimulent. L'inefficacité du système est alors noyée sous une multitude de stocks-tampons. Le pari du Juste-A-temps est donc de mettre à jour ces difficultés plutot que d'adopter une politique mimant alternativement celle de l'autruche et celle du pompier : tous les coûts inutiles disparaitraient alors comme par enchantement. Bien sûr cela reste en partie utopique et atteignable seulement asymptotiquement mais si tout le monde s'y met on peut découvrir des trésors en faisant l'effort nécessaire au lieu de s'arc-bouter pour résister au changement.

Concluons en rappelant que contrairement à ce qu'on pourrait croire en première analyse, ça ne coûte pas plus cher de produire dans l'ordre et la propreté juste ce qui est nécessaire au moment où c'est réclamé. Cela influe même parfois directement sur le moral des travailleurs, le climat de travail, le niveau de la qualité ou le bon fonctionnement des machines. Ainsi, les entreprises qui ont choisi ce tournant il y a quelques années ne s'en portent pas plus mal et s'en félicitent même si certains s'accordent à dire qu'il n'en est pas nécessairement de même pour tous leurs salariés.

1.5 Gérer le système de production en Juste-A-Temps

Pour conclure ce chapitre introductif, nous voyons désormais comment articuler ce qui précéde avec l'objet de notre étude, à savoir l'ordonnancement multicritère de type Juste-A-Temps. En effet, il importe en particulier de bien cerner ce qui dans la philosophie précédente relève de la prise de décision opérationnelle. Rappelons à toutes fins utiles la distinction entre les niveaux stratégiques, tactiques et opérationnels dont une mention est faîte explicitement dans la littérature par [Anthony, 1965]. Les décisions stratégiques traduisent les politiques à long terme. Les décisions tactiques sont prises à moyen terme, elles concernent généralement la planification de la production. Les décisions opérationnelles assurent la flexibilité quotidienne comme la gestion des stocks, des ressources matérielles et de la main d'oeuvre pour satisfaire au mieux la demande des clients tout en s'inscrivant dans le réseau de contraintes inspiré des décisions stratégiques et tactiques. L'orientation du présent mé-

moire sera donc dorénavant focalisée sur l'amélioration du niveau opérationnel, c'est-à-dire l'ordonnancement à court terme qui a pour but de planifier le travail dans un atelier de production.

En effet, selon [Carlier and Chretienne, 1988], ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leur date d'exécution. La réalisation peut être par exemple un programme informatique ou le carnet de commandes d'un atelier. Les tâches sont des processus informatiques ou des traitements à appliquer à certaines pièces par des machines. Les ressources sont alors des processeurs et de la mémoire, ou des machines, ou du personnel. Etablir un ordonnancement revient à coordonner l'exécution de toutes les tâches en utilisant au mieux les ressources disponibles. On rencontre ainsi des problèmes d'ordonnancement dans de très nombreux types de systèmes, dès lors que l'on est amené à organiser des tâches à réaliser, à répartir des ressources, à planifier l'exécution des différentes parties élémentaires d'un tout. La programmation de cette exécution doit naturellement se faire en respectant un certain nombre de contraintes de faisabilité. On peut chercher également parmi les solutions ainsi réalisables celles qui minimisent un ou plusieurs critères.

Bien évidemment le présent travail se situe dans le cadre de l'ordonnancement que certains appellent *statique* c'est à dire que les données sont considérées comme connues des le départ du processus de décision. Ceci est fait en contraste avec ce que certains appellent l'ordonnancement *dynamique* où il se trouve que les jobs peuvent arriver un peu n'importe comment sans qu'on ait de moyens de vraiment savoir ce qui peut ou va advenir. Notons également que l'on reste bien entendu dans le cadre de l'ordonnancement dit déterministe c'est à dire qu'il est hors de question bien évidemment de considérer des données aléatoires dans un environnement stochastique.

Ainsi, parmi les principes et les techniques du JAT, les éléments qui ont mérité notre attention dans le cadre de la phase d'ordonnancement sont : dans un premier temps le respect unique des délais en exécutant les travaux le plus possible à l'heure, c'est-à-dire en évitant les retards aussi bien que les avances, c'est ce qui est explicité dans le chapitre 2 ; puis dans un second temps la réduction des stocks d'en-cours et la fluidification de la production grâce en particulier à l'adaptation de la taille des lots dans le cas de produits divisibles, c'est ce qui est explicité dans la seconde partie du chapitre 3.

Il est clair qu'on peut avoir l'illusion qu'une certaine vision du Juste-A-Temps peut se passer d'une phase organisée et centralisée de type ordonnancement. Ainsi dans [Nishi et al., 2000] les auteurs étudient un atelier virtuel de type flowshop hybride dans l'industrie chimique. Ils proposent une gestion de production dite décentralisée et autonome. Le caractère décentralisé répond à la logique japonaise du flux tiré. Le caratère autonome vise à découpler les étages, réduisant ainsi la propagation des aléas de fabrication. Dans ce modèle, chaque étage échange des données avec les autres étages (en aval) et établit ses propres dates dues "tentatives" (au plus tôt (amont), au plus tard (aval), absolue (livraison finale)). Chaque étage présente donc un problème à une machine (ou à machines parallèles) dont la fonction objectif est une fonction de coût estimant, outre des coûts de préparation dépendants de la séquence, des pénalités reflétant le retard et l'avance sur livraison ainsi que le retard et l'avance sur l'étage suivant (indiquant d'une part la faisabilité, d'autre part les coûts de stockage intermédiaires). Pour chaque étage la résolution se fait en deux temps, d'abord l'établissement de la séquence par recuit simulé puis un calcul des dates de début. Une fois chaque étage ordonnancé, on échange à nouveau les données et on recommence jusqu'à convergence. Des éssais de simulation montrent qu'on ne dégrade pas les performances (en temps et en valeur) par rapport à une gestion centralisée (qui emploierait le même recuit simulé).

De la même façon signalons également que [Ahmadi and Matsuo, 2000] les auteurs s'intéressent à la production en flux tendus, qu'ils appellent eux production JaT lorsqu'il est associé au système kanban. Le flux tendu a pour avantage de réduire les en-cours, de diminuer les coûts et les délais, d'améliorer la qualité. Cependant le JaT classique est adapté surtout aux productions de masse, avec un mix certes important mais ne variant pas trop, avec des produits certes évolutifs mais qui ont une certaine durée de vie, et lorsque le processus de fabrication permet de réduire significativement les temps de changement d'outils. La production de cartes à circuits imprimés qui finance cette étude ne vérifie pas ces dernières conditions et les auteurs développent un système spécifique conservant les avantages du flux tendu, qu'ils appellent 'mini' ligne de production où l'on reconfigure les segments de ligne de production en fonction des variations de charge. Ils proposent de résoudre une succession hiérachisée de problèmes d'optimisation (décrits comme des programmes mathématiques et résolus par les méthodes de la littérature). Niveau1 : estimation du nombre de 'mini' lignes ; Niveau2 : Regroupement des familles de produits dans les 'mini' lignes en fonction d'un 'index de similitude entre familles' ; niveau3 : segmentation des 'mini' lignes ; Ces calculs sont remis à jour tous les trois mois. Niveau4 : l'Ordonnancement réalisé toutes les semaines ; Niveau5 : la conduite de l'atelier se fait en temps réel, le flux tendu est réalisé en maintenant une borne supérieure serrée sur les en-cours.

Mentionnons enfin en passant pour ce qui concerne la planification une approche interessante du point de vue conceptuel et qui est parfois elle aussi appelée ordonnancement de type *juste à temps* : il s'agit du problème de base qui consiste à adapter les ordres de fabrication à un demande diversifiée et le plus possible équilibrée. On peut alors se référer aux travaux de [Kubiak, 2004] ou dans la même optique [Brauner and Crama, 2004].

Chapitre 2

Ordonnancer en Juste-A-Temps

2.1 Introduction

La littérature contient de nombreux travaux de recherche sur des problèmes d'ordonnancement de type Juste-à-Temps. Ces travaux font l'objet de synthèses dans différents état de l'art comme ceux notamment de [Baker and Scudder, 1990], [Hall and Posner, 1991], [Gordon et al., 2002a], [Gordon et al., 2002b], [Kaminsky and Hochbaum, 2004] ou encore [Gordon et al., 2004]. Les problèmes d'ordonnancement de type Juste-à-Temps peuvent être classés de deux façons, suivant la définition de leurs dates dues et suivant la définition des critères à optimiser.

Les dates dues résultent souvent du choix fait par le décideur et constituent des données pour l'analyste. Dans ce cas on considère que les *dates dues sont fixes*. En revanche il peut arriver que les dates dues d'un travail ou d'un ensemble de travaux soient le résultat de négociations entre le décideur et son client. Dans le cadre de cette négociation, le décideur peut alors avoir besoin d'un algorithme qui, rapidement, lui calcule un ordonnancement ainsi que les dates dues en tenant compte de la réalité du terrain. Cela permet de donner à son client une date due "réaliste" pour les commandes qu'il s'apprête à passer. On considère alors, du point de vue du problème d'ordonnancement, que les *dates dues sont inconnues*.
Il arrive souvent que les *dates dues soient distinctes*. Mais on trouve également des cas où elles sont identiques, on parle alors de *date due commune*. Dans ce dernier cas on peut encore distinguer deux cas : une date due commune est dite *non restrictive* lorsque l'augmenter d'une unité ne permet aucune amélioration de l'ordonnancement calculé. Une date supérieure à la somme des temps d'exécution de l'ensemble des travaux est évidemment non restrictive. Sinon on parle de *date due restrictive*.
Notons également que les problèmes à date due inconnue sont équivalents aux problèmes à date due fixe et non restrictive et que pour cette raison même, la seule différence n'est pas dans l'algorithme de résolution mais dans la valeur du résultat. Insistons encore sur le fait que la détermination des dates dues peut être cruciale dans certains problèmes même si dans la plupart de la suite de ce mémoire nous la considérerons comme relevant de la décision de niveau stratégique et donc comme une donnée des problèmes traités par la suite. En fait si l'on fixe à l'avance trop de dates trop proches les unes des autres, on risque d'obtenir l'inverse de l'effet escompté puisque cela reconduit à des niveaux de stocks élevés ce qu'on était justement censé vouloir éviter dans la pratique de la gestion de l'atelier en Juste-à-Temps.
Parmi les modèles les plus classiques de détermination des dates dues, on trouve le modèle *constant*, mais on peut aussi rencontrer le modèle *slack* avec une durée de latence commune

souvent estimée par le $\bar{C}$.

Les problèmes d'ordonnancement de type Juste-à-Temps peuvent également être séparés selon la fonction objectif optimisée. Ils mettent en jeux au minimum une mesure de l'avance et une mesure du retard des travaux, bien souvent aggrégés en une unique fonction objectif. Une première approche consiste à mesure l'avance des travaux non pas par rapport à leur date due mais par rapport à une date de début souhaitée (ce qui peut traduire l'arrivée de la matière première dans l'atelier). Dans ce cas, on parle en réalité de *promptitude* et non plus d'avance. Une seconde approche, qui a été la plus suivie dans la littérature, consiste à mesurer l'avance par rapport aux dates dues, auquel cas on a affaire au critère d'avance maximum E_{max} ou au critère d'avance totale pondérée $\bar{E}^w$.
Considérer l'avance dans une fonction d'aggrégation rend celle-ci non régulière ce qui implique que souvent pour calculer un ordonnancement optimal il peut être intéressant d'insérer des temps morts volontaires au sein de l'ordonnancement (confère section 2.2). Ainsi, il peut être intéressant de délayer le traitement d'un travail de sorte qu'il complète à sa date due, cela au détriment des coûts de stockage des produits semi-finis. Ainsi, pour certains problèmes rencontrés dans la littérature est introduit un critère reflétant ces coûts de stockage. Bien souvent, il s'agit du critère $\bar{C}$ qui constitue plus une estimation de la durée moyenne de séjour dans l'atelier. Nous verrons dans le chapitre 3 comment intégrer les coûts de stockage au sein du problème d'ordonnancement.
La diversité des problèmes conduit à un nombre important de fonctions objectifs dans la littérature sur les problèmes d'ordonnancement de type Juste-à-Temps. Par ailleurs, il n'existe pas à notre connaissance une définition précise d'un ordonnancement Juste-à-Temps. Ces deux raisons ont favorisé l'apparition de nombreux modèles différents. Une synthèse en est présentée dans [T'kindt and Billaut, 2002]. On constate néanmoins qu'un modèle a été largement abordé dans la littérature : il s'agit du modèle avance/retard où la fonction objectif est une combinaison linéaire de l'avance totale pondérée et du retard total pondéré. Cette fonction s'écrit $\bar{E}^\alpha + \bar{T}^\beta$ et on distingue plusieurs catégories de sous-problèmes en fonction des valeurs prises par les poids α_i et β_i :

1. les poids sont asymétriques, *i.e.* $\exists i, i = 1, ..., n$, tel que $\alpha_i \neq \beta_i$,
2. les poids sont symétriques, *i.e.* $\forall i, i = 1, ..., n$, $\alpha_i = \beta_i$,
3. les poids sont indépendants des travaux, *i.e.* $\forall i, i = 1, ..., n$, $\alpha_i = \alpha$ et $\beta_i = \beta$,
4. les poids sont dépendants des travaux, *i.e.* $\exists i, j = 1, ..., n$, tels que $\alpha_i \neq \alpha_j$ ou $\beta_i \neq \beta_j$.

Dans la suite de ce chapitre, nous nous intéressons dans un premier temps à un problème bien particulier qui est celui du calcul des dates de début optimales lorsque la séquence des travaux est fixée. Ce problème survient pour la majorité des problèmes d'ordonnancement de type Juste-à-Temps étant donné que la fonction de coût à optimiser n'est pas régulière. Les problèmes faisant exception à cette règle sont des problèmes à dates dues communes. La section 2.2 présente donc des algorithmes de calcul des dates de début. Ce chapitre est clôturé par la section 2.3 qui présente un état de l'art des travaux portant sur la résolution de problèmes connexes à ceux résolus dans les autres chapitres de la thèse, à savoir des problèmes à dates dues distinctes et minimisation de la fonction de coût $\bar{E}^\alpha + \bar{T}^\beta$

2.2 Calcul des dates de début optimales à séquence fixée

Comme cela a été souligné précédemment, les problèmes d'ordonnancement de type juste-à-Temps mettent en oeuvre des fonctions objectifs non régulières, ce qui implique que

l'ensemble des ordonnancements actifs ou semi-actifs n'est plus nécessairement dominant (voir notamment [Brucker, 1998]). Cela implique qu'il peut être intéressant d'introduire des temps morts volontaires avant le début des travaux dans l'objectif de réduire la valeur de la fonction objectif. Dans la littérature, des auteurs se sont intéressés au problème de calcul des dates de débuts optimales lorsque la séquence des travaux sur les machines est fixée ("optimal timing problem", en anglais). Souvent ce problème peut être résolu en temps polynomial ce qui est utile pour la mise au point d'algorithmes de résolution. Ainsi il devient possible de réduire cette résolution à la recherche du séquencement "optimal", *i.e.* celui qui après résolution du problème de calcul des dates de début conduit à l'ordonnancement optimal.
Dans cette section, nous présentons les principaux problèmes de calcul de dates de début, et leur résolution, abordés dans la littérature.

2.2.1 Le problème $1|d_i, seq|F_\ell(\bar{T}^\alpha, \bar{E}^\beta)$

Les travaux de [Garey et al., 1988] concernent principalement le problème $1|d_i|F_\ell(\overline{T}, \overline{E})$ avec $F_\ell(\overline{T}, \overline{E}) = \overline{T} + \overline{E} = \sum_{i=1}^{n} |C_i - d_i|$ qui est $\mathcal{NP}$-difficile. Ils s'intéressent alors à la résolution de ce problème quand la séquence des travaux est fixée, ce problème se notant $1|d_i, seq|F_\ell(\overline{T}, \overline{E})$, et proposent un algorithme optimal dont la complexité moyenne est en $O(n\log(n))$.

Les travaux sont placés itérativement selon l'ordre imposé. On note σ_{i-1} la séquence partielle obtenue à l'itération $i-1$. Soit J_i le travail à insérer à l'itération i. Si $C_{max}(\sigma_{i-1}) + p_i \leq d_i$ alors le travail J_i débute à la date $d_i - p_i$ et un temps mort est inséré avant J_i. Dans le cas contraire, si $C_{max}(\sigma_{i-1}) + p_i > d_i$ alors le travail J_i est traité immédiatement à la fin de la séquence σ_{i-1}. Ensuite une partie de la séquence $\sigma_i = \sigma_{i-1}//\{J_i\}$ va peut être être décalée pour pouvoir réduire la valeur du côut de cette séquence. On définit un bloc par un ensemble maximum de travaux consécutifs exécutés sans temps morts et nous notons B_k le dernier bloc de σ_i (figure 2.1). Si la majorité des travaux dans B_k sont en retard alors il est intéressant de décaler B_k vers la gauche jusqu'à ce que cela ne soit plus le cas ou jusqu'à ce que B_k ne puisse plus être décalé.

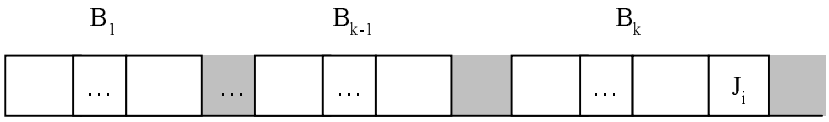

FIG. 2.1 – Définition d'un bloc

Le principe de cet algorithme, noté EGTW1, est présenté dans la figure 2.2.

Diverses extensions directes ont été étudiées dans la littérature notamment le problème $1|d_i, seq|\ F_\ell(\overline{E}^\alpha, \overline{T}^\beta)$ qui est abordé par [Szwarc and Mukhopadhyay, 1994]. Ce problème est résoluble en temps polynomial et les auteurs proposent un algorithme basé sur un découpage de la séquence en blocs. Lorsque $d_{i+1} - d_i \leq p_i$ les travaux J_i et J_{i+1} appartiennent au même bloc, ce qui par itération nous permet de déterminer quels sont les travaux qui doivent appartenir aux mêmes blocs. Par ailleurs il est seulement nécessaire d'insérer des temps morts entre deux blocs. L'algorithme proposé commence par construire un ordonnancement actif,

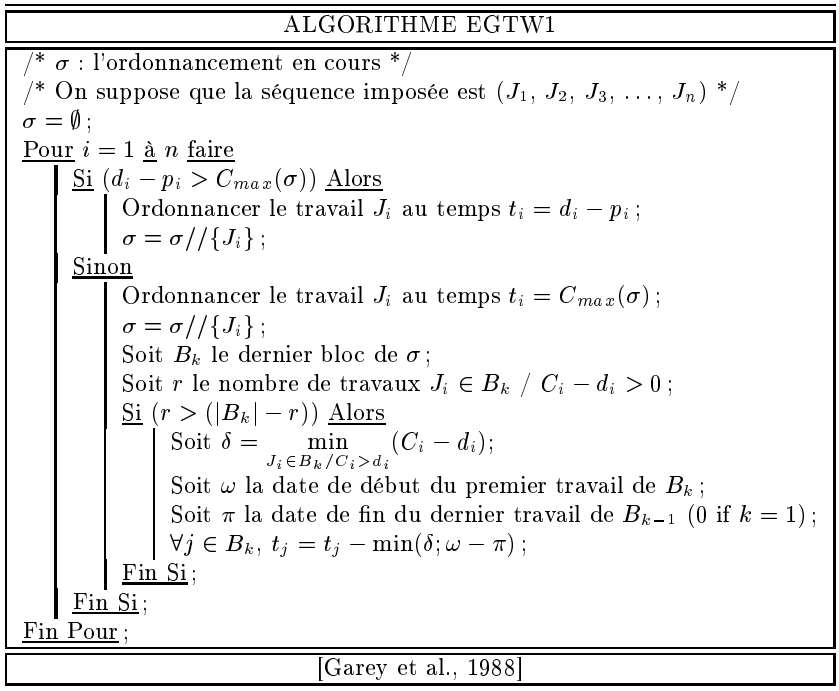

FIG. 2.2 – Un algorithme optimal pour un problème d'optimal timing

i.e. tous les travaux sont traités sans temps mort. En utilisant l'inégalité précédente les blocs sont identifiés. Ensuite les blocs sont décalés à droite en commençant par le premier bloc dans la séquence. Cet algorithme a une complexité temporelle en $O(cn)$ où c est le nombre de blocs et des résultats expérimentaux présentés par [Szwarc and Mukhopadhyay, 1994] montrent que pour des problèmes contenant jusqu'à 500 travaux le temps de calcul moyen est inférieur à 2 secondes. Par ailleurs, pour cette taille de problème, cet algorithme est 30 fois plus rapide que celui proposé par [Davis and Kanet, 1993a] pour le même problème.

2.2.2 Le problème $P\infty|prec, f_i\ convex|\sum_i f_i$

Ce problème d'ordonnancement général a été abordé par [Chrétienne and Sourd, 2003] qui, au premier abord, ne ressemble pas à un problème de calcul de dates de début optimales. Néanmoins, comme nous allons le voir, en particularisant de façon appropriée les fonctions de coût et les contraintes de précédence nous obtenons un problème de calcul de dates de début pour une classe de problèmes d'ordonnancement de type Juste-à-Temps. Chrétienne et Sourd présentent tout d'abord des résultats théoriques et un algorithme général qui sont brièvement retranscrits dans cette section. Notons que chaque travail J_i est défini par une date de début souhaitée s_i, un temps opératoire p_i et une fonction de coût $f_i(t)$.
L'algorithme général fonctionne d'une manière similaire à l'algorithme EGTW1. Les travaux sont triés selon leur rang dans le graphe de précédence et ordonnancés dans cet ordre. La notion de *bloc* est définie comme étant l'ensemble des travaux dans lequel, pour chaque travail soit sa date de début coincide avec la date de fin d'un autre travail de l'ensemble, soit sa date de fin coincide avec la date de début d'un autre travail dans l'ensemble. Quand un nouveau travail est ajouté dans l'ordonnancement partiel il est soit ordonnancé à sa date de début souhaitée si possible, ou ordonnancé à la fin du bloc B avec lequel il est en conflit. Dans ce dernier cas, le bloc B est augmenté et il devient nécessaire de tester s'il faut décaler

vers la gauche ou pas le nouveau bloc B.
Soit t_B la date de début de ce bloc, *i.e.* la plus petite date de début des travaux dans B. S'il n'existe pas $t < t_B$ tel que $\sum_{J_i \in B} f_i(t) < \sum_{J_i \in B} f_i(t_B)$ le block n'est pas décalé à gauche. Dans le cas contraire, soit t_B^* la date de début idéale pour laquelle la contribution du bloc B à la fonction de coût est minimale. B est décalé à gauche jusqu'à ce que soit il débute à t_B^* ou, comme dans l'algorithme EGTW1, il vienne buter sur un autre bloc B'. Dans ce cas le nouveau bloc B est défini par $B = B \cup B'$. Ensuite, on essaye de décaler à gauche le nouveau bloc B si nécessaire pour minimiser le coût total. Il peut également arriver que, lors du décalage d'un bloc B, il soit nécessaire de séparer en deux blocs le bloc B. Cela arrive lorsqu'un sous-bloc b de B est *à l'heure* tandis que le reste du bloc B a encore besoin d'être décalé vers la gauche pour réduire son coût total. Cela est dû au fait que les travaux peuvent être traités en parallèle et, évidemment, cet évènement ne peut pas survenir dans le problème abordé par Garey, Tarjan et Wilfong.
Notons que cet algorithme est polynomial à partir du moment où nous sommes capables de calculer en temps polynomial les dates de début idéales t_B^*. Cela est une conséquence du fait que les fonctions de coût f_i sont des fonctions convexes.

L'algorithme général est particularisé à différents cas spéciaux comme par exemple le problème d'ordonnancement avec des fonctions de coûts f_i linéaires et reflétant les coûts d'avance/retard. Des problèmes avec des contraintes de précédence particulières sont également étudiés. Nous nous intéressons au problème avec coûts d'avance/retard étant donné que le problème correspondant, noté $P\infty|s_i|F_\ell(\bar{E}^\alpha, \bar{T}^\beta)$, nous permet par la suite de résoudre une classe de problèmes de calcul de dates de début optimales. Cela est notamment abordé dans les chapitre 3 et 4.
Pour ce problème, nous supposons que chaque travail J_i est défini par une date de début souhaitée s_i, un temps opératoire p_i, une pénalité unitaire d'avance α_i et une pénalité unitaire de retard β_i. Le coût induit par le travail J_i dans l'ordonnancement est défini par $f_i = \alpha_i \max(0, s_i - t_i) + \beta_i \max(0, t_i - s_i)$. Etant donné que chaque travail ne possède qu'une unique opération, ce coût est équivalent à $\alpha_i E_i + \beta_i T_i$. Nous introduisons tout d'abord des notions théoriques nécessaires pour l'algorithme.
Pour chaque bloc B, soit G le graphe associé des contraintes de précédence avec les temps opératoires sur les arcs. On note $\Gamma(B)$ *l'arbre d'égalité actif couvrant* associé à B. $\Gamma(B)$ est défini comme étant un arbre couvrant sur G dans leque les arcs sont *actifs*, *i.e.* si nous supprimons un tel arc les deux sous-blocs créés B^1 et B^2 sont conflictuels : l'un doit être décalé pour réduire son coût ce qui entraine l'augmentation du coût de l'autre sous-bloc. Notons que l'arbre d'égalité actif couvrant d'un bloc n'a pas besoin d'être calculé à chaque itération de l'algorithm étant donné qu'il est simplement mis à jour lorsqu'on fusionne deux blocs : pour deux blocs fusionnés B and B' l'abre couvrant du nouveau bloc $B = B \cup B'$ est égal à $\Gamma(B) \cup \Gamma(B')$. Nous décrivons maintenant comment sont calculées les dates de début idéales t_B^* pour les fonctions particulières d'avance/retard.
Tout d'abord notons que les fonctions de coût f_i sont linéaires par morceaux, chacune admettant un point de rupture unique. Ainsi, on définit pour chaque bloc B un ensemble de *points singuliers* t_j^B définis par $t_j^B = s_j + t_B - t_j$, $\forall J_j \in B$, et chaque point singulier correspond à un changement dans la pente de la fonction de coût d'au moins un travail du bloc B. Ainsi, dans chaque actif d'égalité actif couvrant $\Gamma(B)$ nous stockons sur les arcs des pentes : pour un arc (i, j) la pente $\ell_{i,j}$ correspond à la contribution unitaire au coût total des travaux associés aux noeuds qui suivent j dans $\Gamma(B)$. Nous notons également par $\ell(B)$ la pente du bloc B qui peut être interprétée comme étant la contribution unitaire au coût total des travaux dans B. Par exemple, is B est constitué d'un travail J_i en avance et d'un travail

J_j en retard, nous avons $\ell(B) = \alpha_i + \beta_j$. Une conséquence est que tous les points singuliers d'un bloc sont les points de rupture de la fonction de coût du bloc, *i.e.* sa contribution au coût total. La date de début idéale t_B d'un bloc B est alors le point de rupture qui conduit au coût minimum et le décalage à gauche d'un bloc est fait d'un point de rupture vers le prochain à moins que l'on rencontre un autre bloc. Dans ce dernier cas, les deux blocs sont fusionnés et décalés à gauche si le nouveau bloc obtenu ne démarre pas à sa date de début idéale (sa pente est négative dans ce cas). Un autre cas peut survenir lorsqu'on rencontre un point de rupture : le bloc courant doit être séparé en deux sous-blocs étant donné qu'un sous-bloc est à l'heure (la pente d'un arc dans le graphe $\Gamma(B)$ est négative). Parmi les deux blocs, celui avec la pente positive est décalé à gauche. Les détails de cet algorithme, noté ECS1, sont donnés dans les figures 2.3 et 2.3.
Chrétienne et Sourd montrent qu'il peut être implémenté en $O(n \max(n, M))$ avec M le nombre de contraintes de précédence. Lorsque le graphe de précédence a une structure d'arbre, la complexité de l'algorithme $ECS1$ devient $O(n^2)$. La complexité temporelle peut encore être réduite à $O(n \log(n))$ dans le cas de contraintes de type chaîne.

L'algorithme $ECS1$ est un algorithme extrêmement important car il permet de résoudre de nombreux problèmes d'optimal timing. Il peut être également appliqué à des problèmes pour lesquels il y a des dates de début au plus tôt et des dates de fin impératives : pour cela il suffit d'introduire dans le graphe des travaux fictifs dans les données sont particularisées (voir [Estève et al., 2004] pour une application).

2.2.3 Le problème $1|f_i$ *linéaire par morceaux, seq*$|F_\ell(\sum_i f_i, \sum_j I_j)$

[Sourd, 2005] considère le problème dans lequel tous les travaux sont déjà séquencés sur une machine unique, sans perte de généralité, dans l'ordre $(J_1, J_2, ..., J_n)$. Chaque travail J_i est défini par un temps opératoire p_i et une fonction de coût f_i qui est linéiare par morceaux avec un nombre de segments qui peut être supérieur à deux. Par conséquent, ce problème généralise le problème d'avance/retard pour lequel les fonctions de coût possèdent seulement deux segments. Par ailleurs, nous supposons que chaque temps mort inséré j induit un coût mesuré par une fonction I_j. L'objectif est alors de résoudre le problème de calcul des dates de début optimales en minimisant $\sum_i f_i + \sum_j I_j$. Comme souligné par Sourd, les fonctions f_i permettent de modéliser un certain nombre de situations réelles comme par exemple la présence de fenêtres de temps dans lequels chaque travail doit être traité : si un travail ne peut pas être traité dans une certaine période de temps alors le coût correspondant f_i est mis à $+\infty$ durant cette période.

Le problème est montré $\mathcal{NP}$-difficile au sens faible. Sourd propose un algorithme de programmation dynamique pour le résoudre dont la complexité est en $O(n^2 BS)$, avec BS une borne supérieure à la valeur du critère C_{max} dans l'ordonnancement optimal. L'algorithme fonctionne également lorsque le problème n'est plus défini comme étant un problème à machine unique mais comme un problème d'ordonnancement général sans contrainte de ressources mais avec des contraintes de précédence de type arbre. D'autres problèmes particuliers sont également considérés, comme par exemple le problème d'avance/retard autour d'une date due commune.

ALGORITHME ECS1 (1)

```
/* A : l'ensemble des contraintes de précédence */
/* S : l'ensemble des travaux classés selon leur rang dans le graphe
     des contraintes de précédence */
b = 0 ; // b est le nombre de blocs
t_i = +∞, ∀i = 1, ..., n ;
Tant que S ≠ ∅ faire
    // Nous créons un nouveau bloc avec seulement un travail J_S[1]
    b = b + 1 ;
    B_b = {J_S[1]} ;
    Si (∀(i, S[1]) ∈ A, s_S[1] ≥ t_i + p_i)) Alors
        // Le travail J_S[1] est à l'heure
        Ordonnancer le travail J_S[1] au temps t_S[1] = s_S[1] ;
        ℓ_B_b = αS[1] ;
        S = S − {S[1]} ;
    Sinon
        // Le travail J_S[1] n'est pas à l'heure
        ℓ_B_b = βS[1] ;
        S = S − {S[1]} ;
        Tant que (∃B tel que ℓ(B) < 0) Faire
            Soit B un bloc avec ℓ(B) < 0 ; // B n'est pas à l'heure
            Décaler à gauche le bloc B jusqu'à ce qu'un des événements surviennent :
                1) ∃J_i ∈ B, tel que J_i débute à la date s_i.
                   // Nous mettons à jour les pentes du bloc en cours
                   ℓ(B) = ℓ(B) − α_i − β_i ;
                   Appeler maj_arbre(B, i, −α_i − β_i) ;
                2) ∃B', J_j ∈ BandJ_i ∈ B' tel quet_j = t_i + p_i.
                   // Nous fusionnons les blocs B et B'
                   Soit J_j ∈ B et J_i ∈ B' tels que t_j = t_i + p_i ;
                   Appeler maj_arbre(B,j,ℓ(B')) ;
                   Appeler maj_arbre(B',i,ℓ(B)) ;
                   G = G ∪ G' ∪ {(i, j)} ;
                   B = B ∪ B' ; ℓ(B) = ℓ(B) + ℓ(B') ;
                   Γ(B) = Γ(B) ∪ Γ(B') ∪ {(i, j)} ;
                   b = b − 1 ; Renuméroter les blocs consécutivement ;
                3) ∃(i, j) ∈ Γ(B) tel que ℓ_i,j < 0.
                   // Nous séparons le bloc B entre les travaux J_i et J_j
                   Soit B = B' ∪ B'' avec B'' le bloc défini par
                     les travaux suivant J_j dans G ;
                   Séparer Γ(B) et G en sous-graphes ;
                   ℓ(B'') = ℓ_i,j ; ℓ(B') = ℓ(B) − ℓ_i,j ;
                   Appeler maj_arbre(B',i,−ℓ(B'')) ;
                   Appeler maj_arbre(B'',j,−ℓ(B')) ;
                   b = b + 1 ; Renuméroter les blocs consécutivement ;
        Fin Tantque ;
    Fin Si ;
Fin Tantque ;
```

[Chrétienne and Sourd, 2003]

FIG. 2.3 – Un algorithme optimal pour calculer les dates de début des travaux pour le problème $P\infty|s_i|F_\ell(\bar{E}^\alpha, \bar{T}^\beta)$

ALGORITHME ECS1 (2)
Procédure maj_arbre(B,i,δ) ; Définir i comme étant visité ; Pour (chaque j non visité tel que $\ell_{i,j} < 0$ ou $\ell_{j,i} < 0$ dans $\Gamma(B)$) Faire Si $(j,i) \in A$ Alors $\ell_{j,i} = \ell_{j,i} + \delta$; Appeler maj_arbre(B,i,δ) ; Fin Pour ;
[Chrétienne and Sourd, 2003]

FIG. 2.4 – La procédure maj_arbre de l'algorithme $ECS1$

2.3 Etat de l'art de certains problèmes avec dates dues distinctes

Dans cette section nous présentons un état de l'art sur les problèmes d'ordonnancement à une machine et de flowshop de type Juste-à-Temps avec dates dues distinctes et fonction objectif de type avance/retard. Ces problèmes correspondent à des problèmes qui seront abordés ultérieurement dans ce document. On considère les cas où cette fonction objectif s'exprime par $\bar{E} + \bar{T}$ ou par $\bar{E}^{\alpha} + \bar{T}^{\beta}$ et on distingue les problèmes où l'insertion de temps mort volontaire est autorisée de ceux où elle est interdite (contrainte *nmit* pour "no machine idle time").

2.3.1 Problèmes sans insertion de temps morts et avec la fonction $\bar{E} + \bar{T}$

Le problème $1|d_i, nmit|\bar{E} + \bar{T}$ est $\mathcal{NP}$-difficile car la version à date due commune, *i.e.* $d_i = d$, $\forall i = 1, ...n,$, l'est également ([Sundararaghavan and Ahmed, 1984]). Lorsque des poids sont pris en compte, *i.e.* lorsqu'on minimise $\alpha\bar{E}+\beta\bar{T}$, [Azizoglu et al., 1991] proposent des heuristiques de résolution ainsi qu'une méthode exacte arborescente. Des résultats expérimentaux montrent que cette dernière est capable de résoudre des problèmes contenant jusqu'à 20 travaux.

Un problème proche du problème $1|d_i, nmit|\bar{E} + \bar{T}$ est celui ou la fonction objectif est définie par $\sum(E_i + T_i)^2$. Pour le problème correspondant, [Gupta and Sen, 1983] proposent une méthode exacte de type procédure par séparation et évaluation et une méthode heuristique. Cette dernière est un méthode arborescente tronquée et des résultats expérimentaux proposés par les auteurs montrent l'efficacité de cette méthode.

2.3.2 Problèmes avec insertion de temps morts et avec la fonction $\bar{E} + \bar{T}$

Le problème $1|d_i|\bar{E} + \bar{T}$ est $\mathcal{NP}$-difficile car la version à date due commune, *i.e.* $d_i = d$, $\forall i = 1, ...n,$, l'est également ([Sundararaghavan and Ahmed, 1984]). Ce problème est abordé par [Kim and Yano, 1994] qui proposent des heuristiques de résolution et une méthode exacte pour résoudre ce problème. Ces heuristiques fonctionnent en deux étapes : une heuristique de liste est appliquée pour obtenir une séquence des travaux, le problème de calcul des dates de début optimales étant ensuite résolu pour chaque séquence obtenue. Ce dernier problème est résolu à l'aide de l'algorithme EGTW1 (section 2.2.1). Kim et Yano proposent également une méthode exacte de type procédure par séparation et évaluation.
[Fry et al., 1996] s'intéressent également à la définition d'une méthode exacte de type procé-

dure par séparation et évaluation. Celle-ci est basée sur un découpage en blocs du problème. Des résultats expérimentaux montrent que des problèmes contenant jusqu'à 25 travaux peuvent être résolus en un temps raisonnable.
Par la suite, [Chang, 1999] se focalise sur la résolution exacte de ce problème à l'aide d'une méthode de type procédure par séparation et évaluation. Il propose une borne inférieure basée sur l'élimination des zones de recouvrement entre travaux : en supposant que tous les travaux terminent à leur date due, une estimation par défaut des décalages temporels nécessaires pour réparer les violations des contraintes disjonctives est calculée. Cette borne inférieure est utilisée dans la procédure par séparation et évaluation et des résultats expérimentaux montrent que des problèmes contenant jusqu'à 35 travaux peuvent être résolus en un temps raisonnable. Aucune comparaison avec la méthode de Kim et Yano ou de Fry et al. n'est présentée.

La prise en compte d'une mesure du coût de stockage est réalisée par [Fry et al., 1987] qui abordent le problème $1|d_i|\bar{E} + \bar{T} + \bar{C}$. Le critère $\bar{C}$ est ici considéré comme une mesure indirecte des coûts de stockage : minimiser le temps de séjour moyen dans l'atelier implique que, globalement, on réduit les coûts de stockage. Ce problème est $\mathcal{NP}$-difficile et est résolu à l'aide d'une procédure par séparation et évaluation. Par ailleurs le problème de calcul des dates de début optimales à séquence fixé, est résolu à l'aide d'un modèle mathématique. Des résultats expérimentaux montrent que des problèmes contenant jusqu'à 15 travaux peuvent être résolus. Ce modèle mathématique avait déjà fait l'objet d'études par [Fry and Leong, 1986] dans le cas où le critère $\bar{C}$ n'est pas pris en compte.

2.3.3 Problèmes sans insertion de temps morts et avec la fonction $\bar{E}^\alpha + \bar{T}^\beta$

Le problème $1|d_i, nmit|\bar{E}^\alpha + \bar{T}^\beta$ a fait l'objet de nombreuses études contrairement au problème sans poids. Il est important de noter que la contrainte *nmit* implique qu'il n'y a pas de réel problème de calcul des dates de début à résoudre et le problème d'ordonnancement se ramène à un problème de séquencement. Au pire, il faut juste déterminer la date de début du premier travail de la séquence, la contrainte *nmit* impliquant juste bien souvent qu'il n'y a pas de temps mort entre les travaux mais qu'il peut y en avoir avant le premier travail de la séquence.
Les travaux précurseurs sont ceux de [Ow and Morton, 1988] et [Ow and Morton, 1989] qui proposent, pour le problème $1|d_i, nmit|\bar{E}^\alpha + \bar{T}^\beta$, des règles de priorité composites *lin-et, exp-et* tenant compte aussi bien des pénalités d'avance que de retard. Ils proposent également une implémentation de l'heuristique de type recherche par faisceaux filtrés qui est une méthode arborescente tronquée. Des résultats expérimentaux montrent que cette dernière heuristique fournie de bons résultats.
[Li, 1997] propose pour ce problème une heuristique par voisinage basé sur des opérateurs de type "non adjacent pairwise interchange". Cette heuristique utilise comme séquence initiale le résultat retourné par une amélioration de l'heuristique EXP-ET de Ow et Morton. Li présente également une méthode exacte de type procédure par séparation et évaluation dont la borne inférieure est basée sur une relaxation lagrangienne du problème. Des résultats expérimentaux montrent que la méthode exacte est capable de résoudre des problèmes contenant jusqu'à 25 travaux. [Liaw, 1999] propose également une méthode exacte de type procédure par séparation et évaluation dont la borne inférieure est calculée par résolution d'une relaxation lagrangienne du problème. La borne supérieure utilisée est une recherche locale sur la séquence calculée par les règle EXP-ET et LIN-ET. En réalité, l'apport théorique de Liaw par rapport aux travaux de Li est réduit. Cela est confirmé par les résultats

expérimentaux qui montrent que la procédure par séparation et évaluation de Liaw possède sensiblement les mêmes performances que celle de Li.
La résolution heuristique de ce problème est investiguée par [Almeida and Centeno, 1998] qui proposent une heuristique appliquant alternativement un algorithme tabou, un algorithme de recuit simulé et un algorithme de descente locale. La séquence initiale sur laquelle sont appliqués ces algorithmes est calculée à l'aide de la règle EXP-ET de Ow et Morton.

[Zegordi et al., 1995] étudient le problème de flowshop à m machines avec pénalités pondérées d'avance et de retard, ce problème se notant $F|nmit, d_i|\bar{E}^\alpha + \bar{T}^\beta$. Ce problème est évidemment $\mathcal{NP}$-difficile puisque le problème à une machine l'est. Les auteurs proposent une heuristique de type recuit simulé dont la particularité est l'opérateur de voisinage. Cet opérateur est basé sur une évaluation par défaut de la perturbation de la fonction objectif pour chaque permutation de travaux réalisable. La meilleure permutation, *i.e.* celle qui augmente le moins le coût, est choisie dans les conditions du recuit simulé.

[Yoon and Ventura, 2002] abordent la résolution du problème $F|nmit, lot-streaming, d_i|\bar{E}^\alpha+\bar{T}^\beta$ dans lequel la contrainte *lot-streaming* indique qu'une opération peut être découpée en sous-lots traités consécutivement. Lorsque sur une machine un sous-lot est traité, il peut être transféré sur la machine suivante, si elle est disponible, sans attendre que l'opération complète soit terminée. Cette contrainte a un sens dans une politique de juste-à-temps puisqu'elle permet de réduire les cycles de production. Les auteurs considèrent différentes configurations de l'atelier en fonction notamment de la capacité des zones de stockage entre les machines. Des modèles mathématiques et des heuristiques par voisinage sont présentés.

2.3.4 Problèmes avec insertion de temps morts et avec la fonction $\bar{E}^\alpha+\bar{T}^\beta$

Le problème $1|d_i|\bar{E}^\alpha+\bar{T}^\beta$ est $\mathcal{NP}$-difficile puisque le problème $1|d_i|\bar{T}^\beta$ est un cas particulier qui l'est également. [Yano and Kim, 1991] s'intéresse au cas particulier où les coefficients de pénalité sont proportionnels aux durées opératoires. Plus précisément on a $\alpha_i = \alpha p_i$ et $\beta_i = \beta p_i$, $\forall i = 1, ..., n$. Les auteurs mettent alors en évidence une condition de dominance qui permet d'éliminer de nombreuses séquences dans une procédure par séparation et évaluation qu'ils proposent. Plusieurs heuristiques sont également présentées : la séquence est déterminée par des listes de priorité et les temps morts sont insérés par un algorithme dédié de programmation dynamique. Les ordonnancements obtenus sont ensuite améliorés à l'aide d'une recherche locale.
[Lee and Choi, 1995] abordent le problème général avec poids quelconques et proposent un algorithme génétique. Chaque individu est représenté par l'ordre des travaux auquel on adjoint une information s'il y a des temps morts ou non entre travaux. Le problème de calcul des dates de début optimales est résolu par une méthode proposée par les auteurs et qui est basée sur la constitution progressive de blocs. Cet algorithme est une extension de l'algorithme $EGTW1$.
[James and Buchanan, 1997] proposent de résoudre le problème $1|d_i|\bar{E}^\alpha + \bar{T}^\beta$ à l'aide d'un algorithme Tabou. L'espace de recherche utilisé n'est pas l'espace des séquences mais un espace particulier où est associée à chaque travail une information binaire : il sera en avance ou il sera en retard. Etant donnée une spécification binaire dans cet espace de recherche, les auteurs développent une heuristique d'insertion dite *en cinq point* pour trouver un bon ordonnancement respectant la spécification binaire. La construction de la séquence et l'insertion de temps morts est conjointe dans cette heuristique. On a ainsi une façon d'évaluer un point de l'espace compressé, on peut alors chercher des voisins binaires et eux aussi les

évaluer. [James and Buchanan, 1998] proposent des améliorations de cette méthode.
Ce problème est repris par [Sourd and Kedad-Sidhoum, 2003] qui présentent une borne inférieure dont le temps de calcul est pseudo-polynomial (en $O(n^2T)$ avec T l'horizon d'ordonnancement). L'efficacité de cette borne est telle que, utilisée dans une PSE, on peut résoudre des problèmes contenant jusqu'à 30 travaux.

[Sridharan and Zhou, 1996] s'intéressent au problème $1|r_i, d_i|\bar{E}^\alpha + \bar{T}^\beta$ et présentent une heuristique qui résout en une passe le problème de séquencement et le problème de calcul des dates de début optimales. A chaque point de décision, tous les travaux dans la file d'attente plus quelques travaux sur le point d'arriver sont considérés pour déterminer quel travail sera le suivant et quand le faire débuter.

Chapitre 3

Une approche multicritère pour l'ordonnancement de type Juste-A-Temps

3.1 Rappels généraux sur l'optimisation multicritère

Nous rappellons dans un premier temps les définitions de base en optimisation multicritère, qui seront utilisées par la suite ([T'kindt and Billaut, 2002]). Soit $\mathcal{S}$ l'ensemble des solutions d'un problème et x un élément de $\mathcal{S}$. On note $Z = \{Z_j, 1 \leq j \leq K\}$ l'ensemble des critères conflictuels considérés et $\mathcal{Z}$ l'image de $\mathcal{S}$ dans l'espace des critères. On note z les images des éléments de $\mathcal{S}$ dans l'espace des critères.

Définition 1 *$x \in \mathcal{S}$ est un optimum de Pareto faible (solution faiblement efficace) si et seulement si $\nexists y \in \mathcal{S}$ tel que $\forall j, 1 \leq j \leq K, Z_j(y) < Z_j(x)$. L'ensemble des optima de Pareto faibles définit dans l'espace des critères la courbe de compensation, appelée aussi courbe d'efficacité.*

Définition 2 *$x \in \mathcal{S}$ est un optimum de Pareto strict (solution efficace) si et seulement si $\nexists y \in \mathcal{S}$ tel que $\forall j, 1 \leq j \leq K, Z_j(y) \leq Z_j(x)$ avec au moins une inégalité stricte. Un optimum de Pareto strict est également un optimum de Pareto faible.*

Ces notions sont illustrées dans la figure 3.1 dans le cas bicritère. Dans cette figure z^2, z^3, z^4 et z^5 sont des Pareto stricts et z^1 et z^6 sont des Pareto faibles non stricts. Bien souvent, on préfèrera s'intéresser à l'ensemble des Pareto stricts plutôt qu'à l'ensemble des Pareto faibles, car ce dernier peut contenir des solutions peu intéressantes pour le décideur.

De nombreuses approches de résolution sont possibles pour aborder un problème d'optimisation multicritère. Le lecteur pourra se référer à [T'kindt and Billaut, 2002] ou [Ehrgott, 2000] pour plus de détails. La méthode qu'il faudra implémenter est fonction de plusieurs facteurs :

- une compensation entre les critères est possible – ou non,
- il est possible – ou non – de pondérer les critères,
- on a – ou non – un objectif à atteindre pour chaque critère.

Les méthodes les plus souvent utilisées dans la littérature sont les suivantes :

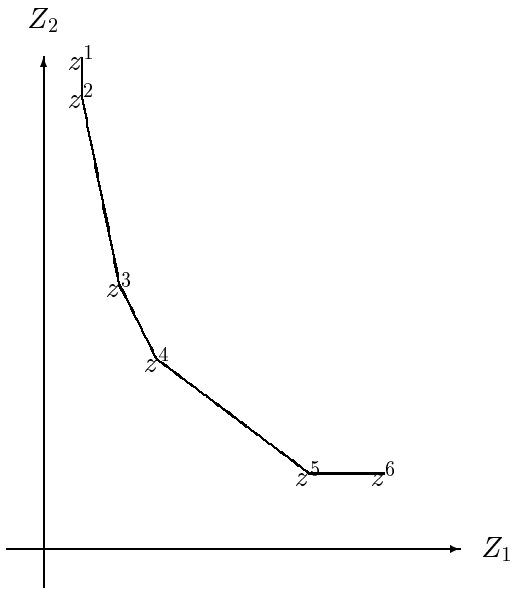

FIG. 3.1 – Illustration des définitions 1 et 2 pour deux critères

- définir un ordre lexicographique entre les critères et chercher une solution qui les optimise dans cet ordre,
- définir une combinaison linéaire des critères et chercher une solution qui optimise cette fonction objectif. Notons que cette technique ne permet d'atteindre que les solutions appartenant à la frontière de l'enveloppe convexe définie par l'ensemble des solutions. Dans le cas où l'espace est discret, il peut arriver qu'on n'atteigne pas certains optima de Pareto stricts potentiellement intéressants pour le décideur.
- définir une borne sur tous les critères sauf un, et chercher une solution qui optimise ce critère en respectant les bornes pour les autres critères. Cette approche, appelée approche ϵ-contrainte, lorsqu'elle est utilisée itérativement permet d'énumérer l'ensemble des optima de Pareto stricts.
- considérer une borne sur chaque critère et optimiser une fonction de l'ensemble des critères. Cette approche s'appelle l'analyse paramétrique.

Par la suite, nous utiliserons l'analyse paramétrique et l'approche ϵ-contrainte, contrairement à la plupart des travaux de la littérature qui abordent les problèmes de type Juste-à-Temps par des combinaisons convexes de critères.

3.2 Une fonction de coût pour le Juste-à-Temps

La littérature en "ordonnancement Juste-à-Temps" considère le plus souvent comme fonctions objectif à optimiser des fonctions de l'avance et du retard des travaux, qui sont des fonctions "en V" (voir figure 3.2). C'est notamment le cas de la fonction objectif $\sum_{1 \leq i \leq n} \alpha_i E_i + \beta_i T_i$, qui peut être vue comme une somme de "courbes en V".

Dans le chapitre 1, nous avons vu que le Juste-à-Temps considère toute action comme ayant un coût qui doit être le plus faible possible. Par analogie, la fonction à optimiser en "gestion de production" ferait donc plutôt intervenir une notion de coût de stockage et de respect des délais. Cette approche a d'ailleurs été utilisée dans l'étude d'un problème

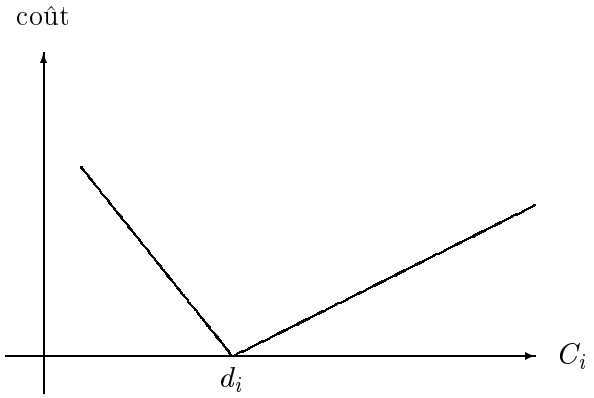

FIG. 3.2 – Fonction de coût en V pour le Juste-à-Temps

d'ordonnancement de type flow shop par [Ouenniche and Boctor, 2001]. L'objectif de notre travail est de mettre en évidence le lien entre la fonction objectif classique de la littérature ordonnancement et la notion de Juste-à-Temps en gestion de production. Ensuite, une fonction de coût étant définie pour chaque travail, des méthodes seront proposées pour résoudre le problème ainsi posé.

On considère un problème d'ordonnancement dans un atelier à m machines. Il s'agit d'ordonnancer n travaux. Chaque travail J_i est disponible à la date 0, et se compose de m opérations, chaque opération j devant s'effectuer sur la machine $M_{\pi_i(j)}$ $(1 \leq j \leq m)$. La séquence $\pi_i = (\pi_i(1), \pi_i(2), ..., \pi_i(m))$ est l'ordre dans lequelles les opérations sont traitées. Si π_i est fixée alors on a affaire à un problème de type jobshop ou flowshop. Dans le cas contraire on a affaire à un problème de type openshop.

3.2.1 Cas général

On considère que toutes les opérations sont divisibles. Elles sont toutes découpées en un nombre de sous-lots noté δ. De plus, la découpe est "équitable", autrement dit pour une opération donnée les sous-lots sont de taille identique.

On note $p_{i,j}$ la durée opératoire de l'opération j du travail J_i (sur la machine $M_{\pi_i(j)}$). Soit q_i le nombre de pièces du travail J_i, on suppose que q_i est un multiple de δ et on a donc q_i/δ pièces dans chaque sous-lot du travail J_i. On notera d_i la date due du travail J_i, donc de la dernière opération du travail J_i. Le *lot-streaming* est autorisé (voir notamment [Yoon and Ventura, 2002] pour une utilisation de cette contrainte dans un problème d'ordonnancement de type Juste-à-Temps), autrement dit un sous-lot d'une opération peut être transféré sur la machine suivante dès qu'il est terminé, sans attendre nécessairement la fin de l'opération complète.

La fonction de coût associée à J_i, notée Z_i, intègre trois facteurs qui s'additionnent :

1. le coût de stockage des pièces entre deux machines, depuis la première machine où s'exécute un travail jusqu'à la dernière, noté S_i,
2. un coût de pénalité lié au retard de la production, noté $\beta_i T_i$ avec T_i le retard du travail J_i et β_i un poids associé à J_i,
3. un coût lié au découpage des travaux en sous-lots, noté $\delta\lambda_i$ avec λ_i le coût unitaire d'un sous-lot.

On a donc :

$$Z_i = S_i + \beta_i T_i + \lambda_i \delta$$

Pour le calcul de S_i, on s'intéresse au coût de stockage des pièces entre deux machines successives de la gamme d'un travail. On note $\pi_i(j)$ le numéro de la machine qui exécute la j-ième opération du travail J_i. Pour un flowshop, on a $\pi_i(j) = j$ pour tout travail J_i. La figure 3.3 illustre l'évolution du stock du produit J_i au cours du temps, en supposant que la machine $M_{\pi_i(j-1)}$ est la première machine où s'exécute le travail J_i ou une machine sur laquelle les parties du travail J_i s'exécutent de façon consécutive.

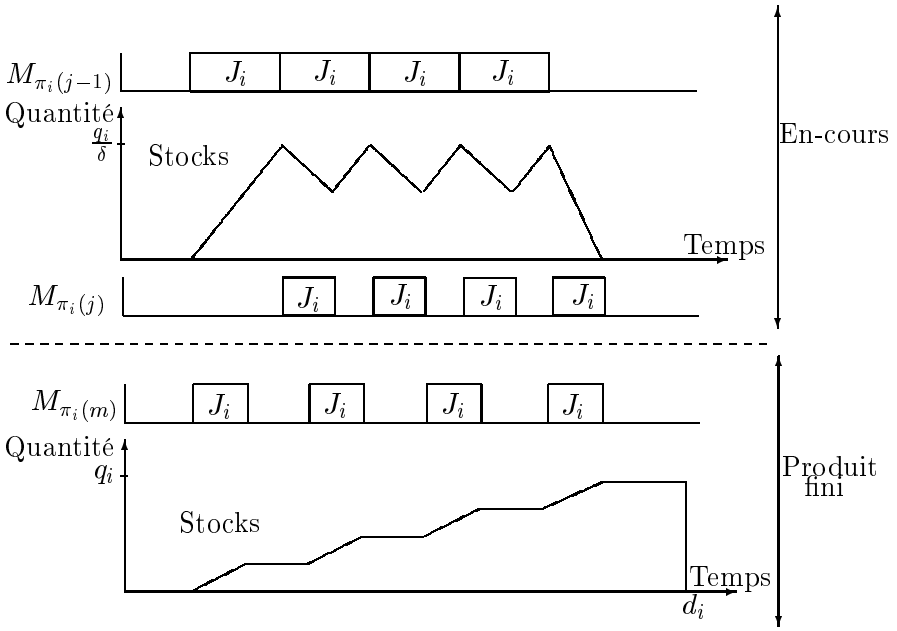

FIG. 3.3 – Evolution des stocks du produit J_i au cours du temps

On utilise les notations suivantes :
- $\gamma_i^{\pi_i(j)}$: le coût unitaire de stockage entre les machines $M_{\pi_i(j)}$ et $M_{\pi_i(j+1)}$ pour le travail J_i par unité de temps,
- k_i : le coût unitaire de stockage du produit fini J_i par unité de temps,
- $t_{i,j,k}$: la date de début du k-ième sous-lot du travail J_i pour l'opération j.

Nous allons maintenant établir la fonction qui donne le coût total associé au stockage du produit J_i. Cette fonction peut être établie de plusieurs façons, la technique que nous présentons consiste à calculer l'aire de la courbe des produits stockés ([Ouenniche and Boctor, 2001]). Cette courbe peut être décomposée en deux courbes, la première correspondant à la production de produit J_i par la machine $M_{\pi_i(j-1)}$, la seconde à la consommation par la machine $M_{\pi_i(j)}$. La figure 3.4 illustre cette production / consommation sur tout l'horizon de production d'une durée T avec T supposée suffisamment grand.

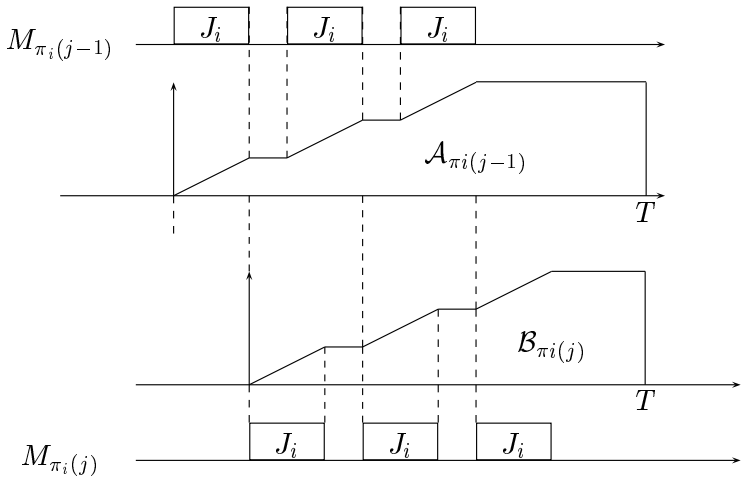

FIG. 3.4 – Illustration du processus de production / consommation

Calculer la fonction de coût revient donc à calculer la surface $\mathcal{A}_{\pi_i(j-1)} - \mathcal{B}_{\pi_i(j)}$ et à multiplier cette surface par le coût unitaire de stockage entre $M_{\pi_i(j-1)}$ et $M_{\pi_i(j)}$. Notons que dans le cas général, les courbes $\mathcal{A}$ et $\mathcal{B}$ ont toutes les deux la même forme, en "marches d'escalier".

Le calcul de $\mathcal{B}_{\pi_i(j)}$ est fonction de la date de début des δ sous-lots du travail J_i. Plus précisément, en considérant les notations des aires indiquées dans la figure 3.5, on a :

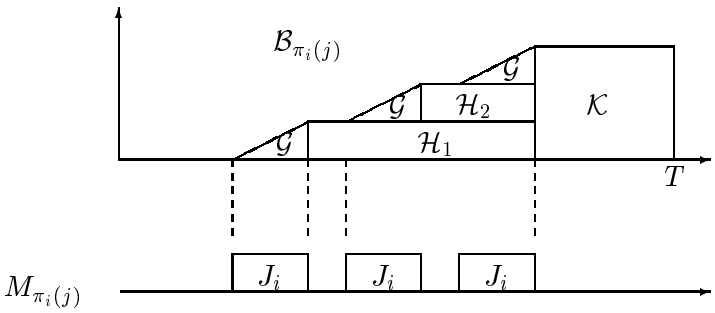

FIG. 3.5 – Calcul de la surface $\mathcal{B}_{\pi_i(j)}$

$$\mathcal{B}_{\pi_i(j)} = \delta \times \mathcal{G} + \sum_{k=1}^{\delta-1} \mathcal{H}_k + \mathcal{K}$$

avec $\mathcal{G} = \dfrac{(q_i/\delta) \times (p_{i,j}/\delta)}{2}$, $\mathcal{H}_k = (q_i/\delta) \times (t_{i,j,\delta} - t_{i,j,k})$ et $\mathcal{K} = q_i(T - t_{i,j,\delta} - \frac{p_{i,j}}{\delta})$.
d'où :

$$\mathcal{B}_{\pi_i(j)} = \frac{q_i}{\delta} \times \left(- \frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1} (t_{i,j,\delta} - t_{i,j,k}) \right) + q_i(T - t_{i,j,\delta})$$

La courbe $\mathcal{B}_{\pi_i(j)}$ qui représente le nombre d'éléments consommés à chaque instant de la machine $M_{\pi_i(j)}$ en produit J_i venant de $M_{\pi_i(j-1)}$ est la même que la courbe $\mathcal{A}_{\pi_i(j)}$ qui

représente le nombre d'éléments produits à chaque instant pour la machine $M_{\pi_i(j+1)}$. La distinction entre les deux stocks se fait par le coût unitaire de stockage qui n'est pas le même entre les couples de machines $(M_{\pi_i(j-1)}, M_{\pi_i(j)})$ et $(M_{\pi_i(j)}, M_{\pi_i(j+1)})$.

La fonction de coût associée au stockage du produit J_i, notée S_i s'écrit :

$$\begin{aligned} S_i = \quad & \gamma_i^{\pi_i(1)} \times (\mathcal{A}_{\pi_i(1)} - \mathcal{B}_{\pi_i(2)}) \\ & + \gamma_i^{\pi_i(2)} \times (\mathcal{A}_{\pi_i(2)} - \mathcal{B}_{\pi_i(3)}) + \ldots \\ & + \gamma_i^{\pi_i(m-1)} \times (\mathcal{A}_{\pi_i(m-1)} - \mathcal{B}_{\pi_i(m)}) \\ & + k_i \times (\mathcal{B}'_{\pi_i(m)} + \mathcal{C}_{\pi_i(m)}) \\ S_i = \quad & \sum_{j=1}^{m-1} \gamma_i^{\pi_i(j)} \times (\mathcal{A}_{\pi_i(j)} - \mathcal{B}_{\pi_i(j+1)}) \\ & + k_i \times (\mathcal{B}'_{\pi_i(m)} + \mathcal{C}_{\pi_i(m)}) \end{aligned}$$

où $k_i(\mathcal{B}_{\pi_i(m)} + \mathcal{C}_{\pi_i(m)})$ représente le coût de stockage du produit J_i fini jusqu'à sa date de livraison, s'il se termine en avance. Par ailleurs, on pose $\mathcal{B}'_{\pi_i(m)} = \mathcal{B}_{\pi_i(m)} - q_i(T - t_{i,\pi_i(m),\delta} - \frac{p_{i,m}}{\delta})$. Le calcul de $\mathcal{C}_{\pi_i(m)}$ est illustré figure 3.6. On a :

$$\mathcal{C}_{\pi_i(m)} = \max(0, (d_i - t_{i,m,\delta} - (p_{i,m}/\delta))) \times q_i$$

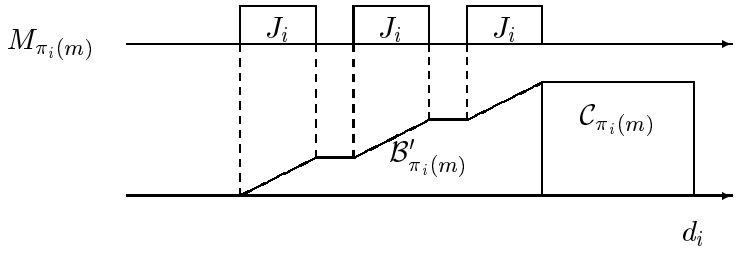

FIG. 3.6 – Calcul de la surface $\mathcal{C}_{\pi_i(m)}$

Puisque $\mathcal{A}_{\pi_i(j)} = \mathcal{B}_{\pi_i(j)}$ on en déduit :

$$S_i = \sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \mathcal{B}_{\pi_i(j)} + k_i \times \mathcal{C}_{\pi_i(m)} - k_i q_i (T - t_{i,\pi_i(m),\delta} - \frac{p_{i,m}}{\delta})$$

avec $\gamma_i^{\pi_i(0)} = 0$ et $\gamma_i^{\pi_i(m)} = k_i$.
On en déduit donc :

$$S_i = \Big(\sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \frac{q_i}{\delta} \times \big(-\frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1} (t_{i,j,\delta} - t_{i,j,k}) + \delta(T - t_{i,j,\delta}) \big) \Big)$$

$$+ \Big(k_i \times \max(0, (d_i - t_{i,m,\delta} - (p_{i,m}/\delta))) \times q_i \Big)$$

$$- k_i q_i (T - t_{i,\pi_i(m),\delta} - \frac{p_{i,m}}{\delta})$$

La fonction de coût globale associée à J_i s'écrit donc :

$$Z_i = \Big(\sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \frac{q_i}{\delta} \times \big(- \frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1} (t_{i,j,\delta} - t_{i,j,k}) \big) + \delta(T - t_{i,j,\delta}) \Big)$$
$$+ \Big(k_i \times q_i \times \max(0, d_i - t_{i,m,\delta} - (p_{i,m}/\delta))) \Big) - k_i q_i (T - t_{i,\pi_i(m),\delta} - \tfrac{p_{i,m}}{\delta})$$
$$+ \beta_i \times (\max(0, t_{i,m,\delta} + (p_{i,m}/\delta) - d_i)) + \lambda_i \delta$$

Chaque produit J_i est décomposé en δ sous-lots. Si on note C_i la date de fin du travail J_i sur la machine $M_{\pi_i(m)}$, donc la date de fin du dernier sous-lot de J_i, on a $C_i = t_{i,m,\delta} + (p_{i,m}/\delta)$ et l'expression de Z_i devient :

$$Z_i = \Big(\sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \frac{q_i}{\delta} \times \big(- \frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1} (t_{i,j,\delta} - t_{i,j,k}) + \delta(T - t_{i,j,\delta}) \big) \Big)$$
$$- k_i q_i (T - t_{i,\pi_i(m),\delta} - \tfrac{p_{i,m}}{\delta}) + k_i q_i E_i + \beta_i T_i + \lambda_i \delta$$

Le terme $\sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \frac{q_i}{\delta} \times \delta(T - t_{i,j,\delta})$ peut être simplifié pour obtenir $k_i q_i T - q_i \sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) t_{i,j,\delta}$.

L'expression de Z_i devient finalement :

$$\boxed{\begin{aligned} Z_i &= \Big(\sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) \times \frac{q_i}{\delta} \times \big(- \frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1} (t_{i,j,\delta} - t_{i,j,k}) \big) \Big) \\ &- q_i \sum_{j=1}^{m} (\gamma_i^{\pi_i(j)} - \gamma_i^{\pi_i(j-1)}) t_{i,j,\delta} + k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta \end{aligned}}$$

Cette expression fait intervenir une combinaison linéaire de C_i, E_i, T_i et des dates de début des sous-lots. Dans le reste de cette section nous donnons quelques particularisation à des cas particuliers intéressants.

3.2.2 Cas particuliers

3.2.2.1 Flowshop et produits non divisibles

Si on suppose que les produits sont non divisibles, alors on peut poser $\delta = 1$ et $q_i = 1$, $\forall i = 1, ..., n$. Comme on a affaire à un problème de type flowshop on a $\pi_i(j) = j$ et l'expression du coût devient :

$$Z_i = - \sum_{j=1}^{m} (\gamma_i^j - \gamma_i^{j-1}) \frac{p_{i,j}}{2} - \sum_{j=1}^{m} (\gamma_i^j - \gamma_i^{j-1}) t_{i,j} + k_i C_i + k_i E_i + \beta_i T_i + \lambda_i$$

On peut remarquer que certains termes du coût sont constants et on peut réécrire Z_i sous la forme :

$$Z_i = - \sum_{j=1}^{m} (\gamma_i^j - \gamma_i^{j-1}) t_{i,j} + k_i C_i + k_i E_i + \beta_i T_i + D$$

avec $D = - \sum_{j=1}^{m} (\gamma_i^j - \gamma_i^{j-1}) \frac{p_{i,j}}{2} + \lambda_i$

On notera que optimiser Z_i est équivalent à optimiser la fonction Z'_i :

$$Z'_i = -\sum_{j=1}^{m}(\gamma_i^j - \gamma_i^{j-1})t_{i,j} + k_i C_i + k_i E_i + \beta_i T_i$$

Lorsque l'atelier n'est constitué que de deux machines (*i.e.* $m = 2$), le coût Z_i devient :

$$Z_i = -\gamma_i t_{i,1} - k_i t_{i,2} + \gamma_i t_{i,2} + k_i C_i + k_i E_i + \beta_i T_i - \gamma_i \frac{p_{i,1}}{2} - k_i \frac{p_{i,2}}{2} + \gamma_i \frac{p_{i,2}}{2} + \lambda_i$$

$$\Leftrightarrow Z_i = \gamma_i(t_{i,2} - t_{i,1}) + k_i E_i + \beta_i T_i + D$$

avec $\gamma_i = \gamma_i^1$, $k_i = \gamma_i^2$ et $D = \gamma_i(\frac{p_{i,2}}{2} - \frac{p_{i,1}}{2}) + k_i \frac{p_{i,2}}{2} + \lambda_i$

3.2.2.2 Flowshop et produits divisibles

Comme on a affaire à un problème de type flowshop on a $\pi_i(j) = j$ et l'expression du coût se réécrit comme suit :

$$Z_i = \Big(\sum_{j=1}^{m}(\gamma_i^j - \gamma_i^{j-1}) \times \frac{q_i}{\delta} \times \big(-\frac{p_{i,j}}{2} + \sum_{k=1}^{\delta-1}(t_{i,j,\delta} - t_{i,j,k})\big)\Big)$$
$$- q_i \sum_{j=1}^{m}(\gamma_i^j - \gamma_i^{j-1})t_{i,j,\delta} + k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta$$

Si on s'intéresse au problème à deux machines (*i.e.* $m = 2$) on obtient :

$$Z_i = -\gamma_i \tfrac{q_i}{2\delta} p_{i,1} - k_i \tfrac{q_i}{2\delta} p_{i,2} + \gamma_i \tfrac{q_i}{2\delta} p_{i,2} + \gamma_i q_i \tfrac{\delta-1}{\delta} t_{i,1,\delta} + k_i q_i \tfrac{\delta-1}{\delta} t_{i,2,\delta} - \gamma_i q_i \tfrac{\delta-1}{\delta} t_{i,2,\delta}$$
$$- \tfrac{q_i}{\delta}\gamma_i \textstyle\sum_{k=1}^{\delta-1} t_{i,1,k} - \tfrac{q_i}{\delta} k_i \sum_{k=1}^{\delta-1} t_{i,2,k} + \tfrac{q_i}{\delta}\gamma_i \sum_{k=1}^{\delta-1} t_{i,2,k} - q_i \gamma_i t_{i,1,\delta} - q_i k_i t_{i,2,\delta} + q_i \gamma_i t_{i,2,\delta}$$
$$+ k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta$$

$$\Leftrightarrow Z_i = \tfrac{q_i}{\delta}\gamma_i(\textstyle\sum_{k=1}^{\delta} t_{i,2,k} - \sum_{k=1}^{\delta} t_{i,1,k}) - \tfrac{q_i}{\delta} k_i \sum_{k=1}^{\delta} t_{i,2,k}$$
$$+ k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta + D$$
avec $\gamma_i = \gamma_i^1$, $k_i = \gamma_i^2$ et $D = -\gamma_i \tfrac{q_i}{2\delta} p_{i,1} - k_i \tfrac{q_i}{2\delta} p_{i,2} + \gamma_i \tfrac{q_i}{2\delta} p_{i,2}$

On notera que optimiser Z_i est équivalent à optimiser la fonction Z'_i :

$$Z'_i = \frac{q_i}{\delta}\gamma_i(\sum_{k=1}^{\delta} t_{i,2,k} - \sum_{k=1}^{\delta} t_{i,1,k}) - \frac{q_i}{\delta} k_i \sum_{k=1}^{\delta} t_{i,2,k} + k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta$$

3.2.2.3 Problème à une machine

Considérons que l'atelier est réduit à une seule machine. Dans ce cas, on considère qu'il n'est pas nécessaire de constituer des sous-lots de fabrication. On pose $m = 1$, $\delta = 1$ et $q_i = 1$, $\forall i = 1, ..., n$. L'expression de la fonction de coût devient :

$$Z_i = \beta_i T_i + k_i E_i + k_i p_i / 2$$

Optimiser cette fonction est équivalent à optimiser Z'_i avec

$$Z_i = \beta_i T_i + k_i E_i$$

.

Ainsi, les problèmes d'ordonnancement à une machine de type Juste-à-Temps de la littérature qui minimisent $\sum_{j=1}^{n} Z_i = \beta_i T_i + k_i E_i$ minimisent bien le coût total d'ordonnancement Juste-à-Temps. Par contre, comme nous l'avons vu, minimiser un tel coût pour un problème de flowshop, par exemple, ne permet pas de minimiser le coût réel total d'ordonnancement Juste-à-Temps.

3.3 Modèle de programmation mathématique

L'expression de Z_i, le coût associé au travail J_i, fait intervenir les dates de début de tous les sous-lots de J_i. Utiliser la programmation mathématique pour modéliser le problème dans son ensemble paraît donc naturel. On considère ici le problème de flowshop de permutation, où $\pi_i(j) = j$, $\forall j, 1 \leq j \leq m$.

Les données du problème sont :
- n nombre de travaux,
- m nombre de machines, et donc d'opérations par travail,
- $p_{i,j}$ durée de la j-ième opération du travail J_i, $1 \leq i \leq n$, $1 \leq j \leq m$,
- d_i date de fin souhaitée du travail J_i,
- HV une grande valeur.

Les variables sont :
- $x_{i,i'}$ une variable booléenne égale à 1 si le travail J_i précède le travail J'_i, 0 sinon.
- $t_{i,j,k}$ la date de début du k-ième sous-lot du travail J_i sur la machine M_j.
- δ, le nombre de sous-lots par travail.
- $C_{i,j}$ la date de fin du travail J_i sur la machine j. Notons que $C_{i,j} = t_{i,j,\delta} + \frac{p_{i,j}}{\delta}$.
- T_i, retard du travail J_i. Rappelons que $T_i = \max(0, C_{i,\delta} - d_i)$.
- E_i, avance du travail J_i. Rappelons que $E_i = \max(0, d_i - C_{i,\delta})$.

Contraintes disjonctives :

$$t_{i',j,\delta} \geq t_{i,j,1} + \frac{p_{i,j}}{\delta} - HV(1 - x_{i,i'}), \forall i, 1 \leq i \leq n, \forall i', i' \neq i, 1 \leq i \leq n, \forall j, 1 \leq j \leq m \quad (3.1)$$

$$t_{i,j,\delta} \geq t_{i',j,1} + \frac{p_{i',j}}{\delta} - HV x_{i,i'}, \forall i, 1 \leq i \leq n, \forall i', i' \neq i, 1 \leq i \leq n, \forall j, 1 \leq j \leq m \quad (3.2)$$

L'expression de ces contraintes garantit que deux travaux consécutifs ne vont pas "s'emmêler", autrement dit le premier sous-lot d'un travail ne peut commencer que lorsque le dernier sous-lot du travail qui le précède est terminé.

Contraintes de succession des sous-lots d'un même travail :

$$t_{i,j,k+1} \geq t_{i,j,k} + \frac{p_{i,j}}{\delta}, \forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq m, \forall k, 1 \leq k \leq \delta - 1 \quad (3.3)$$

Contraintes de gamme entre les sous-lots d'un même travail :

$$t_{i,j+1,k} \geq t_{i,j,k} + \frac{p_{i,j}}{\delta}, \forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq m - 1, \forall k, 1 \leq k \leq \delta \quad (3.4)$$

Expression de la date de fin d'une opération, du retard d'un travail et de l'avance d'un travail :

$$C_{i,j} = t_{i,j,\delta} + \frac{p_{i,j}}{\delta}, \quad \forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq m \tag{3.5}$$

$$T_i \geq 0, \qquad \forall i, 1 \leq i \leq n \tag{3.6}$$

$$T_i \geq C_{i,m} - d_i, \qquad \forall i, 1 \leq i \leq n \tag{3.7}$$

$$E_i \geq 0, \qquad \forall i, 1 \leq i \leq n \tag{3.8}$$

$$E_i \geq d_i - C_{i,m}, \qquad \forall i, 1 \leq i \leq n \tag{3.9}$$

$$\text{expression de } Z_i, \qquad \forall i, 1 \leq i \leq n \tag{3.10}$$

La fonction coût Z_i étant une fonction de C_i, de T_i, de E_i et des dates de début des sous-lots en général, celle-ci peut être exprimée simplement, nous ne rappelons pas ici sa formulation qui vient juste d'être établie.

La variable δ intervient dans l'expression des contraintes au dénominateur. Le modèle proposé est donc non linéaire. Afin de s'affranchir de cet inconvénient en vue d'une résolution du modèle, la valeur de δ est supposée connue. Le problème devient donc linéaire, il peut être résolu par un solver comme CPLEX (ou OPL, XPRESS-MP, ...) et s'il le faut la valeur de δ peut être modifiée pour itérer le processus.

Il ne manque plus au modèle mathématique décrit ci-dessus, que la définition de la fonction objectif à optimiser. Selon le modèle de résolution envisagé, la fonction objectif aura une forme ou une autre, avec éventuellement des contraintes supplémentaires. L'objectif de ce modèle mathématique n'est pas de proposer une méthode de résolution générique, car son efficacité est limitée à un petit nombre de travaux. En revanche, il permet de faire des tests sur de petites instances et offre un cadre unificateur aux modèles envisagés décrits ci-après.

3.4 Modèles de résolution envisagés

Le problème a été modélisé de deux façons. La première consiste à associer un critère à chaque travail et à appliquer une méthode de résolution pour le problème à n critères. La seconde consiste à agréger les coûts en un seul critère et à définir un second critère. Le premier modèle sera appelé par la suite *le modèle multicritère*, le second sera appelé *le modèle bicritère*.

3.4.1 Modèle multicritère

Dans ce modèle on définit un critère Z_i associé à la fonction coût Z_i de chaque travail J_i (d'où la similitude des notations). Le problème à résoudre peut s'écrire de la façon suivante :

$$\begin{array}{l} \text{Minimiser } Z_1, Z_2, ..., Z_n \\ sc \\ (3.1) \text{ ... } (3.10) \end{array}$$

L'intérêt de ce modèle est qu'il permet au décideur de distinguer explicitement les travaux les uns des autres. Le décideur peut donc décider individuellement sur quels coûts il

peut faire jouer une compensation. Cela permet donc une gestion plus fine des travaux, en allant aussi près que possible des exigences du client, à la fois en termes de délais et en termes de réduction de gaspillage (réduction des coûts). Cela présente toutefois l'inconvénient de mettre en oeuvre beaucoup de critères, si la valeur de n croît.
Sur un plan plus théorique, ce modèle permet de réaliser une analyse multicritère des modèles d'ordonnancement de type Juste-à-Temps existants dans la littérature. Notamment, si l'on considère un problème à une machine sans lot-streaming, la fonction $Z_i = \alpha_i E_i + \beta_i T_i$. Le modèle avance/retard de la littérature revient donc à calculer un ou plusieurs optima de Pareto stricts par minimisation d'une combinaison convexe des critères. Or cette approche ne permet pas de calculer tous ces optima ([T'kindt and Billaut, 2002]) ce qui veut dire qu'en pratique certaines solutions potentiellement intéressantes ne peuvent pas être proposées au décideur.

3.4.2 Modèle bicritère

Dans ce modèle, on agrège les fonctions coût en une seule fonction et on ajoute la prise en compte explicite du retard. Le problème peut s'écrire de la façon suivante :

$$\text{Minimiser} \sum_{i=1}^{n} Z_i, T_{max}$$
$$sc$$
$$(3.1) \ldots (3.10)$$

$$T_{max} \geq T_i, \forall i, 1 \leq i \leq n \tag{3.11}$$

Dans ce modèle, tous les coûts de stockage sont agrégés en un coût global. Le critère T_{max} traduit le taux d'insatisfaction du client. En bornant ce critère, on peut garantir que le client ne sera jamais livré avec un trop grand retard. On associe donc un critère lié au coût et un critère qui traduit la satisfaction du client. L'avantage de ce modèle est qu'il ne comporte que deux critères, qui sont dans des grandeurs différentes. Le décideur peut accepter d'augmenter le retard si les coûts sont trop importants, de même il peut accepter un coût supplémentaire pour ne pas mécontenter ses clients.

3.5 Conclusion

Nous avons étudié dans ce chapitre la prise en compte des coûts de stockage dans un atelier où il était possible de constituer des sous-lots. Les fonctions coût mettent en évidence des fonctions de l'avance des travaux, du retard, et bien entendu des dates de début d'exécution des sous-lots.

Deux modèles de résolution sont envisagés : un modèle multicritère où on associe une fonction coût à chaque travail et un modèle bicritère.

Les problèmes abordés par la suite sont : un problème à une machine (chapitre 4) et un problème de flowshop à deux machines (5). Pour chacun de ces problèmes les deux modèles sont étudiés et différentes approches de résolution sont explorées.

Chapitre 4

Problème à une machine

4.1 Introduction

Nous considérons un problème à une machine, défini de la façon suivante : un ensemble de n travaux doit être exécuté sur une machine disjonctive unique, disponible en permanence. Chaque travail j possède une durée d'exécution p_j, une date due d_j. On note α_j et β_j la pénalité d'avance et de retard pour le travail j, respectivement. Il est possible de laisser la machine libre si nécessaire. La préemption n'est pas autorisée.

Etant donné un ordonnancement, on note C_j la date de fin d'exécution du travail j, E_j son avance et T_j son retard.

4.2 Modèle multicritère : calcul d'un optimum de Pareto

Nous définissons le problème comme un problème multicritère en introduisant n critères Z_j où chaque Z_j est égal à $Z_j = \alpha_j E_j + \beta_j T_j$ représentant le coût engendré par le travail j. Autrement dit, Z_j représente la déviation pondérée du travail j par rapport à sa date due. Le but est de déterminer des optima de Pareto stricts.

Dans ce paragraphe, nous nous intéressons au calcul d'un optimum de Pareto strict dans le cadre du modèle de résolution multicritère. Ce calcul est réalisé grâce à l'analyse paramétrique dont la base est le théorème de [Soland, 1979].

Théorème 1 *[Soland, 1979]*
Soit S l'ensemble des solutions et Z l'ensemble des vecteurs de critères correspondants et g une fonction de R^n dans R strictement croissante et bornée inférieurement sur Z, $x^0 \in S$ est un optimum de Pareto strict si et seulement si $\exists b \in R^n$ tel que x^0 est une solution optimale du problème suivant :
minimiser $g(Z(x))$
sous les contraintes :
$x \in S$
$Z(x) \leq b$
où $Z(x)$ est le vecteur de critères associé à la solution x.

Par soucis de simplicité, nous définissons g comme la somme des critères Z_j :

$$g(Z) = \sum_{j=1}^{n} Z_j = \sum_{j=1}^{n} \alpha_j E_j + \beta_j T_j$$

La contrainte $Z_j \leq b_j$, $\forall j, 1 \leq j \leq n$, conduit à l'introduction pour chaque travail j d'une date de disponibilité r_j et d'une date de fin impérative $\tilde{d}_j$ définies par :

$$r_j = \max(0, d_j - p_j - \lfloor \frac{b_j}{\alpha_j} \rfloor)$$

et

$$\tilde{d}_j = d_j + \lfloor \frac{b_j}{\beta_j} \rfloor$$

L'illustration de ces calculs est donnée dans la figure 4.1.

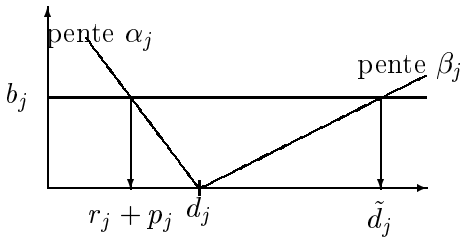

FIG. 4.1 – Calcul des r_j et $\tilde{d}_j$ associés à chaque travail j.

Quand le vecteur b est fixé, la résolution du $1|r_j, d_j, \tilde{d}_j| \sum \alpha_j E_j + \beta_j T_j$ permet de trouver un optimum de Pareto strict pour le problème multicritère.

Ce problème est NP-difficile au sens fort (voir chapitre 2). Remarquons que le problème de faisabilité $1|r_j, \tilde{d}_j|-$ est également NP-difficile au sens fort, même s'il est bien résolu de façon exacte, par exemple avec la PSE de Carlier [Carlier, 1982].

Après la présentation d'une modélisation mathématique du problème, nous décrivons les méthodes de résolution utilisées pour le résoudre, à savoir un algorithme d'insertion optimale de temps morts, une heuristique à base de voisinages et d'algorithmes de liste et enfin une recherche par faisceaux avec récupération des noeuds (*recovering beam search*).

4.2.1 Modèle de Programmation Mathématique

Ce problème peut se modéliser par un programme linéaire en nombres entiers de la façon suivante.

Les données ont été décrites précédemment, les notations sont conservées. On note M une grande valeur. Les variables sont : $x_{j,i}$ est égale à 1 si le travail i précède le travail j, 0 sinon ($1 \leq j \leq n$, $1 \leq i \leq n$) ; s_j la date de début du travail j ($1 \leq j \leq n$) ; T_j le retard du travail j ($1 \leq j \leq n$) ; E_j l'avance du travail j ($1 \leq j \leq n$). La fonction objectif est : Minimiser $Z = \sum_{j=1}^{n} \alpha_j E_j + \beta_j T_j$.

Les contraintes sont :

$$s_j \geq r_j \qquad \forall j, 1 \leq j \leq n \tag{4.1}$$
$$s_j + p_j \leq \tilde{d}_j \qquad \forall j, 1 \leq j \leq n \tag{4.2}$$
$$s_i \geq s_j + p_j - M(1 - x_{j,i}) \qquad \forall i, j, 1 \leq i, j \leq n, i \neq j \tag{4.3}$$
$$x_{i,j} + x_{j,i} = 1 \qquad \forall i, j, 1 \leq i, j \leq n, i \neq j \tag{4.4}$$
$$x_{j,j} = 0 \qquad \forall j, 1 \leq j \leq n \tag{4.5}$$
$$E_j \geq d_j - s_j - p_j \qquad \forall j, 1 \leq j \leq n \tag{4.6}$$
$$T_j \geq s_j + p_j - d_j \qquad \forall j, 1 \leq j \leq n \tag{4.7}$$
$$E_j, T_j, s_j \geq 0 \qquad \forall j, 1 \leq j \leq n \tag{4.8}$$
$$x_{j,i} \in \{0, 1\} \qquad \forall i, j, 1 \leq i, j \leq n \tag{4.9}$$

Ce modèle comporte $n^2 + 3n$ variables et $2n^2 + 4n$ contraintes. Les contraintes (4.1) et (4.2) imposent que les travaux ne peuvent pas commencer avant leur date de disponibilité ni finir après leur date impérative. Les contraintes (4.3), (4.4) et (4.5) sont les contraintes disjonctives du modèle. Les contraintes (4.6) et (4.7) donnent leur valeur aux variables d'avance et de retard.

Ce modèle a été évalué à l'aide du solver CPLEX. Les résultats sont donnés en fin de section.

4.2.2 Insertion optimale de temps morts

Lorsque la séquence des travaux est imposée, c'est-à-dire lorsque les variables $x_{j,i}$ sont fixées, le modèle précédent devient un programme linéaire qui se résoud en temps polynomial. Pourtant, au lieu de le résoudre en tant que tel, nous allons utiliser l'algorithme $ECS1$ en $O(n^2)$ présenté dans le chapitre 2 pour le problème $1|d_j|\sum \alpha_j E_j + \beta_j T_j$ ([Chrétienne and Sourd, 2002]).

Dans ce problème précis les auteurs supposent que les travaux sont seulement définis par p_j, d_j, α_j, et β_j. Dans notre problème, il faut ajouter forcément les dates de disponibilités des travaux et leurs dates de fin impératives. En fait, la présence de r_j distinctes peut aisément être prise en compte en introduisant n travaux fictifs définis comme suit : $p_{j'} = r_j$, $d_{j'} = r_j$, $\alpha_{j'} = \infty$, $\beta_{j'} = \infty$, $\forall j', n+1 \leq j' \leq 2n$. Il faut alors ajouter une contrainte qui impose que le travail $n + j$ précède le travail j, $\forall j, 1 \leq j \leq n$.

De la même façon, la présence des dates de fin impératives est prise en compte par l'introduction encore de n nouveaux travaux définis comme suit : $p_{j''} = 0$, $d_{j''} = \tilde{d}_j$, $\alpha_{j''} = \infty$, $\beta_{j''} = \infty$, $\forall j'', 2n+1 \leq j'' \leq 3n$. Il faut encore ajouter donc la contrainte de précédence que le travail j se doit d'être exécuté avant le travail $2n + j$.

Notre problème, avec des dates de disponibilité et des dates de fin impératives, a été transformé en un problème équivalent avec $3n$ travaux où on associe à chaque travail uniquement une date due. Les premiers n travaux sont ceux du problème original mais sans dates de disponibilité ni date de fin impérative. Les autres $2n$ travaux sont les travaux fictifs introduits ci-avant. Un graphe modélisant les contraintes de précédence peut alors être établi en considérant les contraintes correspondant au fait que la séquence est fixée et celles correspondant aux relations de type $(n + j) \rightarrow j \rightarrow (2n + j)$. Le modèle obtenu peut alors être résolu par l'algorithme de [Chrétienne and Sourd, 2002] dont l'implémentation est en $O(n^2)$. Prenons bonne note enfin pour conclure cette section que le problème d'infaisabilité à séquence fixée est détecté par le retour d'un coût de valeur infinie de la part de l'algorithme.

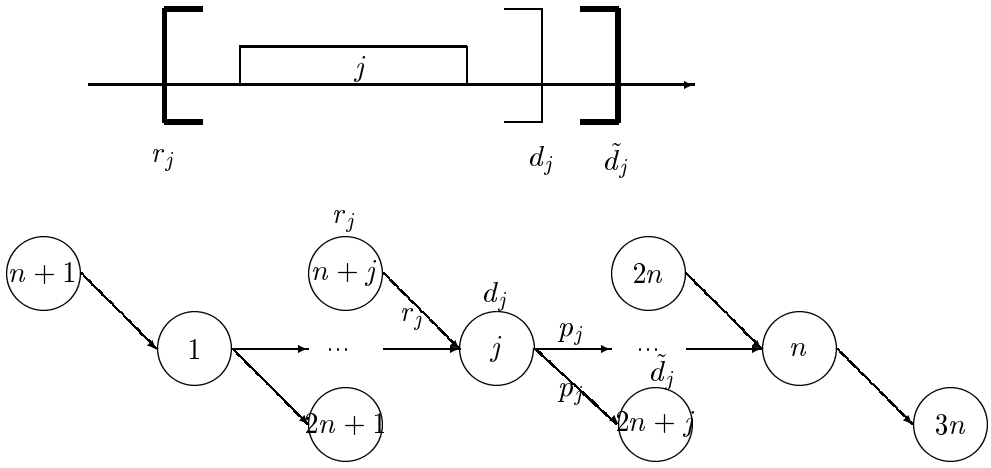

FIG. 4.2 – Instanciation du problème $1|d_j|\sum \alpha_j E_j + \beta_j T_j$

Le schéma de la figure 4.2 illustre l'instanciation du problème $1|d_j|\sum \alpha_j E_j + \beta_j T_j$ pour pouvoir utiliser l'algorithme de [Chrétienne and Sourd, 2002]. Dans cette figure, les sommets représentent les travaux $n+j$, j et $2n+j$, la valeur indiquée sur le sommet est la date due du travail, et la longueur de l'arc est la durée du travail. La liste de travaux est connue, elle est représentée ici en supposant qu'ils sont dans l'ordre 1, ..., n.

4.2.3 Méthodes approchées : algorithmes de listes

• Ow et Morton [Ow and Morton, 1989] proposent deux règles de priorité pour séquencer les travaux dans le problème avec temps morts interdits. La première règle est appelée LIN-ET (LIN pour linéaire et ET pour "early-tardy"). Elle consiste à donner à chaque travail une priorité dynamique (fonction du temps) de la façon suivante :

On pose :
- $m_i = (d_i - p_i) - t$, la marge de i
- $\omega_i = \beta_i / p_i$
- $h_i = \alpha_i / p_i$
- $p = \sum_{i=1}^{n} p_i$
- k un coefficient.

On définit ensuite π_i la priorité du travail i de la façon suivante :

$$\pi_i = \begin{cases} \omega_i & si\ m_i \leq 0 \\ \omega_i - m_i \frac{\omega_i + h_i}{kp} & si\ 0 < m_i \leq kp \\ -h_i & \text{sinon.} \end{cases}$$

Cette fonction peut se représenter graphiquement de la façon indiquée figure 4.3.

La procédure fonctionne itérativement. A chaque itération la priorité des travaux est mise à jour et le travail le plus prioritaire est séquencé à la suite sans temps mort. Cette méthode ne tient pas compte d'éventuelles dates de fin impératives. Afin de rendre cette méthode utilisable nous proposons la modification suivante. Nous introduisons ψ_i défini par : $\psi_i = d_i + p_i - \tilde{d}_i$ et on pose :

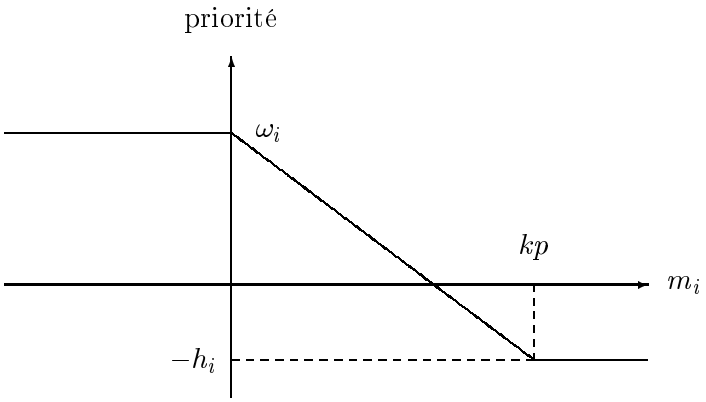

FIG. 4.3 – Fonction de priorité LIN-ET

$$\pi_i = \begin{cases} -h_i m_i / kp & si\ \max(\psi_i, kp) < m_i \\ \omega_i - m_i \frac{\omega_i + h_i}{kp} & si\ \max(\psi_i, 0) < m_i \leq kp \\ \omega_i - \omega_i m_i & \text{si } \psi_i \leq m_i \leq 0 \\ +\infty & \text{sinon.} \end{cases}$$

A chaque itération, le travail i de plus grande priorité est sélectionné. Ensuite, on détermine le travail k pour lequel $d_k - p_k = \min_j(d_j - p_j)$. Si pour tous les travaux restants à ordonnancer on a $d_j < d_k - p_k$ alors on ordonnance le travail i à la date $t = \max_j(t + p_i - 1, d_j - p_j)$. Les priorités sont mises à jour et le processus est itéré jusqu'à ce que tous les travaux soient ordonnancés. On note cette heuristique LIN-ET-mod.

• Ow et Morton proposent une seconde règle de priorité appelée EXP-ET (EXP pour exponentiel) définie de la façon suivante. La priorité d'un travail i est alors définie de la façon suivante :

$$\pi_i = \begin{cases} \omega_i & si\ m_i \leq 0 \\ \omega_i \times exp[\frac{-(\omega_i + h_i)}{h_i} \frac{m_i}{p}] & si\ 0 < m_i \leq \frac{\omega_i}{\omega_i + h_i} kp \\ h_i^{-2} \times (\omega_i - \frac{(\omega_i + h_i) m_i}{kp})^3 & si\ \frac{\omega_i}{\omega_i + h_i} kp < m_i \leq kp \\ -h_i & \text{si } m_i > kp \end{cases}$$

Cette fonction est représentée figure 4.4.

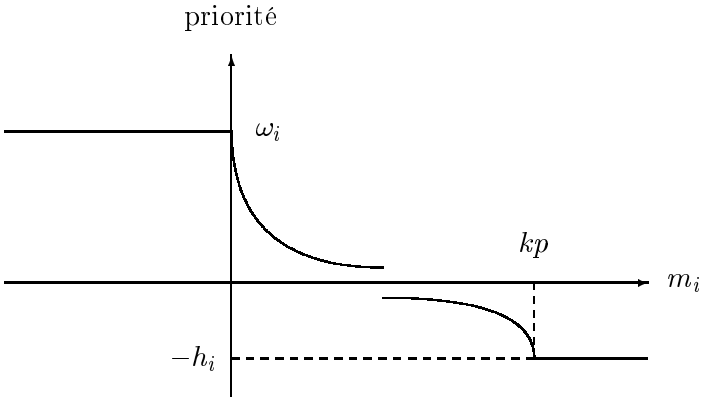

FIG. 4.4 – Fonction de priorité EXP-ET

Pour les mêmes raisons que pour LIN-ET, cet algorithme ne convient pas en l'état pour

résoudre notre problème. On pose à nouveau $\psi_i = d_i + p_i - \tilde{d}_i$ et la priorité :

$$\pi_i = \begin{cases} -h_i m_i / p_i & si\ \max(\psi_i, kp) \leq m_i \\ h_i^{-2} \times (\omega_i - \frac{(\omega_i + h_i) m_i}{kp})^3 & si\ \max(\psi_i, \frac{\omega_i}{\omega_i + h_i} kp) \leq m_i \leq kp \\ \omega_i \times exp[\frac{-(\omega_i + h_i)}{h_i} \frac{m_i}{p}] & si\ \max(0, \psi_i) < m_i \leq \frac{\omega_i}{\omega_i + h_i} kp \\ \omega_i - \omega_i m_i & \text{si } \psi_i \leq m_i \leq 0 \\ +\infty & \text{si } m_i \leq \psi_i \end{cases}$$

On procède alors de la même façon que pour LIN-ET-mod. Cet algorithme est noté EXP-ET-mod par la suite.

Ow et Morton [Ow and Morton, 1989] interdisent les temps morts. En fait, la présence de l'autorisation de l'insertion de temps morts change en profondeur la structure du problème. Cela le rend beaucoup plus difficile à résoudre ne serait-ce que par des algorithmes de liste.

Comme vu au chapitre 2, c'est-à-dire bien vu et pointé du doigt par Davis et Kanet [Davis and Kanet, 1993b], une heuristique qui chercherait à construire la séquence complète – même avec une très bonne règle de priorité éventuellement sans temps morts – puis qui chercherait seulement après à fixer les temps morts, donnerait des résultats très décevants. Cette assertion a été vérifiée expérimentalement par [Estève et al., 2001].

4.2.3.1 Méthode approchée : heuristique par construction

Dans cette section nous fournissons une heuristique dédiée qui construit un ordonnancement en prenant en compte le séquencement aussi bien que l'insertion de temps morts. Cette heuristique appelée HEACT1 construit un ordonnancement de façon itérative.

A chaque itération, HEACT1 sélectionne un travail et décide de sa date de début en tenant compte du fait que les travaux ordonnancés précédemment ne sont pas calés à gauche. La sélection d'un travail se fait en comparant les paires (i, j) des travaux non ordonnancés de façon à décider heuristiquement si le travail i doit précéder le travail j ou le contraire. A la fin de ce test on compare le travail qui doit être en première position avec le prochain travail non ordonnancé. Au final, on retient le travail i qui a été retenu comme étant celui qui doit précéder tous les autres. Il est alors ordonnancé de façon à minimiser sa contribution à la fonction objectif.

Nous utilisons les notations suivantes : σ fait référence à l'ordonnancement partiel des travaux lors de l'itération courante tandis que Ω fait référence à l'ensemble des travaux non encore ordonnancés. On appelle également t la date de début au plus tôt d'un travail quelconque si on l'ordonnançait après σ. Pour décider du prochain travail à ordonnancer après σ on compare les travaux deux à deux. Soit $(i, j) \in \Omega^2$, si nous choisissons d'ordonnancer i avant j alors le travail i sera candidat pour être ordonnancé à la suite de σ. C'est lui qui sera alors comparé à un autre travail de Ω. La décision de sélectionner i ou j est donc fondée sur une estimation de l'intérêt d'ordonnancer le travail i avant le travail j ou le travail j avant le travail i. Celui des deux cas précédents qui conduit à la moins forte contribution à la fonction objectif est le cas sélectionné.

Ainsi, considérons l'exemple où nous testons le cas i avant j et où seulement deux sous cas se manifestent : le travail i peut-être testé aussi près que possible de sa date due ou encore c'est le travail j qui peut être placé aussi près que possible de sa date due. On note $C_{i,1}^{i,j}$ (respectivement $C_{j,1}^{i,j}$) la date de fin du travail i (respectivement j) lorsque i précède j et que c'est le travail i qui se termine le plus près possible de sa date due. De façon similaire

on note $C_{i,2}^{i,j}$ (respectivement $C_{j,2}^{i,j}$) la date de fin du travail i (respectivement j) lorsque i précède j et que c'est le travail i (respectivement j) qui se termine le plus près possible de sa date due. Ces notations sont illustrées figure 4.5 dans le cas où i précède j. Notons que dans la cas général une des deux dates ne coïncide pas forcément avec une date due, cela dépend de t. Notons également que tous les cas ne sont pas faisables si les dates de fin impératives ne sont pas respectées.

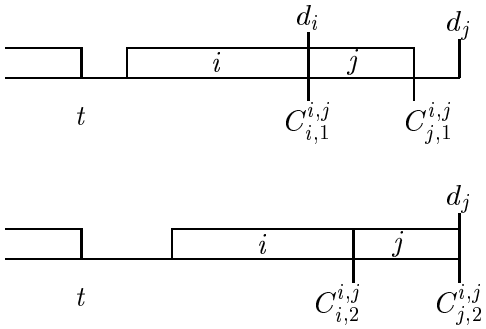

FIG. 4.5 – Différents cas possibles pour deux travaux i et j

Les équations pour déterminer ces dates sont les suivantes.

$$C_{i,1}^{i,j} = \max\left(t, r_i, d_i - p_i\right) + p_i \tag{4.10}$$

$$C_{j,1}^{i,j} = \max\left(C_{i,1}^{i,j}, r_j, d_j - p_j\right) + p_j \tag{4.11}$$

$$C_{j,2}^{i,j} = \max\left(\max\left(t, r_i\right) + p_i, r_j, d_j - p_j\right) + p_j \tag{4.12}$$

$$C_{i,2}^{i,j} = \max\left(t, r_i, C_{j,2}^{i,j} - p_j - p_i\right) + p_i \tag{4.13}$$

On note dans l'algorithme $F_1^{i,j}$ et $F_2^{i,j}$ les contributions à la fonction objectif dans chacun des deux cas. $F^{i,j}$ représente la plus petite des deux, on calcule également $F^{j,i}$ et la meilleure des deux indique le travail entre i et j qui sera sélectionné.

L'algorithme HEACT1 est représenté table 4.1. La notation $\tilde{s}_{i,1}^{C}$ indique la marge du travail i par rapport à sa date de fin impérative dans le cas indiqué par C.

C'est à ce moment là que va intervenir la descente locale, recherche à base de voisinages, une fois que l'heuristique précédente HEACT1 a trouvé un ordonnancement complet, qu'il soit faisable ou non faisable d'ailleurs. L'ordonnancement trouvé – qui est en quelque sorte un ordonnancement *germe* – est alors amélioré au moyen d'un mécanisme d'insertion des travaux. Prenons note du fait que si l'ordonnancement germe est infaisable, la procédure de perfectionnement est malgré tout appliquée et par là on essaie en quelque sorte de réparer les infaisabilités tout en améliorant les performances de la fonction objectif.

Quoi qu'il en soit et quel que soit le statut de l'ordonnancement germe, cette heuristique de voisinage appelée HEACT2 procède de la façon suivante : appelons s l'ordonnancement courant. A l'itération i notons $V_i(s)$ le voisinage de s c'est-à-dire l'ensemble des ordonnancements obtenus en insérant le $i$ème travail en la position j, j prenant toutes les valeurs de 1 à n sauf i bien sûr. La figure 4.6 illustre le voisinage.

Notons qu'après chaque insertion, il est fait recours à la procédure de calcul optimal des dates de début à séquence fixée décrite précedemment. Si l'ordonnancement s est infaisable, alors on va pour le remplacer prendre dans le voisinage $V_i(s)$ l'ordonnancement

DEBUT
$\Omega[i]$ est le $i$ème élément de Ω
$t = 0$; $\sigma = \emptyset$
Tant que ($|\Omega| \neq 0$) Faire
  Soit k tel que $d_k - p_k = min_{i \in \Omega}(d_i - p_i)$
  Si ($\forall i \in \Omega, i \neq k, (d_i - p_i - d_k) > 0$) Alors $t = d_k - p_k$
  $i = \Omega[1]$
  Pour $\ell = 2$ à $|\Omega|$ Faire
    $j = \Omega[\ell]$; $C_{i,1}^{ij} = \max(t, r_i, d_i - p_i) + p_i$; $C_{j,1}^{ij} = \max(C_{i,1}^{ij}, r_j, d_j - p_j) + p_j$
    $F_1^{ij} = \alpha_i \max(0, d_i - C_{i,1}^{ij}) + \beta_i \max(0, C_{i,1}^{ij} - d_i) + \alpha_j \max(0, d_j - C_{j,1}^{ij}) + \beta_j \max(0, C_{i,1}^{ij} - d_j)$
    $C_{i,2}^{ij} = \max(t, r_i, C_{j,2}^{ij} - p_j - p_i) + p_i$; $C_{j,2}^{ij} = \max(\max(t, r_i) + p_i, r_j, d_j - p_j) + p_j$
    $F_2^{ij} = \alpha_i \max(0, d_i - C_{i,2}^{ij}) + \beta_i \max(0, C_{i,2}^{ij} - d_i) + \alpha_j \max(0, d_j - C_{j,2}^{ij}) + \beta_j \max(0, C_{j,2}^{ij} - d_j)$
    $\tilde{s}_{j,1}^{C_{i,1}^{ij}} = \tilde{d}_j - p_j - \max(C_{i,1}^{ij}, r_j)$; $\tilde{s}_{j,2}^{C_{i,2}^{ij}} = \tilde{d}_j - p_j - \max(C_{i,2}^{ij}, r_j)$; $F^{ij} = \infty$
    Si ($\tilde{s}_{j,1}^{C_{i,1}^{ij}} \geq 0$) Alors
      Si ($\tilde{s}_{j,2}^{C_{i,2}^{ij}} \geq 0$) Alors
        Si ($F_1^{ij} \leq F_2^{ij}$) Alors $F^{ij} = F_1^{ij}$
          Sinon $F^{ij} = F_2^{ij}$
        Finsi ;
      Sinon $F^{ij} = F_1^{ij}$
      Finsi
    Sinon Si ($\tilde{s}_{j,2}^{C_{i,2}^{ij}} \geq 0$) Alors $F^{ij} = F_2^{ij}$
    Finsi
    $C_{j,1}^{ji} = \max(t, r_j, d_j - p_j) + p_j$; $C_{i,1}^{ji} = \max(C_{j,1}^{ji}, r_i, d_i - p_i) + p_i$
    $F_1^{ji} = \alpha_j \max(0, d_j - C_{j,1}^{ji}) + \beta_j \max(0, C_{j,1}^{ji} - d_j) + \alpha_i \max(0, d_i - C_{i,1}^{ji}) + \beta_i \max(0, C_{i,1}^{ji} - d_i)$
    $C_{j,2}^{ji} = \max(t, r_j, C_{i,2}^{ji} - p_i - p_j) + p_j$; $C_{i,2}^{ji} = \max(\max(t, r_j) + p_j, r_i, d_i - p_i) + p_i$
    $F_2^{ji} = \alpha_j \max(0, d_j - C_{j,2}^{ji}) + \beta_j \max(0, C_{j,2}^{ji} - d_j) + \alpha_i \max(0, d_i - C_{i,2}^{ji}) + \beta_i \max(0, C_{i,2}^{ji} - d_i)$
    $\tilde{s}_{i,1}^{C_{j,1}^{ji}} = \tilde{d}_i - p_i - \max(C_{j,1}^{ji}, r_i)$ $\tilde{s}_{i,2}^{C_{j,2}^{ji}} = \tilde{d}_i - p_i - \max(C_{j,2}^{ji}, r_i)$; $F^{ji} = \infty$
    Si ($\tilde{s}_{i,1}^{C_{j,1}^{ji}} \geq 0$) Alors
      Si ($\tilde{s}_{i,2}^{C_{j,2}^{ji}} \geq 0$) Alors
        Si ($F_1^{ji} \leq F_2^{ji}$) Alors $F^{ji} = F_1^{ji}$
          Sinon $F^{ji} = F_2^{ji}$
        Finsi
      Sinon $F^{ji} = F_1^{ji}$
      Finsi
    Sinon Si ($\tilde{s}_{i,2}^{C_{j,2}^{ji}} \geq 0$) Alors $F^{ji} = F_2^{ji}$
    Finsi
    Si ($F^{ji} < F^{ij}$) Alors $i = j$
  Fin Pour
  $\sigma = \sigma \cup \{i\}$; $t = t + p_i$; $\Omega = \Omega - \{i\}$
Fin Tant que
Soit s l'ordonnancement obtenu à partir de la séquence σ par insertion optimale des temps morts
Retourner s
FIN

TAB. 4.1 – Algorithme HEACT1

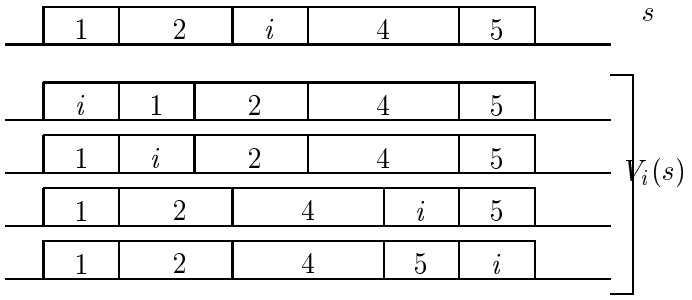

FIG. 4.6 – Définition du voisinage $V_i(s)$ de l'ordonnancement s à l'itération i

faisable qui possède la plus faible valeur de la fonction objectif. Mais si par malheur un tel ordonnancement n'existe pas, il est préférable que personne ne change quoi que ce soit à l'ordonnancement s et de passer directement à l'itération suivante. Dans le cas maintenant où s est faisable on va l'améliorer si c'est possible en prenant dans son voisinage $V_i(s)$ l'ordonnancement faisable qui va bien. En fait à chaque fois que l'ordonnancement s est amélioré, en d'autres termes à chaque fois qu'il est écrasé par un nouvel ordonnancement meilleur, on redémarre la descente locale depuis le début, soit depuis la position 1.

On voit bien qu'on va être obligé de réduire cette exploration des voisinages $V_i(s)$ et nous utilisons pour ce faire une condition de *pseudo-dominance* pour écarter rapidement certains voisins inopportuns. Cette condition est la suivante. On note i et j deux travaux, t_i et t_j leurs dates de début. On dit que i précède immédiatement j si

$$\beta_i(p_j + \max(0, r_j - t_i)) - \Omega_{ij}(\alpha_i + \beta_i) - \beta_j \min(p_i, t_j - r_j) + \Omega_{ji}(\alpha_j + \beta_j) \geq 0$$

où

$$\Omega_{ij} = \begin{cases} 0 & \text{si } \Delta_i < 0 \\ \Delta_i & \text{si } 0 \leq \Delta_i < p_j + \max(0, r_j - t_i) \\ p_j + \max(0, r_j - t_i) & \text{sinon.} \end{cases}$$

$$\Omega_{ji} = \begin{cases} 0 & \text{si } \Delta_j < 0 \\ \Delta_j & \text{si } 0 \leq \Delta_j < \min(p_i, r_j - t_j) \\ \min(p_i, r_j - t_j) & \text{sinon.} \end{cases}$$

et $\Delta_i = d_i - p_i - r_i$ et $\Delta_j = d_j - p_j - \max(r_j, t_i)$.

En fait cette condition est celle introduite par [Mazzini and Armentano, 2001] pour le problème $1|r_j, d_j|\sum \alpha_j E_j + \beta_j T_j$. Cette condition de dominance devient pour notre problème une pseudo-condition de dominance. L'adaptation à notre problème n'a pas pu être démontrée comme une réelle condition de dominance, ce qui fait qu'on pourrait éliminer de mauvais voisins qui n'en sont pas. En fait, en faisant des expérimentations, on s'aperçoit rapidement qu'on ne détériore pas grand chose finalement en ce qui concerne la valeur de la fonction objectif qu'on aurait pu atteindre éventuellement avec le voisinage obtenu sans utilisation de cette condition. Ce que l'on gagne en rapidité, on ne le perd pas beaucoup au niveau de la qualité des résultats. Quoiqu'il en soit et heureusement, il est plus rapide de tester cette condition que d'explorer de nombreux voisins, ce qui fait gagner de la rapidité à l'heuristique HEACT2 sans trop dégrader sa qualité.

Il faut encore ajouter que lors de l'établissement du voisinage $V_i(s)$, en supposant le travail i inséré en position k entre les travaux j et l dans cet ordre par exemple, on fait

appel à la pseudo-condition de dominance de la façon suivante : si la condition ne tient pas pour l'ordre i, j ni pour l'ordre j, k alors on n'a même pas à insérer i en position k ni à faire appel à la procédure d'insertion de temps morts. Le voisinage réduit après cette prise en compte est noté $V_i'(s)$. Le détail de l'algorithme est donné dans la table 4.2.

DEBUT
Soit s la solution retournée par HEACT1
Pour $i = 1$ à n Faire
Calculer $V_i'(s)$
Soit $v' \in V_i'(s)$ l'ordonnancement réalisable avec la plus petite valeur de $\sum \alpha_j E_j + \beta_j T_j$
si (s' existe et s n'est pas réalisable) alors $s = s'; i = 0$
si (s' existe et $\sum \alpha_j E_j(s') + \beta_j T_j(s') < \sum \alpha_j E_j(s) + \beta_j T_j(s)$) alors $s = s'; i = 0$
Fin pour
Retourner s
FIN

TAB. 4.2 – Algorithme HEACT2

4.2.3.2 La Recovering Beam Search

Dans ce paragraphe, nous décrivons une nouvelle heuristique que l'on comparera à l'heuristique par construction décrite dans la section précédente ([Estève et al., 2003a] et [Estève et al., 2005]). Il s'agit d'une recherche par faisceaux, améliorée par une phase de récupération des noeuds. Tout d'abord nous exposons les principes généraux de cette heuristique puis nous décrivons l'évaluation des noeuds et la récupération des noeuds pour notre problème.

Les algorithmes de type recherche par faisceau sont en fait des procédures par séparation et évaluation (PSE) qui sont tronquées et qui donc explorent à chaque étage de l'arbre de recherche des ensembles de solutions (puisque un noeud non terminal est en fait un ensemble de solutions). Ces ensembles de solutions se veulent prometteurs ou considérés comme tels. Les premiers modèles de recherche par faisceau à avoir été appliqués à des problèmes de juste-à-temps l'ont été avec efficacité par [Ow and Morton, 1988] pour le problème à temps morts interdits. En revanche, notre problème autorise les temps morts, notre contribution est donc d'adapter cette méthode innovante à ce problème nouveau. Signalons tout de suite également que le canevas que nous utilisons est celui de [Tadei, 2002] qui s'appelle *recherche par faisceaux améliorée par récupération des noeuds*. Notre apport est donc dans une implémentation originale de ce schéma pour un problème où elle ne s'était jamais montrée sous ce jour auparavant. Tout l'intérêt consiste à tenir compte le plus possible des éléments qui sont spécifiques au problème. L'avantage majeur de ce type d'heuristique est de considérer un nombre limité d'ensembles de solutions en parallèle. A chaque itération, les noeuds fils sont engendrés et seulement ceux qui contiennent les solutions les plus prometteuses sont retenus. Quand un noeud fils est retenu, la phase de récupération consiste à ce moment là, à appliquer une petite recherche locale pour essayer justement de remettre en question toutes les décisions possiblement érronées qui ont conduit à considérer ce noeud. Formellement, donc, on définit un noeud ν pour le problème noté $1|r_j, d_j, \tilde{d}_j| \sum \alpha_i E_i + \beta_i T_i$ comme étant une liste σ de travaux déjà séquencés et un ensemble Ω de travaux restant à ordonnancer.

Un ordonnancement partiel est alors calculé à partir de cette séquence de travaux déjà séquencés en appliquant l'algorithme de calcul optimal des dates de début déjà présenté auparavant. De cette façon, c'est un peu en fait le schéma de séparation, un noeud représente un ensemble d'ordonnancements ayant la particularité d'avoir la même séquence au début. Et le noeud fils en fait est tout simplement construit à partir du noeud courant c'est-à-dire son noeud père, en rajoutant à la fin de la liste des travaux déjà placés, un travail parmi ceux restant à placer. Ensuite tous ces travaux sont d'abord placés à leur date de début au plus tôt possible en vue de vérifier la faisabilité et seuls les fils qui correspondent à un ordonnancement partiel faisable vont être gardés. Et parmi tous les fils retenus à un niveau de profondeur donnée, le rôle de l'algorithme est de ne tenir compte que d'un nombre limité d'entre eux. Ce nombre est appelé la *largeur du faisceau*, est noté w. Ceci veut dire en fait que les noeuds d'un niveau donné vont être triés et selectionnés et à l'aide d'une fonction d'évaluation. Le but est de garder les meilleurs noeuds évalués en nombre limité w, et c'est ensuite qu'on appliquera la phase dite de récupération. Les enfants de ces w noeuds les plus prometteurs du niveau sont engendrés et le processus est itéré jusqu'aux noeuds feuilles, mais on ne remonte pas car ce n'est pas une PSE. L'algorithme détaillé appelé $HEACT3$ est représenté table 4.3.

Nous allons maintenant présenter tous les petits composants de cette heuristique lorsque qu'on cherche à l'implémenter. C'est la force d'une RBS que de tenir compte des spécificités du problème, par rapport aux autres algorithmes de liste. Traitons donc de l'évaluation des noeuds et après de la phase de récupération.

Chaque noeud de l'arbre de recherche ν est évalué grâce à une fonction $f(\nu)$ définie comme $f(\nu) = \gamma LB(\nu) + (1-\gamma)UB(\nu)$ où γ est un poids donné et où $LB(\nu)$ et $UB(\nu)$ font référence respectivement à une borne inférieure et à une borne supérieure au noeud ν. Dans la procédure classique par séparation et évaluation, l'évaluation du noeud ν est généralement calculée en prenant $\gamma = 1$, c'est-à-dire 100% et $1 - \gamma = 0$. Cette évaluation est alors utilisée pour détruire tous les noeuds indésirables, et c'est ce qu'elle fait. Mais dans une recherche par faisceaux améliorée par une récupération des noeuds, cette fonction d'évaluation a seulement pour but – et c'est déja bien – de guider l'heuristique. En quelque sorte, que les bornes soient bonnes ou non n'est pas fondamental, ce qu'il faut c'est une mesure qui permet de dire si le noeud est "prometteur" ou non. L'idée générale est donc de faire une combinaison linéaire entre une évaluation par défaut et une évaluation par excès du noeud.

Borne supérieure. Pour nous, la borne supérieure $UB(\nu)$ va être calculée au noeud ν en appliquant l'heuristique HEACT1, mais pas n'importe comment, uniquement à l'ensemble des travaux restant à ordonnancer. Le résultat de cette heuristique est une séquence partielle de ces dits travaux qui va alors être concaténée à la séquence partielle des travaux déjà séquencés au noeud ν. La séquence complète produit alors un ordonnancement, lorsque la procédure d'insertion optimale de temps morts lui est appliquée.

Borne inférieure. Quant à la borne inférieure, c'est celle qui est proposée par [Sourd, 2004] pour le problème sans dates de disponibilité et sans dates impératives. Pour aller vite, il calcule sa borne en utilisant une certaine forme de préemption et en introduisant pour chaque travail une fonction de coût qu'il veut continue et affine par morceaux. En fait, si on regarde basiquement, chaque fonction de coût f_i est attachée à un travail i et fabriquée à partir de

DEBUT
/* Entrée : γ, poids d'évaluation de la fonction */
Step 1 : /* Initialisation */
Créer le noeud racine ν_0 : $\sigma_0 = ()$, $\Omega_0 = \{1, ..., n\}$;
$Q = \{\nu_0\}$
$F_{best} = \infty$
Step 2 : /* Exploration de l'espace de recherche */
$\ell = |Q|$
Pour $k = 1$ à ℓ Faire
  Soit $\nu = Q[1]$; $Q = Q/\{\nu\}$
  Q_k =\{noeuds fils réalisables ν_j of ν\}
  $\forall \nu_j \in Q_k$ calculer $UB(\nu_j)$, $LB(\nu_j)$ et $f(\nu_j) = \gamma LB(\nu_j) + (1-\gamma)UB(\nu_j)$
Fin pour
$Q' = \bigcup_{i=1}^{\ell} Q_i$
Trier les noeuds de Q' par ordre croissant de la valeur de la fonction f
$k = 0$
Tant que ($k < |Q'|$ et $|Q| < w$) Faire
  /* Phase de récupération */
  Soit $\nu = Q'[k]$
  Appliquer HEACT2 avec σ en entrée et poser σ' l'ordonnancement partiel résultat
  Si (σ' est réalisable et $\sum \alpha_j E_j(\sigma') + \beta_j T_j(\sigma') \leq \sum \alpha_j E_j(\sigma) + \beta_j T_j(\sigma)$
    et $C_{max}(\sigma') < C_{max}(\sigma)$) Alors
    $\sigma = \sigma'$
  Fin si
  Si ($\Omega = \emptyset$ et $\sum \alpha_j E_j(\sigma) + \beta_j T_j(\sigma) < F_{best}$) Alors
    $F_{best} = \sum \alpha_j E_j(\sigma) + \beta_j T_j(\sigma)$
  Fin si
  $Q = Q + \{\nu\}$
  $k = k + 1$
Fin Tant que
Si ($|Q| \neq 0$) Alors Aller Step 2
Step 3 : retourner F_{best}
FIN

TAB. 4.3 – Algorithme HEACT3

quatre segments. L'existence chez nous de dates de disponibilités et de dates impératives se traduit tout simplement en l'introduction de deux segments supplémentaires de valeur évidemment infinie à l'extérieur des montants. La minimisation de la somme de ces fonctions de coût sous les hypothèses de préemption, fournit une sorte de borne inférieure pour le problème original. Le calcul effectif est compliqué car il faut faire appel à un algorithme de sous gradient en recourant à la formulation duale du problème préemptif.

Récupération. La phase de récupération des noeuds a pour but de revenir sur les décisions mauvaises qui auraient pu être prises pour conduire à ce noeud. Cette phase est donc appliquée aux w noeuds fils engendrés par ν et cela consiste à mettre en oeuvre une recherche locale sur l'ordonnancement partiel déja fabriqué au noeud ν. Pour un noeud donné, cette recherche locale suit le principe de HEACT2, mais appliquée uniquement sur la partie déjà choisie à cette profondeur de l'arbre et en plus avec le voisinage réduit déjà évoqué, sinon il serait plus intéressant d'explorer des noeuds que de faire une descente locale en un noeud.

En d'autres termes, si l'ordonnancement partiel obtenu σ' est meilleur et qu'en plus il a une plus petite date de fin (un plus petit *makespan*), alors il est complètement naturel de remplacer σ par σ'. En fait, le résultat de tout ça c'est comme si on avait un saut d'un noeud à un autre, au même niveau de l'arbre de recherche. Cette condition d'acceptation est décrite dans l'algorithm représenté table 4.3. Ce n'est pas exactement une condition de dominance, mais pourtant cela améliore pas mal le comportement de notre heuristique.

On va noter quand même que la valeur que nous considérerons pour la largeur du faisceau est la valeur $w = 1$. D'autres valeurs de w ont été testées, mais elles ne conduisent pas à des résultats significativement meilleurs.

4.2.4 Expérimentations numériques et résultats

Les expériences qui ont été menées ont pour but de comparer les heuristiques HEACT2 et HEACT3 par rapport à la solution optimale pour les problèmes de petite taille d'une part, et d'autre part à la meilleure des deux pour les problèmes de taille moyenne. Les problèmes de petite taille sont ceux pour lesquels $n \in \{4, 6, 8, 10, 20, 30, 40\}$ et les problèmes de taille moyenne sont ceux pour lesquels $n \in \{50, 60, 70, 80, 90, 100, 110, 120, 130\}$.

Les données pour le problème considéré qui vont être générées sont les suivantes : p_j, α_j, β_j, d_j et b_j. En tout état de cause, pour éviter la génération de dates de disponibilité négatives, on utilise un schéma particulier de génération de données. Les durées d'exécution p_j sont tirées au hasard en utilisant une distribution uniforme entre 1 et 100 tandis que pour les poids α_j et β_j on utilise une loi uniforme entre 1 et 10. Pour engendrer les dates dues d_j, on introduit P qui est la somme des durées d'exécution générées, en d'autres termes $P = \sum_{j=1}^{n} p_j$.

Pour chaque taille de problème on considère trois classes de problèmes. Pour la première, les dates dues d_j sont tirées au hasard entre p_j et $2P$. En ce qui concerne la seconde classe, l'intervalle est $[p_j, \frac{7}{2}P]$ tandis que pour la troisième classe, il s'agit de $[p_j, 5P]$. Une fois que les p_j, α_j, β_j, d_j ont été générés, les valeurs des bornes b_j sont tirées au hasard en utilisant une distribution uniforme entre 0 et $(d_j - p_j)$. Tout ça pour garantir d'éviter le cas des dates de disponibilité négatives. Pour chaque classe de dates dues, 30 instances sont générées et cela conduit à un total de 90 instances génerées par taille de problème. Tous les algorithmes ont été développés en langage C. Les tests ont été effectués sur un PC pentium IV 2.8Ghz

avec 256Mb de RAM, sous l'environnement MS Windows.

Notons que pour l'heuristique HEACT3, il reste à déterminer deux paramètres. Le premier est le poids γ dans la fonction utilisée pour évaluer chaque noeud. Des expériences [Estève et al., 2004] montrent que la meilleure efficacité est atteinte en prenant une valeur de $\gamma = 0.7$, c'est-à-dire une contribution à 70% de la borne de [Sourd, 2004] quand elle n'est pas trop longue à calculer. Cela veut donc dire que la borne inférieure contribue un peu plus que la borne supérieure au choix des w noeuds les plus prometteurs d'un niveau donné. Quant au second paramètre, c'est la largeur de faisceau elle même w. Clairement, plus grande est cette valeur et plus efficace est l'heuristique en dépit du fait que cela va requérir beaucoup plus de temps d'exécution sur la machine. Dans une section suivante, on évaluera expérimentalement l'impact de ce paramètre sur le comportement de HEACT3.

Dans le reste de cette section, les notations F_{HEACT2} et F_{HEACT3} font références pour une instance donnée à la valeur de la fonction objectif pour les heuristiques HEACT2 et HEACT3 respectivement. Si une des deux heuristiques échoue pour calculer une solution faisable, on place cette valeur de fonction objectif à l'infini. Dans toutes les expériences, on peut évaluer en $O(n \log(n))$ pour une instance donnée, s'il existe un ordonnancement faisable juste-à-temps où chaque travail est exactement terminé à l'heure. Cette évaluation est réalisée en ordonnançant les travaux dans l'ordre ascendant de leurs dates dues jusqu'à ce que tous les travaux aient été ordonnancés ou que l'un d'eux ne puisse pas être terminé à sa date due. Dans les tables introduites plus bas, la colonne *ot* ("*on time*") fait justement référence à ce pourcentage d'instances pour lequel un ordonnancement à l'heure existe. On appellera également t_{avg} et t_{max} respectivement la durée moyenne et la durée maximale d'exécution sur la machine, en secondes (s). Nous nous intéressons également au pourcentage d'instances pour lesquelles une heuristique donnée n'a pas su retourner une solution faisable. Ce résultat est présenté dans la colonne inf (infaisable). Pour l'heuristique HEACT2 nous nous sommes également interessés au pourcentage moyen d'amélioration de la descente locale par rapport à l'ordonnancement initial donné par l'heuristique HEACT1, lorsque celle-ci arrive à trouver un ordonnancement faisable.

Il y a d'autres résultats encore que nous prenons en considération et qui sont dépendants des expérimentations menées à bien et introduites ci-après.

4.2.5 Résultats sur les problèmes de petite taille

Interessons nous par exemple à la comparaison de nos heuristiques avec les solutions optimales obtenues en utilisant le solveur CPLEX appliqué à notre modèle de programmation linéaire en nombre entiers. Nous testons d'abord trois différentes versions de l'heuristique HEACT3, chacune correspondant à une valeur différente de la largeur du faisceau. Notons F_{opt} la valeur optimale de la fonction objectif calculée par CPLEX pour une instance donnée faisable. Pour chaque version de HEACT3 les colonnes δ_{avg} et δ_{max} font référence aux déviations moyennes et maximales en pourcentage par rapport à la solution optimale respectivement pour les instances où l'on peut calculer une solution faisable non triviale. Pour chaque instance, la déviation par rapport à l'optimal est bien évidemment donnée par

$$\delta = \frac{F - F_{opt}}{F_{opt}}$$

où F fait bien sûr référence à la valeur de la solution retournée par la version considérée de HEACT3.

			$w = 1$				
n	$infp(\%)$	$ot(\%)$	t_{avg} (s)	t_{max} (s)	δ_{avg} (%)	$\delta_{max}(\%)$	$inf(\%)$
4	22.22	51.43	0.00	0	10.92	371.43	22.22
6	21.11	27.14	0.03	1	0.00	0.00	21.11
8	27.78	15.87	0.01	1	0.46	14.05	27.78
10	31.11	10.53	0.22	1	2.79	145.60	31.11
20	34.44	1.69	5.56	11	8.69	118.46	35.56
30	22.22	0.00	40.04	116	12.58	137.00	22.22
40	22.22	0.00	155.51	297	8.05	96.08	22.22

			$w = 2$		
n	t_{avg} (s)	t_{max} (s)	δ_{avg} (%)	$\delta_{max}(\%)$	$inf(\%)$
4	0.01	1	10.92	371.43	22.22
6	0.04	1	0.00	0.00	21.11
8	0.31	1	0.20	11.11	27.78
10	0.52	1	2.82	145.60	31.11
20	10.63	21	9.21	118.46	34.44
30	85.59	176	11.72	137.00	22.22
40	335.39	567	8.01	96.08	22.22

			$w = 3$		
n	t_{avg} (s)	t_{max} (s)	δ_{avg} (%)	$\delta_{max}(\%)$	$inf(\%)$
4	0.01	1	10.92	371.43	22.22
6	0.06	1	0.00	0.00	21.11
8	0.20	1	0.26	14.05	27.78
10	0.54	2	2.77	145.60	31.11
20	14.44	31	9.33	118.46	35.56
30	128.39	245	12.24	137.00	22.22
40	506.01	825	7.81	96.08	22.22

TAB. 4.4 – Influence de w sur la RBS pour $\gamma = 0.7$

Pour chaque taille de problème, on va ainsi calculer le pourcentage d'instances infaisables, appelé $infp$. Les résultats sont donnés dans la table 4.4.

Les résultats montrent que CPLEX est capable de résoudre tous les problèmes contenant moins de 40 travaux, même si à peu près 30% des instances générées s'avèrent infaisables. De plus, parmi les solutions faisables, très peu sont des solutions triviales où tous les travaux peuvent être placés à leur heure, sauf pour les problèmes de moins de 10 travaux. Ces solutions triviales sont calculées très facilement en $O(n \log(n))$ par la règle EDD.

On peut conclure comme on devait s'y attendre que l'accroissement de temps de calcul requis est linéairement proportionnel à la valeur de w, la largeur du faisceau. En plus, si on compare les résultats pour $w = 1$ et $w = 3$ on s'aperçoit qu'accroitre la largeur de faisceau améliore à peine la capacité à trouver de meilleures solutions, sauf pour $n = 20$. Mais si on considère les trois versions testées, on a réellement bien l'impression que celle dont le faisceau est un pinceau unitaire représente un excellent compromis entre le temps passé et la qualité des solutions retournées. C'est pour ça qu'on prendra finalement par la suite une largeur $w = 1$.

	HEACT2						
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$	$imp(\%)$	ord
4	0.00	0	0.28	9.52	22.22	0.29	1
6	0.00	0	5.70	296.15	21.11	1.01	2
8	0.00	0	5.03	125.70	27.78	1.96	5
10	0.01	1	4.69	145.60	36.67	2.76	-
20	0.02	1	6.03	105.89	38.89	3.27	-
30	0.04	1	11.83	104.57	28.89	3.12	-
40	0.11	1	5.86	68.81	26.67	3.08	-

	HEACT3 ($\gamma = 0.7$)					
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$	ord
4	0.00	0	10.92	371.43	22.22	3
6	0.03	1	0.00	0.00	21.11	0
8	0.01	1	0.46	14.05	27.78	2
10	0.22	1	2.79	145.60	31.11	-
20	5.56	11	8.69	118.46	35.56	-
30	40.04	116	12.58	137.00	22.22	-
40	155.51	297	8.05	96.08	22.22	-

TAB. 4.5 – Résultats de HEACT2 et HEACT3 sur des problèmes de petite taille

Une fois qu'on a fait ça, on peut comparer HEACT2 et HEACT3, chacun par rapport à la solution optimale de CPLEX. D'où le nouveau tableau représenté table 4.5. En plus de tout ça, on va considérer l'*évaluation ordinale*, c'est-à-dire le pire résultat calculé par l'heuristique. Pour une instance donnée (jusqu'à $n = 8$), on énumère les $n!$ séquences et on applique l'algorithme de calcul optimal des dates de début. Alors, on obtient un ensemble de valeurs de la fonction objectif qu'on n'a plus qu'à ranger dans l'ordre croissant. Si l'instance est faisable, alors la solution optimale en valeur a pour rang 0, la seconde meilleure solution a pour rang 1 et ainsi de suite va la vie, vous avez compris. On va alors enregistrer la valeur du rang calculé au niveau des heuristiques et c'est ça qu'on présente dans la colonne ord avec son δ_{max} également. La colonne imp indique l'écart relatif moyen entre le résultat de la descente locale HEACT2 et la solution initiale donnée par HEACT1.

On voit sur la table 4.5 que l'heuristique HEACT2 procède très rapidement à trouver une solution ou à décider de l'infaisabilité. Remarquons, comme indiqué dans la colonne inf que cette heuristique arrive la plupart du temps à trouver une solution faisable lorsqu'il en existe une. En fait en ce qui concerne la faisabilité, les pires résultats obtenus le sont pour les problèmes à 30 travaux, où pour 6 instances HEACT2 n'arrive pas à trouver une solution faisable alors que l'instance l'est. Quant aux instances faisables non triviales, on va s'intéresser aux colonnes δ_{avg}, δ_{max} et ord qui donnent une évaluation de l'efficacité. Ainsi, en moyenne, la déviation par rapport à l'optimal est généralement inférieure à 6 %, sauf pour les problèmes à 30 travaux. En ce qui concerne le pire cas maintenant, la déviation peut aller jusqu'à 296.15 % ce qui peut paraître vraiment très élevé. Cependant ne perdons pas de vue les informations données par la colonne ord. Ainsi pour ce problème à 6 travaux où on a la déviation à 296.15 % on remarque que l'évaluation ordinale est de 2, ce qui veut dire que la seconde meilleure solution après la solution optimale en valeur est déjà à 296.15 % de cette dernière. La solution optimale est donc très isolée et si la méthode ne la trouve pas,

la solution qu'elle trouve est donc nécessairement éloignée. Autre remarque intéressante, on voit sur la colonne *imp* que l'amélioration de la descente locale dans HEACT2 par rapport à son point initial donné par HEACT1 existe mais est assez faible (quasiment tout le temps à moins de 3.5 %).

Maintenant, en ce qui concerne l'heuristique HEACT3, on voit tout d'abord sur la table 4.5 que cette heuristique calcule un petit peu plus de solutions faisables que la précédente, ce qui est plutôt une bonne chose. A côté de ça, on voit sur les colonnes δ_{avg}, δmax et ord que HEACT3 donne elle aussi de bon résultats, à savoir pas trop éloignés de l'optimal. On a bien l'impression que pour les très petites tailles d'instances (jusqu'à 10 travaux), HEACT3 a de meilleurs résultats que HEACT2 tandis que ç'est plutôt la situation contraire pour les problèmes entre 20 et 40 travaux. Cela dit, cette tendance se renverse à nouveau lorque l'on passe aux problèmes de taille moyenne, c'est ce que nous verrons dans la section suivante. Si maintenant on se place du point de vue des temps de calcul, les résultats sont moins en faveur de HEACT3, il faut bien le reconnaître. Et le réglage à $\gamma = 70$ % n'est plus tenable à partir de 50 travaux, c'est ce qu'on a remarqué expérimentalement dans un tableau qui n'est pas présenté ici. Ceci est dû au fait que la borne de [Sourd, 2004] est très gourmande en temps, elle sera donc "déconnectée" dans la section suivante.

4.2.6 Résultats sur les problèmes de taille moyenne

On va aussi s'intéresser à la comparaison des deux heuristiques pour les problèmes de taille moyenne.

Notons que pour ces tailles là, on est obligé de s'affranchir du calcul de la borne de [Sourd, 2004], car elle est trop gourmande en temps de calcul. On l'évalue simplement à la racine.

Pour chaque heuristique on a aussi les colonnes δ_{avg} et δ_{max} qui font références à la déviation moyenne et maximale en pourcentage par rapport à la meilleure des deux heuristiques, pour les instances donnant des ordonnancements faisables non triviaux.

Pour une instance donnée, la déviation est bien évidemment donnée par

$$\delta = \frac{F - F_{best}}{F_{best}}$$

où F réfère à la valeur de la solution retournée par l'heuristique considérée et $F_{best} = \min(F_{HEACT2}, F_{HEACT3})$. Les résultats sont présentés table 4.6.

En conclusion, on doit pouvoir dire que toutes ces expérimentations conduisent à la même conclusion. Du point de vue des temps de calcul, l'heuristique HEACT2 significativement a des performances bien meilleures que l'heuristique HEACT3, même si les temps de calcul de l'heuristique HEACT3 restent malgré tout très raisonnables. D'un autre côté, on s'y attendait un peu, puisque la phase de récupération des noeuds de HEACT3 utilise en quelque sorte HEACT2. Cela dit, si maintenant on se place du point de vue de l'efficacité, c'est-à-dire de la valeur de la fonction objectif, c'est HEACT3 qui a des performances meilleures que HEACT2, que ce soit en moyenne ou en valeur maximale de déviations (sauf peut être ponctuellement pour les problèmes à 50 travaux). Ceci veut dire que lorsque les heuristiques trouvent un ordonnancement faisable, celui de l'heuristique HEACT3 est meilleur que celui de l'heuristique HEACT2. Ces deux heuristiques ont des résultats proches pour détecter les infaisabilités ou pour ne pas trouver de solution faisable avec un léger avantage tout de même pour l'heuristique HEACT2.

	HEACT2				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
50	0.16	1	4.57	48.23	22.22
60	0.42	1	3.79	32.94	33.33
70	0.69	1	6.44	48.98	33.33
80	1.10	2	4.28	31.73	30.00
90	1.72	5	4.30	50.12	27.78
100	2.30	4	5.15	40.11	31.11
110	3.24	6	4.12	27.60	31.11
120	4.72	10	3.99	37.78	14.44
130	6.29	12	4.60	28.53	36.67

	HEACT3 ($\gamma = 0\%$)				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
50	6.52	11	1.60	79.22	25.56
60	11.73	22	1.34	22.75	34.44
70	21.57	43	0.44	15.77	35.56
80	36.36	85	1.28	19.61	36.67
90	67.17	129	0.95	15.23	28.89
100	106.27	205	0.46	22.59	32.22
110	157.26	292	1.17	21.86	33.33
120	282.84	442	0.69	11.87	21.11
130	311.58	668	0.89	22.46	37.78

TAB. 4.6 – Résultats de HEACT2 et HEACT3 sur des problèmes de taille moyenne

Ces conclusions ne pouvaient pas être déduites directement du fait que HEACT3 utilise HEACT2 et HEACT1, car HEACT3 n'est pas une simple amélioration de HEACT2, et car HEACT2 et HEACT3 n'explorent pas le même espace des solutions.

4.3 Modèle multicritère : énumération des optima de Pareto

4.3.1 Approche de résolution

Maintenant qu'on sait calculer un optimum de Pareto strict pour le problème correspondant à un vecteur de bornes b, on peut avoir envie d'énumérer un certain nombre d'optima de Pareto en faisant varier le vecteur b.

Le but du travail de ce paragraphe est donc d'établir une liste de solutions. Rappelons qu'une solution conduit en fait à un vecteur de critères, que chaque critère est ici lié à un travail, et représente son coût. Cet ensemble de solutions dans l'espace des critères constitue le front de Pareto ou encore l'hyper-surface de compensation pour notre problème multicritère. Le processus d'énumération reprend le principe développé par Klein et Hannan [Klein and Hannan, 1982] appliqué à leur problème ϵ-contrainte. Nous adaptons ici la philosophie de cette exploration du front de Pareto pour notre problème d'approche paramétrique. Il s'agit d'énumérer à l'intérieur d'un arbre de recherche exploré en profondeur d'abord, chaque noeud correspondant à un vecteur b qui définit en quelque sorte la portion de l'hyper-espace qui va être explorée, les noeuds fils correspondant à un partionnement de cette portion choisie et ainsi de suite.

Grosso modo les étapes de l'énumération sont les suivantes : pour commencer, il faut que toutes les bornes b_j $(1 \leq j \leq n)$ soient fixées à l'infini ; ainsi on se ramène au problème de minimiser le coût total seulement :

$$\begin{array}{l} \text{Minimiser } \sum_{j=1}^{n} Z_j \\ sc \\ Z_j \leq \infty \end{array}$$

Notons S^0 la solution optimale de ce problème. Le problème suivant est alors résolu :

$$\begin{array}{l} \text{Minimiser } \sum_{j=1}^{n} Z_j \\ sc \\ Z_1 \leq Z_1(S^0) \\ \text{ou } Z_2 \leq Z_2(S^0) - 1 \\ \text{ou } Z_3 \leq Z_3(S^0) - 1 \\ \ldots \\ \text{ou } Z_n \leq Z_n(S^0) - 1 \end{array}$$

Du problème initial on déduit donc n problèmes, et pour chaque problème, on peut encore décomposer en n problèmes, et ainsi de suite. Ce processus est illustré figure 4.7 dans un cas où $n = 2$.

Le noeud racine a donc toutes les dates de disponibilité nulles $r_j = 0$ $(1 \leq j \leq n)$ et les dates impératives à l'infini $\tilde{d}_j = \infty$ $(1 \leq j \leq n)$. L'énumération est effectuée en partant du noeud racine, et on continue la recherche en profondeur tant qu'il reste des noeuds non explorés dans l'arbre de recherche. Pour chaque noeud traité, l'opération effectuée est toujours la même. Elle peut se décomposer en cinq étapes à l'itération k :

1. Calcul des nouvelles contraintes du type : $Z_j \leq Z_j(S^k) - 1$

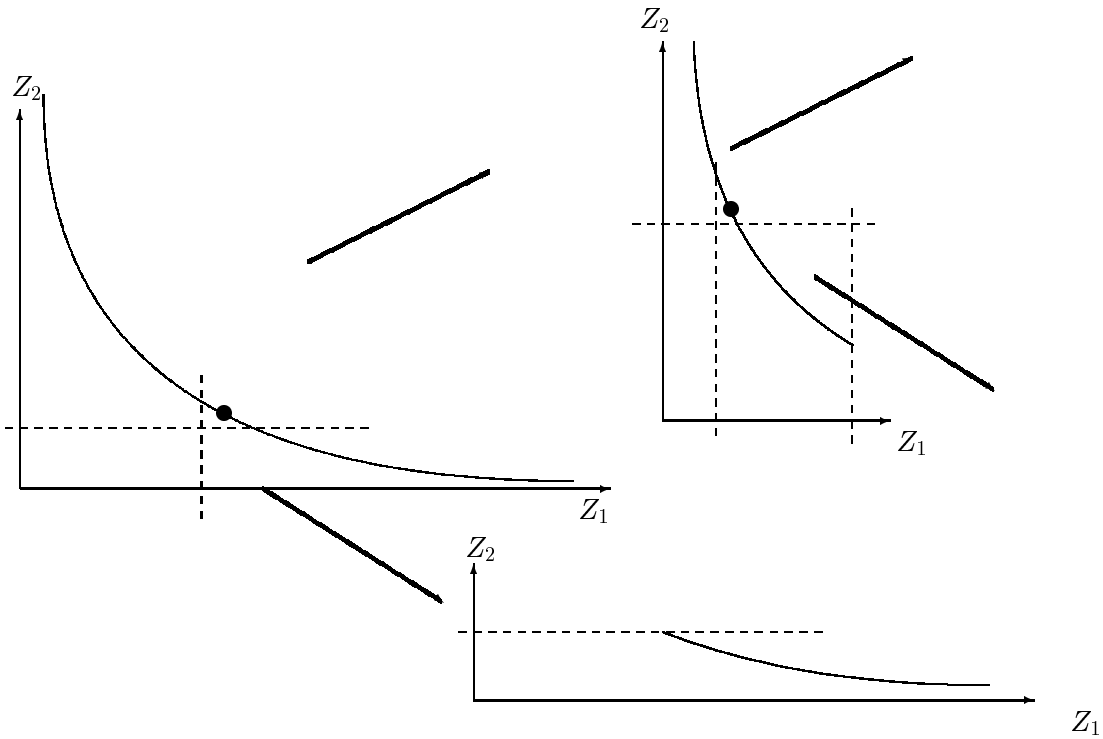

FIG. 4.7 – Illustration du schéma de séparation

2. Résolution heuristique du nouveau problème en utilisant une heuristique vue précédemment.

3. Sauvegarde des vecteurs solutions : cette partie nécessite le tri des solutions et l'élimination des solutions en double (vecteurs de critères identiques) ou des solutions dominées par d'autres.

4. Création des noeuds fils : une condition de coupe a été mise au point afin d'éviter de créer des noeuds fils lorsque leur vecteur b est inférieur ou égal à un vecteur b' d'un noeud non encore traité. En effet, dans ce cas, il ne sert à rien de créer ce noeud car en développant l'autre on obtiendra les mêmes optima (voir figure 4.8).

5. Nouveau traitement.

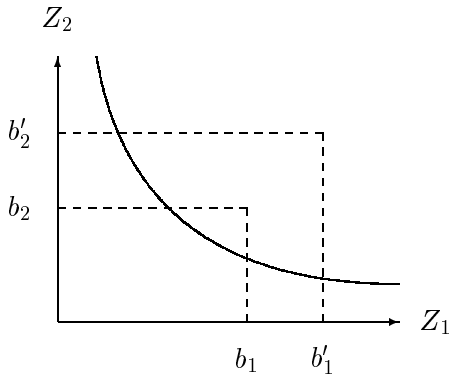

FIG. 4.8 – Dominance entre noeuds

4.3.2 Expérimentations

Il ne nous reste plus maintenant qu'à faire des tests et à évaluer la qualité de notre méthode et des solutions amenées.

N'oublions pas cependant que cette énumération reste une recherche heuristique, dans le sens qu'elle nous donne une approximation seulement du front d'optima de Pareto. L'étude de la section précédente montre que HEACT2 et HEACT3 donnent des résultats satisfaisants comparés à CPLEX. Dans la mesure où HEACT2 n'améliore HEACT1 que de moins de 3% en moyenne (voir tableau 4.5 colonne *imp*), et dans la mesure où HEACT1 est très rapide, c'est cette heuristique qui a été retenue pour résoudre chaque sous-problème. Il est donc raisonnable d'espérer une bonne estimation du front d'optima de Pareto.

Plus le front estimé et le front de Pareto seront proches et plus l'heuristique sera bonne. Donc pour évaluer cette estimation nous avons voulu lancer une campagne de tests en appelant dans un premier temps CPLEX à chaque noeud de l'arbre de recherche, puis sur les mêmes jeux de données, nous avons utilisé l'heuristique HEACT1. Mais des tests préliminaires ont montré que le coût d'utilisation en temps de la version du solveur était trop important même sur des jeux de données de petite taille, vu le nombre important de noeuds générés. Nous n'avons donc pas pu réaliser les mêmes tests que pour l'évaluation des heuristiques c'est-à-dire pour les problèmes de petite taille. Cependant pour se donner une idée de la performance de notre énumération, nous avons tout de même lancé des tests pour $n = 3$. Nous avons ainsi pu comparer les fichiers résultats des deux méthodes comprenant la liste des optima de Pareto. Nous avons alors pu faire quelques statistiques.

On pose $n = 3$, les durées des travaux sont tirées uniformément entre 1 et 100, les coefficients α_j et β_j sont tirés uniformément entre 1 et 10, on pose $P = \sum_{j=1}^{n} p_j$, $r \in \{1, 1.5, 2, 3\}$ et $l \in \{5, 10, 15\}$. Les dates de fin au plus tard d_j sont tirées uniformément entre p_j et $r \times l \times P$. Les résultats sont présentés table 4.7. Notons E l'ensemble des Pareto strics, et F l'ensemble des Pareto retournés par la méthode. On définit :

$$\delta_1 = 100\frac{|E \cap F|}{|F|}$$

et

$$\delta_2 = 100\frac{|E \cap F|}{|E|}$$

Le coefficient δ_1 donne le pourcentage de solutions trouvées par la méthode qui sont dans E et δ_2 donne le pourcentage de solutions de E qui sont trouvées par la méthode.

On peut ainsi voir que pour $n = 3$ on a au minimum 80% d'optima de pareto trouvés par l'heuristique qui sont de véritables optima de Pareto, et au minimum 90% des optima de Pareto sont trouvés par l'heuristique. Des tests plus intensifs devraient être conduits pour renforcer ces conclusions.

Nous avons ensuite mené une campagne de tests afin de vérifier les performances de notre énumération. Le temps de calcul de la méthode – hors temps consacré au tri des solutions – a été limité à t =30 secondes, 120 secondes puis 300 secondes. Le temps de calcul total est donc égal au temps de calcul du tri plus 30 secondes. Les données sont générées de la même façon que précédemment décrit au paragraphe 4.2.4. Le tableau 4.8 présente les résultats des tests pour $t = 30$ secondes.

Dans ce tableau, on constate que le nombre de noeuds explorés diminue quand n augmente. En fait, la procédure HEACT1 prend plus de temps quand n augmente pour traiter un noeud. Dans le temps imparti, elle ne permet donc que de traiter un plus petit nombre de noeuds que pour une petite valeur de n.

		δ_1		δ_2	
l	r	Max	Moy	Max	Moy
5	1	100	90	100	100
5	1,5	100	80	100	90
5	2	100	80	100	100
5	3	100	90	100	90
10	1	100	100	100	100
10	1,5	100	100	100	100
10	2	100	90	100	100
10	3	100	90	100	100
15	1	100	100	100	90
15	1,5	100	100	100	100
15	2	100	100	100	100
15	3	100	100	100	100

TAB. 4.7 – Tests du front de Pareto pour $n = 3$

$t = 30$ secondes

n	noeuds$_{moy}$	noeuds$_{max}$	Pareto$_{moy}$	Pareto$_{max}$	t_{moy}
10	2422.0	8145	1465.6	7760	23.0
20	1977.1	7481	985.5	6793	21.5
30	1266.9	6716	457.8	6017	25.3
40	846.9	5292	128.4	1109	26.6
50	843.1	4028	182.8	3350	29.5
100	571.0	1266	44.2	156	30.0

$t = 120$ secondes

n	noeuds$_{moy}$	noeuds$_{max}$	Pareto$_{moy}$	Pareto$_{max}$	t_{moy}
10	6974.3	27828	4424.9	23134	89.0
20	6269.6	29495	3276.5	21021	82.5
30	3586.9	25192	1470.5	19539	97.9
40	1953.9	17721	334.8	4096	102.7
50	1934.7	14513	516.5	12174	114.7
100	1163.3	1834	93.4	243	119.6

$t = 300$ secondes

n	noeuds$_{moy}$	noeuds$_{max}$	Pareto$_{moy}$	Pareto$_{max}$	t_{moy}
10	13025.2	56477	8365.0	44004	221.0
20	11890.3	66874	6317.2	41274	204.5
30	7217.4	61111	3090.4	39295	242.1
40	3514.0	39497	683.0	9122	255.0
50	3442.4	32556	918.8	15366	285.1
100	1744.5	3080	145.2	340	296.6

TAB. 4.8 – Tests pour $t = 30$, $t = 120$ et $t = 300$ secondes.

De la même façon que pour $t = 30$ secondes, le nombre de noeuds explorés diminue quand la valeur de n augmente. Par rapport à la table avec $t = 30$, le nombre de noeuds explorés a plus que doublé. En revanche, le nombre de Pareto trouvés augmente beaucoup plus proportionnellement, ce qui peut laisser supposer qu'ils se rapprochent davantage de la courbe d'efficacité.

Pour $t = 300$ secondes, le nombre de noeuds explorés en moyenne est presque le double que pour $t = 120$ sauf pour $n = 100$. Le nombre de Pareto a également doublé. Par rapport à $t = 30$, le nombre de Pareto obtenu a environ quintuplé.

4.4 Modèle bicritère

On s'intéresse dans ce paragraphe au modèle bicritère présenté paragraphe 3.4.2. Dans ce modèle, un des deux critères représente le coût total d'avance et de retard, le second est le retard maximum. L'approche proposée pour trouver un optimum de Pareto est l'approche ϵ-contrainte, décrite ci-après.

4.4.1 Approche ϵ-contrainte pour l'obtention d'un optimum de Pareto

On souhaite résoudre le problème suivant :

$$\text{Minimiser} \sum_{i=1}^{n} Z_i$$
$$sc$$
$$(3.1) \ldots (3.10)$$

$$T_i \leq \epsilon, \forall i, 1 \leq i \leq n \tag{4.14}$$

La contrainte est équivalente à introduire des dates de fin impératives pour chaque travail, $\tilde{d}_i = d_i + \epsilon$. Le problème à résoudre peut se noter $1|d_j, \tilde{d}_j| \sum \alpha_j E_j + \beta_j T_j$, ce qui est exactement le même problème que précédemment, avec un autre calcul des dates de fin impératives et des dates de début au plus tôt toutes nulles. Il est donc intéressant de noter que l'on peut utiliser les mêmes algorithmes de résolution à savoir $HEACT1$, $HEACT2$ et $HEACT3$.

Nous avons évalué expérimentalement les algorithmes $HEACT2$ et $HEACT3$ sur le problème $1|d_j, \tilde{d}_j = d_j + \epsilon| \sum \alpha_j E_j + \beta_j T_j$ selon un protocole expérimental similaire à celui utilisé pour le problème multicritère.

Comme pour le problème multicritère on distingue les problèmes de petite taille ($n \in \{4, 6, 8, 10, 20, 30\}$) et les problèmes de taille moyenne ($n \in \{50, 60, 70, 80, 90, 100, 110, 120, 130\}$). Les durées d'exécution p_j sont tirées au hasard en utilisant une distribution uniforme entre 1 et 100 tandis que pour les poids α_j et β_j on utilise une loi uniforme entre 1 et 10. Pour générer les dates dues d_j, on introduit P qui est la somme des durées d'exécution générées, en d'autres termes $P = \sum_{j=1}^{n} p_j$. Pour chaque taille de problème on génère d_j au hasard entre p_j et $P * \phi$ ou ϕ est un autre aléatoire compris entre 1.5 et 2.5. Ce protocol a été mis au point expérimentalement aprés plusieurs essais de sorte à générer au final des instances suffisamment contraintes pour être intéressantes à résoudre.
La borne ϵ à la valeur du critère T_{max} a été générées selon trois configurations : "ϵ petit", "ϵ moyen" et "ϵ grand". Dans le premier cas on considère $0 \leq \epsilon \leq 3n + 1$, dans le second

	HEACT2						
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$	$imp(\%)$	ord
4	0.00	0	0.98	29.31	5.56	2.36	1
6	0.00	0	15.52	353.90	15.56	1.73	15
8	0.00	0	24.67	671.52	27.78	0.04	35
10	0.02	1	21.55	315.47	27.78	0.93	-
20	0.01	1	41.53	1166.67	12.22	10.31	-
30	0.05	1	28.09	319.26	17.78	7.23	-

	HEACT3 ($\gamma = 0.7$)					
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$	ord
4	0.05	1	0.00	0.00	5.56	0
6	0.23	1	10.19	437.60	14.44	3
8	0.86	2	2.64	103.64	27.78	5
10	2.36	4	1.66	29.39	23.33	-
20	38.08	59	8.98	115.28	7.78	-
30	197.35	338	37.65	322.50	14.44	-

TAB. 4.9 – Résultats de HEACT2 et HEACT3 sur des problèmes de petite taille pour $3n + 1 \leq \epsilon \leq 6n + 1$

$3n + 1 \leq \epsilon \leq 6n + 1$ et dans le troisième $6n + 1 \leq \epsilon \leq 9n + 1$. Dans un soucis de synthèse nous ne présentons ici que les résultats obtenus dans le cas ϵ moyen. Pour chaque taille de problème, 30 instances sont générées aléatoirement.
Tous les algorithmes ont été développés en langage C. Les tests ont été effectués sur un PC pentium IV 2.8Ghz avec 256Mb de RAM, sous l'environnement MS Windows.

Les statistiques construites sont identiques à celles réalisées pour le modèle multicritère dans la section 4.2.4. Pour les problèmes de petite taille, la table 4.9 présente les résultats obtenus par comparaison avec le solveur CPLEX.

Ces résultats montrent que le problème $1|\tilde{d}_i|\sum_j \alpha_j E_j + \beta_j T_j$ se résout moins bien que le problème avec dates de début au plus tôt. Concernant l'heuristique HEACT2 on constate pour les trés petites tailles de problème elle améliore peu ou pas les résultats retournées par l'heuristique HEACT1. L'heuristique HEACT3 trouve quand à elle plus de solution faisables que HEACT2 et lorsqu'elle trouve des solutions faisables la déviation à l'optimale est plus faible. Cela tend à confirmer la conclusion obtenue pour le modèle bicritère et qui consiste à dire que HEACT3 donne de meilleurs résultats que HEACT2.
Enfin, on notera que même sans dates de début au plus tôt les écarts dans les valeurs de la fonction objectif pour une instance donnée, peuvent être importants. Ainsi pour $n = 6$, HEACT3 trouve dans le pire des cas une solution à 437.60% de la solution optimale alors que cette solution est la troisième meilleure solution aprés la solution optimale (colonne ord).

La table 4.10 présente les résultats obtenus sur les instances de taille moyenne. On s'aperçoit que même sans utilisation d'une borne inférieure pour guider l'heuristique HEACT3, celle-ci donne de meilleurs résultats que HEACT2 et, trés souvent, résout plus de problèmes (colonne inf). Les temps de calcul de HEACT3, bien que largement supérieurs à ceux de HEACT2, restent raisonnables.

	HEACT2				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
50	0.53	1	7.63	84.11	28.89
60	0.87	2	5.55	52.83	31.11
70	0.53	1	10.50	104.82	22.22
80	3.89	6	8.07	110.66	5.56
90	5.90	11	8.49	60.77	7.78
100	8.47	14	8.84	52.21	7.78
110	11.58	19	10.73	67.90	15.56
120	15.82	25	8.49	54.45	12.22
130	23.14	35	12.73	76.16	7.78

	HEACT3 ($\gamma = 0\%$)				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
50	7.80	11	2.96	52.89	26.67
60	15.34	21	4.96	84.16	31.11
70	27.54	44	2.52	31.45	24.44
80	54.86	76	0.60	6.82	10.00
90	86.08	116	2.61	80.44	5.56
100	126.40	173	3.18	35.86	5.56
110	181.12	259	2.64	57.06	8.89
120	262.35	372	2.33	50.77	8.89
130	369.68	583	1.31	21.14	6.67

TAB. 4.10 – Résultats de HEACT2 et HEACT3 sur des problèmes de taille moyenne pour $3n + 1 \leq \epsilon \leq 6n + 1$

Chapitre 5

Cas du flow-shop de permutation à deux machines

5.1 Introduction

Nous considérons un problème de type flow-shop à deux machines, défini de la façon suivante : un ensemble de n travaux doit être exécuté, chaque travail i se décompose en une opération $o_{i,1}$ qui doit s'exécuter sur la machine disjonctive M_1 avant la seconde opération du travail i, notée $o_{i,2}$, qui doit s'exécuter sur la machine disjonctive M_2. Par rapport aux notations du chapitre 3, on a $\pi_i(1) = 1$ et $\pi_i(2) = 2$.

Chaque opération $o_{i,j}$, $1 \leq i \leq n$, $j \in \{1, 2\}$, possède une durée d'exécution $p_{i,j}$, et chaque travail i possède une date due d_i. On note toujours α_i et β_i la pénalité d'avance et de retard pour le travail i, respectivement. Il est possible de laisser la machine libre si nécessaire. La préemption n'est pas autorisée alors que le lot-streaming l'est. La fonction coût, associée à chaque travail, à optimiser est (confère Chapitre 3, section 3.2.2.2) :

$$Z_i = \frac{q_i}{\delta}\gamma_i(\sum_{k=1}^{\delta} t_{i,2,k} - \sum_{k=1}^{\delta} t_{i,1,k}) - \frac{q_i}{\delta}k_i \sum_{k=1}^{\delta} t_{i,2,k} + k_i q_i C_i + k_i q_i E_i + \beta_i T_i + \lambda_i \delta$$

On devine que la fonction de coût totale va être plus difficile à optimiser efficacement que pour le problème à une machine. Nous ne nous sommes intéressés qu'à la résolution du modèle multicritère, et cela pour des questions de temps. La résolution du modèle bicritère est un travail en cours de réalisation. Comme dans le chapitre 3, nous nous intéressons uniquement à la résolution du problème lorsque le nombre de sous-lots δ est fixé.

Nous proposons principalement deux méthodes approchées pour résoudre le problème : un algorithme génétique et une recherche par faisceaux filtrée avec recouvement des erreurs, méthode dont le principe a été décrit dans le chapitre précédent. Il nous est apparu interessant de poursuivre l'application de cette méthode à des problèmes de type Juste-à-Temps étant donné que pour ces problèmes des algorithmes simples de liste ne donnent pas de bons résultats (cela est dû principalement au fait qu'il y a un problème de calcul des dates de début à résoudre en plus du problème de séquencement).

Finalement, des tests sont effectués pour comparer ces deux méthodes.

5.2 Insertion optimale de temps morts

Dans le chapitre 4 nous avons résolu le problème d'insertion de temps morts à séquence fixée en appliquant l'algorithme $ECS1$ présenté dans le chapitre 2. Cet algorithme est ap-

plicable dès lors qu'à chaque opération est attachée une fonction de coût convexe dont on peut calculer le minimum en temps polynomial. Il nous est apparu difficile d'adapter simplement cet algorithme au problème de flowshop à deux machines : Quelle fonctions de coût doit-on considérer pour les opérations passant sur la première machine ? Comment calculer en temps polynomial les minima des fonctions de coût Z_i ?

Etant donné que, lorsque la valeur de δ est fixée le problème est linéaire, le problème de calcul des dates de début optimales des sous-lots est résoluble en temps polynomial. Cela est dû au fait que le problème est modélisable par un programme linéaire simple obtenu en considérant le modèle mathématique présenté dans le chapitre 3, section 3.3, et en instanciant toutes les variables binaires $x_{i,j}$ en fonction de la séquence sur chaque machine. Ainsi, le modèle mathématique appliqué au problème multicritère est le suivant :

Données :
n le nombre de travaux,
δ le nombre de sous-lots par travail,
m le nombre de machines, et donc d'opérations par travail,
$p_{i,j}$ durée de la j-ième opération du travail J_i, $1 \leq i \leq n$, $1 \leq j \leq 2$,
d_i date de fin souhaitée du travail J_i,
b_i borne sur le critère Z_i,
$x_{i,j}$, vaut 1 si le travail J_i précède le travail J_j.
Variables :
$t_{i,j,k} \in \mathbb{R}$, la date de début du k-ième sous-lot du travail J_i sur la machine M_j,
$C_{i,j} \in \mathbb{R}$, la date de fin du travail J_i sur la machine j,
$T_i \in \mathbb{R}$, retard du travail J_i,
$E_i \in \mathbb{R}$, avance du travail J_i.

Minimiser $\sum_i Z_i$
sc

$t_{i',j,\delta} \geq t_{i,j,1} + \frac{p_{i,j}}{\delta} - HV(1 - x_{i,i'})$	$\forall i, 1 \leq i \leq n, \forall i', i' \neq i, 1 \leq i \leq n, \forall j, 1 \leq j \leq 2$
$t_{i',j,\delta} \geq t_{i,j,1} + \frac{p_{i,j}}{\delta} - HV(1 - x_{i,i'})$	$\forall i, 1 \leq i \leq n, \forall i', i' \neq i, 1 \leq i \leq n, \forall j, 1 \leq j \leq 2$
$t_{i,j,\delta} \geq t_{i',j,1} + \frac{p_{i',j}}{\delta} - HVx_{i,i'}$	$\forall i, 1 \leq i \leq n, \forall i', i' \neq i, 1 \leq i \leq n, \forall j, 1 \leq j \leq 2$
$t_{i,j,k+1} \geq t_{i,j,k} + \frac{p_{i,j}}{\delta}$	$\forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq 2, \forall k, 1 \leq k \leq \delta - 1$
$t_{i,2,k} \geq t_{i,1,k} + \frac{p_{i,1}}{\delta}$	$\forall i, 1 \leq i \leq n, \forall k, 1 \leq k \leq \delta$
$C_{i,j} = t_{i,j,\delta} + \frac{p_{i,j}}{\delta}$	$\forall i, 1 \leq i \leq n, \forall j, 1 \leq j \leq 2$
$T_i \geq 0$	$\forall i, 1 \leq i \leq n$
$T_i \geq C_{i,2} - d_i$	$\forall i, 1 \leq i \leq n$
$E_i \geq 0$	$\forall i, 1 \leq i \leq n$
$E_i \geq d_i - C_{i,2}$	$\forall i, 1 \leq i \leq n$
$Z_i \leq b_i$	$\forall i, 1 \leq i \leq n$

Dans la suite de ce chapitre le problème de calcul des dates de début optimales à séquence fixée est résolu à l'aide du solver CPLEX appliqué au modèle précédent.

5.3 Un algorithme génétique

Les principes de l'algorithme génétique sont dûs à Holland [Holland, 1975]. Ce type d'algorithme fait évoluer un ensemble de solutions à un problème donné (les individus) à l'aide de différents processus tels que la sélection, le croisement, la mutation, etc.

Il s'inspire des phénomènes naturels en considérant qu'une solution à un problème constitue un individu qui va transmettre ses caractéristiques à ses descendants. Le processus de "croisement" décrit la reproduction de deux individus et le processus "mutation" intervient aléatoirement pour modifier le génome d'un individu. Le lecteur pourra se référer à [Portmann and Vignier, 2001] pour une description détaillée des algorithmes génétiques pour la résolution des problèmes d'ordonnancement.

Le principe général de l'algorithme génétique est le suivant :

(a) initialisation : générer une population initiale,
(b) évaluation : évaluer chaque individu de la population,
(c) sélection : choisir des couples d'individus suivant leur performance (principe de la roulette),
(d) croisement des couples d'individus avec une certaine probabilité de croisement (p_{cr}),
(e) mutation de certains individus avec une certaine probabilité de croisement (p_{mut}),
(f) mise à jour de la population,
(g) fin sur un critère d'arrêt (nombre d'itérations sans amélioration par exemple).

Nous présentons maintenant les différentes caractéristiques de l'algorithme génétique, noté $HET1$ ([Estève et al., 2003b]), que nous avons implémenté pour notre problème de calcul d'un optimum de Pareto strict.
Il faut noter que le critère d'arrêt utilisé est le *nombre d'itération sans amélioration de la meilleure solution*. Expérimentalement, ce nombre a été fixé à 5. Il constitue un bon compromis entre temps de recherche et efficacité de l'algorithme.

5.3.1 Initialisation de la population

Pour initialiser la population de l'algorithme $HET1$, nous avons utiliser une heuristique simple qui trie les travaux selon leur date de fin souhaitée croissante pour construire une séquence sur les deux machines. Le problème de calcul des dates de début optimales est alors résolu pour cette séquence, conduisant ainsi à un ordonnancement. On notera que celui-ci peut être infaisable à cause des bornes b_i sur les critères Z_i.

Une partie de la population est générée selon cette heuristique tandis que l'autre partie est générée aléatoirement. Le pourcentage de solutions générées aléatoirement est noté p_{gnr}.

5.3.2 Codage et évaluation d'une solution

Dans $HET1$, un individu est une séquence de travaux. Les dates de début de chaque sous-lot sont calculées en résolvant le programme linéaire simple présenté dans la section 5.2. La résolution de ce programme, pour un individu donné, peut conduire à insérer des temps morts entre les sous-lots d'une même opération. Par ailleurs, cette résolution permet de calculer la valeur de la fonction objectif $\sum_j Z_j$ associée à l'individu ou d'établir qu'il n'existe pas d'ordonnancement faisable. Dans ce dernier cas, la valeur de la fonction objectif associée à l'individu est égal à HV.

5.3.3 Opérateurs de croisement et de mutation

Le croisement entre deux individus qui a été retenu est *le croisement en deux points*. Une étape du croisement est représentée figure 5.1. On définit aléatoirement deux points de coupe dans les individus parents I_1 et I_2. La zone à l'intérieur des deux points s'appelle la zone de mapping. On prend chaque travail de la zone de mapping, soit J_5 le premier travail, on regarde la position de ce travail dans l'autre parent, par exemple la position 1 et on l'échange avec le travail qui se trouve à cette position, soit le travail J_4. On procède de la même façon pour tous les travaux de la zone de mapping de I_1 et I_2. Les autres travaux ne sont pas changés.

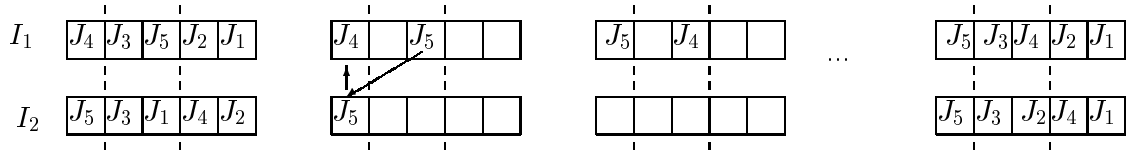

FIG. 5.1 – Croisement de deux individus

Deux individus sélectionnés dans la population ont une probabilité notée p_{crois} d'être croisés. Les individus fils issus du croisement de deux parents ont une probabilité notée p_{mut} de subir une mutation. Celle-ci consiste à changer aléatoirement la position d'un travail dans la séquence. Les nouveaux individus obtenus sont évalués comme indiqué dans la section précédente.

5.3.4 Paramétrage de la méthode

Une étape importante dans la mise au point d'un algorithme génétique consiste à établir les valeurs les plus pertinentes de ses paramètres pour le problème résolu. Les paramètres d'un tel algorithme, pour l'implémentation que nous en avons faite, sont la probabilité de croisement p_{crois}, la probabilité de mutation p_{mut}, le pourcentage d'individus p_{gnr} générés aléatoirement dans la population initiale, le nombre d'individus N dans la population et le nombre d'itérations I sans amélioration de la meilleure solution provoquant l'arrêt de l'algorithme.

La détermination de ces paramètres a été réalisée grâce à des expérimentations numériques ([Badille, 2004]). Ainsi différentes valeurs pour ces paramètres ont pu être testées sur des instances générées aléatoirement. Il en ressort que le meilleur compromis efficacité/temps de calcul requis est obtenu pour les valeurs suivantes :

$$\begin{array}{ll} p_{crois} & : 70\% \\ p_{mut} & : 20\% \\ p_{gnr} & : 80\% \\ N & : 40 \\ I & : 5 \end{array}$$

Il faut noter que la résolution du programme linéaire simple requiert un temps de calcul important dès lors qu'elle est réalisée de façon répétitive dans l'algorithme. Ainsi, dans l'absolu, il est possible d'obtenir un meilleur algorithme génétique en augmentant les valeurs de N et I, mais au prix d'une augmentation importante du temps de calcul.

5.4 La Recovering Beam Search

Les principes de cette méthode ont été présentés dans la section 4.2.3.2. Nous avons également appliqué cette méthode au problème de flowshop à deux machines de type Juste-à-Temps. Notre idée directrice est de vérifier expérimentalement que cette méthode donne de bons résultats sur ce type de problème, par comparaison avec des heuristiques plus "classiques".

L'arborescence explorée par cette heuristique, notée $HET2$, est construite de la façon suivante. Chaque noeud contient une séquence partielle, notée σ, de travaux déjà ordonnancés en tête d'ordonnancement. On lui associe donc un ensemble de travaux, noté Ω, restant à placer. La création d'un noeud consiste à placer un travail non ordonnancé en fin de séquence. La figure 5.2 illustre ce schéma de séparation.

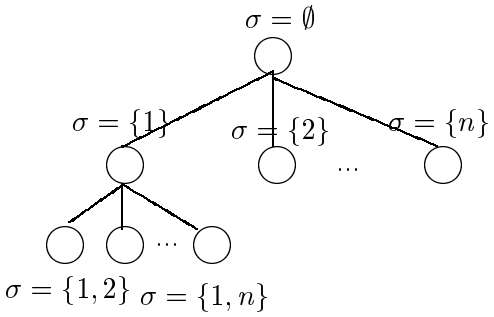

FIG. 5.2 – Séparation des noeuds de la RBS

Nous présentons maintenant chacun des éléments constituant l'heuristique $HET2$.

5.4.1 Le filtre

Etant donnée la complexité de la fonction de coût, aucune condition de dominance n'a pu être établie pour le problème. Nous avons néanmoins utilisé quelques régles de pseudo-dominance qui en pratique ne détériore pas trop le comportement de l'heuristique $HET2$ et permettent de gagner du temps. Soit $J_i, J_j \in \Omega$ deux travaux à ordonnancer pour le noeud en cours d'examen. On cherche à savoir si le noeud fils défini par $\sigma//\{J_i\}$ doit être conservé ou pas. On considère que le noeud défini par $\sigma//\{J_i\}$ est coupé si une des conditions suivantes est vérifiée :

1. $d_j \leq d_i - p_{i,2}$,
2. $d_j - p_{j,2} \leq d_i - p_{i,2} - p_{i,1}$,
3. $d_i - p_{i,2} - C_2(\sigma) \geq p_{j,2}$,
4. $d_i - p_{i,2} - p_{i,1} - C_1(\sigma) \geq p_{j,1}$,

avec $C_k(\sigma)$ la date de fin de la séquence σ sur la machine M_k. Même s'il est évident que l'application des conditions dans le filtre peut conduire à supprimer des noeuds conduisant à des bonnes solutions, on constate expérimentalement qu'elles contribuent à accélérer le comportement de $HET2$ sans détériorer énormément la qualité des solutions produites.

5.4.2 Evaluation d'un noeud

Dans un algorithme de type recherche par faisceaux filtrée, chaque noeud créé est évalué par une borne inférieure et une borne supérieure. L'objectif de ces bornes n'est pas de couper le noeud mais simplement d'orienter la recherche heuristique dans l'arbre de recherche. Ainsi, on n'attend que les bornes encadrent précisement la meilleure solution atteignable dans une sous-arborescence, mais plutôt qu'en comparant deux noeuds elles soient capable de détecter celui conduit à la meilleure solution.

Pour notre problème, à chaque noeud ν, la borne inférieure est calculée par une relaxation linéaire partielle du modèle mathématique (section 3.3). Ainsi les variables correspondant aux travaux de σ sont fixées tandis que les variables associées aux travaux de Ω restent libres. Ces dernières sont alors considérées comme des variables à valeur réelle comprises dans l'interval $[0;1]$. La valeur de la fonction objectif retournée par le solveur est notée LB et constitue une borne inférieure au noeud ν. Il convient de noter également que si aucune solution faisable n'est trouvée alors on pose $LB = HV$ avec HV une valeur suffisament grande.

De même, à chaque noeud ν, une borne supérieure, notée UB est calculée à l'aide de l'algorithme présenté dans la table 5.1. Si aucune solution faisable n'est construite alors on pose $UB = HV$.

DEBUT
Pour $i \in \Omega$ Faire
Poser $C_i = d_i$ et calculer Z_i^{ref} la contribution de i à la fonction coût
Poser $C_i = d_i - 1$ et calculer Z_i^g la contribution de i à la fonction coût si i est décalée de 1 à gauche de sa date due
Poser $C_i = d_i + 1$ et calculer Z_i^d la contribution de i à la fonction coût si i est décalée de 1 à droite de sa date due
Calculer : $\pi_i^g = Z_i^g - Z_i^{ref}$, $\pi_i^d = Z_i^d - Z_i^{ref}$, $\pi_i = \max(0, \pi_i^g, \pi_i^d)$
Fin pour
Trier les travaux de Ω par π_i croissant
Appel au PLS pour calculer les dates de début des travaux
Retourner la valeur de UB
FIN

TAB. 5.1 – Borne supérieure en ν

Chaque noeud est évalué par une combinaison linéaire $f = \alpha LB + (1-\alpha)UB$ à partir du moment où $LB, UB \neq HV$. Dans le cas contraire on pose $f = \min(LB, UB)$. A un niveau donné de l'arbre de recherche, l'heuristique $HET2$ choisi les w noeuds de plus petite évaluation avec w la largeur du faisceaux.

5.4.3 La phase de recouvrement

Une fois que w noeuds ont été sélectionnés aprés le processus d'évaluation, on leur applique la phase de recouvrement. Celle-ci consiste à réaliser sur chaque séquence σ une recherche locale dans l'objectif de détecter un meilleur arrangement des travaux déjà ordonnancés dans la séquence σ considérée. Conceptuellement, pour une séquence σ accepter

comme meilleure un réarrangement σ' est équivalent à réaliser un saut dans l'arbre de recherche vers un noeud de même profondeur.

La recherche locale appliquée dans l'heuristique $HET2$ réalise pour chaque travail $J_i \in \sigma$ une insertion à toutes les positions de σ. A chaque nouvelle insertion, le problème de calcul des dates de début optimales est résolu (confère section 5.2). Cette phase requiert évidemment un temps de calcul important notamment dû à la résolution de ce dernier problème.

Un dernier point à aborder est lié à la condition d'acceptation d'une séquence σ' comme meilleure qu'une séquence σ, ces deux séquences contenant le même ensemble de travaux. On note $C_k(\sigma)$ la date de fin de la séquence σ sur la machine M_k aprés résolution du problème de calcul des dates de début optimales. Le noeud associé à la séquence σ' est accepté en remplacement du noeud associé à la séquence σ si les trois conditions suivantes sont vérifiées :

1. $C_1(\sigma') \leq C_1(\sigma)$,
2. $C_2(\sigma') \leq C_2(\sigma)$,
3. $\sum_j Z_j(\sigma') \leq \sum_j Z_j(\sigma)$,
4. il existe des ordonnancements faisables associés à σ et σ'.

On considère plus précisement qu'une des inégalité parmi les conditions 1, 2 et 3 doit être stricte. Lorsque, dans le voisinage de la solution σ courante, une solution σ' est acceptée la phase de recouvrement est arrêtée. Une recherche locale plus exhaustive aurait pu être considérée, mais au détriment du temps de calcul requis par l'algorithme. Nous avons donc opté pour une stratégie de recherche plus "légère".

5.4.4 Paramétrage de la méthode

Contrairement à un algorithme génétique, une recherche par faisceaux filtrée avec recouvrement des erreurs possède peu de paramètres à régler expérimentalement. Les paramètres à considérer sont la largeur du faisceau, notée w, et la valeur du poids, notée α, de chaque borne dans l'évaluation d'un noeud. Le paramètre w suit une règle simple : plus w augmente et plus le temps de calcul requis augmente (linéairement) et plus l'heuristique risque d'être performante. Dans les expérimentations réalisées, nous considérons que $w = 1$ et comparons donc la version "la plus mauvaise" de l'heuristique $HET2$ à l'heuristique $HET1$.
Concernant le paramètre α nous avons exécuté l'heuristique $HET2$ pour plusieurs valeurs de α sur des instances générées aléatoirement ([Badille, 2004]). Nous retenons alors la valeur de ce paramètre qui conduit aux meilleurs résultats pour les instances générées. Au final, nous avons $\alpha = 0.5$ ce qui implique que la borne inférieure et la borne supérieure ont le même poids dans l'évaluation d'un noeud.

5.4.5 Expérimentations numériques et résultats

Les expériences qui ont été menées ont pour but de comparer les heuristiques $HET1$ et $HET2$ par rapport à la solution optimale pour les problèmes de petite taille d'une part, et d'autre part à la meilleure des deux pour les problèmes de taille moyenne. Les problèmes de petite taille sont ceux pour lesquels $n \in \{4, 6, 8, 10, 20, 30\}$ et les problèmes de taille moyenne sont ceux pour lesquels $n \in \{10, 20, 30, 40, 50, 60\}$.

Les données pour le problème considéré qui vont être générées sont les suivantes : $p_{i,j}$, k_i, γ_i, q_i, δ, λ_i, β_i, d_i et b_i. Nous donnons ci-dessous le mode de génération aléatoire de ces

données, la notation $U[X, Y]$ signifiant que la variable est générée selon une loi uniforme dans l'interval $[X, Y]$.

- $p_{i,j} \in U[10, 100]$,
- $\gamma_i, k_i \in U[1, 10]$,
- $\beta_i \in U[10, 500]$,
- $\lambda_i \in U[10, 100]$,
- $q_i \in U[40, 100]$,
- $\delta \in U[2, Minqi]$ avec $Minqi = \min_i q_i$ le minimum des q_i générés pour l'instance en cours,
- $d_i \in U[p_{i,1} + p_{i,2}, \nu\bar{P}]$ avec $\bar{P} = \sum_{i,j} p_{i,j}$ et $\nu \in \{2, 3.5, 5\}$. On a ainsi trois classes de problèmes reflétant des configurations des dates dues différentes,
- $b_i \in U[\Gamma_i, 2\Gamma_i]$ avec $\Gamma_i = \beta_i(\bar{P} - d_i) + k_i q_i d_i + \gamma_i q_i \bar{P} + \lambda_i q_i/2$.

Pour chaque taille de problème n et chaque valeur du paramètre ν 30 instances sont générées aléatoirement. Les résultats présentés sont aggrégés par taille de problème ce qui implique notamment que les moyennes sont calculées sur la base de 90 instances.
Tous les algorithmes ont été développés en langage C. Les tests ont été effectués sur un PC pentium IV 1.7Ghz avec 256Mb de RAM, sous l'environnement MS Windows.

Dans le reste de cette section, les notations F_{HET1} et F_{HET2} font références pour une instance donnée à la valeur de la fonction objectif pour les heuristiques $HET1$ et $HET2$ respectivement. Si une des deux heuristiques échoue pour calculer une solution faisable, on place cette valeur de fonction objectif à l'infini. Dans les tableaux de résultat présenté, on note t_{avg} et t_{max} respectivement la durée moyenne et la durée maximale d'exécution sur la machine, en secondes (s). Nous nous intéressons également au pourcentage d'instances pour lesquelles une heuristique donnée n'a pas su retourner une solution faisable. Ce résultat est présenté dans la colonne inf (infaisable).

Il y a d'autres résultats encore que nous prenons en considération et qui sont dépendants des expérimentations menées à bien et introduites ci-après.

Comparaison des heuristiques sur les problèmes de petite taille
Nous nous intéressons à la comparaison des heuristiques $HET1$ et $HET2$ par rapport à la solution optimale obtenue par application du solveur CPLEX sur le programme en nombres entiers. Notons F_{opt} la valeur optimale de la fonction objectif calculée par CPLEX pour une instance donnée faisable. Pour chaque heuristique les colonnes δ_{avg} et δ_{max} font référence aux déviations moyennes et maximales en pourcentage par rapport à la solution optimale respectivement pour les instances où l'on peut calculer une solution faisable non triviale (où tous les travaux sont à l'heure). Pour chaque instance, la déviation par rapport à l'optimal est bien évidemment donnée par

$$\delta = \frac{F - F_{opt}}{F_{opt}}$$

où F fait bien sûr référence à la valeur de la solution retournée par l'heuristique considérée. Les résultats sont présentés dans la table 5.2.

Les résultats présentés dans la table 5.2 montrent que l'heuristique $HET2$, sur des instances de petite taille, retourne systématiquement la solution optimale lorsque le problème est faisable. Le temps de calcul requis est relativement réduit. L'heuristique $HET1$ quant à elle fournis des résultats dont la qualité se détériore lorsque la taille du problème augmente. Le temps de calcul requis est relativement important ce qui peut être lié au fait que l'algorithme génétique explore de nombreuses solutions avant d'arriver à un optimum local. Ce

	$HET1$				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
4	1.24	7	0.00	0.00	16.67
6	2.35	7	8.45	106.38	26.67
8	5.52	23	17.59	129.98	31.11
10	9.09	39	30.53	124.02	36.67
20	51.72	285	104.71	249.54	54.44
30	97.83	1184	301.87	1121.07	67.78

	$HET2$				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
4	0.05	1	0.00	0.00	16.67
6	0.13	1	0.00	0.00	26.67
8	0.34	1	0.00	0.00	30.00
10	0.55	2	0.00	0.00	36.67
20	5.10	12	0.00	0.00	54.44
30	22.16	61	0.00	0.00	67.78

TAB. 5.2 – Résultats de $HET1$ et $HET2$ sur des problèmes de petite taille

temps de calcul important est également une conséquence du temps requis pour résoudre le problème de calcul des dates de début optimales. Par ailleurs, comme pour le problème à une machine abordé dans le chapitre 4, l'écart dans le pire des cas entre la solution optimale et la solution retournée par $HET1$ est relativement important. Cela peut être expliqué par la grande variabilité dans l'échelle des valeurs associées aux ordonnancements faisables.

Comparaison des heuristiques sur les problèmes de taille moyenne
Pour chaque heuristique les colonnes δ_{avg} et δ_{max} font références à la déviation moyenne et maximale en pourcentage par rapport à la meilleure des deux heuristiques, pour les instances donnant des ordonnancements faisables. Pour une instance donnée, la déviation est bien évidemment donnée par

$$\delta = \frac{F - F_{best}}{F_{best}}$$

où F réfère à la valeur de la solution retournée par l'heuristique considérée et $F_{best} = \min(F_{HET1}, F_{HET2})$. Les résultats sont présentés dans la table 5.3.

Les résultats présentés dans la table 5.3 montrent que l'heuristique $HET2$ est la plus performante en terme de temps de calcul et en terme d'efficacité. Ces résultats confirment ceux obtenus sur les problèmes de petite taille. Ils confirment également le fait que l'algorithme génétique met en oeuvre un processus trop lourd en temps de calcul pour pouvoir calculer de bonnes solutions en un temps raisonnable.

	$HET1$				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
10	8.02	26	35.16	147.45	34.44
20	52.21	296	94.83	297.68	52.22
30	138.98	769	298.45	1028.45	58.89
40	249.62	1857	377.22	1128.48	67.78
50	409.47	4138	599.90	1218.63	70.00
60	678.74	6690	824.25	1646.30	67.78

	$HET2$				
n	t_{avg} (s)	t_{max} (s)	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$inf(\%)$
10	0.53	2	0.00	0.00	34.44
20	5.22	12	0.00	0.00	52.22
30	23.63	60	0.00	0.00	58.89
40	72.71	169	0.00	0.00	66.67
50	193.44	445	0.00	0.00	70.00
60	409.75	895	0.00	0.00	67.78

TAB. 5.3 – Résultats de $HET1$ et $HET2$ sur des problèmes de taille moyenne

Conclusion et Perspectives

Ce travail de thèse a porté sur l'étude et la résolution des problèmes d'ordonnancement de type Juste-à-Temps. On distingue deux catégories de littérature : celle portant principalement sur les problèmes de gestion et d'organisation de la production et celle portant principalement sur les problèmes d'ordonnancement. Les travaux présentés dans la première catégorie, et qui sont liés aux systèmes Juste-à-Temps, traitent de concepts généraux sur la façon de produire : comment organiser physiquement son atelier pour bien produire en Juste-à-Temps ? Quelles sont les conséquences au niveau de la gestion des ressources humaines pour bien produire en Juste-à-Temps ? *etc.* On peut donc considérer que cette littérature présente, notamment, des recommendations générales pour toutes les phases de la production, y compris la phase d'ordonnancement. La littérature sur les problèmes d'ordonnancement contient de nombreux travaux sur les problèmes d'ordonnancement de type Juste-à-Temps. Ceux-ci considèrent essentiellement que l'aspect Juste-à-Temps d'un ordonnancement se mesure en terme de décalage des dates de fin des travaux par rapport à des dates dues. Plusieurs façons de mesurer ce décalage existent, sans qu'elles soient comparer entre elles : laquelle reflète au mieux une politique d'ordonnancement Juste-à-Temps ?

Le chapitre 1 présente un des premiers travaux réalisé dans cette thèse, à savoir mener une étude de la problématique de production Juste-à-Temps pour pouvoir en retenir les concepts relevant de la phase d'ordonnancement. Ainsi, nous montré que certaines notions de la production en Juste-à-Temps (la notion de "chaîne de machine", régulation des flux, réduction des coûts de production inutiles, respect des délais) devaient être pris en compte dans la phase d'ordonnancement. Plus précisemment, elles peuvent être prises en compte par un meilleur contrôle des coûts de stockage, notamment.

Dans le chapitre 2 nous avons présenté un état de l'art de certains problèmes d'ordonnancement de type Juste-à-Temps abordés dans la littérature. Cet état de l'art fait ressortir les différents modèles considérés et les principaux algorithmes présentés. Par ailleurs, nous mettons en évidence dans ce chapitre la problématique de calcul des dates de début optimales dès lors que la séquence des travaux est connue. Résoudre ce problème s'avère nécessaire dès lors que la fonction objectif ne constitue pas un critère régulier, ce qui est trés souvent le cas des problèmes d'ordonnancement Juste-à-Temps.

Dans le chapitre 3 nous proposons, à partir des notions retenues dans le chapitre 2, une modélisation nouvelle des problèmes d'ordonnancement Juste-à-Temps. Cette modélisation met en évidence le fait que les modèles de type avance/retard résolus dans la littérature ordonnancement ne sont représentatifs d'une politique Juste-à-Temps que dans certains cas trés particuliers. Les fonctions de coûts proposées reflètent des coûts de stockage des produits semi-finis et finis, des coûts de non respect des délais de production et des coûts liés à la réduction des cycles de production. A chaque travail est associé une fonction de coût.

Ces fonctions de coûts rendent le problème d'ordonnancement multicritère, ce qui intuitivement peut se comprendre : ordonnancer un travail pour qu'il se termine à sa date prévue tout en séjournant le moins possible dans les stocks peut conduire à avoir du retard ou une augmentation des coûts de stockage pour un ou plusieurs autres travaux.
Dans le chapitre 3, nous proposons également deux modèles possibles : un modèle multicritère où l'objectif est de minimiser les n fonctions de coût et un modèle bicritère où l'on minimise le coût total, *i.e.*la somme des fonctions de coût, ainsi qu'un critère de qualité externe.

Dans le chapitre 4, nous proposons l'application des modèles d'ordonnancements Juste-à-Temps multicritère et bicritère à un environnement à machine unique. Pour ce problème la fonction de coût se ramène à la fonction de coût classique en ordonnancement Juste-à-Temps, à savoir une somme des pénalités d'avance et de retard. Tout d'abord nous abordons le modèle multicritère pour lequel nous considérons le problème de calcule d'un optimum de Pareto strict à l'aide de l'analyse paramétrique. Ce problème se ramène alors à la résolution d'un problème $1|r_i, d_i, \tilde{d_i}|\sum_i \alpha_i E_i + \beta_i T_i$ particulier. Nous proposons des algorithmes de nature heuristique dont notamment une heuristique par voisinage et une recherche par faisceaux filtrée avec recouvrement des erreurs. Ce type d'algorithme, récent dans sous cette forme là, n'a jamais été appliqué à notre connaissance sur ce problème. Nous validons au travers d'expérimentations numérique l'efficacité de cette heuristique. Ces expérimentations confirment l'intuition initiale que nous avions, à savoir que ce type d'heuristique pouvait bien fonctionner pour des problèmes complexes où calculer simplement des séquences ne suffit pas pour avoir un bon ordonnancement. Une explication du bon fonctionnement de cet algorithme vient du schéma de branchement et de la possibilité, grâce aux bornes, à chaque niveau de choisir l'ensemble de solutions atteignables (un noeud) le plus prometteur. Nous étudions également dans ce chapitre le problème de l'énumération des optima de Pareto stricts lorsqu'on considère les n critères séparemment. Une heuristique et des résultats expérimentaux sont présentés.
Le modèle bicritère est également étudié. Nous considérons la minimisation du coût total sachant que le critère de qualité externe, le critère T_{max}, est borné par une constante. Autrement dit, nous appliquons l'approche ϵ-contrainte pour calculer un optimum de Pareto strict pour le problème bicritère. Calculer cet optimum revient à résoudre le problème $1|d_i, \tilde{d_i}|\sum_i \alpha_i E_i + \beta_i T_i$ pour lequel nous appliquons l'heuristique par voisinage et la recherche par faisceaux filtrée avec recouvrement des erreurs que nous avons présenté pour le modèle multicritère. Les résultats expérimentaux réalisés montrent encore une fois la supériorité de la recherche par faisceaux filtrée.

Dans le chapitre 5, nous nous intéressons au problème de type flowshop à deux machines et nous concentrons sur le modèle multicritère. Nous supposons également que les opérations sont constituées de pièces et donc divisibles. La contrainte de "lot-streaming" est alors imposée de sorte de pouvoir réduire les temps de fabrication des produits, ce qui va en faveur d'une politique Juste-à-Temps. Nous considérons le problème de calcul d'un optimum de Pareto strict en utilisant l'analyse paramétrique, ce qui nous conduit à résoudre le problème $F2|lot-streaming, d_i, Z_i \leq b_i|\sum_i Z_i$ où Z_i est la fonction de coût associée au travail J_i. Ce coût représente un coût de production et de stockage. Pour résoudre ce problème nous proposons, étant la complexité du problème, un algorithme génétique et une recherche par faisceaux filtrée avec recouvrement des erreurs. Des expérimentations numériques permettent de conclure à la supériorité de la recherche par faisceaux filtrée et confirment les conclusions obtenues pour le problème à une machine.

Les travaux menés dans cette thèse sont des travaux qui doivent être poursuivis car ils ouvrent une perspective intéressante, à notre avis, dans le domaine de l'ordonnancement Juste-à-Temps. Non seulement car ils montrent les limites des modèles abordés dans la littérature, mais également car ils montrent que les problèmes Juste-à-Temps sont des problèmes multicritères difficiles à résoudre. Les travaux menés montrent également qu'une méthode de type recherche par faisceaux filtrée avec recouvrement des erreurs peut être bien adaptée pour résoudre ce type de problème. Cela rejoint une conclusion obtenue par [Ow and Morton, 1988] et [Ow and Morton, 1989] qui avaient appliquée une recherche par faisceaux filtrée (sans recouvrement des erreurs et avec un mécanisme légèrement différent) à un problème d'ordonnancement à une machine Juste-à-Temps particulier.
A l'avenir il pourrait être intéressant, à notre avis, d'explorer plusieurs pistes :

- Adapter les heuristiques proposées pour le problème de flowshop multicritère au problème bicritère et étudier leur efficacité sur ce problème.
- Reprendre les problèmes étudiés dans ce manuscrit et proposer non plus des méthodes heuristiques mais des méthodes exactes. Il nous parait trés intéressant de travailler sur ce point étant donné que les problèmes abordés sont particuliers avec des fonctions de coût complexes. Il y a donc matière à proposer de nouveaux outils de résolution inexistant dans la littérature.
- Etendre les problèmes abordés à des problèmes encore plus proches d'une réalité industrielle. Notamment, il peut être intéressant d'appliquer cette approche à des problèmes réels ou simplement d'introduire dans les problèmes abordés ici des contraintes/éléments de la réalité, comme par exemple introduire des moyens de transport (chariots, tapis roulant, *etc.*), et étudier comment proposer un outil d'aide à la décision efficace.
- Intégrer les problématiques d'approvisionnement. Une autre voie de recherche envisageable peut être d'intégrer dans les modèles proposés dans le cadre d'un calcul des besoins en phase de plannification à moyen terme. Ainsi, on pourrait avoir une estimation plus précise des matières premières à fournir et à quelle date pour pouvoir réaliser une bonne production Juste-à-Temps. Cela permettrait de limiter des déchets dans le système impliqués par un calcul des besoins qui serait complètement indépendant de l'ordonnancement réalisé. C'est malheureusement ce qui se fait à l'heure actuel.

Bibliographie

[Ahmadi and Matsuo, 2000] Ahmadi, R. H. and Matsuo, H. (2000). A mini-line approach for pull production. *European Journal of Operational Research*, 125 :340–358.

[Almeida and Centeno, 1998] Almeida, M. T. and Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers and Operations Research*, 25(7/8) :625–635.

[Anthony, 1965] Anthony, R.-N. (1965). Planning and control systems : a framework for analysis. *Harvard University Press.*

[Azizoglu et al., 1991] Azizoglu, M., Kondakci, S. K., and Kirca, O. (1991). Bicriteria scheduling problem involving total tardiness and total earliness penalties. *International Journal of Production Economics*, 23 :17–24.

[Badille, 2004] Badille, J. (2004). Développement d'algorithmes pour la résolution de problèmes d'ordonnancement de type "juste-à-temps". *Rapport de stage, Laboratoire d'Informatique de l'Université François-Rabelais de Tours.*

[Baker and Scudder, 1990] Baker and Scudder (1990).

[Brauner and Crama, 2004] Brauner and Crama (2004).

[Brucker, 1998] Brucker, P. (1998). *Scheduling Algorithms.* Springer-Verlag, Berlin.

[Carlier, 1982] Carlier, J. (1982). The one machine sequencing problem. *European Journal of Operational Research*, 11 :42–47.

[Carlier and Chretienne, 1988] Carlier, J. and Chretienne, P. (1988). Problèmes d'ordonnancement, modélisation, complexité, algorithmes. *Masson.*

[Chang, 1999] Chang, P. (1999). A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers and Mathematics with Applications*, 37 :133–144.

[Chrétienne and Sourd, 2003] Chrétienne, P. and Sourd, F. (2003). PERT scheduling with convex cost functions. *Theoretical Computer Science*, 292 :145–164.

[Chrétienne and Sourd, 2002] Chrétienne, P. and Sourd, F. (2002). Scheduling with convex cost functions. *Theoretical Computer Science, to appear.*

[Davis and Kanet, 1993a] Davis, J. and Kanet, J. (1993a). Single machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40 :85–101.

[Davis and Kanet, 1993b] Davis, J. S. and Kanet, J. J. (1993b). Single machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40 :85–101.

[Durden, 1999] Durden, C.-H. (1999). Cost accounting and performance measurement in a just-in-time production environment. *Asia Pacific Journal of Management*, 16 :111–125.

[Ehrgott, 2000] Ehrgott, M. (2000). *Multicriteria Optimization.* Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.

[Estève et al., 2004] Estève, B., Aubijoux, C., Chartier, A., and T'kindt, V. (2004). A recovering beam search algorithm for the single machine just-in-time scheduling problem. *European Journal of Operational Research*, 39 :27.

[Estève et al., 2001] Estève, B., Aubijoux, C., Chartier, A., and T'Kindt, V. (2001). Résolution heuristique d'un problème d'ordonnancement multicritère à une machine de type juste-à-temps. *Quatrième conférence de la Société Francaise de Recherche Opérationnelle et d'Aide à la Décision*, ROADEF'02, Paris.

[Estève et al., 2003a] Estève, B., Aubijoux, C., Chartier, A., and T'kindt, V. (2003a). A recovering beam search algorithm for the single machine just-in-time scheduling problem. In *Sixth Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'03)*, pages 125–126, Aussois, France.

[Estève et al., 2005] Estève, B., Aubijoux, C., Chartier, A., and T'kindt, V. (2005). A recovering beam search algorithm for the single machine just-in-time scheduling problem. *European Journal of Operational Research*, in press :xx–xx.

[Estève et al., 2003b] Estève, B., Chopin, A., Mirault, A., and T'kindt, V. (2003b). Just-in-time scheduling of a 2-machine flowshop problem with lot-streaming. In *1st Multi-disciplinary International Conference on Scheduling : Theory and Applications (MISTA 2003)*, pages 473–474, Nottingham (UK).

[Estève et al., 2004] Estève, B., Sorel, E., Yahi, N., and T'kindt, V. (2004). A recovering beam search algorithm for the just-in-time scheduling of a two-machine problem with lot-streaming. In *9th International Workshop on Project Management and Scheduling (PMS'2004)*, pages 363–366, Nancy, France.

[Fry et al., 1996] Fry, T. D., Armstrong, R. D., Darby-Dowman, K., and Philipoom, P. R. (1996). A branch and bound procedure to minimize mean absolute lateness on a single processor. *Computers and Operations Research*, 23(2) :171–182.

[Fry and Leong, 1986] Fry, T. D. and Leong, G. K. (1986). Bi-criterion single-machine scheduling with forbidden early shipments. *Engineering Costs and Production Science*, 10(2) :133–137.

[Fry et al., 1987] Fry, T. D., Leong, G. K., and Rakes, T. R. (1987). Single machine scheduling : a comparison of two solution procedures. *Omega*, 15(4) :277–282.

[Fullerton and McWatters, 2001] Fullerton, R.-R. and McWatters, C.-S. (2001). The production performance benefits from jit implementation.

[Garey et al., 1988] Garey, M. R., Tarjan, R. E., and Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13(2) :330–348.

[Golhar and Stamm, 1991] Golhar, D.-Y. and Stamm, C.-L. (1991). The just-in-time philosophy : a litterature review. *International Journal on Production Research*, 29-4 :657–676.

[Gordon et al., 2002a] Gordon, V., Proth, J.-M., and Chu, C. (2002a). Due date assignment and scheduling : SLK, TWK and other due date assignement models. *Production Planning and Control*, 13(2).

[Gordon et al., 2002b] Gordon, V., Proth, J.-M., and Chu, C. (2002b). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139(1).

[Gordon et al., 2004] Gordon, V., Proth, J.-M., and Strusevich, V. (2004). Scheduling with due-date assignment. In [Leung, 2004], chapter 21.

[Gupta and Sen, 1983] Gupta, S. K. and Sen, T. (1983). Minimizing a quadratic function of job lateness on a single machine. *Engineering costs of Production Economic*, 7(3) :187–194.

[Hall and Posner, 1991] Hall and Posner (1991).

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems.* MIT press, Cambridge, Mass.

[Huson and Nanda, 1995] Huson, M. and Nanda, D. (1995). The impact of just-in-time manufacturing on firm performance in the u.s. *Journal of Operations management*, 12 :297–310.

[James and Buchanan, 1997] James, R. J. W. and Buchanan, J. T. (1997). A neighbourhood scheme with a compressed solution space for the early/tardy scheduling problem. *European Journal of Operational Research*, 102 :513–527.

[James and Buchanan, 1998] James, R. J. W. and Buchanan, J. T. (1998). Performance enhancements to tabu search for the early/tardy scheduling problem. *European Journal of Operational Research*, 106 :254–265.

[Kaminsky and Hochbaum, 2004] Kaminsky, P. and Hochbaum, D. (2004). Due-date quotation models and algorithms. In [Leung, 2004], chapter 20.

[Kim and Yano, 1994] Kim, Y.-D. and Yano, C. A. (1994). Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics*, 41 :913–933.

[Klein and Hannan, 1982] Klein, D. and Hannan, E. (1982). An algorithm for the multiple objective integer linear programming problem. *European Journal of Operational Research*, 9 :378–385.

[Kubiak, 2004] Kubiak (2004).

[Lea and Min, 2003] Lea, B.-R. and Min, H. (2003). Selection of management accounting systems in just-in-time and theory of constraints based manufacturing. *International Journal of Production Researchs*, 42-13 :2879–2910.

[Lee and Choi, 1995] Lee, C. and Choi, J. (1995). A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers and Operations Research*, 22.

[Leung, 2004] Leung, J.-T., editor (2004). *Handbook of Scheduling : Algorithms, Models and Performance Analysis.* Chapman & Hall/CRC Computer and Information Science serie, Volume 1.

[Li, 1997] Li, G. (1997). Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, 96 :546–558.

[Liaw, 1999] Liaw, C. F. (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers and Operations Research*, 26 :679–693.

[Mazzini and Armentano, 2001] Mazzini, R. and Armentano, A. (2001). Heuristic for single machine scheduling with early and tardy costs. *European Journal of Operational Research*, 128 :129–146.

[Molet, 1989] Molet, H. (1989). La nouvelle gestion de production. *Hermes*, page 62pp.

[Monden, 1983] Monden, Y. (1983). Toyota production system. *Institute of Industrial Engineers Atlanta.*

[Nishi et al., 2000] Nishi, T., Sakata, A., Hasebe, S., and Hashimoto, I. (2000). Autonomous decentralized scheduling system for just-in-time production. *Computers and Chemical Engineering*, 24 :345–351.

[Nollet et al., 1994] Nollet, J., Kélada, J., and Diorio, M. (1994). La gestion des opérations et de la production : une approche systémique. *Gaetan Morin Editeur (in French)*, page 682.

[Ouenniche and Boctor, 2001] Ouenniche, J. and Boctor, F. (2001). The two-group heuristic to solve the multi-product, economic lot sizing and scheduling problem in flow shops. *European Journal of Operational Research*, 129(3) :539–554.

[Ow and Morton, 1988] Ow, P. S. and Morton, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26(1) :35–62.

[Ow and Morton, 1989] Ow, P. S. and Morton, T. E. (1989). The single machine early/tardy problem. *Management Science*, 35(2) :177–190.

[Portmann and Vignier, 2001] Portmann, M. C. and Vignier, A. (2001). Algorithmes génétiques et ordonnancement. *Ordonnancement de la production, Hermès science, P. Lopez et F. Roubellat editeurs*, pages 95–130.

[Sale and Inman, 2003] Sale, M. L. and Inman, A. (2003). Survey-based comparison and change in performance of firms using traditional manufacturing, jit and toc. *International Journal of Production Research*, 41(4) :829–844.

[Schonberger, 1986] Schonberger, R.-J. (1986). World class manufacturing. *New York The Free Press.*

[Soland, 1979] Soland, R. (1979). Multicriteria optimisation : a general characterization of efficient solutions. *Decision sciences*, 10 :27–38.

[Sourd, 2004] Sourd, F. (2004). The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions. *INFORMS Journal of Computing*, 16 :198–208.

[Sourd, 2005] Sourd, F. (2005). Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, 165 :82–96.

[Sourd and Kedad-Sidhoum, 2003] Sourd, F. and Kedad-Sidhoum, S. (2003). The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling*, 6(6) :533–549.

[Sridharan and Zhou, 1996] Sridharan, V. and Zhou, Z. (1996). A decision theory based scheduling procedure for single-machine weighted earliness and tardiness problems. *European Journal of Operational Research*, 94 :292–301.

[Sundararaghavan and Ahmed, 1984] Sundararaghavan, P. S. and Ahmed, M. U. (1984). Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. *Naval Research Logistics Quarterly*, 31(2) :325–333.

[Szwarc and Mukhopadhyay, 1994] Szwarc, W. and Mukhopadhyay, S. (1994). Optimal timing schedules in earliness-tardiness single machine sequencing. *Naval Research Logistics*, 42 :1109–1114.

[Tadei, 2002] Tadei, F. D. M. G. R. (2002). Recovering beam search : a hybrid heuristic method for combinatorial optimisation problems. *Research Report, Politecnico di Torino (Italy).*

[T'kindt and Billaut, 2002] T'kindt, V. and Billaut, J. (2002). Multicriteria scheduling : Theory, models and algorithms. *Springer Verlag.*

[Yano and Kim, 1991] Yano, C. A. and Kim, Y. D. (1991). Algorithms for a class of single machine weighted tardiness and earliness problems. *European Journal of Operational Research*, 52 :167–178.

[Yoon and Ventura, 2002] Yoon, S.-H. and Ventura, J. (2002). Minimizing the mean weighted absolute deviation from due dates in lot-streaming flow shop scheduling. *Computers & Operations Research*, 29 :1301–1315.

[Zegordi et al., 1995] Zegordi, S. H., Itoh, K., and Enkawa, T. (1995). A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. *International Journal of Production Research*, 33(5) :1449–1466.