Utilisation d'ordres partiels pour la caractérisation de solution robustes en ordonnancement

Hoang Trung La

Thèse

Présentée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du grade de
Docteur de l'Institut National des Sciences Appliquées de Toulouse

Spécialité : **Systèmes Industriels**

par

LA Hoang Trung
Ingénieur en Informatique de l'Institut Polytechnique de Hanoi

---

UTILISATION D'ORDRES PARTIELS POUR LA CARACTÉRISATION DE SOLUTIONS ROBUSTES EN ORDONNANCEMENT

---

soutenue le 24 Janvier 2005 devant le jury :

Rapporteurs :	Jean-Charles BILLAUT	Professeur de l'Université de Tours
	Willy HERROELEN	Professeur de l'Université Catholique de Leuven
Examinateurs :	Marie-Claude PORTMANN	Professeur de l'INPL, Nancy
	Philippe BAPTISTE	Chargé de recherche CNRS
Directeurs de thèse :	Cyril BRIAND	Maître de Conférences de l'UPS, Toulouse
	Jacques ERSCHLER	Professeur de l'INSA, Toulouse
Membre invité :	François ROUBELLAT	Directeur de recherche CNRS

Cette thèse a été préparée au Laboratoire d'Analyse et d'Architecture des Systèmes du C.N.R.S
7 avenue du Colonel Roche, 31077 TOULOUSE cedex 4

Remerciements

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Analyse et d'Architecture des Systèmes (L.A.A.S) du C.N.R.S et, à ce titre, je tiens à remercier Messieurs Jean-Claude Laprie et Malik Ghallab, directeurs successifs du L.A.A.S de m'avoir accueilli dans ce laboratoire.

Je remercie également Messieurs Robert Valette, Directeur de Recherche et Pierre Lopez, Chargé de Recherche au C.N.R.S, de m'avoir accueilli respectivement au sein des groupes " Organisation et Conduite de Systèmes Discrets " et " Modélisation, Optimisation et Gestion Intégrée de Systèmes d'Activités " dont ils ont eu la responsabilité.

Ce travail doit beaucoup à mes deux directeurs de thèse, Monsieur Cyril Briand, Maître de Conférences à l'Université Paul Sabatier et Monsieur Jacques Erschler, Professeur à l'Institut National des Sciences Appliquées de Toulouse. Je tiens à exprimer toute ma profonde gratitude à Cyril Briand pour m'avoir accordé sa confiance dès notre première contact, pour m'avoir aimablement assisté durant les périodes de recherche de bourse et de demande de dispense de D.E.A, et finalement pour m'avoir encadré pendant ces trois années de thèse. Sa compétence, sa disponibilité, sa sympathie et sa patience (surtout lors de la phase de rédaction tellement difficile pour un non francophone comme moi) m'ont permis de mener à bien ce travail et je l'en remercie sincèrement. J'exprime également ma reconnaissance à Jacques Erschler pour ses encouragements, pour ses conseils pertinents et pour ses remarques très précieuses. Je souhaite ici leur exprimer toute mon amitié et tous mes meilleurs vœux pour leur avenir.

Je remercie chaleureusement Messieurs Jean-Charles Billaut, Professeur à l'Université de Tours et Willy Herroelen, Professeur à l'Université Catholique de Leuven, pour avoir accepté d'être rapporteurs de mon manuscrit de thèse, pour toutes les remarques constructives qu'ils ont émises et pour avoir accepté de participer à mon jury de soutenance.

Je remercie aussi vivement Madame Marie-Claude Portmann, Professeur à l'École de Mines de Nancy pour sa participation au jury et pour l'honneur qu'elle m'a fait de présider mon jury.

Je remercie sincèrement Messieurs Philippe Baptiste, Chargé de Recherche au C.N.R.S

et François Roubellat, Directeur de Recherche au C.N.R.S pour avoir accepté de participer à mon jury de thèse, pour l'attention et l'intérêt qu'ils ont bien voulu porter à mon travail, et surtout pour leur lecture très attentive du manuscrit.

Je tiens à remercier chaleureusement mes collègues du groupe MOGISA, pour l'ambiance de travail très agréable qu'ils créent. Je remercie en particulier Emmanuelle Despontin-Monsarrat, ma collègue de bureau, pour m'avoir assisté dès mon arrivé à Toulouse, pour m'avoir aidé à comprendre le mode de vie français, le fonctionnement du L.A.A.S, pour m'avoir guidé au maniement de Latex lors de la rédaction de thèse, et pour tous les mots et expressions courantes du français qu'elle m'a expliqué et que j'ai appris jour après jour. Je remercie également les permanents : Marie-José Huguet, Marcel Mongeau, Colette Mercé, Gérard Fontan, Patrick Esquirol, Jean-Claude Hennet pour leur amitié et leurs remarques constructives sur ma présentation de soutenance.

Un grand merci à Jean-Luc Santamaria, ex-stagiaire de DEA au L.A.A.S, qui a si parfaitement développé l'interface homme-machine, utilisée pour mettre en valeur certains résultats obtenus dans le cadre de ma recherche.

Mes remerciements vont également aux personnels administratifs et techniques au L.A.A.S, en particulier à Eliane Dufour pour l'organisation consciencieuse de ma soutenance, et à Christian Berty pour la reproduction de ma thèse.

Hors du L.A.A.S, je tiens à remercier tous mes amis vietnamiens à Toulouse, pour leurs encouragements, leur amitié, leur aide, ... qui ont rendu inoubliable mon séjour en France.

Pour finir, une spéciale dédicace à toi, Hang, pour ce que nous avons partagé, les joies, les moments difficiles et pour les jours à venir ...

Je veux rendre un hommage du fond du cœur à mes parents, qui m'ont assisté, m'ont encouragé dans toute circonstance, dans tous les instants de ma vie. Qu'ils veuillent trouver ici le témoignage de mon affection.

Table des matières

Table des figures

*

Liste des tableaux

INTRODUCTION

L'ordonnancement joue un rôle essentiel dans de nombreux secteurs d'activités : la conception (de bâtiments, de produits, de systèmes, ...), l'administration (gestion d'emplois du temps, gestion du personnel), l'industrie (gestion de production), l'informatique (ordonnancement de processus, ordonnancement de réseaux). Il s'agit généralement d'organiser dans le temps l'exécution de tâches soumises à des contraintes de temps et de ressources, tout en satisfaisant au mieux un ou plusieurs objectifs. Les méthodes d'ordonnancement foisonnent dans la littérature. Elles se différencient par la nature du problème considéré (nombre de ressources, structure particulière du problème, ...), la nature des contraintes prises en compte, les objectifs à satisfaire (minimisation des coûts, de la durée totale de mise en œuvre, ...) et la nature de l'approche de résolution adoptée (heuristiques, méthodes exactes, métaheuristiques, approches par contraintes, ...).

La plupart de ces méthodes opèrent sous l'hypothèse classique que les données du problème d'ordonnancement sont parfaitement connues : l'ensemble des tâches à ordonnancer est fini et déterminé a priori, les durées opératoires, les dates de début au plus tôt et de fin au plus tard des tâches sont fixées et stables et les capacités des ressources sont connues à tout instant. Cette hypothèse permet en effet de traiter un problème de façon déterministe, ce qui simplifie considérablement la résolution, celle-ci demeurant toutefois ardue dans la plupart des cas.

Une tendance, de plus en plus affirmée lorsqu'on analyse la littérature scientifique ces dix dernières années, concerne la remise en cause de cette hypothèse. En effet, s'il est indiscutable que le développement de nouvelles méthodes d'ordonnancement déterministe, de plus en plus efficaces, est théoriquement une avancée indéniable, il est difficile d'omettre que les ordonnancements produits sont avant tout destinés à être mis en œuvre dans un environnement dont la caractéristique principale est d'être incertain, mais surtout pas déterministe. En gestion de production par exemple, à quoi bon disposer d'un plan de production optimal à 8:00, si celui-ci s'avère inutilisable à 8:05 parce qu'un ouvrier est absent, qu'une machine est en panne ou qu'un composant indispensable est manquant ? Il serait évidemment possible, pour pallier ce problème, de recalculer un nouveau plan optimal, à chaque fois qu'un événement imprévu survient (dans la mesure où le temps de calcul nécessaire à sa détermination serait faible relativement à la dynamique du système). Toutefois, il en résulterait une organisation fortement instable, peu propice à une organisation

efficace du travail (notamment lorsque les ressources sont des hommes), ne permettant pas de garantir une performance sûre.

Ce constat a justifié l'émergence d'une nouvelle thématique de recherche appelée *ordonnancement robuste*. Il s'agit toujours d'organiser dans le temps l'exécution de tâches soumises à des contraintes de temps et de ressources, tout en satisfaisant au mieux un ou plusieurs objectifs, mais en s'interdisant cette fois de négliger le caractère perturbé de l'environnement de mise en œuvre. La solution produite doit alors non seulement résister aux perturbations de l'environnement, c'est à dire être toujours utilisable en cas d'imprévus, mais aussi posséder une qualité stable face à ces perturbations. Comme nous le verrons dans ce mémoire, les approches d'ordonnancement robuste se distinguent notamment selon la nature des incertitudes considérées (et la façon de les modéliser), selon qu'elles manipulent ou non un ordonnancement de référence, servant de guide tout au long de la mise en œuvre, et selon qu'elles tentent d'anticiper ou non les éventuelles perturbations.

Ce mémoire s'intéresse à une approche de l'ordonnancement robuste, de type proactif-réactif, privilégiant l'anticipation des perturbations lors de la phase de construction de l'ordonnancement de référence, afin de faciliter les décisions à prendre lors de la phase de mise en œuvre, en réaction aux perturbations issues de l'environnement. Dans notre cas, l'anticipation des perturbations consiste à déterminer un ensemble flexible de solutions de performance connue. La flexibilité est séquentielle, c'est à dire que l'ordre des tâches sur chaque ressource est différent d'une solution à l'autre de l'ensemble. Ce type de flexibilité est intéressant car il permet, lorsqu'une ressource se libère, de disposer de plusieurs choix quant à la prochaine tâche à traiter. Si suite à un aléa, une tâche ne peut commencer, par exemple parce qu'un composant requis est indisponible, ou parce qu'une autre tâche devant la précéder est en retard, il est alors possible de ne pas laisser une ressource oisive en choisissant de réaliser une autre tâche, tout en garantissant le même niveau de performance. De plus, la flexibilité séquentielle induisant une flexibilité temporelle, il est également possible d'absorber les retards liés à la réalisation de certaines tâches, sans détérioration de performance, au prix cependant d'une perte de flexibilité séquentielle.

La détermination d'un ensemble flexible de solutions possédant, à la fois, une performance au pire la meilleure possible, et offrant une flexibilité séquentielle la plus importante possible, est un problème difficile, devant être abordé dans l'optique d'une recherche de compromis flexibilité / performance. En effet, plus un ensemble de solutions est flexible et moins sa performance est bonne en général, et vice-versa. Ce mémoire présente quelques approches permettant d'aborder cette recherche de compromis par utilisation de conditions dominantes ou suffisantes, vis-à-vis de l'optimalité de solutions en ordonnancement. Une caractéristique commune de ces approches est de se livrer à une analyse préalable de structures d'intervalles, ces structures étant spécifiques au type de problème considéré, dans le but de définir des ordres partiels dominants ou suffisants vis-à-vis de l'optimalité. Une propriété intéressante de ces ordres partiels est qu'ils permettent, de par leur structure, un calcul aisé de la performance de l'ensemble des solutions qu'ils caractérisent. De plus, ils

autorisent, sous certaines conditions, de traiter le cas où des intervalles de réalisation sont associés aux durées, aux dates de début au plus tôt et de fin au plus tard des tâches. Une performance au pire peut alors être calculée, robuste vis-à-vis de l'ensemble des scénarios de réalisation de l'ordonnancement caractérisé grâce au modèle par intervalles.

Le mémoire de thèse est organisé de la manière suivante.

La première partie décrit le contexte de notre étude ainsi que quelques notions nécessaires à sa compréhension. Dans un premier temps quelques notions d'ordonnancement sont rappelées et la problématique de l'ordonnancement robuste est présentée. Le deuxième chapitre est consacré à l'analyse des méthodes d'ordonnancement robuste proposées dans la littérature et précise davantage les motivations de notre étude. Dans le chapitre suivant, la notion d'ordre partiel nécessaire, suffisant ou dominant est présentée, ainsi que quelques concepts relatifs à l'analyse de structures d'intervalles.

La deuxième partie se focalise sur l'étude du problème à une machine. Dans un premier temps, un théorème de dominance démontré dans les années quatre-vingts est présenté. Nous montrons alors comment déterminer les performances au mieux et au pire de l'ensemble de séquences dominantes que ce théorème caractérise. Puis, nous présentons comment ce théorème peut être utilisé afin de considérer un ensemble de scénarios potentiels de réalisation d'un ordonnancement, sur lequel une performance robuste peut être déterminée. Dans un deuxième temps, une procédure par séparation et évaluation pour l'ordonnancement robuste est proposée, appliquée au problème déterministe à une machine. Cette procédure permet de caractériser toutes les solutions optimales contenues dans l'ensemble dominant initial, défini par le théorème précédent, par une énumération de structures d'intervalles. Une expérimentation est également présentée mettant en évidence les performances de cette procédure. Le chapitre 6 décrit deux approches d'aide à la décision pour l'ordonnancement robuste qui utilisent les résultats des chapitres 4 et 5. Il s'agit de guider un décideur pour la détermination d'un compromis flexibilité / performance acceptable.

La dernière partie s'intéresse à des problèmes à plusieurs machines. Le septième chapitre s'intéresse tout d'abord au problème de flow shop de permutation à deux machines pour lequel un ordre partiel suffisant est proposé. Cet ordre partiel permet de caractériser un large ensemble de solutions optimales, cet ensemble restant optimal pour de nombreux scénarios potentiels de réalisation de l'ordonnancement. Enfin, le dernier chapitre décrit quelques pistes pour résoudre, de façon robuste, le problème de job shop. Un résultat pour le cas du job shop à deux machines est tout d'abord donné. Puis, le problème de job shop est étudié de façon générale, et deux algorithmes sont proposés pour le résoudre.

Première partie

NOTIONS PRÉLIMINAIRES ET CONTEXTE DE L'ÉTUDE

Chapitre 1

Notions préliminaires en ordonnancement

Après le rappel de quelques notions centrales en ordonnancement, ce chapitre s'intéresse à la gestion des incertitudes en ordonnancement et en particulier, à la problématique de l'ordonnancement robuste. Il décrit plus précisément les sources et les types d'incertitudes à prendre en compte et comment celles-ci sont généralement modélisées, puis gérées.

1.1 Quelques rappels d'ordonnancement

1.1.1 Définition

En gestion de production comme en gestion de projet, l'ordonnancement joue un rôle privilégié, s'inscrivant dans des niveaux de décision à la fois tactique et opérationnel [Esquirol & Lopez 99, Giard 91]. Il s'agit de contrôler, à court ou moyen terme, l'activité d'un ensemble de ressources disponibles en quantité limitée, en gérant les conflits d'utilisation que pose la réalisation d'un ensemble d'activités sur un horizon de temps généralement imposé.

Plusieurs définitions d'un problème d'ordonnancement sont données dans la littérature, nous indiquons ici celle proposée dans [Esquirol & Lopez 99] :

« Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, ...) et de contraintes portant sur l'utilisation et la disponibilité de ressources requises. »

Le mot "ordonnancement" désigne soit une solution au problème d'ordonnancement, soit par abus de langage, le processus ayant conduit à la détermination de cette solution. Une solution d'ordonnancement décrit les dates prévues pour l'exécution des tâches et l'allocation des ressources au cours du temps. La méthode utilisée pour l'élaboration d'un

ordonnancement vise souvent à satisfaire un ou plusieurs objectifs (coûts, délais, qualité), exprimés au sein d'un ou plusieurs critères de performance. Les paragraphes suivants précisent ces notions de tâche, ressource, objectif, ...et introduisent quelques notations.

Pour une description plus détaillée de la problématique de l'ordonnancement, le lecteur est prié de se référer aux ouvrages [Coffman 76], [Blazewicz *et al.* 96] ou [Pinedo 95].

1.1.2 La notion de tâche

Une tâche i, appartenant à un ensemble T fini de tâches à ordonnancer, est une entité élémentaire de travail caractérisée par une date de début au plus tôt r_i, une date de fin au plus tard d_i et une durée opératoire p_i. Sa localisation dans le temps est définie par une date de début $s_i \geq r_i$ et/ou une date de fin $f_i \leq d_i$. Dans le cas où la durée opératoire $p_i = f_i - s_i$ est connue, la valeur de la variable f_i peut être déduite de la valeur de s_i (et vice-versa). La réalisation de la tâche i nécessite l'utilisation d'une ou plusieurs ressources $k \in R$, où R est l'ensemble de ressources. Les ressources concernées par la réalisation de la tâche sont généralement supposées connues a priori. Parfois, les ressources sont à choisir dans des ensembles de ressources (ou *pools* de ressources), chaque ensemble correspondant à une compétence particulière. L'intensité de ressource k consommée pour la réalisation de i est notée q_{ik} (supposée constante lors de l'exécution de la tâche). Si les tâches peuvent être exécutées par morceaux, alors le problème est dit *préemptif*. Dans ce cas, la durée p_i d'une tâche ne peut être déterminée qu'une fois l'ordonnancement construit.

Dans le cas de l'ordonnancement d'atelier, notons que le terme *opération* remplace parfois celui de tâche. Dans le domaine de la gestion de projet, c'est le terme *activité* qui est préféré.

1.1.3 La notion de ressource

Une ressource k est un moyen technique ou humain, disponible en quantité limitée, destiné à être utilisé pour la réalisation de plusieurs tâches. La disponibilité est généralement exprimée par une *capacité* propre à chaque ressource k, notée Q_k ($Q_k \geq 1$). Une ressource est dite renouvelable si, après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les hommes, les machines ...). Dans le cas contraire, elle est dite consommable (matières premières, budget ...). Dans certains problèmes d'ordonnancement, la notion d'*état de ressource* est également utilisée afin de prendre en compte des contraintes technologiques liées à son utilisation. Dans ce cas, la capacité de la ressource dépend de l'état considéré. Un changement d'état de la ressource peut être produit par l'occurrence d'événements externes incontrôlables (panne, maintenance, ...) ou par le début ou la fin de certaines tâches du problème.

Remarquons que dans le cas de l'ordonnancement d'atelier, le terme *machine* est généralement utilisé à la place de ressource.

1.1.4 Contraintes de temps et de ressources

La notion générique de contrainte est relative à un ensemble de variables de décision. Elle exprime une restriction sur les domaines de valeur de ces variables. Il y a deux sortes de variables en ordonnancement. Certaines variables concernent les décisions de localisation des tâches dans le temps (variables d'ordonnancement s_i et f_i) ; d'autres concernent les décisions d'affectation des tâches sur les ressources (variables d'affectation). Dans le cadre de cette thèse, nous nous intéressons seulement aux variables d'ordonnancement et aux contraintes qui les caractérisent : les *contraintes temporelles* et les *contraintes de ressources*.

Les contraintes temporelles permettent d'exprimer les interdépendances temporelles entre tâches. Dans la pratique, il s'agit de prendre en compte les aspects technologiques ou logistiques qui caractérisent l'organisation d'une production ou d'un projet. Ces contraintes sont classiquement formalisées sous la forme d'une inégalité de potentiels de type $t_j - t_i \geq \delta_{ij}$ où t_j et t_i sont deux variables d'ordonnancement distinctes, correspondant à des débuts ou fins de tâches, et δ_{ij} est une constante entière. Ce type de contrainte est aussi appelé *contrainte de précédence généralisée* et permet d'exprimer des écarts temporels minimaux ou maximaux entre les variables. Les inéquations, $s_i \geq r_i$ et $f_i \leq d_i$, sont des contraintes particulières de ce type.

Les contraintes de ressources permettent de représenter à la fois, les caractéristiques de consommation des tâches sur les ressources et les caractéristiques de disponibilité des ressources. L'ordonnancement produit doit assurer que la consommation des tâches sur les ressources est en cohérence, à tout instant, avec leurs disponibilités.

Remarquons que les contraintes de ressources peuvent être traduites en contraintes temporelles grâce à la notion d'*ensemble critique* de tâches [Nabeshima 73]. Un ensemble critique de tâches I_c est un ensemble minimal de tâches non réalisables simultanément. Il est défini par :

$$\exists k \in R \text{ tel que } \sum_{i \in I_c} q_{ik} > Q_k \text{ et } \nexists (k', I) \text{ avec } k' \in R, I \subset I_c \text{ tels que } \sum_{i \in I} q_{ik'} > Q_{k'}$$

Ces ensembles étant minimaux, une condition nécessaire et suffisante pour respecter les contraintes de ressources est d'assurer que, dans chaque ensemble critique, il existe au moins deux tâches qui s'exécutent sur des intervalles de temps disjoints. Ceci peut s'exprimer en disant qu'au moins une inégalité de potentiels parmi celles d'un ensemble $H_{I_c} = \{s_j - s_i \geq p_i | i \neq j \text{ et } (i,j) \in I_c\}$ doit être satisfaite. Dans le cas particulier où les

ressources sont disjonctives ($Q_k = 1$), tous les ensembles critiques sont constitués de deux tâches, on parle alors de *paires de disjonction*.

1.1.5 La notion d'objectif et de critère

Selon le domaine d'application, la fonction ordonnancement peut avoir d'autres objectifs que celui de veiller au simple respect des contraintes de temps et de ressource. Il peut s'agir de satisfaire des objectifs économiques, de respecter une législation du travail en vigueur dans une entreprise ou, comme dans notre cas, de gérer au mieux le risque en présence d'incertitudes.

Lorsque ces objectifs sont quantifiables et exprimables en fonction des variables d'ordonnancement, ils sont injectés dans le problème d'ordonnancement, soit par ajout de nouvelles contraintes, soit par ajout d'un ou plusieurs critères d'optimisation. Dans le premier cas, la fonction ordonnancement doit produire une solution *admissible*, c'est-à-dire satisfaisant l'ensemble des contraintes formulées. Dans le second cas, la solution produite doit non seulement être admissible, mais aussi minimiser ou maximiser la valeur du ou des critères considérés. Ce problème peut être représenté par un programme linéaire en nombres entiers, ainsi que l'illustre l'exemple suivant (où Z est un critère à minimiser quelconque et où seules les contraintes temporelles sont prises en compte) :

$$\left.\begin{array}{lll} \text{Minimiser} & Z & \\ \text{Sous contraintes} & s_i \geq r_i & i \in T \\ & s_j - s_i \geq \delta_{ij} & < i,j > \in T \times T \end{array}\right\}$$

Pour les approches d'optimisation, les critères d'évaluation numériques peuvent être :

- soit liés au temps, comme dans le cas de la minimisation du temps total d'exécution ($C_{\max}$) ou des retards vis-à-vis de dates d'échéance fixées ($L_{\max}$ ou $T_{\max}$) ;
- soit liés aux ressources, comme dans le cas de la minimisation de la quantité de ressources nécessaires à la réalisation des tâches ou la charge de chaque ressource ;
- soit liés aux coûts (de lancement, de production, de transport, etc.) qu'un ordonnancement induit.

Un critère est dit *régulier* si l'on ne peut pas le dégrader en avançant l'exécution d'une tâche. Les critères liés au temps, $C_{\max}$ ou $L_{\max}$, sont par exemple réguliers, à la différence de la plupart des critères liés aux coûts ou aux ressources.

1.1.6 Modélisation par un graphe potentiels-tâches

La manière la plus naturelle de modéliser les contraintes d'un problème d'ordonnancement est d'utiliser un *graphe potentiels-tâches* [Roy 70]. Ce graphe est noté $G(X, U)$ où

chaque sommet $x \in X$ correspond à une tâche à ordonnancer et où un arc $u = (i, j) \in U$, de longueur δ_{ij}, correspond à une inégalité de potentiels entre les deux sommets i et j ($t_j - t_i \geq \delta_{ij}$). Le potentiel t_i d'un sommet i est généralement associé à la date de début de la tâche correspondante, ce qui permet, lorsque les durées des tâches sont connues, de représenter n'importe quel type d'inégalité de potentiels. Des *tâches fictives* sont parfois utilisées, en particulier pour représenter l'origine des temps, ou la fin de l'ordonnancement.

Une condition nécessaire et suffisante, bien connue, pour que les contraintes temporelles soient cohérentes (c'est-à-dire pour qu'une solution existe) est que le graphe conjonctif ne comporte aucun circuit de longueur positive [Bartush *et al.* 88].

Pour représenter les ensembles non-conjonctifs d'inégalités de potentiels (cf. 1.1.4), liés à la prise en compte de contraintes de ressource, il est nécessaire d'utiliser un graphe non-conjonctif $G(X, U, F)$, où F est l'ensemble des cliques non-conjonctives d'arcs associées à chaque ensemble H_{I_C}. Deux arcs $u \in U$ et $v \in F$, de longueur respective δ_u et δ_v, peuvent relier deux mêmes sommets i et j. Dans ce cas, si $\delta_u \geq \delta_v$, alors la clique non-conjonctive contenant v peut être éliminée de F. Sinon, les deux arcs doivent être conservés.

À titre d'exemple, considérons un problème caractérisé par cinq tâches et deux ressources A et B, respectivement disponibles en quantité $Q_A = 4$ et $Q_B = 1$. Les durées opératoires et les consommations de ressources sont indiquées dans la table 1.1. Les contraintes temporelles sont les suivantes :

$$\begin{cases} s_1 & \geq & 0 \\ s_2 & \geq & 0 \\ s_3 & \geq & 0 \\ s_4 - s_1 & \geq & 0 \\ s_5 - s_1 & \geq & 0 \\ s_4 - f_2 & = & 0 \\ f_5 - s_3 & \geq & 0 \\ -f_4 & \geq & -20 \\ -f_5 & \geq & -20 \end{cases}$$

Tâche i	1	2	3	4	5
p_i	10	2	3	5	6
q_{iA}	3	1	1	0	0
q_{iB}	0	0	0	1	1

TAB. 1.1: Durées opératoires et consommation de ressources

Le graphe non-conjonctif associé à ce problème est représenté sur la figure 1.1. Ce graphe comporte six sommets, $X = \{0, 1, 2, 3, 4, 5\}$, où 0 est un sommet fictif représentant

l'origine des temps. Les arcs correspondant à la partie conjonctive U sont indiqués en trait plein sur la figure. La partie non-conjonctive F du graphe est déduite de l'existence de deux ensembles critiques minimaux : $I_{c1} = \{1, 2, 3\}$ (relatif à l'utilisation de la ressource A) et $I_{c2} = \{4, 5\}$ (relatif à l'utilisation de la ressource B). F comporte donc deux cliques non-conjonctives d'arcs, F_1 et F_2, représentées en pointillés sur la figure. Déterminer une solution admissible consiste alors à sélectionner un arc dans chacune des deux cliques, F_1 et F_2, de sorte que leur ajout dans la partie conjonctive U ne crée pas de circuit de longueur positive.

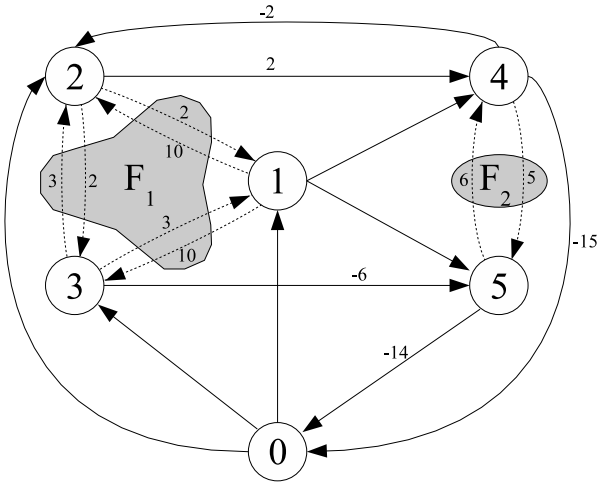


FIG. 1.1: Un exemple de graphe potentiels-tâches non-conjonctif

1.1.7 Classifications des problèmes d'ordonnancement

Selon la nature des variables mises en jeu, la nature des contraintes, ou encore la structure particulière du graphe non-conjonctif associé à un problème, plusieurs classifications des problèmes d'ordonnancement sont proposées dans la littérature.

Dans le domaine de l'ordonnancement d'atelier, la notion structurante de *gamme opératoire* est souvent utilisée comme critère de classification. En effet, une gamme opératoire impose un ordre de passage des produits sur les machines, et donc un ordre des opérations associées à chaque travail. De ce fait, le graphe potentiels-tâches non-conjonctif associé au problème possède parfois une structure particulière, qu'il est possible d'exploiter pour élaborer des méthodes d'ordonnancement efficaces. Dans [MacCathy & Liu 93], sept types de problèmes d'ordonnancement d'atelier sont ainsi distingués :

- le problème *job shop* où chaque travail a sa gamme opératoire propre ;
- le problème *flow shop* où chaque travail a une gamme identique ;

- le problème *open shop* où l'ordre de passage sur les machines est libre pour chaque travail ;
- le problème *flow shop de permutation* où chaque travail à une gamme identique et où l'ordre de passage des travaux est le même pour chaque machine ;
- le problème *à une machine* où chaque travail est assimilé à une opération unique, exécutée par une seule et même machine m de capacité $Q_m = 1$;
- le problème à *machines parallèles* où chaque travail est assimilé à une opération unique, exécutée par une machine à sélectionner dans un ensemble ;
- le problème *job shop à machines dupliquées* où chaque travail a sa gamme opératoire propre et où chaque opération est réalisée par une machine à sélectionner dans un ensemble.

On peut synthétiser et généraliser cette typologie ainsi que l'illustre la figure 1.2, proposée dans [Billaut 99].

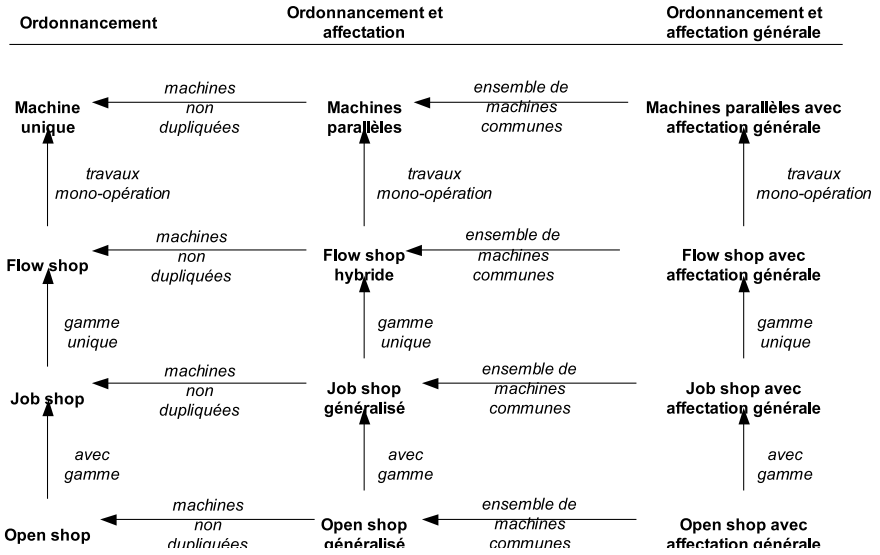


FIG. 1.2: Une typologie des problèmes d'ordonnancement (d'après [Billaut 99])

S'inspirant de cette typologie, la notation $\alpha|\beta|\gamma$ proposée dans [Graham *et al.* 79], puis reprise dans [Blazewicz *et al.* 96], s'est rapidement imposée comme faisant référence. Elle permet en effet de caractériser un problème d'ordonnancement de manière précise. Le champ α décrit la structure du problème et se décompose généralement en deux sous-champs α_1 et α_2, le premier indiquant la nature du problème (job shop, flow shop, etc.), le second précisant le nombre de machines ou de pools. Le champ β décrit les types de contraintes prises en compte. Enfin, le champ γ indique la fonction objectif considérée. À titre indicatif, quelques valeurs classiques de α_1, β et γ sont décrites dans les tableaux 1.2,

1.3 et 1.4.

Valeur	Description
$\emptyset$	machine unique
P	machines parallèles identiques
Q	machines parallèles proportionnelles
R	machines parallèles non reliées
F	flow shop
J	job shop
O	open shop
FH	flow shop hybride
JG	job shop généralisé
OG	open shop généralisé
XAG avec $X \in \{P, Q, R, F, J, O\}$	problème X avec affectation générale

TAB. 1.2: Quelques valeurs du champ α_1

Valeur	Description
$prec$	il existe des contraintes de précédence générale entre les opérations
r_i	une date de début au plus tôt r_i est associée à chaque travail i
d_i	une date d'échéance préférentielle d_i est associée à chaque travail i
$\tilde{d}_i$	une date d'échéance stricte $\tilde{d}_i$ est associée à chaque travail i
$p_i = 1$	les durées opératoires sont unitaires
$pmtn$	la préemption des opérations est autorisée
$no - wait$	les opérations de chaque travail doivent se succéder sans attente
$Snsd(Rsnd)$	les ressources doivent être préparées avant et/ou après chaque exécution indépendamment de la séquence des travaux

TAB. 1.3: Quelques valeurs du champ β

En ordonnancement de projet, la notion de gamme opératoire est généralement absente. Toutefois une notation de type $\alpha|\beta|\gamma$ a également été proposée dans [Brucker *et al.* 99], qui se veut proche de celle définie en ordonnancement d'atelier. La principale différence concerne les valeurs du champ α qui dépend, non plus de l'existence de gamme opératoire, mais de la nature des variables de décision (ordonnancement et/ou affectation) et de la nature des ressources (renouvelable, non-renouvelable). Le tableau 1.5 indique quelques

Valeur	Fonction objectif à minimiser
$C_{\max}$	la durée totale (makespan)
$L_{\max}$	le plus grand retard algébrique
$T_{\max}$	le plus grand retard vrai
$\sum U_i$	le nombre de travaux en retard
$\sum [w_i]C_i$	la durée moyenne ou pondérée des travaux
$\sum [w_i]U_i$	le nombre moyen ou pondéré de travaux en retard
$\sum [w_i]T_i$	le retard moyen ou pondéré

TAB. 1.4: Quelques valeurs du champ γ

valeurs possibles du champ α. On remarque qu'un problème de type job-shop $Jm||C_{\max}$ devient alors un cas particulier d'un problème de type $PSm, 1, 1||C_{\max}$. Notons également qu'une autre classification des problèmes d'ordonnancement de projet est proposée dans [Herroelen *et al.* 99].

Valeur	Description
PS	ordonnancement de projet
MPS	ordonnancement de projet avec plusieurs modes d'exécution par activités
PSm, σ, ρ	problème PS avec m ressources, de capacité σ, et chaque activité requiert au plus ρ unités de ressource
$MPSm, \sigma, \rho; \mu, \tau, \omega$	problème MPS avec m ressources renouvelables, de capacité σ, consommables en au plus ρ unités et avec μ ressources non-renouvelables, de capacité τ, consommables en au plus ω unités

TAB. 1.5: Valeurs du champ α en gestion de projet

Remarquons enfin que plusieurs extensions ont été proposées pour les champs β et γ (voir [Blazewicz *et al.* 96, T'kindt & Billaut 02]) afin de prendre en compte des catégories de problème particulières (stochastiques, répétitifs, cycliques, dynamiques, multicritères, etc.) et qu'il est probable que dans l'avenir, d'autres extensions soient encore créées.

1.1.8 Méthodes classiques de résolution

Plusieurs auteurs [MacCathy & Liu 93, Esswein 03] partitionnent les méthodes de résolution en trois classes : les méthodes optimales efficaces, les méthodes énumératives et les méthodes heuristiques.

Les méthodes optimales efficaces garantissent, pour un problème et un critère donnés, la détermination d'une solution optimale en un temps de calcul polynomial, i.e le temps maximal consommé pour trouver une solution optimale est une fonction polynomiale des données du problème (nombre de ressources, de tâches). De telles méthodes ne sont évidemment disponibles que pour des classes réduites de problèmes d'ordonnancement. Parmi les plus connues, citons :

- la règle SPT (Shortest Processing Time) pour le problème $1||\sum C_j$;
- la règle WSPT (Weighted Shortest Processing Time) pour le problème $1||\sum w_j C_j$;
- la règle EDD (Earliest Due Date) pour le problème $1|d_j|L_{\max}$;
- l'algorithme de Moore et Hodgson pour le problème $1|d_j|\sum U_j$;
- l'algorithme de Johnson pour le problème $F2|prmu|C_{\max}$.

Les méthodes optimales énumératives procèdent quant à elles à une énumération partielle de l'espace de recherche. Leur complexité temporelle théorique est généralement exponentielle, mais elles fournissent en pratique, sur des problèmes de taille moyenne, des solutions optimales en un temps raisonnable. Dans cette classe, on peut distinguer :

- les Procédures par Séparation et Évaluation (PSE) qui énumèrent par une recherche arborescente un ensemble de solutions, en éliminant les branches de l'arbre de recherche montrées non-optimales (utilisation de bornes inférieure et supérieure du critère) ;
- les méthodes de Programmation Linéaire (PL), modélisant les critères et les contraintes comme des fonctions linéaires de variables mixtes (réelles, entières) ;
- les méthodes basées sur la Programmation Dynamique (PD) qui procèdent à une décomposition en sous-problèmes, que l'on résout optimalement à rebours, en tenant compte à chaque étape des informations issues de la résolution du sous-problème précédent.

Les méthodes heuristiques sont souvent utilisées pour traiter des problèmes que les méthodes optimales sont incapables de résoudre en un temps acceptable. Elles produisent généralement une solution faisable de bonne qualité en un temps relativement court. La qualité d'une heuristique doit être évaluée sur plusieurs jeux d'instance de taille variable afin de mesurer d'une part, la déviation moyenne du critère par rapport à sa valeur optimale, et d'autre part, l'évolution du temps de calcul en fonction de la taille ou de la structure

du problème. Ces heuristiques sont classiquement classifiées en trois grands types :

- les algorithmes gloutons dans lesquels les décisions d'ordonnancement sont prises progressivement, à temps croissant, au fur et à mesure que les ressources se libèrent, grâce à des règles de priorité simples de type SPT, EDD, etc. ; et pour lesquels on ne remet jamais en cause une décision qui a été prise.
- les méthodes de recherche locale (tabou, recuit simulé, algorithmes génétiques, etc.) qui, partant d'une solution initiale, définissent un voisinage, qui est ensuite exploré pour trouver des solutions meilleures ; (en s'autorisant parfois à dégrader la solution courante pour augmenter les chances d'obtenir une solution meilleure).
- les méthodes de recherche arborescente tronquée, proches des PSE, excepté que l'arbre de recherche est volontairement restreint, quitte à perdre des solutions optimales, afin de gagner en temps de calcul.

1.2 Incertitudes et robustesse

1.2.1 Les sources d'incertitudes

Que ce soit en gestion de production ou en gestion de projet, une caractéristique importante de l'environnement dans lequel s'applique un ordonnancement réside dans son caractère incertain. En effet, l'ordonnancement dépend de *paramètres internes*, propres au système assurant sa réalisation (capacité des ressources, durées opératoires, ...), et de *paramètres externes*, correspondant aux informations données par les fournisseurs et les sous-traitants participant indirectement à sa mise en œuvre (délais, volumes de production, ...). Or, lors de la construction de l'ordonnancement, ces paramètres sont souvent supposés donnés et constants alors qu'en réalité, ils sont mal connus et susceptibles de varier dans le temps de façon plus ou moins prévisible.

Lorsque le système d'information associé à l'organisation le permet, les perturbations sur les paramètres sont signalées en temps réel, par des événements, afin de permettre un *ordonnancement réactif*. Remarquons que le qualificatif *réactif* doit ici être pris dans le sens automatique du terme, un ordonnancement réactif pouvant être assimilé à un système bouclé, ainsi que le représente la figure 1.3, inspirée de [Artigues *et al.* 02]. Comme nous le verrons par la suite, cette notion d'ordonnancement réactif nous semble centrale et relativement indissociable de la notion d'ordonnancement sous incertitudes.

L'ordonnancement réactif agit sur un système d'activités. Ce système est composé des acteurs et des moyens participant directement (ressources) ou indirectement (fournisseurs, sous-traitants) à la réalisation de l'ordonnancement ; il est éventuellement traversé par

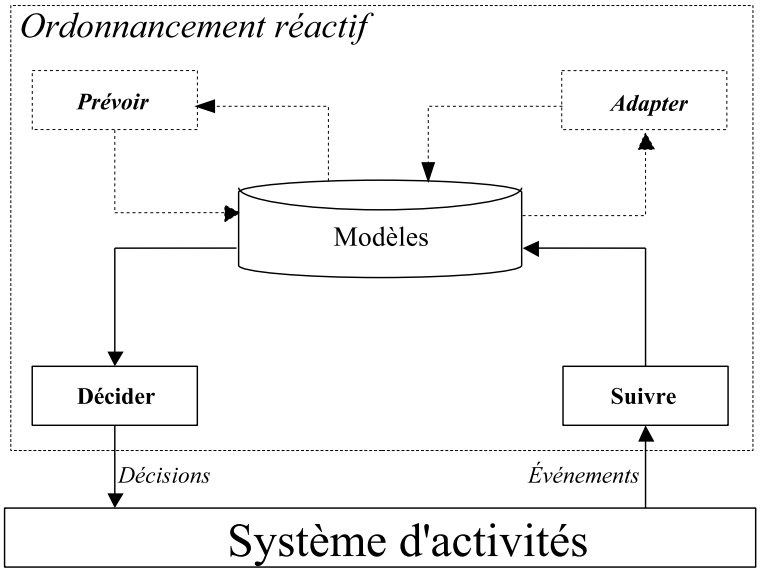


FIG. 1.3: Une vision automaticienne d'un ordonnancement réactif

un flux de produits, comme par exemple dans le cas de l'ordonnancement de production. L'ordonnancement réactif produit un ensemble de décisions, en réaction à des événements attendus (par exemple la fin d'une tâche, la libération d'une ressource, ...) ou contingents (par exemple la panne d'une ressource, un retard de livraison, ...), issus du système commandé. Il utilise pour cela un ou plusieurs modèles, ceux-ci synthétisant l'ensemble des décisions possibles à un instant donné (compatibles avec les contraintes prises en compte et l'état de l'environnement).

Sur la figure 1.3, l'ordonnancement réactif est assimilé à la réalisation de quatre fonctions : *suivre*, *décider*, *prévoir* et *adapter*. Les deux premières sont des fonctions indispensables à la réactivité : la fonction *suivre* permet de mettre à jour, au fur et à mesure de la réception des événements, les modèles utilisés par la deuxième fonction, pour l'élaboration des décisions d'ordonnancement. Les deux dernières sont des fonctions non obligatoires, utilisées lorsque l'on cherche à anticiper les événements, pour respectivement prévoir un ensemble de décisions d'ordonnancement, et éventuellement l'adapter si cela s'avère nécessaire (par exemple à l'occurrence d'un événement contingent).

Notons que les événements contingents, relatifs aux perturbations, peuvent survenir non seulement durant la phase de mise en œuvre de l'ordonnancement, mais aussi avant. Par exemple, en ordonnancement de production, un client peut souhaiter modifier le délai relatif à sa commande alors que cette dernière n'a pas encore été mise en œuvre.

Notons également que les fonctions *décider* et *prévoir* sont souvent désignées sous les noms respectifs *d'ordonnancement en ligne* ou *d'ordonnancement hors-ligne* (ces noms sont utilisés dans la suite du texte). On parle parfois également d'ordonnancement *dynamique*

et d'ordonnancement *statique*.

En s'inspirant de l'article [Davenport & Beck 00], la table 1.6 illustre à titre d'exemple quelques types de perturbations.

Paramètres internes	- Variation de la capacité Q_k d'une ressource - Variation de la durée opératoire p_j d'une tâche - Ajout / Suppression d'une contrainte - Modification de la fonction objectif
Paramètres externes	- Modification des r_j et/ou d_j d'une tâche - Insertion / Annulation d'une tâche

TAB. 1.6: Quelques exemples de perturbations

1.2.2 Modèles de prise en compte des incertitudes

MacKay et al. [Mackay *et al.* 89] distinguent, en fonction du niveau de connaissance, trois catégories d'incertitudes :

- les incertitudes complètement inconnues, correspondant à des événements totalement imprévisibles, par exemple une grève, une catastrophe naturelle ;
- les suspicions du futur, qui sont issues de l'intuition et de l'expérience des décideurs, mais difficilement modélisables ;
- les incertitudes connues, ou partiellement connues, pour lesquelles un modèle d'information est disponible.

Dans les deux premiers cas, la prise en compte des incertitudes, lors de la construction de l'ordonnancement, ne peut être traitée que de façon implicite. Par contre, pour le dernier cas, il est possible de prendre explicitement en compte les caractéristiques des incertitudes, au sein des modèles utilisés par l'ordonnancement.

Dans [Billaut *et al.* 05], quatre types de modèles sont distingués : les *modèles stochastiques*, les *modèles flous*, les *modèles par intervalles* et les *modèles par scénarios*.

Un modèle stochastique peut être utilisé lorsque l'on dispose d'informations concernant l'amplitude, la fréquence ou plus généralement l'impact des perturbations sur les données du problème. Ces informations sont issues d'analyses statistiques appliquées à l'historique d'une organisation. Des variables aléatoires et des distributions de probabilité peuvent alors être associées aux paramètres du problème d'ordonnancement.

Les modèles flous sont souvent préférés aux précédents, car ils contournent le problème de la détermination des densités de probabilité, en préférant une représentation plus intuitive, associant aux paramètres des ensembles flous. Il s'agit donc davantage de représenter l'imprécision que l'incertitude. Un ensemble flou F est défini par l'intermédiaire d'une fonction d'appartenance $\mu_F(\omega)$ qui associe à chaque élément $\omega \in \Omega$, compatible avec le concept de F, une valeur comprise dans $[0, 1]$. Par exemple, sur la figure 1.4, est représenté l'ensemble flou M des variations typiques de durées opératoires au sein d'une organisation ; Ω est l'ensemble des durées opératoires. Si une durée augmente de plus de 20% ou diminue de plus de 10% par rapport à sa valeur nominale alors la variation est atypique ($\mu_M(\omega) = 0$). Si une durée varie entre $[-5\%, +10\%]$ alors la variation est typique ($\mu_M(\omega) = 1$). Entre $[-10\%, -5\%]$ et $[+10\%, +20\%]$ de variation, plus la valeur de μ_M est grande et plus les valeurs de ω sont compatibles avec le concept de variation typique.

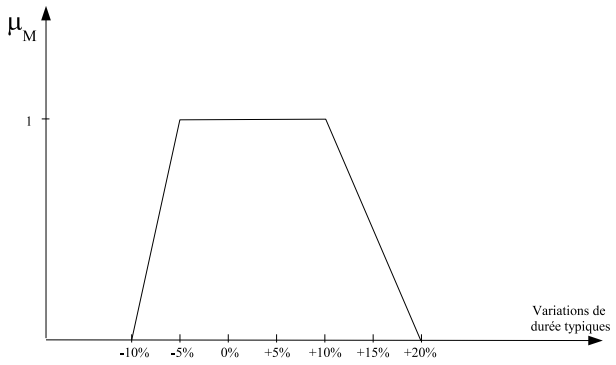


FIG. 1.4: Un exemple d'ensemble flou

Un modèle par intervalles peut être vu comme un cas particulier des modèles précédents puisque chaque paramètre peut prendre sa valeur, selon une densité de probabilité uniforme, dans un intervalle de valeurs. Les paramètres auquels est associé ce type de modèle sont généralement les dates de début au plus tôt des tâches ($r_j \in [\underline{r}_j, \overline{r}_j]$), les dates de fin des tâches ($d_j \in [\underline{d}_j, \overline{d}_j]$) et les durées opératoires ($p_j \in [\underline{p}_j, \overline{p}_j]$). En associant à une tâche j quatre variables, r_j, s_j, f_j et d_j, il est possible de représenter ces informations au sein d'un graphe potentiels-bornes [Esquirol *et al.* 95], ainsi que l'illustre la figure 1.5. Notons cependant que dans cette représentation, il est important de distinguer la nature des variables : les variables s_j et f_j sont *contrôlables* puisqu'elles sont fixées par l'ordonnancement ; les variables r_j et d_j sont *incontrôlables*, puisqu'elles sont fixées par l'environnement externe.

Un modèle par scénarios énumère pour l'ensemble P des paramètres relatifs à un problème, plusieurs ensembles de valeurs possibles (i.e. $P \in \{P_1, \ldots, P_k\}$). Chaque ensemble P_i correspond à un scénario, c'est-à-dire à un problème d'ordonnancement distinct. Une probabilité peut éventuellement être associée à chaque scénario, sinon ils sont équiprobables. La différence avec les deux modèles précédents réside dans le fait que les scénarios

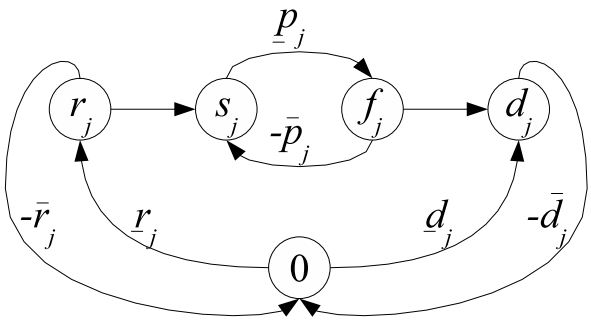


FIG. 1.5: Représentation sur un graphe potentiels-bornes d'un modèle d'incertitude par intervalles

sont explicitement énumérés, alors que dans les cas précédents ils sont caractérisés.

On note que dans [Roy 02], le terme de *version* est préféré à celui de scénario. Dans [Vincke 99], c'est le terme d'*instance* qui est utilisé. Ainsi que le soulignent ces auteurs, et quel que soit le terme choisi, une modélisation des incertitudes est de toute façon souhaitable, dans le cadre de la robustesse, afin de spécifier vis-à-vis de quoi un résultat peut être robuste. Cette notion de robustesse est discutée dans la partie suivante.

1.2.3 Notion de robustesse en ordonnancement

Le terme *robustesse* apparaît de plus en plus souvent en ordonnancement, mais aussi en aide à la décision et recherche opérationnelle. Nous reportons ici quelques définitions trouvées dans la littérature pour ce terme :

- *« Le paradigme classique en programmation mathématique est de développer un modèle qui suppose que les données d'entrée sont connues et égales à leur valeur nominale. Cependant, cette approche ne prend pas en compte l'effet des incertitudes sur ces données qui influence la qualité et la faisabilité du modèle. Il est en effet possible que les valeurs des données diffèrent de leur valeur nominale, induisant ainsi le viol de plusieurs contraintes, et rendant la solution optimale trouvée à partir des valeurs nominales, de mauvaise qualité, voire infaisable. Cette constatation montre l'intérêt de concevoir des approches de résolution qui soient insensibles aux incertitudes, c'est-à-dire robustes. »* - [Bertsimas & Sim 04].
- *« Nous utilisons le terme de robustesse pour caractériser la performance d'un algorithme ou plutôt d'un processus complet de construction d'un ordonnancement en présence d'aléas. »* - [GOThA 02].

- *« La robustesse est la capacité d'un ordonnancement prédictif à faire face à des événements imprévus. »* - [Davenport & Beck 00].
- *« ...un ordonnancement robuste est un ordonnancement capable d'absorber certaine quantité d'événements inattendus sans avoir à être ré-ordonnancé. »* - [Davenport & Beck 00].
- *« ...un ordonnancement est dit robuste quand on prévoit sa bonne performance par rapport aux autres face à un certain ensemble de scénarios, et quand on utilise la technique de décalage à droite pour ré-ordonnancer. »* - [Jensen 01].
- *« ...un ordonnancement robuste est un ordonnancement qui est insensible à des perturbations imprévues de l'atelier de production, étant donnée une politique de contrôle a priori. »* - [Leon *et al.* 94].
- *« ...un ordonnancement robuste est un ordonnancement capable de satisfaire des exigences de performance a priori dans un environnement incertain. »* - [Le Pape 91].
- *« ...un ordonnancement robuste est un ordonnancement dont les performances sont acceptables en présence d'incertitudes. »* - [Esswein 03].
- *« Un ordonnancement robuste est un ordonnancement dont la performance (par rapport à l'ordonnancement optimal associé) est relativement insensible aux réalisations potentielles des durées opératoires. »* - [Kouvelis *et al.* 00].
- *« Un ordonnancement robuste est un ordonnancement ayant la meilleure performance au pire cas par rapport à la solution optimale correspondante sur toutes réalisations potentielles des durées opératoires. »* - [Daniels & Kouvelis 95].
- *« Le terme stabilité ou robustesse de solution concerne l'insensibilité des dates de début des activités à des changements sur les données d'entrée. Le terme robustesse de qualité est utilisé pour parler de l'insensibilité de la performance de l'ordonnancement par rapport à la valeur de la fonction objectif. »* - [Herroelen & Leus 04b].
- *« La robustesse de qualité est une propriété d'une solution dont la qualité, mesurée par la valeur de la fonction objectif, ne dévie pas trop par rapport à l'optimalité lorsqu'il y a des petits changements sur les données du problème. La robustesse de solution peut être décrite comme la robustesse dans l'espace de solutions. La robustesse de solution apparaît quand une solution ne change pas beaucoup lorsqu'il y a des petits changements sur les données du problème, c'est-à-dire la méthode de résolution est capable de trouver une solution proche de la solution actuellement utilisée. »* - [Sevaux & Sörensen 04].
- *« un ordonnancement est robuste si sa performance est peu sensible à l'incertitude des données [...] la robustesse d'un ordonnancement caractérise donc cette performance. »* -[Billaut *et al.* 05].

- *« Un des souhaits souvent formulés par les décideurs vis-à-vis de la méthode multicritère qu'ils utilisent est d'avoir une idée de la robustesse du résultat. C'est la raison pour laquelle l'étude de la sensibilité des méthodes multicritères fait partie des grandes axes de recherche du domaine. Cette demande traduit la volonté de savoir dans quelle mesure une variation des données due par exemple à une erreur de mesure ou d'estimation risque d'affecter le résultat donné par la méthode »* - [Durand & Trentesaux 00].

Nous constatons que s'il y a un consensus général pour une définition informelle de la robustesse (capacité de faire face à des aléas, de résister à l'à peu près, de gérer le risque, ...), les définitions divergent dès que l'on considère un champ applicatif particulier. La raison de cette hétérogénéité réside dans le fait que le terme robustesse s'applique à des objets différents selon les travaux, et que la robustesse est donc également mesurée de façon différente.

Avant de discuter des différentes mesures de robustesse, la partie suivante montre les liens unissant le concept de robustesse et celui de flexibilité. C'est ce lien que nous exploitons dans nos travaux.

1.2.4 Flexibilité et robustesse

Le terme *flexibilité* est très utilisé, notamment en gestion de production, où la recherche de flexibilité apparaît comme un des éléments les plus marquants dans l'évolution des systèmes de production. De façon générale, la flexibilité d'un objet, d'un système, est une propriété qui recouvre généralement deux aspects complémentaires mais distincts [Erschler & Terssac 88] :

- un aspect interne lié à une capacité de changement, de déformation, à une variété d'états possibles ;
- un aspect externe lié à une capacité d'adaptation à des modifications de l'environnement, à des perturbations.

Le premier type de flexibilité est lié à la nature même de l'objet alors que le second concerne les interactions avec son environnement, sa fonction au sein d'un ensemble plus vaste. Si la capacité de changement peut être mesurée indépendamment de l'environnement de l'objet, sa capacité d'adaptation dépend du type de perturbations qu'il doit être capable d'absorber. On voit donc bien ici que la notion de flexibilité peut être reliée à celle de robustesse. Notons que dans [Artigues *et al.* 02], les auteurs utilisent respectivement les notions de *flexibilité statique* et *flexibilité dynamique* pour désigner ces flexibilités interne et externe.

Dans [GOThA 02], pour le domaine de l'ordonnancement, les auteurs indiquent que la flexibilité peut être exprimée comme l'existence de modifications possibles dans un ordonnancement, calculé hors-ligne, entraînant une perte de performance acceptable. Elle peut

aussi être associée à un voisinage de solutions pouvant être examiné lors de l'exécution, ou à une famille d'ordonnancements, sans privilégier un ordonnancement en particulier. Il s'agit donc là d'une flexibilité statique pouvant avoir plusieurs formes [Billaut *et al.* 05]. Nous distinguons :

- *la flexibilité temporelle*, concernant les dates de début des tâches, et autorisant sous certaines limites une dérive de ces dernières dans le temps ;
- *la flexibilité séquentielle* qui autorise une modification de l'ordre des tâches sur les ressources ;
- *la flexibilité sur les affectations* qui, dans le cas où une tâche peut être réalisée par une ressource à choisir dans un pool, et/ou dans le cas où une tâche est associée à plusieurs modes d'exécution, laisse libre certains choix d'affectation.

La flexibilité temporelle est souvent implicite en ordonnancement. En effet, dès lors qu'un ordonnancement est déterminé, il est possible de construire un graphe potentiels-tâches conjonctif (satisfaisant les contraintes de ressource), où le respect des performances temporelles (valeurs de $C_{\max}$ ou $L_{\max}$) est imposé grâce à l'ajout d'arcs de longueur négative, entre certains nœuds et le nœud origine 0. Il est alors possible de déterminer pour chaque tâche j, une date de début au plus tôt $\underline{s}_j$, en calculant le chemin le plus long d_{0j} entre les nœuds 0 et j ($\underline{s}_j = d_{0j}$), et une date de début au plus tard $\overline{s}_j$, en calculant le chemin le plus long d_{j0} entre les nœuds j et 0 ($\overline{s}_j = -d_{j0}$). La date de début de la tâche j peut donc varier entre ces deux valeurs (i.e. $s_j \in [\underline{s}_j, \overline{s}_j]$), sans modification de la performance, ce qui correspond bien à une flexibilité temporelle. Cette forme de flexibilité est donc naturellement présente dans une solution, et peut éventuellement être augmentée dans un objectif de robustesse, comme nous le verrons par la suite.

La flexibilité séquentielle est plus rarement utilisée. En effet, l'introduction de flexibilité séquentielle impose de manipuler, non pas un seul ordonnancement, mais une famille d'ordonnancements. De ce fait, il est plus difficile, en évitant la combinatoire liée à une énumération exhaustive et systématique des solutions, de déterminer la performance d'une famille d'ordonnancements, ces derniers ne possédant pas tous une performance identique. Ce point est abordé dans les chapitres suivants. Notons que l'introduction de flexibilité séquentielle induit obligatoirement une augmentation de la flexibilité temporelle (l'inverse étant faux).

La flexibilité sur les affectations, lorsqu'elle existe initialement dans le problème, est rarement conservée à l'issue de l'ordonnancement hors ligne. En effet, comme dans le cas de la flexibilité séquentielle, se pose également ici la difficulté de manipulation d'une famille d'ordonnancements et de son évaluation vis-à-vis de la fonction objectif. Néanmoins, la flexibilité sur les affectations est souvent utilisée au cours de la phase d'ordonnancement réactif lorsque, par exemple suite à la défection d'une ressource, une tâche est déplacée de

la ressource défectueuse vers une autre de façon dynamique. La position d'insertion est souvent déterminée par analyse de la flexibilité temporelle disponible sur la ressource de destination, afin de minimiser les conséquences sur la performance [Artigues & Roubellat 00].

Les trois formes de flexibilité précédemment décrites sont de type statique. La flexibilité dynamique d'un ordonnancement réside dans la capacité de l'ordonnancement réactif à adapter l'ordonnancement déterminé hors ligne, en présence de perturbations, de façon à satisfaire au mieux les exigences propres à ce niveau de décision.

1.2.5 Le compromis robustesse / performance

Il est important de pouvoir mesurer la robustesse afin de pouvoir déterminer quelle solution est plus robuste qu'une autre, ce qui est utile dans un contexte d'optimisation. Les mesures diffèrent selon les spécificités du domaine applicatif considéré et selon que l'on s'intéresse à l'insensibilité aux incertitudes de la fonction objectif (quality robustness) ou à l'insensibilité d'une solution d'ordonnancement particulière (solution robustness) [Herroelen & Leus 02]. Le lecteur est prié de se référer à [Billaut *et al.* 05, GOThA 02] où plusieurs mesures génériques sont proposées pour ces deux types de robustesse.

Plusieurs auteurs [Bertsimas & Sim 04, Esswein 03, Briand *et al.* 03a] mettent en évidence que la robustesse a un prix : plus un résultat est robuste (insensible aux incertitudes), et plus en contrepartie sa performance est mauvaise, et inversement. À titre d'exemple, les paragraphes suivants présentent brièvement l'approche récente de Bertsimas et Sim [Bertsimas & Sim 04], qui nous semble particulièrement illustrative du point de vue du compromis robustesse / performance. Les auteurs s'intéressent à un problème d'optimisation très général formulé de la façon suivante :

$$\left.\begin{array}{ll} \text{Maximiser} & c'x \\ \text{Sous contraintes} & Ax \leq b \\ & l \leq x \leq u. \end{array}\right\}$$

Dans cette formulation, seuls les éléments de la matrice A sont supposés affectés par les incertitudes. J_i désignant les coefficients de la ligne i de la matrice A, chaque coefficient a_{ij}, $j \in J_i$, est modélisé par une variable aléatoire $\tilde{a}_{ij}$ prenant valeur dans l'intervalle $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$, où a_{ij} correspond à la valeur nominale du coefficient. La distribution de probabilité est supposée symétrique sur cet intervalle, mais inconnue. Les auteurs introduisent ensuite pour chaque ligne i, le paramètre Γ_i, prenant valeur dans l'intervalle $[0, |J_i|]$, indiquant le nombre de coefficients de J_i susceptibles de différer de leur valeur nominale. Ils montrent alors que ce problème peut être reformulé de façon linéaire (sans que cela induise une augmentation très pénalisante de la taille du modèle) et s'intéressent à déterminer, en fonction de la valeur de Γ_i, d'une part la probabilité pour qu'une contrainte

soit violée (i.e. que la solution devienne infaisable), et d'autre part, sur la base d'une analyse de sensibilité, la dégradation, provoquée par une augmentation de Γ_i, sur le critère. Les auteurs constatent que, pour un niveau d'incertitude donné (c'est-à-dire pour une valeur de Γ_i fixe), plus on souhaite que la robustesse de solution soit importante (c'est-à-dire que la probabilité d'infaisabilité soit faible), et plus il faut accepter en contrepartie une dégradation de qualité importante. Ils montrent aussi, en considérant plusieurs problèmes d'optimisation combinatoire, que la probabilité d'infaisabilité peut cependant être réduite en dessous de 1%, pour des valeurs de Γ_i conséquentes relativement à celles des $|J_i|$, sans que cela pénalise la fonction objectif de plus de quelques pour cent.

Cet exemple met donc en évidence, dans le cadre très général de l'optimisation, d'une part que la robustesse doit être envisagée dans une optique de recherche de compromis et, d'autre part, que les mesures de robustesse de solution et de robustesse de performance sont fortement reliées.

1.2.6 Une classification des approches d'ordonnancement robuste

Depuis une dizaine d'années, plusieurs méthodes d'ordonnancement robuste ont été proposées dans la littérature. Plusieurs classifications ont également été décrites.

La classification, proposée dans [Davenport & Beck 00], et reprise dans [Billaut *et al.* 05], constitue une référence souvent utilisée dans le domaine de l'ordonnancement robuste. Cette classification distingue trois types d'approches : les approches *proactives*, les approches *réactives* et les approches *proactives-réactives*.

Les approches proactives tentent de prendre en compte l'incertain lors de la phase d'ordonnancement hors-ligne uniquement. Il s'agit d'anticiper les incertitudes, en jouant sur la flexibilité, de sorte à produire un ordonnancement, ou une famille d'ordonnancements, relativement insensible aux incertitudes. Un modèle d'incertitudes est pour cela généralement supposé disponible. Si le degré d'incertitude est très élevé, ou si aucun modèle d'incertitudes n'est disponible a priori, ou encore si l'environnement de l'ordonnancement est très dynamique, alors une approche réactive peut être plus appropriée.

Les approches réactives prennent en compte l'incertain lors de la phase d'ordonnancement dynamique uniquement. On ne cherche donc pas à anticiper les incertitudes, mais plutôt à réagir en temps réel, de façon opportune, lorsque des aléas surviennent. La stratégie de réaction est généralement élaborée sur la base de l'état courant du système d'activités (état des ressources et des tâches) et exploite éventuellement des informations relatives à l'aléa considéré (durée d'une panne, estimation d'un retard). Un ordonnancement de référence, de nature déterministe et déterminé hors ligne, est aussi parfois utilisé. Le temps d'élaboration des décisions d'ordonnancement, nécessaires à la prise en compte d'un aléa, doit être suffisamment court relativement à la dynamique du système d'activités considéré.

Cette contrainte est parfois primordiale, comme dans le cas de l'ordonnancement de processus informatiques dans un système d'exploitation. Seules des règles de priorité simples peuvent alors être utilisées.

Les approches proactives-réactives tentent de combiner avantageusement les deux techniques précédentes. L'idée est de mettre à profit l'utilisation d'une technique proactive pour faciliter l'élaboration des stratégies de réaction, de sorte que les décisions d'ordonnancement soient de meilleure qualité, et produites en un temps plus court. Un autre avantage des approches proactives-réactives réside tout simplement dans le fait qu'elles prennent en compte les incertitudes tout au long du cycle de vie de l'ordonnancement. Il est ainsi possible pour un décideur de gérer le compromis robustesse / performance durant la totalité de ce cycle, ce qui est sain par essence, voire indispensable dans le domaine de la conduite de projets.

Très proche de celle de Davenport et Beck, une autre classification est proposée dans [Mehta & Uzsoy 98]. Celle-ci comporte quatre catégories : l'approche totalement réactive, l'approche prédictive - réactive, l'approche robuste et l'approche à base de connaissances. Les deux premières scindent la catégorie des approches réactives de Davenport et Beck en deux, selon qu'un ordonnancement déterministe de référence est exploité lors de la phase réactive ou non. La troisième catégorie recouvre quant à elle les approches proactives et proactives-réactives de Davenport et Beck. Enfin, la catégorie des approches à base de connaissance correspond aux méthodes dont l'objectif est de fournir un mécanisme pour la sélection dynamique d'une politique de ré-ordonnancement appropriée, parmi un ensemble d'alternatives possibles. On peut donc la voir comme une sous-catégorie particulière des approches réactives.

Dans [Herroelen & Leus 02], six types d'approches d'ordonnancement sous incertitudes sont distingués : les approches réactives, les approches stochastiques, les réseaux de projets stochastiques, les approches floues, les approches proactives et les approches basées sur l'analyse de sensibilité. Sans décrire les caractéristiques de chaque catégorie (les approches stochastiques et floues sont discutées dans les parties suivantes), nous pouvons dire, qu'à la différence des deux classifications précédentes, celle-ci est davantage guidée par la nature des outils utilisés pour modéliser ou gérer l'incertitude, et non par une distinction méthodologique de fond.

Sans remettre en question ces classifications, nous nous proposons de brosser un état de l'art de l'ordonnancement sous incertitudes en distinguant trois catégories : les approches réactives, les approches prédictives-réactives et les approches proactives-réactives. En effet, comme nous l'avons indiqué dans la partie 1.2.1, il nous semble sain, lorsqu'on s'intéresse à la prise en compte d'incertitudes, d'adopter une vision automaticienne de l'ordonnancement réactif, celui-ci étant supposé couplé à un environnement évoluant dynamiquement. D'autre part, nous pensons que les approches se distinguent essentiellement par le fait qu'elles utilisent ou non un ordonnancement de référence et, lorsque c'est le cas, par le

fait qu'elles tentent d'intègrer ou non, de façon proactive, l'existence d'incertitudes. Nous illustrons ces trois classes d'approches par les schémas illustratifs des figures 1.6, 1.7 et 1.8.

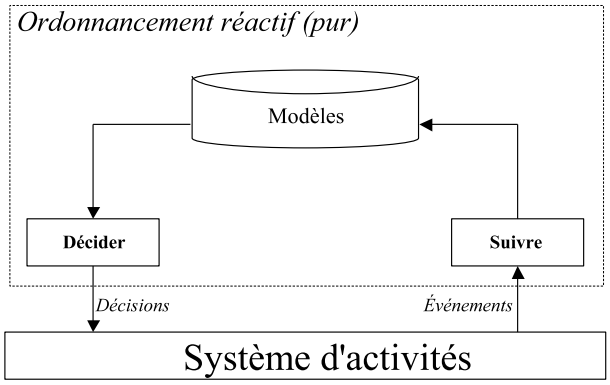


FIG. 1.6: Approches réactives

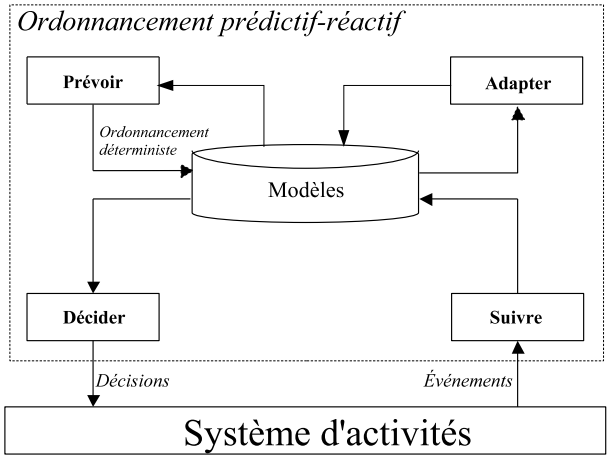


FIG. 1.7: Approches prédictives - réactives

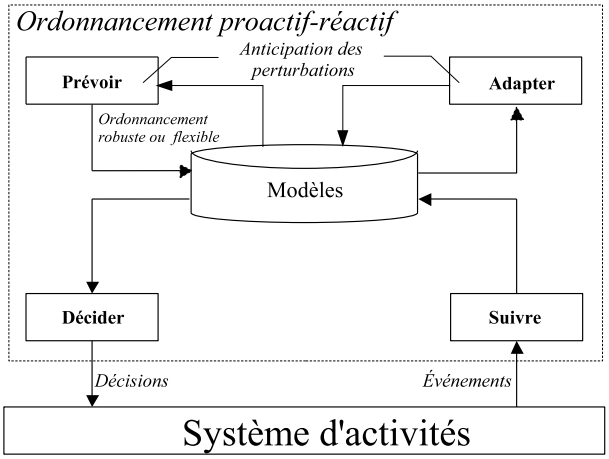


FIG. 1.8: Approches proactives - réactives

Chapitre 2

Un tour d'horizon des méthodes d'ordonnancement robuste

Dans le respect de la classification présentée dans la partie 1.2.6, ce chapitre propose un tour d'horizon des méthodes d'ordonnancement robuste. Il s'agit de décrire les différentes façons de prendre en compte les incertitudes, soit hors-ligne, soit en ligne, dans le cadre d'un ordonnancement réactif (au sens automatique du terme). Les avantages et inconvénients de chaque approche sont discutés et une synthèse est proposée, exprimant plus précisément les motivations et les objectifs de notre étude.

2.1 Approches réactives

2.1.1 Généralités

Les méthodes d'ordonnancement réactives sont parfois qualifiées de "totalement réactives", "réactives pures", ou "dynamiques". On parle parfois aussi de *pilotage réactif*. Ces approches réactives sont souvent utilisées dans des environnements fortement perturbés, où les incertitudes sont fréquentes et de fortes amplitudes. Dans un tel environnement, un ordonnancement de référence peut rapidement s'avérer de mauvaise qualité ou même infaisable. Par conséquent, on suppose qu'il n'existe pas de tel ordonnancement, et que toutes les décisions sont prises en temps réel, en utilisant des stratégies qui privilégient la rapidité des décisions sur leur qualité. Un modèle à événements discrets (réseaux de Petri, Grafcet, Statecharts, ...) est généralement utilisé pour maintenir une image interne de l'état courant du système d'activités, cet état évoluant en fonction des occurrences d'événements. Il indique généralement à tout instant l'ensemble des décisions possibles, c'est-à-dire compatibles avec les contraintes du système, étant donné un état. Les contraintes prises en compte dans ces modèles sont souvent plus détaillées que celles prises en compte lors d'un ordonnancement hors ligne. On retrouve bien sûr les contraintes de disponibilité des ressources, de précédence entre tâches, mais aussi des contraintes portant sur les capacités de stockage, l'utilisation ou la réservation de moyens de transport, etc.

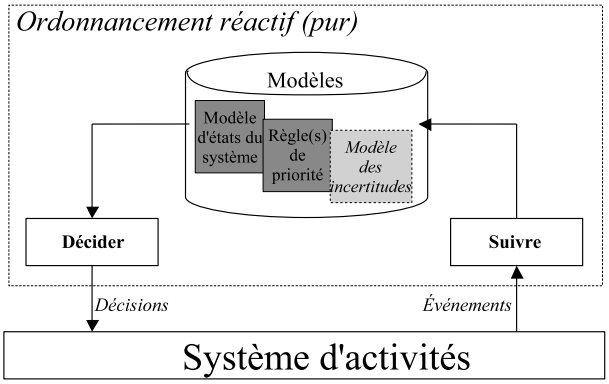


FIG. 2.1: Architecture classique d'un ordonnancement réactif

Lorsque le modèle à événements discrets indique que plusieurs décisions concurrentes sont possibles à un instant donné, il est nécessaire de faire des choix, ceux-ci conditionnant la performance finale de l'ordonnancement. Ces choix sont généralement produits à l'aide de règles de priorité (voir figure 2.1). Cette approche présente en effet plusieurs avantages, dont le principal réside dans le fait que les prises de décision sont très rapides et intuitivement faciles à comprendre pour les utilisateurs. Les règles de priorité peuvent varier selon la fonction objectif considérée (minimisation du makespan, du retard, ...).

2.1.2 Règles de priorité

Dans [Pinedo 95], une étude générale sur les règles de priorité est présentée. Celles-ci peuvent être classifiées de plusieurs manières. On distingue tout d'abord les règles statiques et les règles dynamiques. Les règles statiques sont indépendantes du temps, elles sont fonction des données associées aux tâches ou aux ressources (par exemple la règle WSPT - Weighted Shortest Processing Time first). Les règles dynamiques sont dépendantes du temps, et donc des décisions d'ordonnancement, par exemple la règle MS (Minimum Slack first). Une autre distinction concerne le caractère local ou global d'une règle. Une règle locale examine seulement une sous-partie des informations disponibles, par exemple la file d'attente des tâches en attente de la libération d'une ressource ; à la différence d'une règle globale qui exploite toutes les informations disponibles. Le tableau 2.1, issu de [Pinedo 95], présente quelques règles de base et leur contexte d'utilisation.

En pratique, les objectifs considérés sont souvent multiples et les règles sont alors combinées afin de pouvoir prendre en compte plusieurs indicateurs en même temps. Par exemple, dans [Pinedo 95], est présentée la règle ATC (Apparent Tardiness Cost) qui combine les règles WSPT et MS. Dans [Holhaus & Rajendran 97] et [Holhaus & Rajendran 00], les auteurs étudient également la combinaison de règles simples et montrent l'intérêt de certaines combinaisons du point de vue de la performance.

Règle	Donnée	Problème
SIRO (Service In Random Order)	—	—
ERD (Earliest Release Date first)	r_j	$1\|r_j\|Var(\sum(s_j - r_j)/n)$
EDD (Erlieast Due Date first)	d_j	$1\|\|L_{\max}$
MS (Minimum Slack first)	d_j	$1\|\|L_{\max}$
SPT (Shortest Processing Time first)	p_j	$P\|\|\sum s_j$ ou $F\|p_{ij} = p_j\|\sum s_j$
WSPT (Weighted SPT)	w_j, p_j	$P\|\|\sum(w_j s_j)$
LPT (Longest Processing Time first)	p_j	$P\|\|C_{\max}$
CP (Critical Path)	p_j, prec	$P\|prec\|C_{\max}$
LNS (Largest Number of Successors first)	p_j, prec	$P\|prec\|C_{\max}$
LAPT (Longest Alternate Processing Time first)	p_j	$O2\|\|C_{\max}$

TAB. 2.1: Quelques exemples de régles et leur utilisation

2.1.3 Choix dynamique de règles

Une extension naturelle des méthodes d'ordonnancement par règles consiste à permettre au système de choisir ou de modifier dynamiquement la règle à utiliser, selon le contexte. Selon [Priore *et al.* 01], cette extension permet d'améliorer considérablement la performance du système. La sélection d'une règle peut être réalisée soit à l'aide de simulations, soit à l'aide d'un modèle de connaissances.

Dans le premier cas, l'idée consiste, au moment d'une prise de décision, à simuler l'application d'un ensemble de règles, en se projetant éventuellement sur le futur proche, puis de sélectionner celle fournissant la meilleure performance. Dans [Chong *et al.* 03], les auteurs proposent un mécanisme d'ordonnancement à base de simulations conforme à cette orientation. Les performances des différentes règles sont tout d'abord comparées, à l'aide de simulations hors ligne, exécutées en tenant compte éventuellement des perturbations potentielles. Les résultats de ces simulations sont ensuite utilisés comme des indicateurs pour sélectionner en ligne les règles les plus prometteuses, étant donné un état du système et un contexte. Une simulation en ligne des règles les plus prometteuses peut aussi être réalisée pour déterminer celle la plus adaptée, le temps nécessaire à la simulation ralentissant évidemment la prise de décision.

Dans le deuxième cas, des techniques issues de l'Intelligence Artificielle sont utilisées. Le modèle de connaissances est construit hors ligne lors d'une phase d'apprentissage. La principale difficulté consiste à "calibrer" le modèle, de sorte qu'il soit suffisamment intelligent pour déterminer en ligne quelle règle, ou quelle décision, est la plus appropriée, étant donné un contexte. Dans [Priore *et al.* 98], les auteurs présentent la construction hors-ligne d'un modèle à base de connaissances, fondée sur l'itération de cinq étapes : la définition des paramètres de contrôle du modèle de connaissances, la génération des exemples d'apprentissage, la définition des conditions d'activation d'une règle (une condition d'activation est vraie si la valeur courante des paramètres de contrôle appartient à un domaine prédéfini),

la sélection de la meilleure règle parmi celles activées et la vérification de performance. Tant que la performance n'est pas satisfaisante, les étapes sont réitérées en introduisant des changements soit dans les paramètres de contrôle, soit dans les exemples d'apprentissage, soit encore sur les domaines mis en jeu dans les conditions d'activation. Une autre méthode à base de connaissances est présentée dans [Chen & Yih 96] où les auteurs focalisent leur étude sur la détermination des paramètres pertinents pour la sélection des règles, en utilisant une approche neuronale.

2.1.4 Ordonnancement coopératif

D'autres approches d'ordonnancement réactif, issues des travaux en Intelligence Artificielle, s'inspirent de la notion de systèmes multi-agents. Il s'agit de proposer un modèle d'auto-organisation coopératif pour le pilotage réactif [Pujo & Brun-Picard 02]. Dans ce modèle, le système est constitué d'un ensemble d'agents autonomes dont chacun essaie d'optimiser localement sa fonction objectif, éventuellement différente dans chaque agent, en répondant aux sollicitations des autres agents. Dans [Lim & Zhang 02], les auteurs proposent un mécanisme d'attracteur itératif pour faciliter le processus d'affectation des tâches et manipuler la négociation entre agents. Ce mécanisme permet de créer des plans de production de manière concurrente. Le point fort de cette approche est la grande liberté dont dispose chaque agent, mais la difficulté se situe au niveau de la définition des objectifs de chaque agent, ceux-ci devant être ajustés pour assurer une bonne performance globale du système.

2.1.5 Discussion

Nous constatons que les méthodes réactives présentent l'avantage de construire en temps réel, et de façon très souple, une solution faisable. Les types d'incertitudes que ces méthodes peuvent prendre en compte sont très variés : variation d'une durée, d'une date de début au plus tôt ou au plus tard, prise en compte en temps réel d'une nouvelle tâche, pannes de ressource, etc. Cependant, de telles approches fournissent en général un ordonnancement de performance faible, qu'il n'est pas possible d'anticiper du fait qu'on ne connaît l'ordonnancement qu'une fois celui-ci réalisé.

Pour pallier ce problème, certains auteurs proposent d'intégrer dans la phase de décision en ligne, des informations relatives aux incertitudes. C'est en particulier le cas dans les approches stochastiques d'ordonnancement de projet où les décisions d'ordonnancement sont prises à des moments stochastiques, en fonction du passé et d'une connaissance a priori des distributions de probabilité associées aux paramètres des activités. L'objectif lors de chaque décision est généralement de minimiser l'espérance mathématique de la durée totale d'exécution de l'ordonnancement. C'est aussi le cas pour les approches floues d'ordonnancement de projet, qui élaborent un ordonnancement pour lequel des ensembles flous sont

associés aux dates de début des activités. Cet ordonnancement peut être utilisé en ligne pour guider les décisions. Pour plus de détail sur l'ordonnancement de projet stochastique et flou, le lecteur est prié de se référer à l'état de l'art proposé dans [Herroelen & Leus 02].

2.2 Approches prédictives-réactives

2.2.1 Généralités

Les méthodes prédictives-réactives sont également souvent référencées dans la littérature pour faire face à des aléas. Dans ces approches, ainsi que l'illustre la figure 2.2, un ordonnancement déterministe de référence est construit hors-ligne. Cet ordonnancement est ensuite utilisé, en ligne, pour guider les décisions. Il est alors éventuellement adapté en temps réel, pour tenir compte des perturbations, lorsque sa flexibilité temporelle intrinsèque ne permet plus d'absorber un aléa. Ce type d'approche pallie d'une certaine façon les inconvénients des approches réactives puisque l'ordonnancement hors-ligne est susceptible d'améliorer la performance globale de l'ordonnancement réactif, en imposant certaines décisions. De plus, avoir en permanence à disposition un ordonnancement de référence peut être fort utile pour des décideurs, ceux-ci disposant ainsi d'une vision prévisionnelle de l'organisation, sur la base de laquelle il est possible d'extraire des informations en termes d'évolution des besoins dans le temps, de coûts ou de satisfaction des clients [Mehta & Uzsoy 98, Herroelen & Leus 02].

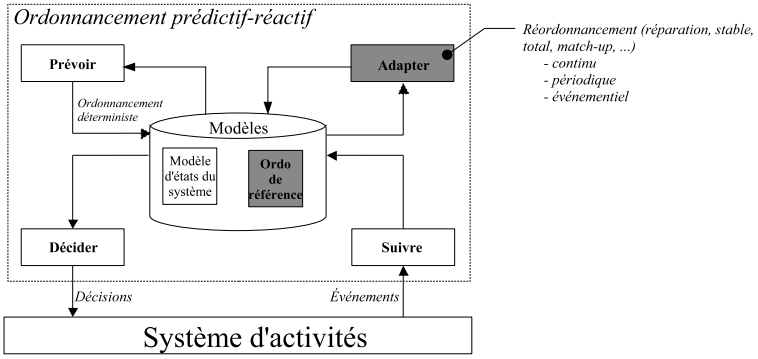


FIG. 2.2: Schéma classique d'un ordonnancement prédictif-réactif

C'est dans la façon d'aborder l'adaptation de l'ordonnancement de référence que les approches se distinguent. Dans [Sabuncuoglu & Bayiz 00], les auteurs proposent une étude générale des méthodes prédictives-réactives, en considérant en particulier les problèmes de type job shop. Ils distinguent les approches selon que les modifications apportées à l'ordonnancement de référence impliquent un ré-ordonnancement total, un ré-ordonnancement partiel ou uniquement le ré-ordonnancement local d'une tâche. Un ré-ordonnancement total implique que toutes les tâches non-réalisées sont ordonnancées à nouveau en tenant

compte de l'état courant du système. Un ré-ordonnancement partiel n'ordonnance qu'un sous-ensemble des tâches non-réalisées. Enfin, un ré-ordonnancement local (parfois appelé réparation d'ordonnancement) ne modifie que l'ordonnancement ou l'affectation d'une seule tâche. Ainsi que l'illustre la figure 2.3, issue de [Esswein 03], ces méthodes peuvent être rangées sur un axe selon le degré de modification apporté à l'ordonnancement initial.

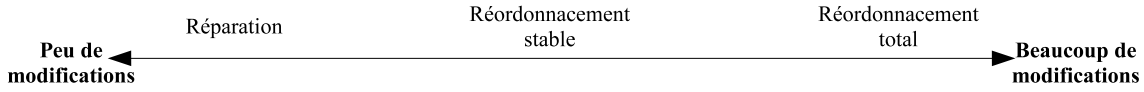


FIG. 2.3: Position des approches prédictives-réactives relativement au degré de modification de l'ordonnancement de référence

D'autres études proposent un état de l'art des approches prédictives-réactives [Mehta & Uzsoy 99, Aytug *et al.* 03] dans lequel la classification s'appuie sur les réponses aux deux questions " Quand ré-ordonnancer ?" et "Comment ré-ordonnancer ?".

En fonction de la réponse à la première question, on distingue trois types de ré-ordonnancement :

- le *ré-ordonnancement continu* recalcule un nouvel ordonnancement chaque fois qu'une perturbation rend l'ordonnancement de référence infaisable. Ce type de réaction permet d'obtenir de bonnes performances pour l'ordonnancement finalement exécuté, bien qu'il nécessite des temps de calcul conséquents, pas toujours compatibles avec la dynamique du système d'activités. De plus, le changement récurrent de l'ordonnancement est susceptible d'engendrer une instabilité de l'atelier relativement néfaste, notamment lorsque les acteurs de l'ordonnancement sont humains. Cet inconvénient est parfois désigné sous le nom de *nervosité d'un atelier* dans [Mackay *et al.* 89], dans le cas de l'ordonnancement de production.
- le *ré-ordonnancement périodique*, comme son nom l'indique, consiste à ré-ordonnancer, non plus lorsque l'ordonnancement devient infaisable, mais régulièrement, à la fin d'une période de temps pré-déterminée. Si une perturbation survient entre deux ré-ordonnancements, des règles simples de réparation locale sont appliquées pour y remédier. Cette technique est évidemment moins gourmande en temps de calcul que la précédente ; néanmoins, elle est également moins performante, notamment lorsque les perturbations sont fréquentes, puisque l'ordonnancement est susceptible de dériver considérablement entre deux ré-ordonnancements.
- le *ré-ordonnancement événementiel* est un compromis entre les deux précédentes techniques. L'ordonnancement est recalculé lorsqu'une déviation de performance conséquente a été détectée (ce qui correspond au franchissement d'un seuil et provoque la levée d'un événement)

Pour la deuxième question, quatre réponses sont décrites :

- la méthode de décalage à droite qui consiste à simplement retarder l'exécution des tâches posant problème, en maintenant la stabilité de l'atelier, au prix d'une dégradation conséquente de la performance.
- le ré-ordonnancement total (voir ci-dessus).
- le ré-ordonnancement multi-objectif qui s'intéresse à la recherche d'un compromis entre la performance du nouvel ordonnancement produit et la stabilité de l'atelier (ce qui pose le problème de la mesure de la stabilité).
- le ré-ordonnancement par retour vers l'ordonnancement initial (appelé aussi match-up rescheduling) qui consiste à ne modifier que sur une période de temps, la plus réduite possible, les décisions de l'ordonnancement de référence, de sorte que ce dernier soit de nouveau utilisable à la fin de la période (il s'agit donc ici aussi de maintenir la stabilité de l'organisation).

Une autre classification consisterait à trier les méthodes selon le problème qu'elles considèrent ou encore, la nature des incertitudes qu'elles permettent de prendre en compte. En effet, les techniques d'adaptation d'un ordonnancement sont relativement différentes selon la nature du problème et le type de perturbation considéré : pannes de ressources, variations de durées opératoires, arrivées impromptues de nouvelles tâches à prendre en compte, etc.

Les paragraphes suivants proposent de balayer, en tentant d'illustrer chacune des classes précédentes, quelques travaux de la littérature proposant une approche prédictive-réactive.

2.2.2 Quelques approches

La technique triviale de décalage à droite de tâches est discutée dans [Smith 95]. Elle est appliquée pour résoudre les incohérences, provoquées par une perturbation, liées au respect des contraintes de temps et de ressource. Un décalage pouvant impliquer le non-respect d'autres contraintes, il est nécessaire d'en propager les effets sur les tâches situées en aval de celle décalée jusqu'à recouvrer la faisabilité (on suppose ici l'absence de circuit dans le graphe potentiels-tâches associé au problème). Cette méthode de ré-ordonnancement permet de reconstruire très rapidement un ordonnancement cohérent suite à des aléas. Cependant, elle peut conduire à la création de temps d'inactivités sur les ressources, et donc à une dégradation conséquente de la performance.

Une technique de ré-ordonnancement total est proposée dans [Snoek 01]. Le problème considéré est un job shop et les perturbations concernent l'arrivée imprévisible de travaux. L'objectif est de minimiser le retard moyen des travaux. La méthode de ré-ordonnancement proposée est de type événementiel : un ré-ordonnancement est exécuté à chaque arrivée de

nouveaux travaux. Pour pallier la complexité en temps de calcul du ré-ordonnancement total, une décomposition du problème est réalisée : l'horizon d'ordonnancement est décomposé en n périodes ce qui amène les auteurs à considérer n sous-problèmes, qu'ils résolvent en séquence. Une approche génétique est utilisée pour résoudre chaque sous-problème. Afin de permettre une intégration plus aisée des travaux, une anticipation des marges disponibles pour la réalisation de chaque sous-problème est également réalisée.

Dans leur article, Sabuncuoglu et Bayiz proposent, outre l'état de l'art déjà évoqué plus haut, une méthode prédictive-réactive pour les problèmes de type job shop. Les critères considérés sont le retard moyen et la durée totale. Les perturbations correspondent à des pannes de ressources. Dans une phase hors ligne, un ordonnancement de référence est généré par une procédure, basée sur une recherche arborescente tronquée. L'ordonnancement est partiel dans la mesure où sa longueur est limitée à un horizon plus court que l'horizon total. Lors de la phase en ligne, un ordonnancement partiel est périodiquement reconstruit pour tenir compte des pannes de machines. La période de ré-ordonnancement n'est pas fixe : un ré-ordonnancement a lieu à chaque dépassement d'un seuil relatif à la charge de travail réalisée par les ressources depuis le dernier ré-ordonnancement. Entre deux ré-ordonnancements, la technique de décalage à droite est utilisée, en cas de perturbation, pour conserver la faisabilité de l'ordonnancement. Par simulation, les auteurs étudient ensuite l'influence de la période de ré-ordonnancement et de la longueur de l'horizon d'ordonnancement, vis-à-vis des performances, en comparant notamment leur approche avec une méthode réactive pure utilisant les régles de priorité MWR (Most Work Remaining) et SPT (Shortest Processing Time) pour faire face aux incertitudes générées de façon aléatoire.

Dans [Akturk & Gorgulu 99], les auteurs proposent une méthode prédictive-réactive, utilisant la technique " match-up rescheduling" lors de la phase de révision de l'ordonnancement de référence. Cette méthode est appliquée dans le cadre de problèmes de type flow shop modifié. Le flow shop modifié est une extension du flow shop classique dans laquelle les gammes des travaux sont éventuellement différentes, mais compatibles avec une séquence imposée de machines. Les incertitudes sont relatives à des pannes de ressources (la durée de la panne est connue). À chaque fois qu'une panne de machine se produit, un ré-ordonnancement est lancé. La technique utilisée consiste tout d'abord à choisir un instant particulier dans l'ordonnancement de référence, appelé *match-up point*, à partir duquel le ré-ordonnancement doit redevenir identique à l'ordonnancement de référence. Le ré-ordonnancement est ensuite lancé sur l'intervalle de temps défini par l'instant courant et le match-up point. Les objectifs considérés sont la stabilité (le match-up point doit être le plus tôt possible) et d'autre part, la minimisation des retards induits (entre la nouvelle solution et celle de référence). La méthode décrite par les auteurs est basée sur l'itération de deux étapes. Lors de la première étape un point de match-up est défini pour la ressource défectueuse puis, l'ensemble des travaux et ressources devant être ré-ordonnancés est construit. La seconde étape ordonnance en premier les travaux de la ressource défectueuse (considérée goulot), par utilisation d'une procédure par séparation et évaluation utilisant un résultat de dominance, puis les travaux sur les ressources amont et aval. En

cas d'infaisabilité (i.e. il n'est pas possible d'atteindre le point de match-up dans les temps impartis), une procédure d'élargissement des pools de travaux et de ressources à considérer est proposée, et les deux étapes sont ré-itérées. Les auteurs mettent en particulier en évidence l'importance d'avoir un ordonnancement de référence flexible pour obtenir de bons résultats.

Dans la même orientation de ré-ordonnancement stable, El Sakkout, Richards et Wallace proposent dans [Sakkout *et al.* 97, Sakkout *et al.* 98] une méthode pour minimiser la perturbation du ré-ordonnancement vis-à-vis de l'ordonnancement de référence. La perturbation est mesurée par la somme des différences entre les dates de début et de fin des tâches des deux ordonnancements. En minimisant cette déviation, on peut maintenir la stabilité de l'ordonnancement. Les problèmes considérés sont relativement quelconques du fait d'une modélisation générique sous forme de problème de satisfaction de contraintes dynamique. Les incertitudes concernent les variations de capacité des ressources. Remarquons que la stabilité d'un ordonnancement est abordée de façon générale dans [Sotskov 98], pour des incertitudes concernant les durées opératoires, où les auteurs produisent une analyse de sensibilité fondée sur le concept de rayon de stabilité.

2.2.3 Discussion

Les méthodes prédictives-réactives, lorsqu'elles peuvent être mises en œuvre, sont généralement plus performantes que les méthodes réactives pures. Pour améliorer encore ces performances, il est cependant possible de tenter d'anticiper les perturbations, soit lors de la phase d'ordonnancement hors-ligne, soit lors de la réaction en ligne, en tirant par exemple parti d'informations disponibles sur les incertitudes. Nous qualifions ces méthodes de proactives-réactives.

2.3 Approches proactives-réactives

2.3.1 Généralités

La proactivité désignant la volonté d'anticiper les perturbations avant qu'elles ne se produisent, nous distinguons trois types d'approches selon que :

- l'anticipation a lieu lors de la phase hors-ligne et consiste à construire une solution robuste ;
- l'anticipation a lieu lors de la phase hors-ligne et vise à construire une famille flexible de solutions ;
- l'anticipation a lieu lors de la phase d'adaptation en ligne d'un ordonnancement de référence.

Remarquons que la majorité des approches présentées dans cette partie s'intéresse soit à la proactivité hors-ligne, soit à la proactivité en ligne. Rares sont celles qui considèrent le processus complet d'ordonnancement où la proactivité concerne à la fois l'ordonnancement hors-ligne et celui en ligne.

2.3.2 Proactivité par construction d'un ordonnancement robuste

Nous nous intéressons ici à des approches élaborant une solution d'ordonnancement robuste non flexible (cf. figure 2.4). Par non-flexible, nous entendons que la flexibilité temporelle, naturellement contenue dans la solution, n'est pas volontairement augmentée pour faire face aux incertitudes. Ces approches supposent généralement qu'un modèle des incertitudes est disponible et il s'agit de construire une solution "bonne" relativement aux différents scénarios caractérisés par ce modèle. Nous distinguons dans la suite du texte plusieurs catégories d'approches que nous illustrons, non exhaustivement, par quelques travaux s'y rattachant.

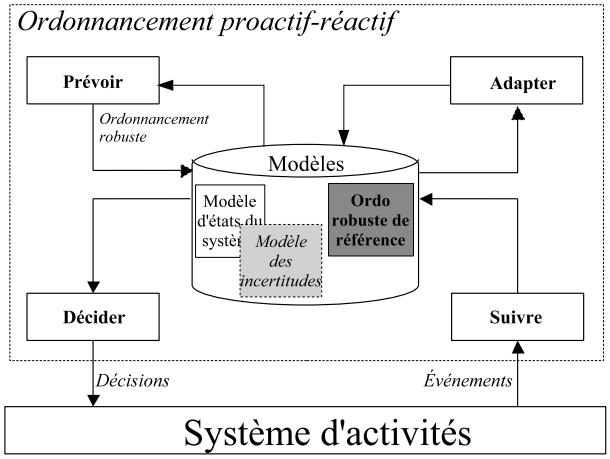


FIG. 2.4: Proactivité par construction d'un ordonnancement robuste

Un première catégorie d'approches utilise une représentation par scénarios des incertitudes (les scénarios étant soit énumérés soit caractérisés), dans le but d'optimiser la performance en moyenne ou au pire d'une solution. Les incertitudes considérées concernent généralement les durées opératoires.

Dans [Kouvelis *et al.* 00], les auteurs considèrent par exemple le problème de flow shop de permutation à deux machines avec minimisation du makespan. Les incertitudes sur les durées opératoires de tâches sont considérées explicitement lors de la phase statique pour construire un ordonnancement de qualité robuste. Les durées des tâches sont variables et représentées par des scénarios, chaque scénario correspondant à un ensemble de durées

opératoires et possédant une probabilité positive de réalisation. L'objectif est de minimiser le *regret*, c'est-à-dire l'écart de performance sur l'ensemble des scénarios entre la solution robuste choisie et la solution optimale déterminée à l'aide de la règle de Johnson pour un scénario donné. Ce problème est démontré NP-difficile. Une procédure par séparation et évaluation est proposée, ainsi qu'une heuristique afin de construire une séquence robuste, au sens de la minimisation du regret.

Des résultats proches sont présentés dans [Daniels & Kouvelis 95] et [Daniels & Carrillo 97]. Dans le premier article, le problème à une machine, où l'incertitude des durées opératoires est modélisée à l'aide d'intervalles, est abordé. L'objectif est de minimiser la date de fin moyenne des travaux. Dans le deuxième, les auteurs cherchent un ordonnancement possédant une probabilité maximale de produire une performance supérieure ou égale à un certain seuil pré-déterminé.

Un autre résultat proche est celui présenté dans [Yang & Yu 02] où les auteurs modélisent également les incertitudes sur les durées opératoires par un ensemble fini de scénarios. L'objectif est de minimiser la date de fin moyenne des travaux dans le cas d'un problème à une machine. Les auteurs montrent que le problème est NP-difficile et proposent un algorithme exact de programmation dynamique pour le résoudre efficacement. Des heuristiques sont également proposées pour réduire la complexité temporelle. Des résultats expérimentaux sont produits pour des instances de 10 à 20 travaux et les auteurs montrent que la complexité augmente avec le nombre de scénarios.

Une autre catégorie d'approches s'intéresse à la détermination d'un ordonnancement robuste au sens de l'évaluation de la fonction objectif considérée. Contrairement à la catégorie précédente, il n'est pas nécessaire ici de disposer d'un ensemble de scénarios, ni de disposer d'une solution optimale pour chaque scénario.

Une approche de ce type est présentée dans [Jensen 01]. L'auteur définit un ordonnancement robuste comme un ordonnancement conservant une bonne performance lorsque l'environnement fluctue. Les incertitudes considérées sont relatives aux pannes de machine. Une méthode est proposée, basée sur la qualité du voisinage d'un ordonnancement s, dont l'objectif n'est plus de minimiser le makespan, mais de minimiser une mesure de robustesse, définie sur le voisinage $N(s)$ de la solution s : $R_{C_{\max}(s)} = \sum(1/card(N(s)).C_{\max}(s')$ avec $s' \in N(s)$. Cette mesure de robustesse est intégrée au sein d'un algorithme génétique multi-critère.

Un résultat proche est celui proposé par Sevaux et Sörensen [Sevaux & Sörensen 04] où un algorithme génétique est également décrit, les incertitudes concernant les dates de disponibilité des tâches. La fonction d'évaluation de chaque individu x est définie par une somme pondérée des évaluations obtenues pour un ensemble d'individus $x + \delta_i$, où δ_i correspond à une perturbation appliquée sur les paramètres du problème.

Une dernière catégorie d'approches s'intéresse à déterminer une solution robuste par le biais de l'analyse de sensibilité [Hall & Posner 04]. L'analyse de sensibilité s'intéresse à la question "Que se passerait-il si ...?" qui se pose lorsque les paramètres d'un problème sont susceptibles de changer. On s'intéresse généralement à caractériser pour une solution un périmètre d'insensibilité. C'est ce périmètre que l'on cherche à maximiser pour obtenir une solution robuste.

Un exemple de ce type d'approche est présenté dans [Burns *et al.* 99] dans le cas d'un problème d'ordonnancement mono-processeur. Les auteurs introduisent la notion de garantie probabiliste pour une tolérance aux fautes en temps réel (Fault-tolerant real-time systems). Il s'agit de satisfaire les délais de toutes les tâches en présence d'incertitudes. Cette garantie probabiliste est une garantie d'ordonnançabilité avec une probabilité associée. Une garantie de 99% ne signifie pas que 99% des tâches vont satisfaire leurs délais, mais plutôt que dans 99% des exécutions possibles de cet ordonnancement, toutes les tâches vont satisfaire leurs délais. Burns et al. utilisent l'analyse d'ordonnançabilité et l'analyse de sensibilité pour calculer la fréquence maximale de fautes. Le modèle de fautes est ensuite utilisé pour affecter une probabilité à cette valeur.

Une autre approche de ce type est présentée dans [Penz *et al.* 01] pour le problème d'ordonnancement à machines parallèles. Les durées opératoires sont incertaines et sont modélisées selon la forme $(1+\epsilon_i)p_i$, où $\epsilon_i \in]-1, +\infty[$ représente le pourcentage de confiance qui peut être accordé à p_i. Les auteurs définissent une garantie d'insensibilité relativement à un algorithme d'ordonnancement hors-ligne donné et à une perturbation sur les valeurs des ϵ_i. Une approche semblable est également décrite dans [Rossi 02].

2.3.3 Proactivité par construction d'un ordonnancement flexible

Dans le cadre d'un ordonnancement proactif-réactif, un autre type d'approches s'intéresse non pas à fournir une solution robuste, mais à déterminer un ensemble flexible de solutions, la flexibilité étant ici vue comme un moyen de faire face aux perturbations. En effet, avoir à disposition un ordonnancement flexible permet, lors de l'ordonnancement en ligne, de disposer d'un ensemble de choix possibles, de qualité éprouvée, afin d'absorber au mieux les aléas. Ce type d'approche est illustré sur la figure 2.5. Nous distinguons deux catégories d'approches selon que la flexibilité volontairement insérée est de nature temporelle ou séquentielle.

2.3.3.1 Méthodes fournissant de la flexibilité temporelle

Dans [Chiang & Fox 90] et [Gao *et al.* 95], les auteurs proposent d'augmenter la flexibilité de l'ordonnancement initial en utilisant des protections temporelles. Le problème considéré est de type job shop et les incertitudes sont relatives aux pannes de machines qui

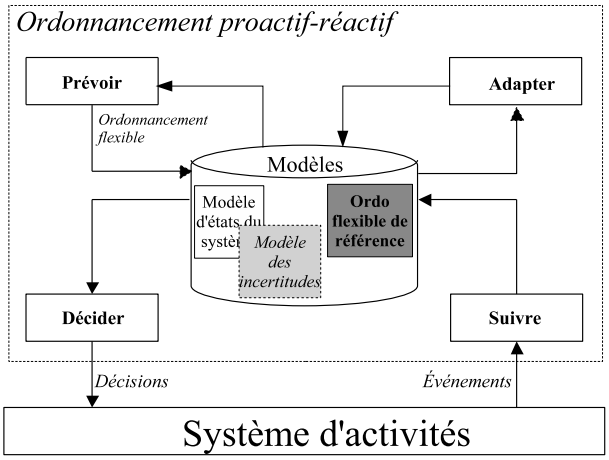


FIG. 2.5: Proactivité par construction d'un ordonnancement flexible

peuvent influencer les durées opératoires des tâches. Les durées opératoires des tâches utilisant ces ressources sont volontairement augmentées en fonction des durées et des fréquences des pannes. L'ordonnancement est ensuite construit en tenant compte des nouvelles durées.

Dans [Davenport *et al.* 01], les auteurs proposent une amélioration de la technique précédente, permettant d'éviter que les marges temporelles résultant de l'insertion de protections temporelles soient perdues en cas d'absence de pannes. Ils introduisent deux nouvelles techniques : TWS (Time Window Slack) et FTWS (Focused Time Window Slack). Plutôt que d'étendre les durées opératoires des tâches, la technique TWS modifie la définition du problème pour assurer que chaque tâche aura au moins une quantité spécifique de marge de protection, cette marge étant prise dans une marge générale partagée par toutes les tâches du problème. La technique FTWS vise à mieux répartir au fil de l'ordonnancement la quantité de marge réservée en tenant compte des probabilités de pannes. Ainsi, si une tâche est séquencée tard, alors la probabilité qu'une perturbation survienne avant son exécution est plus grande, donc la tâche a besoin de plus de marge. Les résultats expérimentaux montrent que ces techniques sont plus performantes que celles consistant à insérer des protections temporelles ajoutées aux durées opératoires.

Dans [Leon *et al.* 94], les auteurs étudient le problème de job shop dont le critère à optimiser est le makespan. Les incertitudes prises en compte sont également relatives aux pannes de machines (les lois de distribution des temps d'arrivée et des durées de pannes sont supposées connues). Le but des auteurs est de maintenir une bonne performance de l'ordonnancement prévisionnel (calculé hors ligne) en présence d'incertitudes et en utilisant la technique de "décalage à droite " pour la phase réactive. Pour construire un ordonnancement robuste, ils incorporent une expression de robustesse dans la fonction d'évaluation. Trois mesures de la robustesse sont développées et évaluées, dont en particulier la mesure robuste $RM3$ basée sur la valeur moyenne de la marge. Soit S un ordonnancement prévisionnel spécifiant la séquence des tâches et $M0(S)$ le makespan de

l'ordonnancement prévisionnel (sans perturbation). L'expression de RM3 est la suivante : $RM3 = M0(S) - \sum_{i \in N_f} slack_i/|N_f|$ où N_f est l'ensemble des activités i exécutées sur des machines pouvant tomber en panne. Un algorithme génétique est ensuite développé pour construire un ordonnancement robuste dont la fonction objectif est une combinaison linéaire de $RM3$ et de la marge moyenne.

Dans [Mehta & Uzsoy 98] et [Mehta & Uzsoy 99], les auteurs proposent d'insérer des temps morts (*idle time*) dans l'ordonnancement de référence, pour les problèmes $J||L_{\max}$ et $1||\sum T_i$ respectivement. Supposant les distributions des fréquences et durées de pannes connues, ils cherchent à minimiser les avances ou retards des tâches relativement à un ordonnancement déterministe prédéterminé, tout en maintenant une performance. Les résultats montrent que l'insertion des temps morts peut considérablement améliorer la performance de l'ordonnancement prédictif en présence des incertitudes sans trop dégrader la qualité de l'ordonnancement réellement exécuté.

Tavares et al. [Tavares *et al.* 98] proposent une méthode pour gérer le risque dans le cadre de la gestion de projet. Il s'agit de prendre en compte les incertitudes relatives aux durées opératoires et aux coûts. Selon les auteurs, ces deux types d'incertitudes sont antagonistes puisque commencer une activité à sa date de début au plus tôt (au plus tard) réduit (augmente) le risque de retard mais augmente (réduit) le coût d'une instabilité : un compromis est donc nécessaire. La notion de facteur de flottement est proposée pour aider le manager du projet dans ce compromis. Ce facteur de flottement α permet de contrôler la date de début d'un ensemble de tâches : $s_i(\alpha) = es_i + \alpha(ls_i - es_i)$. En suivant cette approche, Herroelen et Leus dans [Herroelen & Leus 04a] proposent, à la place d'un facteur α commun à toutes les tâches, d'utiliser un facteur pour chaque tâche i, déterminé selon les activités précédant ou succédant i. Les résultats expérimentaux montrent une bonne amélioration de performance par rapport à la méthode précédente.

Basée également sur l'insertion de temps morts (appelés tampons ou buffers), la méthode CC/BM (Critical Chain Scheduling and Buffer Management), proposée initialement par Goldratt [Goldratt 97], a suscité beaucoup d'intérêt. L'idée centrale de cette méthode consiste à insérer des buffers temporels, à des endroits déterminés par la topologie du chemin critique, afin de protéger l'ordonnancement de référence face à des perturbations. Trois types de buffer sont utilisés : le *project buffer* ajouté à la fin du chemin critique, les *feeding buffers* insérés chaque fois qu'un arc issu d'une tâche non critique rejoint une tâche critique, et les *resource buffers* insérés chaque fois que deux tâches consécutives du chemin critique n'utilisent pas la même ressource. L'utilisation de ces buffers est illustrée sur la figure 2.6 extraite de [Herroelen & Leus 01]. Sur cet exemple, la chaîne critique correspond au chemin $Start - 1 - 2 - 4 - End$. Le conflit de ressource entre les tâches 1 et 2 est résolu arbitrairement ainsi que matérialisé par l'arc en pointillé. Un *project buffer* a été placé à la fin du projet. Un *feeding buffer* a été inséré entre les tâches 3 et *End*. Un *resource buffer* est placé entre les tâches 2 et 4, toutes deux critiques mais n'utilisant pas les mêmes ressources ; le rôle de ce dernier étant de signaler à la ressource Y, avant que la tâche 2 ne

se termine, de se tenir prête à réaliser l'activité 4. L'expérience montre que l'utilisation de buffers constitue une façon simple de protéger un ordonnancement face aux incertitudes. Cependant, la taille de chaque buffer est souvent surestimée puisqu'elle est fixée par la règle des 50 % (la taille du buffer de projet est égale à 50 % de la durée totale du projet, celles des buffers de feeding et de ressource égales à la moitié du chemin le plus long conduisant à l'activité précédant le buffer). De plus, en pratique, les emplacements d'insertion de buffer ne sont pas toujours pertinents. Les avantages et faiblesses de la méthode CC/BM, ainsi que certaines améliorations potentielles, sont discutés en détail dans [Herroelen & Leus 01].

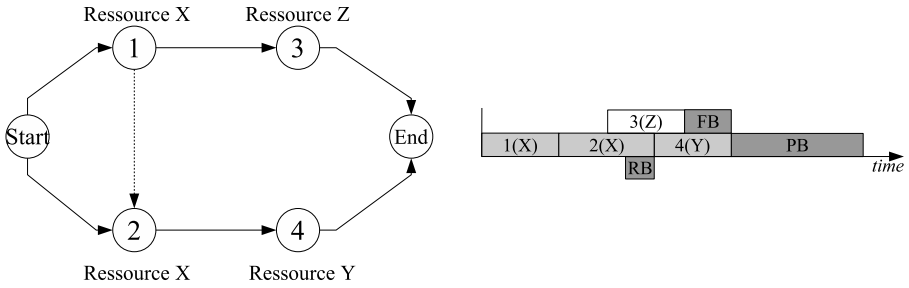


FIG. 2.6: Un ordonnancement de chaîne critique

2.3.3.2 Méthodes fournissant de la flexibilité séquentielle

Si l'exécution d'une tâche est retardée du fait que les composants nécessaires à sa réalisation ne sont pas disponibles, ou du fait qu'une autre tâche la précédant est elle même en retard, la flexibilité temporelle proactivement insérée dans l'ordonnancement peut ne pas suffire à absorber la perturbation. Or, en pratique, au lieu d'attendre qu'une tâche devienne prête au risque de laisser des ressources oisives trop longtemps, il est souvent envisageable de lancer l'exécution d'une autre tâche, dans la mesure où cette réaction ne détériore pas la performance globale. Ce type de réaction est souvent exclu par les méthodes fournissant exclusivement de la flexibilité temporelle puisque l'ordre des tâches est imposé sur chaque ressource dans la solution qu'elles fournissent. Pour pallier cet inconvénient, quelques travaux se sont intéressés à la conception de méthodes hors-ligne intégrant de la flexibilité séquentielle au sein de la solution d'ordonnancement qu'elles proposent. Ici aussi, la flexibilité séquentielle est vue comme un moyen de faire face aux perturbations. Il s'agit de fournir ou de caractériser un ensemble flexible de solutions, de performance acceptable, exploitables en ligne pour réagir au mieux aux fluctuations de l'environnement.

L'approche proposée dans [Drummond *et al.* 94] se situe dans cette catégorie. Les auteurs présentent un algorithme, nommé Just-In-Case scheduling (JIC), pour une application d'observation astronomique par télescope. Il s'agit d'un problème à une machine, dont la seule ressource est le télescope, les tâches correspondant aux actions d'observation dont les durées sont incertaines. Chaque observation doit s'exécuter, pour des raisons de condi-

tions expérimentales, dans un intervalle de temps défini par l'utilisateur. L'algorithme JIC prend en entrée un ordonnancement déterministe et le "robustifie" en prenant en compte les incertitudes sur les durées pour identifier les points potentiels d'incohérence. Si la probabilité qu'une incohérence se produise est forte, alors un ordonnancement alternatif est déterminé pour les actions restantes. Tout ordonnancement alternatif peut lui même être reconsidéré, selon la même méthode, et peut ainsi donner naissance à d'autres ordonnancements alternatifs. Plusieurs expérimentations ont été effectuées montrant des résultats encourageants. Néanmoins, cette méthode devient naturellement fortement combinatoire dès que le nombre de tâches augmente.

Pour pallier la complexité liée à l'énumération de plusieurs ordonnancements, d'autres travaux s'intéressent à l'utilisation d'*ordres partiels*. Brièvement, un ordre partiel permet de caractériser, sans les énumérer, un ensemble de solutions. Nous revenons sur cette notion au cours du troisième chapitre.

Dans [Wu *et al.* 99], les auteurs considèrent le problème de job shop avec minimisation de la somme pondérée des retards en présence d'incertitudes. La méthode proposée, baptisée PFSL (Preprocess First Schedule Later), consiste à fixer de façon hors-ligne une partie des décisions de séquencement, en laissant libres certains choix, résolus en ligne, au fur et à mesure de l'exécution de l'ordonnancement. Elle se compose de deux phases : la phase de décomposition (ou preprocess) et la phase d'ordonnancement en ligne qui utilise des règles de priorité. La phase de décomposition utilise une méthode par séparation et évaluation qui partitionne l'ensemble des tâches en sous-ensembles ordonnés (i.e si le sous-ensemble s précède le sous-ensemble t alors toute tâche de s précède toute tâche de t), puis à l'intérieur de chaque sous-ensemble, certaines relations d'ordre sont imposées afin de maîtriser la performance au pire. Une expérimentation de cette méthode, dans un contexte où les durées opératoires sont perturbées, montre que PFSL est mieux adaptée que certaines méthodes statiques (IATC [Byeon *et al.* 93]) ou réactives pures (règle ATC [Vepsalainen & Morton 87]) dans des cas où le niveau d'incertitudes est petit ou moyen. Cependant cette méthode est de complexité exponentielle et donc limitée par la taille du problème à résoudre.

Dans [Moukrim *et al.* 03], les auteurs abordent, suivant la même logique, le problème d'ordonnancement à machines parallèles identiques et temps de communication incertains. Un temps de communication entre deux tâches i et j, reliées par la contrainte $i \prec j$, est pris en compte lorsque deux tâches ne s'exécutent pas sur la même machine. Les temps de communication pris en compte hors-ligne sont estimés, les valeurs réelles dépendant du contexte courant d'exécution. La méthode proposée se compose de trois étapes. La première détermine hors ligne une affectation des tâches aux processeurs sur la base des estimations des temps de communication. La deuxième fixe un ordre partiel en considérant les tâches affectées sur chaque machine. Enfin, la dernière étape utilise en ligne certaines règles de priorité, pour établir un ordonnancement complet, en cohérence avec l'affectation et l'ordre partiel obtenus dans les deux premières étapes. Les résultats expérimentaux montrent une

meilleure performance en comparaison avec une approche réactive pure.

Une autre approche est celle développée au LAAS-CNRS (Laboratoire d'Analyse et d'Architecture des Systèmes), il y a plus de vingt cinq ans, basée sur le concept de groupes de tâches. L'idée générale de cette approche est de construire, durant la phase hors ligne, un ordonnancement partiel en déterminant pour chaque ressource, une séquence de groupes de tâches, les tâches d'un même groupe étant totalement permutables. Un avantage de cet ordre partiel réside dans la possibilité de pouvoir évaluer en temps polynomial la performance au pire de l'ordonnancement (c'est-à-dire la permutation la plus défavorable vis-à-vis de la minimisation du plus grand retard algébrique). La méthode utilise une heuristique qui construit une séquence de groupes initiale, puis une méthode de type tabou est utilisée pour améliorer les performances. Cet ordonnancement est ensuite exploité en temps réel : le séquencement des tâches d'un même groupe est défini, au fur et à mesure de l'exécution de l'ordonnancement, par un décideur qui dispose d'indicateurs quant à la qualité de chaque choix vis-à-vis de la performance. De nombreux travaux ont été développés autour de cette approche, baptisée ORABAID (ORdonnancement d'Atelier Basée sur une AIde à la Décision) [Demmou 77, Thomas 80, Le Gall 89, Billaut 93, Artigues 97]. Les articles [Artigues & Roubellat 01, Artigues *et al.* 02, Esswein *et al.* 05] fournissent une description synthétique de cette approche et nous invitons le lecteur à s'y référer pour plus de détails. Nous reviendrons néanmoins sur quelques caractéristiques de cette dernière au cours du troisième chapitre.

S'inspirant de l'ordre partiel précédent, Aloulou et al. proposent dans [Aloulou *et al.* 02] et [Aloulou 02] une approche proactive-réactive, de type génétique, pour le problème d'ordonnancement multicritères à une machine . Le modèle proposé est constitué de deux algorithmes : un proactif et un réactif. L'algorithme proactif a pour but de construire, en reprenant le concept de groupes et en limitant les possibilités de permutation au sein d'un groupe, un ensemble de solutions. L'algorithme réactif guide, en fonction de la politique retenue de pilotage (actif, semi-actif, sans-délai, ...), et des événements issus de l'environnement, les choix non-résolus de séquencement des tâches d'un même groupe.

Une autre approche intéressante, utilisant le concept de groupes de tâches permutables, est proposée dans [Esswein 03]. L'idée originale de cette approche réside dans le fait que la flexibilité séquentielle est insérée a posteriori au sein de l'ordonnancement (ce qui permet de définir une performance au pire *et* au mieux) : une solution initiale est considérée, idéalement optimale, puis les groupes sont créés dans un deuxième temps, en maîtrisant les éventuelles dégradations de performance. Pour cela plusieurs algorithmes bi-critères, combinant un objectif de minimisation du nombre de groupes (i.e. de maximisation de la flexibilité séquentielle) et un objectif de minimisation de la dégradation de performance, sont proposés. Plusieurs types de problème sont abordés : les problèmes à une machine, à deux machines, de flowshop, de job shop, etc. Des résultats expérimentaux sont présentés montrant que l'utilisation d'un ordonnancement de groupes est bénéfique en moyenne, la qualité de l'ordonnancement finalement exécuté s'avérant supérieure à celle d'un ordon-

nancement non flexible, face à des incertitudes de nature variable (variation de durées opératoires, de dates de disponibilité, de dates de fin, ...).

Remarquons que les approches basées sur le concept de groupes de tâches permutables n'utilisent pas de modèles d'incertitudes. D'un certain côté, il s'agit d'un avantage puisqu'elles sont de ce fait très génériques et permettent de faire face à des aléas très variés. Néanmoins, cela constitue aussi un inconvénient puisqu'il n'est pas possible de préciser vis-à-vis de quoi et dans quelle mesure une solution est robuste. Insérer de la flexibilité séquentielle dans une solution permet donc de créer une robustesse "potentielle", mais il n'est pas possible de garantir que cette robustesse sera a priori suffisante, en regard d'un ensemble de scénarios de perturbations.

Bien que ne revendiquant pas explicitement leur caractère robuste, une autre catégorie d'approches mérite d'être citée dans cette partie comme cela est souligné dans [Briand *et al.* 05a]. Il s'agit des approches d'ordonnancement par propagation de contraintes qui permettent de déterminer s'il existe des solutions satisfaisant les contraintes d'un problème (c'est-à-dire de savoir si le problème est ou non consistant), sans pour autant les énumérer. Pour cela, les mécanismes de propagation de contraintes caractérisent un espace de solutions par un processus de déduction logique exprimant des conditions nécessaires d'admissibilité basées sur la considération des contraintes temporelles et de ressources [Esquirol *et al.* 01]. La propagation permet soit de réduire des domaines des variables temporelles, soit de déduire des affectations de tâches interdites, soit de définir des relations d'ordre partiel entre les tâches (précédences obligatoires, précédences interdites, non-insérabilité d'une tâche dans un ensemble), soit encore de déterminer des distances minimales entre certaines variables temporelles. Les mécanismes de propagation permettent donc d'avoir une caractérisation plus précise du problème considéré. Ils peuvent être inclus au sein d'une méthode de résolution et évitent de nombreuses tentatives de résolution vouées à l'échec. Ils vérifient également des exigences de robustesse car, d'une part, ils aboutissent à un ensemble de solutions (sans les énumérer explicitement) en ayant supprimé certaines incohérences du problème et d'autre part, ils sont bien adaptés à une évolution réactive du modèle qui se traduit par l'ajout ou la suppression d'une ou plusieurs contraintes. Dans le premier cas, les solutions déjà obtenues doivent être remises en cause mais les déductions issues des propagations déjà effectuées, elles, peuvent être conservées. Dans le second cas, les déductions issues des propagations déjà effectuées sont remises en cause, mais on peut conserver les solutions (une solution du problème initial est aussi une solution du problème relâché).

2.3.4 Proactivité lors de la phase d'adaptation en ligne d'un ordonnancement de référence

Une autre approche proactive-réactive envisageable consiste à placer la proaction, non essentiellement dans la phase de détermination d'un ordonnancement de référence, mais

lors de la phase d'adaptation en ligne. Cette approche est illustrée sur la figure 2.7.

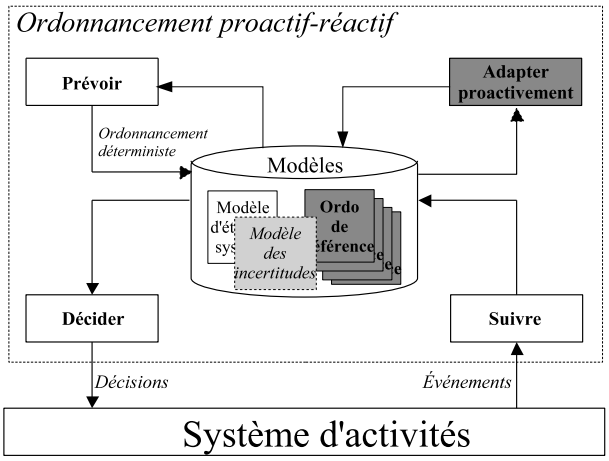


FIG. 2.7: Proactivité lors de la phase d'adaptation en ligne

Bidot et al. présente par exemple dans [Vidal *et al.* 03] et [Bidot 03], une approche de ce type, appliquée à la gestion de projet, assez proche de l'approche JIC de Drummont. Dans une première phase, une solution d'ordonnancement déterministe est déterminée hors-ligne prenant en compte les durées moyennes des tâches et minimisant le makespan sur un horizon donné. Cette solution sert de référence mais, au fur et à mesure de l'exécution de l'ordonnancement, de nouveaux ordonnancements sont développés proactivement, "au cas où", pour faire face à des perturbations dont la probabilité d'occurrence augmente dangereusement.

Si une perturbation survient, pour laquelle un ordonnancement avait été préparé "au cas où", on passe de la configuration courante à une nouvelle. Les différents contextes d'exécution sont modélisés au sein d'un automate à états. L'automate est développé petit à petit pendant l'exécution en fonction de l'évolution des probabilités d'occurences des événements et des événements déjà apparus. Un contexte d'exécution est défini par un réseau de contraintes, un ensemble de données et un algorithme d'exécution. Le passage d'un contexte à un autre se produit lors d'un franchissement de bornes temporelles (passage d'un horizon d'ordonnancement à un nouveau), lorsque des contraintes temporelles sont violées ou lorsque la performance se dégrade au delà d'un seuil.

2.4 Synthèse

Dans ce chapitre, nous avons souligné qu'un ordonnancement réactif doit, pour posséder la capacité de s'adapter au caractère incertain de son environnement d'application,

gérer les événements contingents issus de cet environnement, signalant les dérives des paramètres internes ou externes qui caractérisent le problème. Il s'agit d'une problématique de robustesse devant être abordée dans l'optique d'une recherche de compromis robustesse / performance : la recherche d'un ordonnancement "résistant" à l'incertain conduisant en effet à une dégradation de performance, et inversement. Nous avons balayé les méthodes d'ordonnancement robustes selon qu'elles manipulent ou non un ordonnancement de référence et, lorsque c'est le cas, par le fait qu'elles tentent d'anticiper ou non, de façon proactive, l'existence d'incertitudes. Cette distinction a conduit à considérer trois classes d'approches : les approches réactives, les approches prédictives-réactives et les approches proactives-réactives.

Les approches réactives présentent l'avantage de construire en temps réel, et de façon très souple, une solution faisable en prenant en compte des incertitudes de nature très variées. Cependant, elles fournissent en général un ordonnancement de performance faible, qu'il n'est pas possible d'anticiper du fait qu'on ne connaît l'ordonnancement qu'une fois celui-ci réalisé.

Les approches prédictives-réactives manipulent de façon explicite un ordonnancement de référence, qui est adapté, au fur et à mesure de son exécution, de façon continue, périodique ou événementielle, en procédant à des ré-ordonnancements locaux, partiels ou totaux. La stabilité de ce ré-ordonnancement est souvent recherchée afin de ne pas perturber outre mesure l'organisation initiale. De telles approches, cherchant à maîtriser les dégradations de performance induites par les aléas de mise en œuvre, nous paraissent relativement saines. De plus, contrairement aux approches réactives, l'existence d'un ordonnancement de référence permet aux décideurs d'avoir à tout instant une vision à moyen terme de l'organisation, ce qui permet d'améliorer ses conditions de mise en œuvre. Toutefois, l'ordonnancement de référence étant déterministe, il donne une vision optimiste de l'organisation qui ne tient pas compte des perturbations futures.

Afin de disposer d'une vision plus réaliste moins optimiste, les approches proactives-réactives tentent d'anticiper les fluctuations de l'environnement en construisant un ordonnancement qui intègre explicitement les incertitudes. Nous avons proposé de distinguer trois catégories d'approches proactives-réactives : celles qui, sur la base d'une modélisation des incertitudes, cherchent à produire une solution non-flexible robuste unique (la solution produite étant de qualité acceptable pour un ensemble de plusieurs scénarios de réalisation possibles), celles qui insèrent de la flexibilité temporelle et/ou séquentielle au sein d'une solution d'ordonnancement (la flexibilité étant vue comme un moyen de résister à l'incertain) et celles qui anticipent en ligne les perturbations, en fonction de leur probabilité d'occurrence, en développant de façon dynamique plusieurs ordonnancements "de secours".

Les travaux développés dans ce manuscrit s'intéressent à la classe des approches proactives-réactives qui nous semble la plus adaptée pour faire face aux incertitudes. L'intérêt est particulièrement porté sur les approches fournissant de la flexibilité séquentielle, dans une

optique de recherche de compromis flexibilité / performance. En ce sens, nos travaux se rapprochent de ceux relatifs à l'utilisation du concept de groupe de tâches permutables (cf. partie 2.3.3.2), et plus particulièrement des travaux récents développés dans la thèse de doctorat de Carl Esswein [Esswein 03]. Notons que notre étude ne se focalisera pas sur la façon d'exploiter en ligne une famille flexible de solutions, cet aspect devant faire l'objet d'un prochain volet de recherche.

Notre ambition est de tenter de dépasser certaines limites propres à l'utilisation du concept de groupe de tâches permutables. En effet, bien que ce concept soit très riche, notamment du fait qu'il permet de déterminer une performance au pire pour la famille d'ordonnancements caractérisée, la flexibilité qu'il est possible d'exhiber est relativement limitée puisque seules des tâches contiguës, affectées à une même ressource, peuvent être permutées. Par exemple, il n'est pas possible de caractériser à l'aide d'une seule séquence de groupes deux solutions, de la forme $i \prec S \prec j$ et $j \prec S \prec i$, qui ne se différencieraient que par une permutation de deux tâches i et j, autour d'un ensemble ordonné de tâches S, i, j et S utilisant la même ressoure R. Or, ce type de permutation est fréquent lorsqu'on tente de représenter un ensemble de solutions optimales. Un premier objectif est donc de définir un ordre partiel exhibant davantage de flexibilité, tout en conservant la possibilité de déterminer une performance au pire.

Un autre inconvénient des approches fondées sur le concept de groupe de tâches permutables réside dans le fait qu'elles ne prennent pas en compte a priori de modèles d'incertitudes. Autrement dit, il n'est pas possible de garantir a priori que la flexibilité séquentielle exhibée sera suffisante pour garantir la robustesse vis-à-vis d'un ensemble de scénarios potentiels de réalisation d'un ordonnancement. On peut seulement affirmer que l'on a plus de chance d'être robuste en ayant à disposition un ensemble flexible de solutions. Un deuxième objectif du travail présenté dans ce manuscrit est de proposer quelques améliorations sur ce point. Pour cela nous montrons, pour des problèmes d'ordonnancement particuliers, comment le concept de *dominance* peut être exploité pour caractériser un ensemble flexible de solutions, sur la base d'un modèle d'incertitudes particulier.

Chapitre 3

Ordres partiels et structures d'intervalles

Ce chapitre présente quelques notions nécessaires à la compréhension des mécanismes mis en jeu dans les approches d'ordonnancement robuste que nous présentons dans les parties 3 et 4. Il s'intéresse tout d'abord à la notion d'*ordre partiel* et à son utilisation en ordonnancement. Les notions de suffisance, de nécessité ou de dominance d'un ordre partiel sont en particulier présentées. Dans un deuxième temps, la notion de *structure d'intervalles*, ainsi que les concepts de *sommet*, *base* et *pyramides* s'y rattachant, sont décrits. Ce sont ces concepts que nous utilisons en vue de caractériser des ordres partiels intéressants vis-à-vis de certains problèmes d'ordonnancement.

3.1 Notion d'ordre partiel

3.1.1 Définition d'un ordre partiel

Nous donnons ici une définition de la notion d'ordre partiel, telle qu'elle est présentée dans [Cheng *et al.* 02], ainsi qu'un rappel de quelques propriétés des ordres partiels.

Un ordre partiel P est caractérisé par une paire $P = (X, \leq_P)$, X étant un ensemble d'éléments, où toute relation $\leq_P$ sur $X \times X$ est réflexive, antisymétrique et transitive. Pour $u, v \in X$, $u \leq_p v$ est interprété comme u est plus petit ou égal à v. De la même façon, $u <_P v$ signifie que, $u \leq_p v$ et $u \neq v$. Un élément v est *minimal* (respectivement *maximal*) dans P s'il n'existe aucun élément u tel que $u \leq_p v$ (respectivement $v \leq_p u$).

Un ordre partiel $P = (X, \leq_p)$ est un *ordre total* (aussi appelé *ordre linéaire* ou *ordre complet*) si pour tout couple $(u, v) \in X \times X$ la relation $u \leq_p v$ ou la relation $v \leq_p u$ est vérifiée.

Étant donnés deux ordres partiels $P = (X, \leq_P)$ et $Q = (X, \leq_Q)$ sur le même ensemble

X, Q est une *extension* de P (ou P est un *sous-ordre* de Q) si $u <_P v$ implique $u <_Q v$ pour tout couple $(u, v) \in X$. Un ordre linéaire $Q = (X, \leq_Q)$ est une *extension linéaire* d'un ordre partiel $P = (X, \leq_P)$ si Q étend P.

L'*intersection* de deux ordres partiels $P_1 = (X, \leq_{P_1})$ et $P_2 = (X, \leq_{P_2})$ sur le même ensemble X est l'ordre partiel $P_1 \cap P_2 = (X, \leq_{P_1 \cap P_2})$ où $u \leq_{P_1 \cap P_2} v$ si et seulement si $u \leq_{P_1} v$ et $u \leq_{P_2} v$, $\forall u, v \in X$.

Un sous-ensemble $I \subseteq X$ est un *idéal* de P si pour tout couple (u, v) où $v \in I$ et $u \in X$ tels que $u \leq_P v$, alors on a $u \in I$. Un sous-ensemble $F \subseteq X$ est un *filtre* de P si pout tout couple (u, v) où $u \in F$, $v \in X$ tels que $u \leq_P v$, alors on a $v \in F$. Pour tout $u \in X$, l'*idéal principal* $I(u)$ est défini par $I(u) = \{v \in X | v \leq_P u\}$ et le *filtre principal* $F(u)$ est défini par $F(u) = \{v \in X | u \leq_P v\}$.

3.1.2 Intérêt des ordres partiels en ordonnancement

Les ordres partiels permettent de caractériser des ensembles flexibles de solutions, en évitant l'explosion combinatoire liée à leur énumération. C'est pourquoi ils sont souvent utilisés en ordonnancement par exemple au sein de méthodes par séparation et évaluation progressive pour représenter les noeuds intermédiaires d'une arborescence de recherche, dans les approches par contraintes pour représenter un ensemble de solutions non démontré incohérent, ou encore pour introduire de la flexibilité séquentielle au sein d'une solution d'ordonnancement.

Un ordre partiel présente parfois une structure particulière qui favorise le calcul de performance au mieux ou au pire de l'ensemble de solutions qu'il caractérise. Cette propriété est particulièrement intéressante puisque, les solutions caractérisées ne possédant pas nécessairement des performances identiques du point de vue de la fonction objectif considérée, savoir déterminer une performance au mieux et / ou au pire, en un temps de calcul acceptable, peut s'avérer appréciable du point de vue de l'optimisation, notamment au sein de procédures par séparation et évaluation progressive. Il devient ainsi possible de comparer deux ordres partiels en fonction de leur performance.

Une autre propriété intéressante liée à la structure d'un ordre partiel réside dans la possibilité de calculer rapidement le nombre de solutions caractérisées. Cette propriété est particulièrement intéressante en conjonction avec la précédente, dans l'optique d'une recherche de compromis flexibilité / performance.

À titre d'exemple, le concept de *groupe de tâches*, déjà évoqué dans le chapitre 2, définit un ordre partiel qui possède les deux propriétés précédentes. Rappelons que ce concept permet de caractériser une famille de solutions en associant à chaque ressource une *séquence de groupes* [Demmou 77, Thomas 80]. Les groupes sont ordonnés mais les tâches de chaque groupe sont totalement permutables (ou partiellement permutables [Aloulou 02,

Wu *et al.* 99]), ce qui définit bien un ordre partiel. La structure particulière de cet ordre partiel rend possible, en temps polynomial, le calcul de la pire des solutions contenue dans l'ensemble, vis-à-vis d'un critère régulier de type *minmax* ($L_{\max}$, $C_{\max}$, ...). En effet, l'existence de groupes induit l'ajout de contraintes de précédence particulières dans le graphe potentiels-tâches associé au problème. Ainsi, pour chaque couple de tâches i et j, reliées par une contrainte de gamme opératoire de type $i \prec j$, on ajoute la contrainte $s_j - s_i \geq \overline{\delta}_{ij}$, où $\overline{\delta}_{ij}$ est déterminé en fonction de la permutation la plus défavorable pour i au sein du groupe auquel i appartient. Une contrainte similaire est également ajoutée entre chaque couple de tâches i et j, appartenant à deux groupes G_i^k et G_j^k consécutifs de la ressource k. La performance au pire des cas peut alors être déterminée par un calcul de plus long chemin dans le graphe [Thomas 80]. Du fait de cette propriété, cet ordre partiel peut être utilisé soit pour calculer en ligne une séquence de groupes [Billaut 93, Le Gall 89] de performance satisfaisante, soit pour introduire de la flexibilité séquentielle au sein d'une solution initiale (éventuellement optimale) [Esswein 03]. Notons qu'insérer de la flexibilité à partir d'une solution optimale permet de garantir une performance au mieux optimale pour la séquence de groupes (ce qu'il est difficile de faire autrement). Comme il est également facile de dénombrer les solutions qu'une séquence de groupes caractérise, la recherche d'un compromis flexibilité / performance est aussi envisageable.

Pour illustrer cet ordre partiel, nous reprenons un exemple proposé dans [Esswein 03]. Il s'agit d'un problème de job shop à trois machines.

i	j	$M_{i,j}$	$p_{i,j}$
1	1	1	3
1	2	2	3
1	3	3	3
2	1	2	4
2	2	3	3
2	3	1	1
3	1	3	2
3	2	1	2
3	3	2	2

TAB. 3.1: Une instance de job shop à trois machines

La figure 3.1 représente un ordonnancement de groupes réalisable pour cette instance. Cet ordonnancement de groupes est constitué d'un total de 7 groupes : deux groupes de deux opérations (1,3) sur M_1, (1,2) sur M_3 et cinq groupes d'une seule opération (2) sur M_1, (2), (3), (1) sur M_2 et (3) sur M_3.

Si on choisit un ordre arbitraire des opérations au sein de chaque groupe, on obtient un des quatre ordonnancements de la figure 3.2.

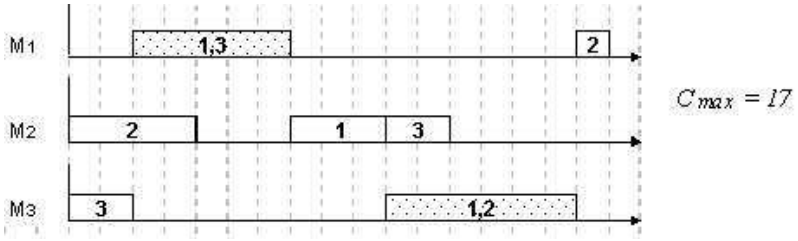


FIG. 3.1: Un ordonnancement de groupes

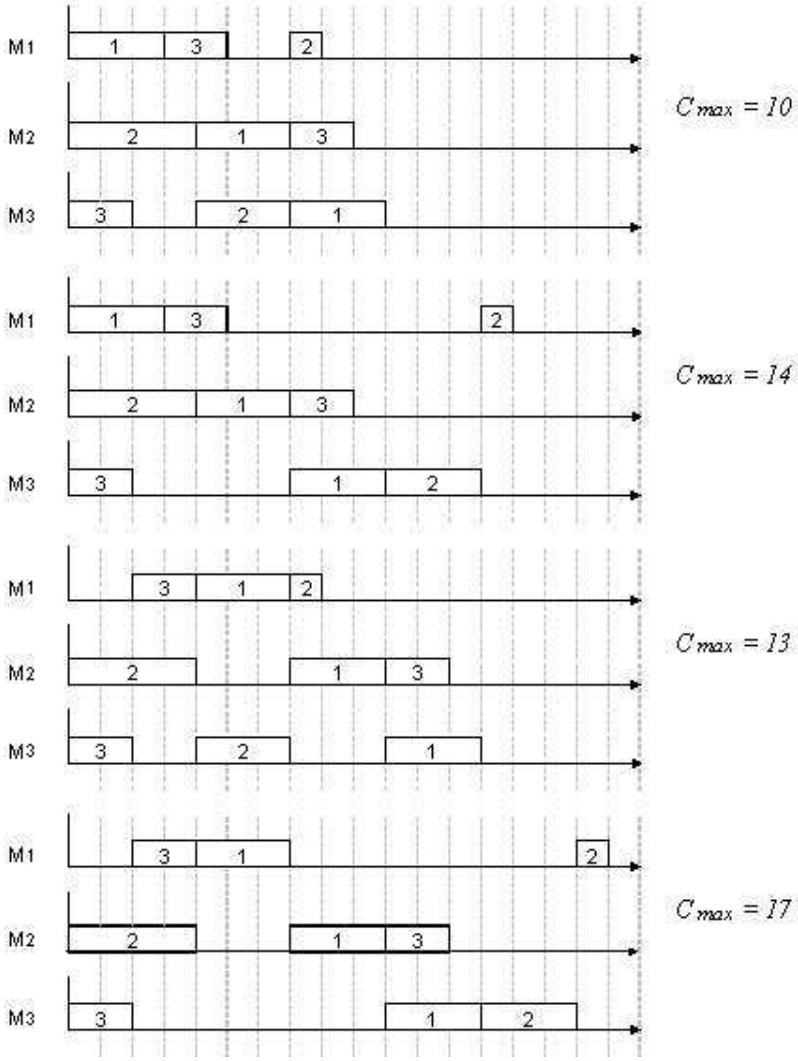


FIG. 3.2: Quatre ordonnancement réalisables

3.2 Ordre partiel nécessaire, suffisant et dominant en ordonnancement

3.2.1 Généralités

En ordonnancement, les notions de dominance et de conditions nécessaires et suffisantes, relatives soit à l'admissibilité d'un problème, soit à l'optimisation d'un critère, se traduisent souvent par la définition d'ordres partiels. Par extension, nous parlons alors d'ordre partiel nécessaire, suffisant ou dominant.

Dans le cadre de l'admissibilité, la recherche de conditions nécessaires et suffisantes par l'*analyse sous contraintes* du problème vise à déterminer *toutes* les solutions satisfaisant les contraintes du problème. Elle est théoriquement possible, mais pose un problème de complexité [Erschler 76]. Elle a de ce fait donné lieu à deux problématiques de recherche distinctes, l'une concernant des conditions d'admissibilité suffisantes, l'autre des conditions d'admissibilité nécessaires.

Les approches par conditions *suffisantes* caractérisent un sous-ensemble de l'ensemble des ordonnancements admissibles (ou optimaux relativement à un critère donné). Les ensembles de solutions satisfaisant des conditions *nécessaires* comprennent tous les ordonnancements admissibles (resp. optimaux), plus certains ordonnancements non admissibles (resp. non optimaux).

D'autres approches caractérisent des ordonnancements suivant des conditions *dominantes* vis-à-vis de l'optimalité ou de l'admissibilité. Dans le cadre de l'étude de l'admissibilité, une séquence Seq_1 domine une séquence Seq_2 si et seulement si Seq_2 admissible implique Seq_1 admissible [Erschler *et al.* 83]. Les séquences dominantes peuvent être considérées comme potentiellement meilleures que les autres dans le sens où, si aucune n'est admissible, alors cela implique qu'il n'existe pas de solution admissible pour le problème considéré.

Dans le cadre de l'optimisation et pour certains critères, la définition de la dominance fait intervenir la notion de circuits dans des graphes conjonctifs de type $G(X,U)$, où X désigne l'ensemble des tâches associées au problème à ordonnancer et U, l'ensemble des arcs conjonctifs reliant deux tâches du problème [Roy 70]. Considérons deux séquences Seq_1 et Seq_2 décrites respectivement par les graphes conjonctifs $G_1(X,U_1)$ et $G_2(X,U_2)$. La séquence Seq_1 domine la séquence Seq_2 si et seulement tout circuit de $G_2(X,U_2)$ est de longueur supérieure ou égale au plus long circuit de $G_1(X,U_1)$ [Fontan 80].

3.2.2 Concept de corps d'hypothèses

Lorsque l'on recherche des conditions nécessaires, suffisantes ou dominantes d'admissibilité ou d'optimalité, il est intéressant de préciser le *corps d'hypothèses* que l'on considère, c'est-à-dire l'ensemble des informations sur les données et leurs relations que l'on suppose disponibles pour la définition des conditions [Fontan 80].

Souvent, les conditions d'admissibilité ou d'optimalité nécessitent une connaissance précise des valeurs des paramètres d'un problème (dates de disponibilité r_i, dates de fin d_i, durées opératoires des tâches p_i). Le corps d'hypothèses considéré est alors dit *complet*. D'autres conditions ne considèrent pas les valeurs explicites des paramètres, mais plutôt les relations d'ordre existant entre celles-ci. On parle alors de *corps d'hypothèses restreint* ou *partiel*. Dans [Fontan 80], sont distingués six corps d'hypothèses principaux, présentés dans le tableau 3.2 suivant :

corps d'hypothèses	*dates de début et de fin*	*durées opératoires*
CH0	connues	connues
CH1	ordre relatif des dates	inconnues
CH2	ordre relatif des intervalles	inconnues
CH3	connues	inconnues
CH4	ordre relatif des dates	ordre relatif
CH5	connues	ordre relatif

TAB. 3.2: Quelques corps d'hypothèses

Sur ce tableau, seulement le corps d'hypothèses CH0 est complet, tous les autres sont restreints avec certains éléments inconnus ou partiellement connus. Remarquons que l'utilisation de corps d'hypothèses restreints pour la détermination de conditions nécessaires, suffisantes ou dominantes est particulièrement intéressante dans une optique de robustesse. En effet, les paramètres r_i, p_i, d_i étant incertains, il est avantageux de disposer de conditions dont les déductions seront relativement insensibles aux perturbations sur ces paramètres (contrairement aux déductions déterminées par considération d'un corps d'hypothèses complet). En revanche, ces déductions seront évidemment plus "pauvres" que celles que l'on pourrait élaborer par considération d'un corps d'hypothèses complet.

3.2.3 Ordre partiel nécessaire

Les conditions nécessaires d'admissibilité sont principalement utilisées dans les approches d'ordonnancement par contraintes. De telles conditions permettent en effet de déduire de nouvelles contraintes qui peuvent être propagées dans le but d'élaguer l'espace de recherche en restreignant les possibilités d'affectation des tâches aux ressources, les fenêtres temporelles d'exécution des tâches et les possibilités de séquencement entre tâches. Ces conditions nécessaires sont généralement élaborées sur la base d'un corps d'hypothèses complet.

La caractéristique principale des approches par contraintes est une séparation claire du modèle, des méthodes d'analyse et des méthodes de résolution. Dans ce type d'approche, un problème est représenté par l'ensemble des variables et l'ensemble des contraintes portant sur ces variables (domaines, contraintes binaires, etc.). Avec un tel formalisme, le critère est intégré en tant que variable contrainte. Par exemple, pour un problème de minimisation, on pose $expression\_critère < valeur\_critère$, où $expression\_critère$ est une combinaison des variables du problème et $valeur\_critère$ correspond à une valeur imposée de la performance au pire des solutions caractérisées. On peut ainsi utiliser cette formulation du critère pour résoudre itérativement un problème d'optimisation, en faisant progressivement décroître la valeur de la performance au pire.

Les mécanismes de propagation de contraintes permettent de déterminer s'il existe des solutions satisfaisant les contraintes d'un problème (c'est-à-dire de savoir si le problème est ou non cohérent) sans pour autant les énumérer. Pour cela, ces mécanismes caractérisent un espace de solutions par un processus de déduction logique exprimant des conditions nécessaires d'admissibilité. Ainsi, toute solution supprimée de l'espace obtenu est effectivement incohérente (les mécanismes sont dits *sains*). En revanche, une solution appartient à cet espace parce qu'elle n'a pas été montrée incohérente par propagation, mais elle peut néanmoins l'être puisque les mécanismes de propagation sont *incomplets* dans le cas général. Pour certains problèmes, par exemple les *problèmes temporels simples* [Dechter *et al.* 91], on dispose de conditions nécessaires et suffisantes d'admissibilité ; dans ce cas, l'espace de solutions obtenu correspond à l'espace des solutions admissibles.

À partir d'un ensemble de contraintes C, toute condition nécessaire d'admissibilité contribue à la définition d'un nouvel ensemble de contraintes C' incluant C. Une fois ces conditions nécessaires d'admissibilité obtenues, le processus de propagation peut être itéré sur le nouvel ensemble de contraintes dont on dispose et ce jusqu'à ne plus obtenir de nouvelle déduction ou à aboutir à la détection d'une incohérence. En fin de propagation, on dispose alors, pour chaque variable temporelle et pour chaque variable d'affectation, d'un domaine de valeurs non démontrées impossibles et pour chaque ressource, d'un ordre partiel pour la réalisation des tâches (ensemble de séquencements non démontrés impossibles).

Les conditions nécessaires d'admissibilité permettent classiquement d'imposer des ordres

partiels nécessaires entre les tâches. Hormis les relations d'ordre entre deux tâches ($i \prec j$), il faut remarquer que ces ordres partiels ne sont pas mémorisés et sont directement interprétés en termes d'ajustement des fenêtres temporelles associées aux variables. On distingue trois types d'ordre partiel :

1. *précédences obligatoires entre une tâche i et un ensemble de tâches S* : $i \prec S$ ou $S \prec i$;
2. *précédences interdites entre une tâche i et un ensemble de tâches S* : $i \not\prec S$ ou $S \not\prec i$;
3. *non-insérabilité d'une tâche i dans un ensemble de tâches S* : $i \nmid S$;

Notons que la non-insérabilité d'une tâche dans un ensemble est quant à elle utilisée en complément de la condition de tâche non première ou non dernière dans un ensemble afin de déduire des conditions de séquencement obligatoire entre une tâche et un ensemble :

$$\text{si } i \nmid S \text{ et } i \not\prec S \text{ alors } S \prec i \; ; \quad \text{si } i \nmid S \text{ et } S \not\prec i \text{ alors } i \prec S.$$

À titre d'exemple, nous décrivons ici une condition nécessaire d'admissibilité, basée sur la propagation des contraintes de ressource, conduisant à déduire des relations de précédence interdites entre une tâche i et un ensemble de tâches S. On suppose ici que i et S sont inclus dans un ensemble disjonctif maximal (c'est-à-dire un ensemble dont tout couple de tâches est une paire de disjonction). La règle est indiquée ci-dessous, où $\underline{s}_i$ désigne la date de début au plus tôt de i et $lst(S)$ représente une borne maximale de la date de début au plus tard sur l'ensemble des séquences admissibles de S (elle est établie en relâchant les contraintes de début au plus tôt des tâches de S). Cette règle permet de déduire qu'il existe au moins une tâche $s \in S$ telle que $s \prec i$.

$$\text{si } lst(S) - \underline{s}_i < p_i \text{ alors } i \not\prec S$$

Une règle symétrique existe permettant de déduire $S \not\prec i$, manipulant la date de fin au plus tard $\overline{f}_i$ de i et la fonction $eft(S)$, correspondant à une borne minimale de la date de fin au plus tôt sur l'ensemble des séquences admissibles de S. Et de même, si $S \not\prec i$ alors il existe au moins une tâche $s \in S$ telle que $i \prec s$.

Ces déductions peuvent entraîner des ajustements de la fenêtre temporelle $[\underline{s}_i, \overline{f}_i]$ ainsi que décrit dans [Levy 96, Torres & Lopez 00].

Pour plus de détail sur les techniques de propagation de contraintes en ordonnancement, le lecteur peut se référer à [Esquirol *et al.* 01, Carlier & Pinson 89, Le Pape 91, Carlier & Pinson 94, Nuijten 94, Baptiste *et al.* 01].

3.2.4 Ordre partiel suffisant

L'utilisation de conditions suffisantes d'admissibilité ou d'optimalité en ordonnancement est généralement limitée à des problèmes d'ordonnancement simples. En effet, à moins qu'il n'existe pas de contraintes de distance maximale entre les variables temporelles (auquel cas tout ordonnancement satisfaisant les contraintes de distances minimales et les contraintes de ressource est admissible), il n'existe pas de condition suffisante d'admissibilité pouvant s'appliquer de façon générique à des problèmes quelconques.

Les problèmes étudiés dans le cadre de la recherche de conditions suffisantes sont généralement des problèmes à une ou deux ressources, les conditions suffisantes d'optimalité variant en fonction du critère considéré et de la nature du problème (flow shop, job shop, open shop). Elles se basent souvent sur la considération d'un corps d'hypothèses restreint. Il n'est pas dans notre intention de dresser ici la liste exhaustive de ces conditions suffisantes, celle-ci étant exposée dans tout ouvrage consacré à l'ordonnancement (voir par exemple [Esquirol & Lopez 99] ou [Pinedo 95]). À titre illustratif, nous avons toutefois choisi de présenter la célèbre *règle de Johnson* [Johnson 54], utilisée dans le cadre de problèmes flow shop de permutation à deux machines. On considère un ensemble T de n travaux à séquencer sur deux machines M_1 et M_2. Les durées opératoires du travail $j \in T$ sur chacune des deux machines sont notées p_{j1} et p_{j2} respectivement. Les travaux passent d'abord sur M_1, puis sur M_2, et les séquences des travaux sur les deux machines sont identiques. La règle de Johnson stipule que toute séquence de travaux vérifiant la condition $\min(p_{i1}, p_{j2}) \leq \min(p_{j1}, p_{i2}) \iff i \leq_J j$ est optimale vis-à-vis de la minimisation du $C_{\max}$. Cette règle définit un ordre partiel suffisant $J = (X, \leq_J)$ sur l'ensemble X des travaux, tel que tout ordre total P, extension linéaire de J, est optimal. N'étant basée que sur les relations d'ordre entre les durées opératoires, elle utilise donc un corps d'hypothèses restreint.

Travaux	1	2	3	4	5	6	7	8	9
p_{i1}	1	3	8	8	2	4	7	9	5
p_{i2}	6	8	12	2	4	5	7	11	3

TAB. 3.3: Un problème $F2|prmu|C_{\max}$

Considérons par exemple le problème représenté sur le tableau 3.3. L'ordre partiel défini par la règle de Johnson permet de caractériser trois solutions, $1 \prec 5 \prec 2 \prec 6 \prec 7 \prec 3 \prec 8 \prec 9 \prec 4$, $1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 7 \prec 8 \prec 9 \prec 4$, $1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 8 \prec 7 \prec 9 \prec 4$, dont la durée totale optimale est $C^*_{\max} = 59$. Notons que nous définirons pour ce problème, au cours de la partie 3, un nouvel ordre partiel suffisant, dont l'ordre partiel suffisant de Johnson est une extension.

3.2.5 Ordre partiel dominant

Rappelons qu'un sous-ensemble de solutions est dit dominant pour l'optimisation d'un critère donné, s'il contient au moins un optimum pour ce critère. De manière analogue, il est dit dominant par rapport à un ensemble de contraintes lorsque ce sous-ensemble contient au moins une solution admissible, s'il en existe [Esquirol & Lopez 99]. On peut donc dire qu'une solution dominante est en quelque sorte "plus admissible" (ou "plus optimale") que d'autres [Fontan 80].

Pour un problème d'optimisation, si on utilise la notion d'ordre partiel, on dit qu'un ordre partiel P, sur un ensemble de tâches d'un problème d'ordonnancement, est un *ordre partiel dominant* s'il existe une séquence optimale qui est une extension linéaire de cet ordre partiel P [Cheng *et al.* 02].

Contrairement au cas des conditions suffisantes, il existe des conditions dominantes génériques d'optimalité relatives à tout critère régulier. Ces conditions exploitent les notions d'ordonnancements semi-actifs et actifs et sont basées sur un corps d'hypothèses complet.

- Un ordonnancement est dit *semi-actif* si on ne peut pas avancer le début d'une tâche sans modifier la séquence sur la ressource. On peut montrer que l'ensemble des ordonnancements semi-actifs est dominant pour tout critère régulier.
- Un ordonnancement est dit *actif* si on ne peut pas commencer une tâche plus tôt sans reporter le début d'une autre. Notons qu'un ordonnancement actif est semi-actif et on peut également montrer que l'ensemble des ordonnancements actifs est dominant pour tout critère régulier.

La figure 3.3 illustre ces types d'ordonnancement, étant données cinq tâches 1, 2, 3, 4, 5 et les contraintes de précédence imposées $1 \prec 2 \prec 5$ et $3 \prec 4$.

Comme les ensembles d'ordonnancements semi-actifs et actifs sont très vastes, exploiter leur propriété de dominance au sein de procédures d'optimisation ne permet pas de limiter très efficacement l'espace de recherche. C'est pourquoi on préfère souvent, au détriment de la généricité, exploiter la structure particulière d'un problème pour déduire des conditions de dominance plus efficaces en termes de réduction de l'espace de recherche.

Par exemple, dans [Tadei *et al.* 98], les auteurs proposent une nouvelle procédure par séparation et évaluation progressive pour résoudre le problème $F2|r_i|C_{max}$ qui exploite des propriétés de dominance particulièrement efficaces. Les durées opératoires des n travaux à séquencer sur les machines M_1 et M_2 sont respectivement notées p_{i1} et p_{i2}. Les auteurs considèrent une séquence partielle σ, $C_1(\sigma)$ et $C_2(\sigma)$ sont respectivement les durées totales de la séquence partielle sur les machines M_1 et M_2, et A^σ et B^σ les chemins critiques correspondants. Ils définissent également, étant donnée une séquence partielle σ, une date de début au plus tôt actualisée $R_i = \max(r_i, C_1(\sigma))$ pour tout travail $i \notin \sigma$. Ils énoncent alors les quelques propriétés de dominance suivantes :

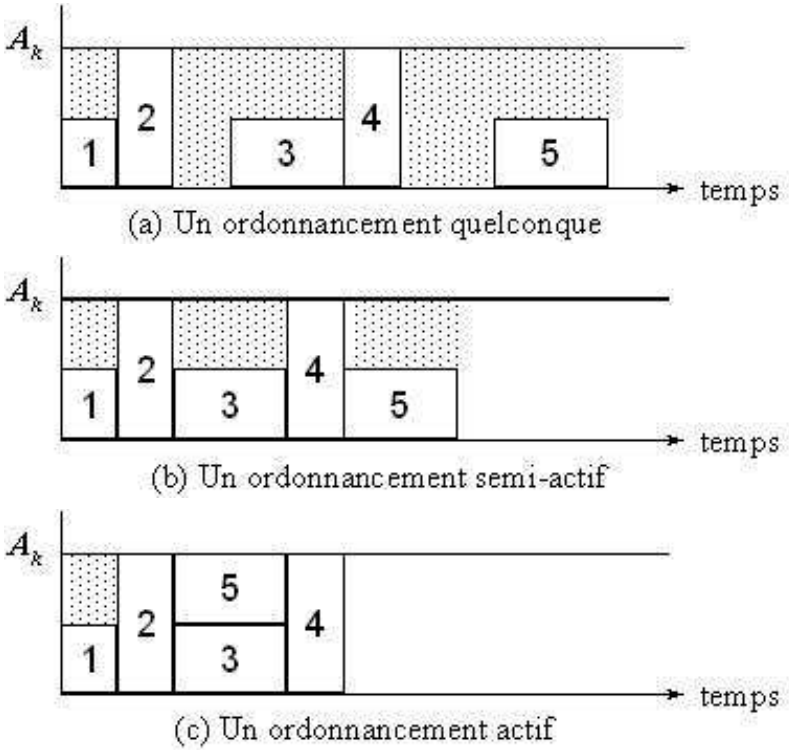


FIG. 3.3: Différents ordonnancements dominants

1. Si $r_i \leq r_j$ ET $p_{i1} \leq \min(p_{j1}, p_{i2})$ ALORS le travail i précède le travail j dans au moins une solution optimale.
2. Étant données deux séquences partielles $\sigma \prec i \prec j$ et $\sigma \prec j \prec i$, si $C_1(\sigma \prec i \prec j) \leq C_1(\sigma \prec j \prec i)$ *ET* $C_2(\sigma \prec i \prec j) \leq C_2(\sigma \prec j \prec i)$ *ALORS* $\sigma \prec j \prec i$ est dominée par $\sigma \prec i \prec j$.
3. σ étant une séquence partielle optimale et i étant un travail $\notin \sigma$, SI $r_i \leq \min_{k \notin \sigma} R_k$ ET $R_i + p_{i1} \leq C_2(\sigma)$ ALORS il existe au moins une solution optimale commençant par la sous-séquence $\sigma \prec i$.

3.3 Notion de structure d'intervalles

Cette partie décrit la notion de structure d'intervalles et les concepts qui lui sont rattachés. Ces notions sont en effet au cœur des approches fondées sur la manipulation d'ordres partiels pour la caractérisation de solutions que nous présentons dans les parties 2 et 3.

3.3.1 Structure d'intervalles et algèbre de Allen

Une structure d'intervalles est définie par un couple $< I, C >$ où $I = i_1, \ldots, i_n$ est un ensemble d'intervalles et C est un ensemble de contraintes sur $I \times I$. Chaque intervalle i_j est

défini par un couple de points x_j et y_j tel qu'une relation d'ordre $x_j \prec_R y_j$ quelconque soit vérifiée. Une contrainte entre deux intervalles i_j et i_k peut être exprimée soit en spécifiant un ordre total entre les points définissant ces intervalles soit, de façon équivalente, en utilisant une des treize relations de l'algèbre de Allen [Allen 91], récapitulées sur la figure 3.4 dans le cas où la relation d'ordre $\prec_R$ porte sur le temps. Remarquons que les six premières relations illustrées sur la figure 3.4 ont des symétriques, obtenus en inversant les intervalles A et B dans les relations.

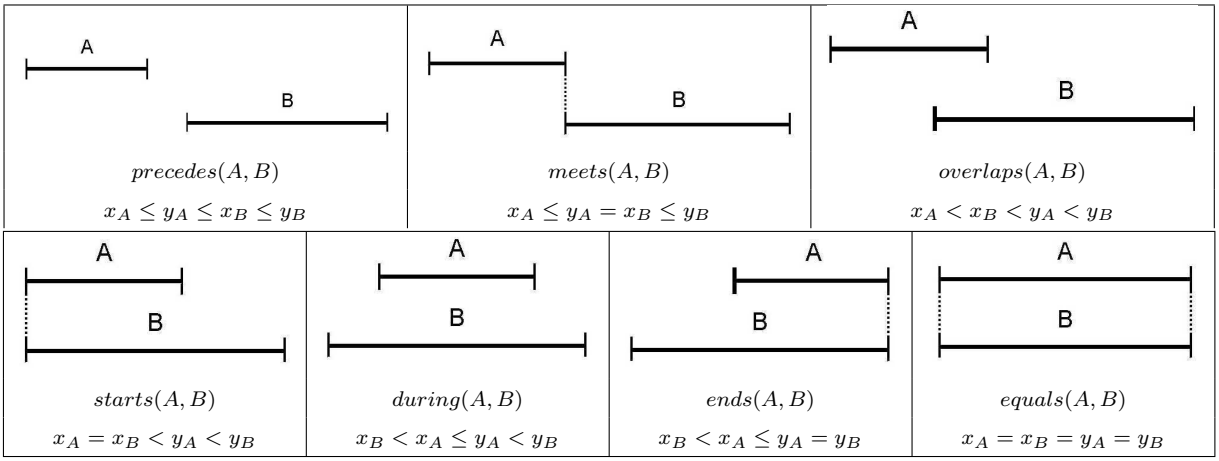


FIG. 3.4: Algèbre de Allen

3.3.2 Notions de sommet et de base

Par considération de la relation de Allen *during*, deux types particuliers d'intervalles peuvent être mis en évidence : l'intervalle de type *sommet* et celui de type *base*.

Définition 1. *Un intervalle s est dit sommet d'une structure d'intervalles $< I, C >$ si $\nexists i \in I$ tel que during(i, s).*

Définition 2. *Un intervalle b est dit base d'une structure d'intervalles $< I, C >$ si $\nexists i \in I$ tel que during(b, i).*

En utilisant ces notions de sommet et de base, les notions de *sommet-pyramide* (noté *s-pyramide*) et de *base-pyramide* (noté *b-pyramide*) peuvent être définies [Esquirol & Lopez 99].

3.3.3 s-pyramide et b-pyramide

Définition 3. *Une s-pyramide P_s, associée au sommet s d'une structure d'intervalles $< I, C >$, est le sous-ensemble d'intervalles $i \in I$ tel que during(s, i).*

Définition 4. *Une b-pyramide P_b, associée à la base b d'une structure d'intervalles $< I, C >$, est le sous-ensemble d'intervalles $i \in I$ tel que during(i, b).*

3.3.4 Exemple illustratif

Pour illustrer les quatre définitions précédentes, considérons la structure d'intervalles représentée sur la figure 3.5. Sur cet exemple, on identifie les sommets : C, D et E qui caractérisent respectivement les s-pyramides : $P_C = \{A, B, G\}$, $P_D = \{G\}$ et $P_E = \{F, G\}$. De même, on identifie les bases A, B, F et G qui caractérisent les b-pyramides correspondantes : $P_A = \{C\}$, $P_B = \{C\}$, $P_F = \{E\}$ et $P_G = \{C, D, E\}$. Remarquons qu'en cohérence avec la définition de la notion de pyramide, un sommet n'appartient pas à la pyramide qu'il caractérise.

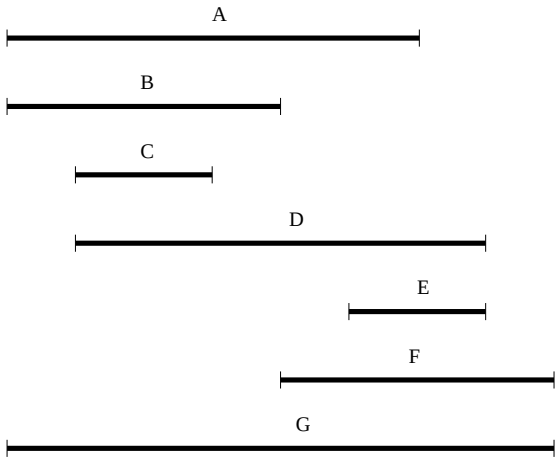


FIG. 3.5: Exemple d'une structure d'intervalles

3.3.5 Intérêt des structures d'intervalles

L'utilisation de structures d'intervalles permet une représentation intéressante des caractéristiques d'un problème. En ordonnancement, il est par exemple possible d'exprimer par un intervalle, les rangs possibles d'affectation d'une tâche sur une ressource, la structure d'intervalles ainsi obtenue étant exploitable du point de vue de l'amélioration des techniques de propagation de contraintes (voir par exemple [Levy 96]). De plus, comme nous le verrons dans les parties suivantes, une structure d'intervalles est bien adaptée à la représentation de corps d'hypothèses restreints, les sommets, bases et pyramides d'une structure pouvant être définis sans avoir connaissance des valeurs explicites des paramètres d'un problème.

Deuxième partie

CAS DU PROBLÈME À UNE MACHINE

Chapitre 4

Un ordre partiel dominant

Dans ce chapitre, nous décrivons un ordre partiel dominant pour le problème à une machine, défini par un théorème de dominance démontré dans les années quatre-vingts. Le problème à une machine est tout d'abord décrit, ainsi qu'une méthode efficace permettant de le résoudre, puis, le théorème de dominance est présenté. Nous présentons ensuite deux algorithmes permettant de borner respectivement la qualité au mieux et au pire de l'ensemble dominant caractérisé par ce théorème. Cette qualité est déterminée en construisant, pour chaque travail j, la séquence la plus favorable et celle la plus défavorable vis-à-vis du retard algébrique, conduisant respectivement aux retards algébriques au mieux (noté $L_j^{\min}$) et au pire (noté $L_j^{\max}$). Enfin, quelques réflexions sont proposées concernant l'insensibilité de l'ensemble dominant relativement aux incertitudes.

4.1 Description du problème et d'une méthode de résolution

4.1.1 Problème considéré

Nous considérons ici un problème V constitué d'un ensemble de travaux $T = \{1, 2, \ldots, n\}$ à ordonnancer sur une machine disponible en un seul exemplaire. Chaque travail j est défini par sa durée opératoire p_j, sa date de début au plus tôt r_j et sa date de fin au plus tard d_j. Le critère considéré est le plus grand retard algébrique $L_{\max}$ (la contrainte $f_i \leq d_i$ étant relâchée).

Ce problème, noté $1|r_i|L_{\max}$, a été démontré NP-difficile [Lenstra *et al.* 77]. Néanmoins, une solution optimale peut être obtenue par application de l'efficace PSEP de Carlier [Carlier 82], qui fournit en quelques secondes, une séquence optimale, pour des problèmes comptant plus de 1000 travaux.

La méthode arborescente de Carlier est conçue pour minimiser le makespan $C_{\max}$ dans un problème à une machine avec dates de début au plus tôt r_j et durées de latence q_j (q_j est

un temps incompressible entre la fin de j et la fin de l'ordonnancement). Toutefois, comme indiqué dans [McMahon & Florian 75] et [Levy 96], le problème $1|r_i|L_{\max}$ étant équivalent au problème avec durées de latence en imposant $q_i = \max_{j \in T} d_j - d_i$, la méthode de Carlier peut lui être appliquée. La partie suivante décrit cette méthode, celle-ci étant utilisée au chapitre 5.

4.1.2 La méthode de Carlier

La méthode de Carlier construit une arborescence dans laquelle chaque nœud est associé à un problème à une machine partiellement séquencé. Le choix des relations de précédence est guidé par l'algorithme de Schrage et exprimé par des actualisations de dates de début au plus tôt et de durées de latence.

L'algorithme de Schrage correspond à une reformulation de l'algorithme EDD (Earliest Due Date) de Jackson (les travaux sont séquencés en ordre croissant des d_j). Il sélectionne à chaque itération le travail prêt de plus grande durée de latence q_j. L'algorithme de Schrage est décrit sur la figure 4.1.

Proc SCHRAGE($solution, C_{\max}, r, q, p$)
  $C_{\max} \leftarrow 0$; $solution \leftarrow$ tableau vide à n positions ;
  $Fin \leftarrow 0$; $A\_sequencer \leftarrow T$;
  Pour i de 1 à n faire
    $Debut \leftarrow \max(Fin, min_{j \in A\_sequencer} r_j)$;
    $Solution[i] \leftarrow$ travail $k \in A\_sequencer$ prêt ($r_k \leq Debut$) de plus grande durée de latence ;
    $A\_sequencer \leftarrow A\_sequencer \backslash \{k\}$;
    $Fin \leftarrow Debut + p_k$;
    $C_{\max} \leftarrow max(C_{\max}, Fin + q_k)$;
  FinPour
FinProc

FIG. 4.1: Algorithme de Schrage

À partir de la séquence de Schrage, Carlier développe une arborescence de recherche où une évaluation optimiste du critère est associée à chaque nœud , *i.e.* une borne inférieure du makespan $C_{\max}$. Le calcul de cette borne est basé sur la formule suivante :

$$\forall T' \subset T, h(T') = \min_{i \in T'} r_i + \sum_{i \in T'} p_i + \min_{i \in T'} q_i$$

La séquence de Schrage associée à chaque nœud est modélisée par un graphe conjonctif dont les sommets correspondent aux travaux, plus deux sommets fictifs source (o) (marquant le début des travaux) et puits $(*)$ (marquant la fin des travaux). Pour chaque travail $i \in T$, l'arc (o, i) est valué par r_i, l'arc $(i, *)$ par $(q_i + p_i)$ et l'arc (i, j), tel que j suit immédiatement i dans la séquence de Schrage, par p_i. Soit $C = [i_1, \ldots, i_p]$ le *chemin critique* du graphe, *i.e.* le plus long chemin de (o) à $(*)$. Deux cas sont à considérer :

- Si $\forall i_k \in C \backslash \{i_p\}, q_{i_k} \geq q_{i_p}$, alors le nœud considéré ne peut pas être séparé et la séquence de Schrage est localement optimale ;

- Sinon, un *travail critique* J_c et un *ensemble critique de travaux* J sont définis :
  - J_c est le travail de $C \backslash \{i_p\}$ le plus proche du sommet $(*)$ de durée de latence inférieure à q_{i_p} ;
  - J est formé de la partie $[J_{c+1}, \ldots, i_p]$ du chemin C, où J_{c+1} est le travail qui suit J_c sur C.

Il est démontré dans [Carlier 82] que si le sommet considéré contient une solution optimale, elle est telle que, soit J_c est séquencé avant tous les travaux de J, soit tous les travaux de J sont séquencés avant J_c. L'arborescence est donc développée en considérant deux nouveaux problèmes partiellement ordonnancés :

- l'un où J_c est séquencé avant tous les travaux de J, ce qui s'obtient en modifiant la valeur de la durée de latence de façon suivante :

$$q_{J_c} \leftarrow max(q_{J_c}, \sum_{i_k \in J} p_{i_k} + q_{i_p});$$

- l'autre où J_c est séquencé après tous les travaux de J, ce qui s'obtient en modifiant la valeur de la date de début au plus tôt de façon suivante :

$$r_{J_c} \leftarrow max(r_{J_c}, \min_{i_k \in J} r_{i_k} + \sum_{i_k \in J} p_{i_k})$$

Le travail critique et l'ensemble critique des travaux sont par ailleurs utilisés pour déterminer une borne inférieure du makespan $C_{\max}$: l'ensemble T' de l'évaluation $h(T')$ est fixé à J pour le sommet père (avant séparation) et à $J \cup J_c$ pour les sommets fils (après la séparation).

Nous décrivons l'algorithme de Carlier sur la figure 4.2. Dans cet algorithme, N désigne l'ensemble des nœuds. À chaque nœud $\beta \in N$ est associé un triplet $(f_\beta, r^\beta, q^\beta)$, où f_β représente l'évaluation du nœud β, r^β et q^β sont les dates de début au plus tôt et les durées de latences actualisées selon l'ordonnancement partiel du nœud β. La valeur du makespan

Proc CARLIER($solution, L_{\max}, r, d, p$)
$d_{\max} \leftarrow max_{j \in T} d_j$; Pour i de 1 à n faire $q_i \leftarrow d_{\max} - d_i$; FinPour ; **(1)**
SCHRAGE($meilleur\_sol, F, r, q, p$) ;
rechercher le chemin critique $C = [i_1, \ldots, i_p]$ associé à $meilleur\_sol$;
Si $min_{i_j \in C} q_{i_j} = q_{i_p}$ alors $sommet\_separable \leftarrow$ **faux** ;
Sinon
déterminer J et J_c ; $f_o \leftarrow h(J)$; $N \leftarrow o(f_o, r, q)$;
$sommet\_separable \leftarrow$ **vrai** ; $\beta \leftarrow o$;
FinSi
TantQue $sommet\_separable =$ **vrai** faire
à partir du sommet β : **(2)**
• considérer le problème β_1 où $J_c \prec J$:
$r^{\beta_1} \leftarrow r^{\beta}$; $q^{\beta_1} \leftarrow q^{\beta}$; $q^{\beta_1}_{J_c} \leftarrow max(q^{\beta_1}_{J_c}, \sum_{i_j \in J} p_{i_j} + q^{\beta_1}_{i_p})$;
$f_{\beta_1} \leftarrow max[f_\beta, h(J \cup J_c)]$; Si $f_{\beta_1} < F$ alors $N \leftarrow N \cup \beta_1$; FinSi $(*)$
• considérer le problème β_2 où $J \prec J_c$:
$r^{\beta_2} \leftarrow r^{\beta}$; $q^{\beta_2} \leftarrow q^{\beta}$; $r^{\beta_2}_{J_c} \leftarrow max(r^{\beta_2}_{J_c}, \min_{i_j \in J} r^{\beta_2}_{i_j} + \sum_{i_j \in J} p_{i_j}$;
$f_{\beta_2} \leftarrow max[f_\beta, h(J \cup J_c)]$; Si $f_{\beta_2} < F$ alors $N \leftarrow N \cup \beta_2$; FinSi $(*)$
$N \leftarrow N \backslash \beta$;
$parcours \leftarrow$ **vrai** ;
TantQue $N \neq \emptyset$ **et** $parcours =$**vrai** faire **(3)**
rechercher le sommet β d'évaluation f_β minimale ;
SCHRAGE($sol\_sch, C\_sch, r^\beta, q^\beta, p$) ;
Si $C\_sch < F$ alors
$F \leftarrow C\_sch$; $meilleure\_sol \leftarrow sol\_sch$;
PourTout $\nu \in N$ faire
Si $f_\nu > F$ alors $N \leftarrow N \backslash \nu$; FinSi
FinPourTout
FinSi
rechercher le chemin critique $C = [i_1, \ldots, i_p]$ associé à sol_sch ;
Si $min_{i_j \in C} q_{i_j} = q_{i_p}$ alors $sommet\_separable \leftarrow$ **faux** ; $N \leftarrow N \backslash \beta$;
Sinon
déterminer J et J_c ; $f_\beta \leftarrow max(f_\beta, h(J))$; $(*)$
$parcours \leftarrow$**faux** ; $sommet\_separable \leftarrow$ **vrai** ;
FinSi
FinTantQue
FinTantQue
$solution \leftarrow meilleure\_sol$; $L_{\max} \leftarrow F - d_{\max}$;
FinProc
$(*)$ Le calcul de $h(J \cup J_c)$ et de $h(J)$ est effectué avec les dates de début au plus tôt et les durées de latence du sommet considéré.

FIG. 4.2: Algorithme de Carlier

$C_{\max}$ pour la meilleure solution connue constitue une borne supérieure du makespan ; elle est mémorisée et notée F. Trois phases (marquées **(1)**, **(2)** et **(3)**) apparaissent dans le déroulement de la procédure : la première correspond à l'initialisation de l'arborescence, la seconde au développement de l'arborescence à partir du nœud courant et la troisième au parcours de l'arbre pour la recherche du prochain nœud à séparer. La solution optimale est atteinte quand il ne reste plus de nœud séparable dans l'arborescence ; elle correspond alors à la meilleure solution connue.

La méthode de Carlier détermine une solution optimale unique. Dans l'optique de proposer une solution robuste, incorporant de la flexibilité séquentielle, le chapitre 5 propose une nouvelle méthode, basée sur un théorème de dominance, démontré dans les années quatre-vingt ([Couzinet-Mercé 79], [Erschler *et al.* 83], [Erschler *et al.* 85]). Par considération d'un corps d'hypothèses restreint, ce théorème permet de définir un ordre partiel dominant qui caractérise un ensemble de séquences dominantes vis-à-vis de l'admissibilité, ou de l'optimalité (pour un critère de retard de type $T_{\max}$ ou $L_{\max}$). La partie suivante décrit ce théorème.

4.2 Le théorème des pyramides

4.2.1 Généralités

Précisons tout d'abord que le théorème des pyramides a été formulé, en dehors de tout contexte de recherche de robustesse, dans le but premier de limiter la complexité algorithmique liée à la vérification de la consistance d'un problème d'ordonnancement (existence de solutions admissibles).

Le corps d'hypothèses restreint que les auteurs étudient en vue de la dominance prend en compte uniquement l'ordre relatif des dates de début au plus tôt r_j et de fin au plus tard d_j de l'ensemble T des n travaux. Les durées opératoires p_j ainsi que les valeurs explicites des r_j et d_j de chaque travail $j \in T$ ne sont donc pas considérées. Les résultats suivants restent alors valides quelles que soient les durées opératoires et quelles que soient les valeurs de r_j et d_j dans le respect de l'ordre relatif retenu dans le corps d'hypothèses.

Pour caractériser l'ensemble de séquences dominantes, les notions de structure d'intervalles, de sommet et de s-pyramide sont utilisées (voir chapitre 3). On considère la structure d'intervalles $< I, C >$ où un intervalle $i_j \in I$, caractérisé par une date de début r_j et une date de fin d_j (i.e. $i_j = [r_j, d_j]$), est associé à chaque travail $j \in T$ du problème.

Les sommets sont supposés indicés dans l'ordre croissant de leurs r_j, ou en cas d'égalité, dans l'ordre croissant de leurs d_j. Ceci est équivalent à affirmer que, s_α et s_β étant deux sommets de la structure d'intervalles, alors $\alpha < \beta$ si et seulement si $r_{s_\alpha} \leq r_{s_\beta}$ et $d_{s_\alpha} \leq d_{s_\beta}$.

Si deux sommets possèdent des dates de début au plus tôt et de fin au plus tard identiques, alors ils peuvent être indicés de façon quelconque. La s-pyramide notée P_α désigne la pyramide caractérisée par le sommet s_α. On note $u(i_j)$ (resp. $v(i_j)$) l'indice de la première s-pyramide à laquelle le travail i_j appartient (resp. l'indice de la dernière pyramide à laquelle le travail i_j appartient). Un ordre partiel dominant peut alors être caractérisé grâce au théorème suivant.

4.2.2 Énoncé du théorème

Théorème 1. *Un ensemble dominant de séquences peut être constitué par les séquences telles que :*

1. *les sommets sont ordonnés dans l'ordre de leurs indices ;*
2. *avant le premier sommet, seuls sont placés des travaux appartenant à la première s-pyramide, rangés dans l'ordre croissant de leurs r_j ou, en cas d'égalité, dans un ordre arbitraire ;*
3. *après le dernier sommet, seuls sont placés des travaux appartenant à la dernière s-pyramide, rangés dans l'ordre croissant de leurs d_j ou, en cas d'égalité, dans un ordre arbitraire ;*
4. *entre deux sommets s_k et s_{k+1}, sont placés en premier des travaux appartenant à la s-pyramide P_k et n'appartenant pas à P_{k+1} dans l'ordre croissant de leurs d_j (dans un ordre arbitraire en cas d'égalité), puis des travaux communs aux deux pyramides P_k et P_{k+1} dans un ordre arbitraire, et enfin des travaux appartenant à la pyramide P_{k+1} et n'appartenant pas à P_k dans l'ordre croissant de leurs r_j (dans un ordre arbitraire en cas d'égalité).*

Le théorème des pyramides définit un ordre partiel dominant dans la mesure où il impose des relations de précédence entre les sommets d'une part, puisque $s_k \prec s_{k+1}$, et entre les travaux non-sommets et les sommets d'autre part, puisque $s_{u(j)-1} \prec j \prec s_{v(j)+1}$. Cependant, nous remarquons que le théorème, en imposant par ailleurs de classer les travaux affectés devant un sommet par ordre croissant des r_j, et ceux affectés derrière par ordre croissant des d_j, exclut aussi certains ordres totaux qui sont pourtant des extensions linéaires de l'ordre partiel précédent. Ainsi, si deux travaux non-sommets i et j, tels que $r_i < r_j$, appartenant à une seule pyramide de sommet s, sont tous deux séquencés avant s alors $i \prec j$ et l'ordre total $j \prec i \prec s$ est interdit alors qu'il correspond pourtant à une extension linéaire de l'ordre partiel imposé par le théorème des pyramides. Ce théorème introduit donc, outre un ordre partiel dominant, des relations de dominance conditionnelles, liées à la position des travaux relativement aux sommets, et à l'ordre relatif des dates de début au plus tôt et de fin au plus tard.

4.2.3 Exemple

Pour illustrer le théorème des pyramides, nous utilisons l'exemple extrait de l'article [Carlier 82]. Il s'agit d'un problème à sept travaux dont les dates de début, les dates de fin

et les durées opératoires sont indiquées dans le tableau 4.1. Remarquons que ces valeurs sont ici données de façon explicite bien que, pour l'application du théorème des pyramides, seul l'ordre relatif entre les dates de fin au plus tôt et de fin au plus tard soit nécessaire.

Travail	r_j	d_j	p_j
1	10	44	5
2	13	25	6
3	11	27	7
4	20	30	4
5	30	43	3
6	0	34	6
7	30	51	2

TAB. 4.1: Une instance de problème à une machine

Le diagramme des intervalles associé à cet exemple est donné sur la figure 4.3, ainsi que la décomposition en pyramides correspondante. Les durées opératoires sont indiquées par des rectangles, les sommets sont foncés.

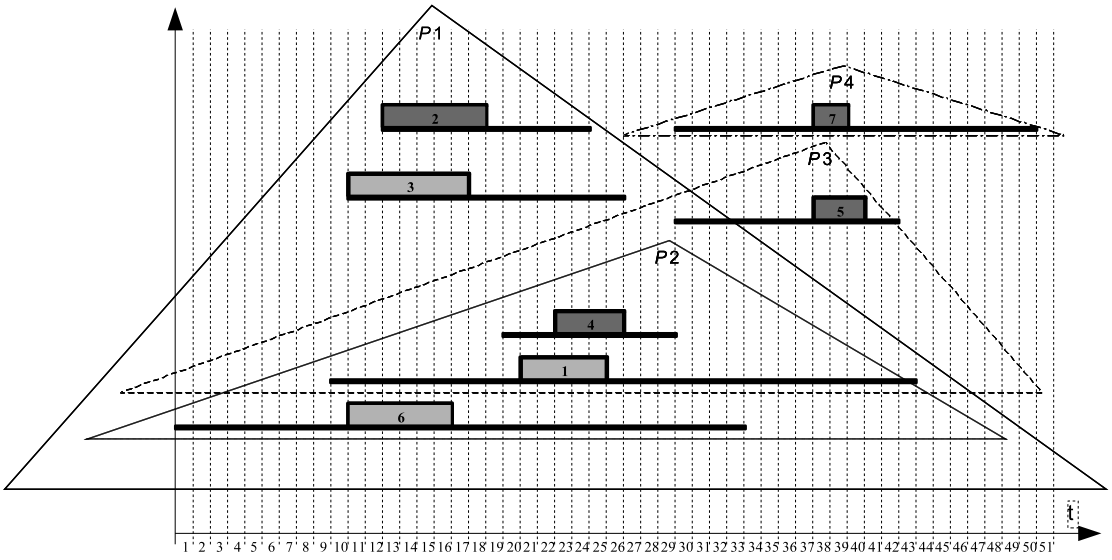


FIG. 4.3: Diagramme des intervalles, sommets et pyramides

Compte tenu de la structure d'intervalles associée à cet exemple, on distingue quatre sommets : $s_1 = 2$, $s_2 = 4$, $s_3 = 5$ et $s_4 = 7$, caractérisant les quatre pyramides : $P_1 = \{1, 3, 6\}$, $P_2 = \{1, 6\}$, $P_3 = \{1\}$ et $P_4 = \emptyset$. Remarquons que, compte tenu de la définition d'une s-pyramide, les sommets n'appartiennent pas à la pyramide qu'ils carac-

térisent.

En appliquant le théorème des pyramides à cette structure d'intervalles, nous obtenons un ensemble de séquences dominantes, par exemple : $6 \prec 1 \prec 3 \prec 2 \prec 4 \prec 5 \prec 7$, $1 \prec 2 \prec 3 \prec 6 \prec 4 \prec 5 \prec 7$, $2 \prec 3 \prec 4 \prec 6 \prec 5 \prec 1 \prec 7$, etc. La séquence $1 \prec 2 \prec 6 \prec 3 \prec 4 \prec 5 \prec 7$ ne fait pas partie de l'ensemble dominant de séquences puisque le travail 3, n'appartenant qu'à la pyramide P_1, ne peut pas être placé après le travail 6 possédant une date de fin plus grande ($d_6 > d_3$).

4.2.4 Une mesure de flexibilité

Une propriété intéressante du théorème des pyramides est qu'il permet de mesurer la cardinalité de l'ensemble de séquences dominantes S_{dom} obtenu lors de l'application du théorème. Cette cardinalité est calculée grâce à la formule :

$$card(S_{dom}) = \prod_{q=1}^{N} (q+1)^{n_q}$$

où n_q désigne le nombre de travaux non sommets appartenant exactement à q pyramides et N le nombre total de pyramides.

L'ordre des travaux placés entre les sommets étant imposé par la considération des dates de début au plus tôt et de fin au plus tard (ordre croissant), cette formule dénombre en fait les affectations différentes des travaux selon s'ils sont placés avant ou après un sommet s_k, tel que $u(j) \leq k \leq v(j)$.

Illustrons l'application de cette formule sur l'exemple précédent. Parmi les trois travaux non sommets $\{1, 3, 6\}$, le travail 3 appartient à une seule pyramide, le travail 6 appartient à deux pyramides et le travail 1 appartient à trois pyramides. L'ordre partiel imposé par le théorème des pyramides peut être représenté par le schéma de la figure 4.4 sur lequel sont indiquées les positions possibles de chaque travail non sommet. Par exemple, le travail 6 peut être placé soit avant le sommet 2, soit entre les deux sommets 2 et 4, soit après le sommet 4, le travail 3 ne peut être placé qu'avant ou après le sommet 2. La cardinalité de S_{dom} est donc $card(S_{dom}) = (1+1)^1 \cdot (2+1)^1 \cdot (3+1)^1 = 24$, constitué des séquences suivantes :

$6 \prec 1 \prec 3 \prec 2 \prec 4 \prec 5 \prec 7$,	$1 \prec 3 \prec 2 \prec 6 \prec 4 \prec 5 \prec 7$,	$1 \prec 3 \prec 2 \prec 4 \prec 6 \prec 5 \prec 7$,
$6 \prec 1 \prec 2 \prec 3 \prec 4 \prec 5 \prec 7$,	$1 \prec 2 \prec 3 \prec 6 \prec 4 \prec 5 \prec 7$,	$1 \prec 2 \prec 3 \prec 4 \prec 6 \prec 5 \prec 7$,
$6 \prec 3 \prec 2 \prec 1 \prec 4 \prec 5 \prec 7$,	$3 \prec 2 \prec 6 \prec 1 \prec 4 \prec 5 \prec 7$,	$3 \prec 2 \prec 1 \prec 4 \prec 6 \prec 5 \prec 7$,
$6 \prec 2 \prec 3 \prec 1 \prec 4 \prec 5 \prec 7$,	$2 \prec 3 \prec 6 \prec 1 \prec 4 \prec 5 \prec 7$,	$2 \prec 3 \prec 1 \prec 4 \prec 6 \prec 5 \prec 7$,
$6 \prec 3 \prec 2 \prec 4 \prec 1 \prec 5 \prec 7$,	$3 \prec 2 \prec 6 \prec 4 \prec 1 \prec 5 \prec 7$,	$3 \prec 2 \prec 4 \prec 6 \prec 1 \prec 5 \prec 7$,
$6 \prec 2 \prec 3 \prec 4 \prec 1 \prec 5 \prec 7$,	$2 \prec 3 \prec 6 \prec 4 \prec 1 \prec 5 \prec 7$,	$2 \prec 3 \prec 4 \prec 6 \prec 1 \prec 5 \prec 7$,
$6 \prec 3 \prec 2 \prec 4 \prec 5 \prec 1 \prec 7$,	$3 \prec 2 \prec 6 \prec 4 \prec 5 \prec 1 \prec 7$,	$3 \prec 2 \prec 4 \prec 6 \prec 5 \prec 1 \prec 7$,
$6 \prec 2 \prec 3 \prec 4 \prec 5 \prec 1 \prec 7$,	$2 \prec 3 \prec 6 \prec 4 \prec 5 \prec 1 \prec 7$,	$2 \prec 3 \prec 4 \prec 6 \prec 5 \prec 1 \prec 7$.

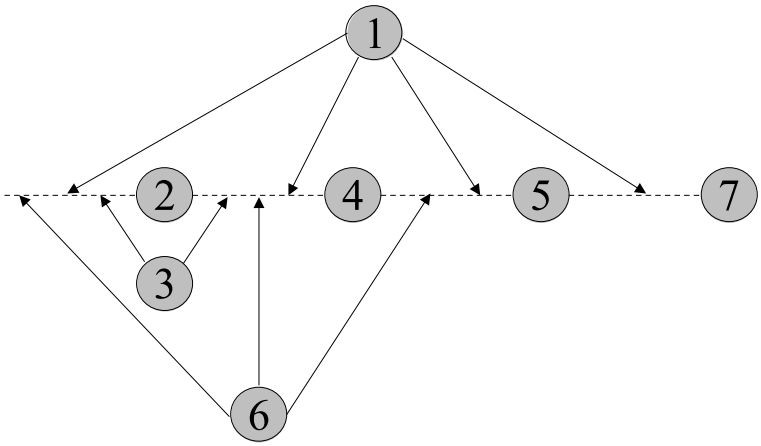


FIG. 4.4: Ensemble dominant de séquences associé à l'exemple

Remarquons que cette formule ne prend pas en compte les possibilités de permutation des travaux placés entre deux sommets s_k et s_{k+1} et appartenant en même temps aux s-pyramides P_k et P_{k+1}. En effet, on peut démontrer que ces permutations sont strictement équivalentes en terme de qualité. Par exemple, considérons le cas où les trois travaux non sommets 1, 3 et 6 sont tous placés entre les deux sommets 2 et 4. Dans ce cas, seule la séquence $2 \prec 3 \prec 6 \prec 1 \prec 4 \prec 5 \prec 7$ est comptabilisée puisque le travail 3, appartenant seulement à la pyramide s_1 du sommet 2, doit nécessairement précéder les deux travaux 1 et 6. Cependant, selon le théorème, les deux travaux 1 et 6, appartenant tout deux aux pyramides de sommets s_1 et s_2, peuvent être séquencés dans n'importe quel ordre. La séquence $2 \prec 3 \prec 1 \prec 6 \prec 4 \prec 5 \prec 7$ respecte donc aussi le théorème, bien qu'elle ne soit pas comptabilisée. La valeur $card(S_{dom})$ est donc une borne inférieure du nombre réel de séquences dans l'ensemble dominant.

4.2.5 Extensions du théorème

Dans cette partie, nous nous proposons d'affiner le théorème des pyramides en étudiant deux cas limites : le cas où deux sommets possèdent des dates de début identiques et celui où deux sommets possèdent des dates de fin identiques. Pour ces deux cas, nous énonçons deux propositions que nous démontrons et dont nous illustrons l'intérêt dans la partie suivante.

Proposition 4.2.1. *Si un travail j appartient à deux pyramides P_1 et P_2 dont les sommets s_1 et s_2 possèdent des dates de début égales, alors toute séquence plaçant j entre s_1 et s_2 est dominée par au moins une autre séquence de l'ensemble dominant.*

Démonstration. La démonstration est relativement évidente. Le travail j appartenant aux deux pyramides P_1 et P_2 de sommets s_1 et s_2 avec $r_{s_1} = r_{s_2} = r_s$ et $d_{s_1} < d_{s_2}$ (cf. figure

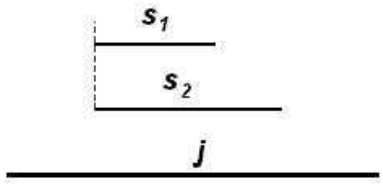


FIG. 4.5: Deux sommets possédant la même r_i

4.5), nous avons la relation suivante : $r_i < r_s \wedge d_i > d_{s_2}$. Selon le théorème, toute séquence dominante est telle que $s_1 < s_2$. Si le travail j est placé après le sommet s_1, alors sa date de début peut être actualisée telle que $r_j = r_{s_1} = r_s$. Les trois intervalles associés aux travaux s_1, s_2 et j forment alors une structure en escalier (*i.e.* $r_j = r_{s_1} = r_{s_2}$ et $d_{s_1} < d_{s_2} < d_j$). Or, pour une telle structure, on sait que la séquence $s_1 \prec s_2 \prec j$, rangeant les travaux par date de fin croissante, est dominante, en particulier devant la séquence $s_1 \prec j \prec s_2$ plaçant j entre les deux sommets. □

Proposition 4.2.2. *Si un travail j appartient à deux pyramides P_1 et P_2 dont les sommets s_1 et s_2 possèdent des dates de fin égales, alors toute séquence plaçant j entre s_1 et s_2 est dominée par au moins une autre séquence de l'ensemble dominant.*

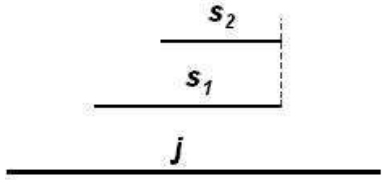


FIG. 4.6: Deux sommets possédant la même d_i

Démonstration. Le raisonnement est la même que celui de la démonstration précédente. On obtient une structure en escalier inversée (*i.e.* $d_j = d_{s_1} = d_{s_2}$ et $r_j < r_{s_1} < r_{s_2}$) dont on déduit que la séquence $s_1 \prec j \prec s_2$ est dominée par la séquence dominante $j \prec s_1 \prec s_2$ rangeant les travaux par date de début croissante.

□

En conséquence, nous déduisons la proposition suivante dont la démonstration est évidente à partir des deux premières.

Proposition 4.2.3. *Si un travail j appartient à deux pyramides P_1 et P_2 dont les sommets s_1 et s_2 possèdent des dates de début et de fin égales, alors toute séquence plaçant j entre s_1 et s_2 est dominée par au moins une autre séquence de l'ensemble dominant.*

À partir de ces extensions, il est intéressant d'étudier l'apport, en termes de flexibilité séquentielle, du théorème des pyramides comparativement au concept de groupes de tâches permutables. La flexibilité séquentielle est ici assimilée au nombre de séquences différentes, contenues dans l'ensemble dominant S_{dom} d'un problème, pouvant potentiellement être caractérisées grâce au théorème des pyramides.

La flexibilité séquentielle pouvant potentiellement être mise en évidence par le théorème des pyramides recouvre celle pouvant être offerte par le concept de groupes. En effet, un groupe de tâches totalement permutables correspond dans notre cas à un ensemble de sommets possédant des dates de début et de fin identiques dans le sens où le dernier caractérise exactement le même ensemble de séquences que le premier. La figure 4.7 illustre cette observation.

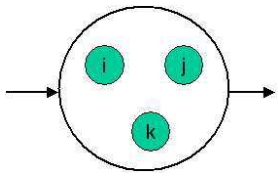
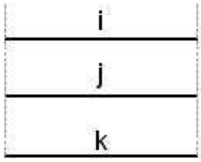
Groupe de tâches	Structure d'intervalles équivalente	Séquences caractérisées
		$i<j<k,\ i<k<j$ $j<i<k,\ j<k<i$ $k<i<j,\ k<j<i$

FIG. 4.7: Un groupe de tâches et la structure d'intervalles équivalente

En revanche, il est impossible d'exprimer un ensemble dominant quelconque de séquences S_{dom} à l'aide d'une et une seule séquence de groupes de tâches permutables, comme l'illustre la figure 4.8.

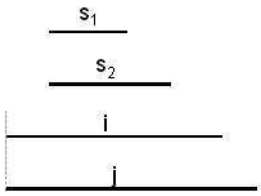
Structure d'intervalles	Séquences caractérisées	Groupe de tâches équivalent
	$i<j<s_1<s_2$ $j<i<s_1<s_2$ $i<s_1<s_2<j$ $j<s_1<s_2<i$ $s_1<s_2<i<j$	$\emptyset$

FIG. 4.8: Une structure d'intervalles sans groupe équivalent

Un autre intérêt du théorème des pyramides réside dans sa capacité à caractériser des

permutations de travaux placés de façon non contiguë sur la ressource, ce qui est aussi illustré sur la figure 4.8 : les deux travaux non sommets i et j peuvent être placés soit avant, soit après les deux sommets s_1 et s_2, mais jamais entre s_1 et s_2.

4.3 Mesure de la qualité d'un ensemble dominant

Le théorème des pyramides associe à chaque structure d'intervalles définie par un problème V à résoudre, un ensemble de séquences dominantes S_{dom}. Dans cette partie, nous indiquons comment évaluer les performances d'un tel ensemble du point de vue du retard algébrique. Pour cela, nous montrons comment il est possible de calculer en temps polynomial, pour chaque travail j, les valeurs $L_j^{\max}$ et $L_j^{\min}$ correspondant respectivement au pire et au meilleur retard de j parmi toutes les séquences de S_{dom}. Il est important de remarquer que le calcul de ces valeurs est basé sur l'évaluation, pour chaque travail, de la séquence la plus favorable et de celle la plus défavorable vis-à-vis du retard algébrique. Ces séquences peuvent être établies sur la base d'un corps d'hypothèses restreint, c'est-à-dire en ne considérant que les relations d'ordre entre les r_j et d_j des travaux. Toutefois, pour calculer les valeurs précises de $L_j^{\max}$ et $L_j^{\min}$, la connaissance explicite des valeurs des paramètres r_j, p_j et d_j est requise.

Dans la suite du texte, on rappelle que $u(j)$ et $v(j)$ désignent respectivement les indices de la première et de la dernière pyramide à laquelle peut être affecté le travail j.

4.3.1 Calcul du retard au mieux d'un travail

Intéressons nous tout d'abord au calcul de $L_j^{\min}$. De façon évidente, le but étant de minimiser la longueur du chemin critique associé au travail j, seuls les travaux nécessairement séquencés avant j (en cohérence avec le théorème des pyramides) sont à considérer (i.e. moins le nombre de travaux placés avant j est important et plus le retard de j sera petit). Pour cette raison, nous supposons que j est affecté à la pyramide d'indice $u(j)$ et que seul l'ensemble $Pred_j^{\min}$ des travaux k tels que $v(k) < u(j)$ est pris en compte (seuls les travaux $k \in Pred_j^{\min}$ précèdent nécessairement j dans toutes les séquences de l'ensemble dominant).

Dans la pyramide d'indice $u(j)$, il est toujours possible de séquencer j en première position. Déterminer $L_j^{\min}$ revient alors à minimiser la durée totale $C_{\max}$ du problème d'ordonnancement constitué des travaux de l'ensemble $Pred_j^{\min}$. Le retard des travaux de $Pred_j^{\min}$ n'étant pas significatif pour le calcul de $L_j^{\min}$, nous fixons arbitrairement leur date de fin à la date d_j. La structure d'intervalles de $Pred_j^{\min}$ ainsi obtenue définissant une structure en escalier, une séquence $Seq_j^{\min}$ optimale (pour minimiser le $C_{\max}$) est obtenue par l'application de la règle de Jackson séquençant les travaux dans l'ordre croissant de leurs dates de début. Remarquons que cette règle de construction de $Seq_j^{\min}$ respecte le théorème des pyramides puisqu'elle affecte chaque travail $k \in Pred_j^{\min}$ à la pyramide $u(k)$ ainsi que l'illustre la figure 4.9.

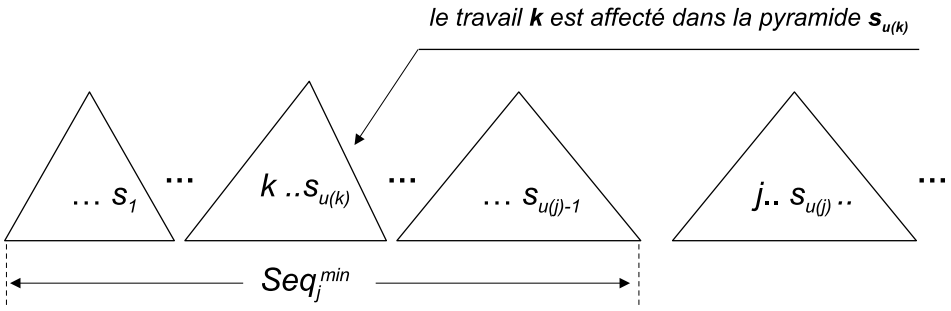


FIG. 4.9: Construction de la meilleure séquence pour le travail j

Soient $s_k^{Seq_j^{\min}}$ les dates de début des travaux $k \in Pred_j^{\min}$ dans $Seq_j^{\min}$ et $C_{\max}^{Seq_j^{\min}}$ le makespan correspondant. On a :

$$C_{\max}^{Seq_j^{\min}} = \max(s_k^{Seq_j^{\min}} + p_k), \forall k \in Pred_j^{\min}$$

D'où :

$$L_j^{\min} = \max(C_{\max}^{Seq_j^{\min}}, r_j) + p_j - d_j$$

L'algorithme pour calculer $L_j^{\min}$ est décrit sur la figure 4.10. Sa complexité temporelle est en $O(n \log n)$.

Pour illustration, reprenons l'exemple précédent du tableau 4.1 et considérons le travail 5. Celui-ci est un sommet, il n'appartient donc qu'à la pyramide qu'il caractérise et $u(5) = 3$. L'ensemble $Pred_5^{min}$, comprenant les travaux nécessairement placés avant 5 dans toute séquence dominante, est donc constitué des travaux k tels que $v(k) < 3$, d'où : $Pred_5^{\min} = \{2, 3, 6, 4\}$. En appliquant la règle de Jackson sur l'ensemble $Pred_5^{\min}$, on obtient la séquence $Seq_5^{\min} = 6 \prec 3 \prec 2 \prec 4$, ayant une durée $C_{\max}^{Seq_5^{\min}} = 28$. Nous en déduisons :

$$L_5^{\min} = \max(C_{\max}^{Seq_5^{\min}}, r_5) + p_5 - d_5 = 30 + 3 - 43 = -10.$$

La sous-séquence $6 \prec 3 \prec 2 \prec 4$ respecte bien le théorème des pyramides. La séquence la plus favorable pour le travail 5 (celle produisant le retard algébrique le plus petit) est donc $6 \prec 3 \prec 2 \prec 4 \prec 5$ avec $L_5^{\min} = -10$.

Proc CalculerLjMin(j, T) (*)
$Pred_j^{\min} = \emptyset$;

Pour chaque $i \in T$ faire
Si $v(i) < u(j)$ alors $Pred_j^{\min} \leftarrow Pred_j^{\min} \cup i$; FinSi
FinPour

Pour chaque $i \in Pred_j^{\min}$ faire
Affecter i à la pyramide d'indice $u(i)$;
FinPour

Pour chaque pyramide P_k, telle que $k < u(j)$ faire
$Seq_{P_k}^{\min} \leftarrow$ travaux séquencés par ordre croissant des r_i ;
FinPour

$Seq_j^{\min} \leftarrow Seq_{P_0}^{\min} \prec \cdots \prec Seq_{P_{u(j-1)}}^{\min} \prec j$
Calculer $L_j^{\min}$ pour la séquence $Seq_j^{\min}$;
FinProc

(*) Dans cet algorithme, $Pred_j^{\min}$ est l'ensemble de travaux devant précéder j.
T est l'ensemble de travaux à ordonnancer.

FIG. 4.10: Algorithme de calcul de $L_j^{\min}$

4.3.2 Calcul du retard au pire d'un travail

Considérons à présent le calcul de $L_j^{\max}$. De façon symétrique au cas précédent, afin de maximiser le retard du travail j, celui-ci est supposé affecté à la pyramide d'indice $v(j)$, c'est-à-dire le plus tard possible. D'autre part, les travaux k devant nécessairement être séquencés après j (*i.e.* les travaux k tels que $u(k) > v(j)$) ne sont pas considérés. Nous notons $Pred_j^{\max}$ l'ensemble des travaux restants. Trouver $L_j^{\max}$ consiste alors à maximiser le makespan $C_{\max}$ des travaux de $Pred_j^{\max}$ tout en respectant le théorème des pyramides.

Comme précédemment, les retards de ces travaux n'étant pas significatifs pour le calcul de $L_j^{\max}$, leur date d'échéance peut être fixée à une valeur arbitraire. La structure d'intervalles ainsi obtenue étant à nouveau "en escalier", nous pourrions alors utiliser la version inversée de la règle de Jackson qui consiste à classer les travaux par ordre de r_i décroissant pour maximiser leur $C_{\max}$. Toutefois, la séquence obtenue ne respecterait alors pas le théorème des pyramides. Pour pallier ce problème, nous nous proposons de considérer d'une part les travaux k tels que $v(k) < v(j)$, (*i.e.* appartenant à des pyramides antérieures à celle de j) et d'autre part les travaux k tels que $u(k) \leq v(j) \leq v(k)$ (*i.e.* pouvant être affectés à la même pyramide que celle de j).

Afin de maximiser la longueur du chemin critique associé à j, nous affectons les travaux k tels que $v(k) < v(j)$ à la pyramide d'indice $v(k)$, de façon à ce qu'ils soient ordonnancés le plus tard possible. Les affectations des travaux aux pyramides étant établies, le problème de maximiser le $C_{\max}$ des travaux situés dans les $v(j) - 1$ pyramides d'indice inférieur à $v(j)$ se décompose alors en $v(j) - 1$ problèmes de maximisation indépendants. Chaque problème consiste à maximiser le makespan $C_{\max}^{P_i}$ des travaux affectés à la pyramide P_i. Il peut être résolu optimalement, avec respect du théorème des pyramides, en construisant une sous-séquence dans laquelle les travaux sont séquencés par ordre de d_i croissant. En effet, de cette façon, le sommet est toujours placé en tête de la sous-séquence, puisqu'il possède par définition la date d'échéance d_i la plus petite. De plus, le sommet possédant également la date de début r_i la plus grande, on a $C_{\max}^{P_i} = r_s + \sum_{j \in P_i - \{s\}} p_j$ et il n'existe clairement aucune séquence produisant une valeur supérieure à celle-ci. Remarquons aussi que la sous-séquence ainsi obtenue respecte bien le théorème des pyramides.

La séquence, notée S_j^1, conduisant au $C_{\max}$ le plus grand sur l'ensemble des travaux situés dans les pyramides d'indice inférieur à $v(j)$ correspond alors à la juxtaposition des sous-séquences déduites précédemment (cf. figure 4.11).

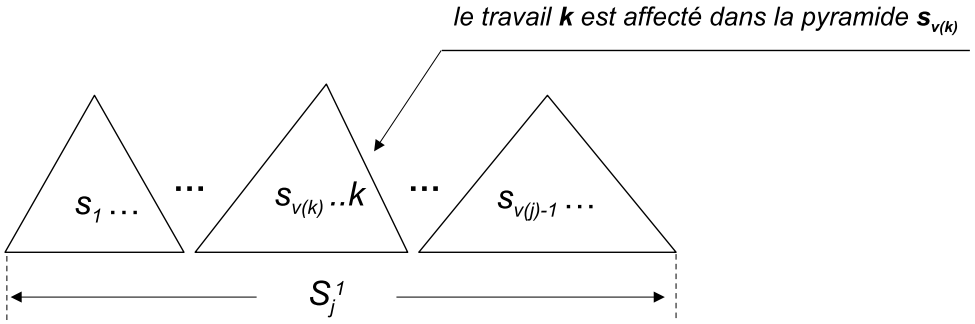


FIG. 4.11: La sous-séquence S_j^1 avant la pyramide d'indice $v(j)$

Considérons à présent les travaux k tels que $u(k) \leq v(j) \leq v(k)$. Pour maximiser le retard de j, nous affectons ces travaux k tels que $u(k) \leq v(j) \leq v(k)$ à la pyramide d'indice $v(j)$ (celle de j). En effet, plus le nombre de travaux placés avant j est important et plus ceux-ci sont ordonnancés tard, plus le retard algébrique de j sera important. Le problème est alors de déterminer, en respectant le théorème des pyramides, quelle séquence parmi celles de l'ensemble dominant plaçant j en dernier dans la pyramide $v(j)$, possède le $C_{\max}$ le plus grand. En cohérence avec le théorème des pyramides, il est facile de montrer que cette séquence correspond à celle où le plus de travaux possibles sont placés après le sommet, c'est-à-dire entre le sommet de la pyramide $v(j)$ et le travail j. Pour l'obtenir, il suffit de placer tous les travaux k tels que $d_k > d_j$ avant le sommet et tous les travaux restants après le sommet (cf. démonstration suivante). En effet, si les travaux k tels que $d_k > d_j$

étaient placés après le sommet, ils seraient séquencés après j (cf. théorème des pyramides). En suivant cette règle, nous pouvons donc garantir que le travail j est mis à la dernière position dans la pyramide d'indice $v(j)$ et que le $C_{\max}$ des travaux dans cette pyramide est maximisé.

Proposition 4.3.1. *Soit une structure d'intervalles caractérisée par une unique s-pyramide P de sommet s, la séquence la plus défavorable vis-à-vis du retard algébrique d'un travail $j \in P$ est obtenue en plaçant tout travail $k \in P - \{s\}$ tel que $d_k > d_j$ avant s par ordre de r_i croissant, sinon après s par ordre de d_i croissant.*

Démonstration. Selon la proposition précédente, toute séquence S obtenue est de la forme $A \prec s \prec B \prec j$ ($A \prec s$ dans le cas où le travail j considéré est le sommet s de la pyramide). A correspond à l'ensemble des travaux $k \in P - \{s\}$ tels que $d_k > d_j$ et B à l'ensemble des travaux $k \in P - \{s\}$ tels que $d_k \leq d_j$. Le makespan de j dans cette séquence S est $C^S_{\max}(j) = \max(C^A_{\max}, r_s) + p_s + \sum_{k \in B} p_k + p_j$. Nous montrons dans la suite de cette démonstration que séquencer un travail de A (resp. de B) après (resp. avant) le sommet s peut diminuer la valeur de $C^S_{\max}(j)$ mais jamais l'augmenter, d'où le résultat énoncé dans la proposition 4.3.2.

Supposons qu'un travail i de A est séquencé après le sommet s. Comme $d_i > d_j$, le théorème des pyramides produit une nouvelle séquence S' de la forme $A' \prec s \prec B \prec j \prec i$, où $A' = A - \{i\}$. Le makespan de j dans cette nouvelle séquence S' est $C^{S'}_{\max}(j) = \max(C^{A'}_{\max}, r_s) + p_s + \sum_{k \in B} p_k + p_j$. Comme les travaux de A et A' sont classés par ordre croissant des dates de début au plus tôt, on a $C^A_{\max} \geq C^{A'}_{\max}$ et donc $C^S_{\max}(j) \geq C^{S'}_{\max}(j)$.

On en conclut que pour maximiser la date de fin de j, aucun travail de A ne doit être placé après le sommet s.

Supposons à présent qu'un travail i de B est séquencé avant le sommet s. Le théorème des pyramides produit une nouvelle séquence S' de la forme $A' \prec s \prec B' \prec j$, où les travaux de $A' = A + \{i\}$ sont ordonnés par ordre croissant des dates de début au plus tôt et les travaux de $B' = B - \{i\}$ sont ordonnés par ordre croissant des dates de fin au plus tard. Le makespan de j dans cette nouvelle séquence S' est $C^{S'}_{\max}(j) = \max(C^{A'}_{\max}, r_s) + p_s + \sum_{k \in B} p_k + p_j - p_i$. D'après l'expression de $C^S_{\max}(j)$, on en déduit $C^{S'}_{\max}(j) - C^S_{\max}(j) = \max(C^{A'}_{\max}, r_s) - \max(C^A_{\max}, r_s) - p_i$. La valeur de $C^{A'}_{\max}$ étant nécessairement supérieure ou égale à $C^A_{\max}$, nous considérons deux cas :

1. Si $C^A_{\max} \leq C^{A'}_{\max} \leq r_s$ alors $C^{S'}_{\max}(j) - C^S_{\max}(j) = -p_i \leq 0$.
2. Sinon nous montrons que $C^{A'}_{\max} - C^A_{\max} \leq p_i$ et donc $C^{S'}_{\max}(j) - C^S_{\max}(j) \leq 0$ est également vrai.
   On a $A' = A_1 \prec i \prec A_2$ et $A = A_1 \prec A_2$ où $\forall k \in A_1$, $r_k \leq r_i$ et $\forall k \in A_1$, $r_k > r_i$. Nous notons respectivement $C^{A_1}_{\max}$, et $C^{A_2}_{\max}$ le makespan des travaux de A_1 et de A_2. La valeur r_{A_2} désigne la date de début au

plus tôt la plus petite des travaux de A_2 en se souvenant que $r_i \leq r_{A_2}$ par définition. On a :

$$C^A_{\max} = \max(C^{A_1}_{\max}, r_{A_2}) + C^{A_2}_{\max}$$

et

$$C^{A'}_{\max} = \max(C^i_{\max}, r_{A_2}) + C^{A_2}_{\max}$$

où

$$C^i_{\max} = \max(C^{A_1}_{\max}, r_i) + p_i$$

Intéressons nous à la valeur de $C^i_{\max}$. Si $C^{A_1}_{\max} \geq r_i$ alors $C^i_{\max} = C^{A_1}_{\max} + p_i$ et donc $C^{A'}_{\max} - C^A_{\max} = \max(C^{A_1}_{\max} + p_i, r_{A_2}) - \max(C^{A_1}_{\max}, r_{A_2})$, d'où $C^{A'}_{\max} - C^A_{\max} \leq p_i$.

Sinon si $C^{A_1}_{\max} < r_i$ alors $C^i_{\max} = r_i + p_i$ et donc $C^{A'}_{\max} - C^A_{\max} = \max(r_i + p_i, r_{A_2}) - \max(C^{A_1}_{\max}, r_{A_2})$. Comme $r_i \leq r_{A_2}$ et $C^{A_1}_{\max} < r_i$, il vient : $C^{A'}_{\max} - C^A_{\max} = \max(r_i + p_i, r_{A_2}) - r_{A_2} = \max(r_i + p_i - r_{A_2}, 0) \leq p_i$.

On en conclut que pour maximiser la date de fin de j, aucun travail de B ne doit être placé avant le sommet s. □

Soit S^2_j la sous-séquence construite par application de la règle de la proposition 4.3.2 en considérant la pyramide d'indice $v(j)$ (cf. figure 4.12).

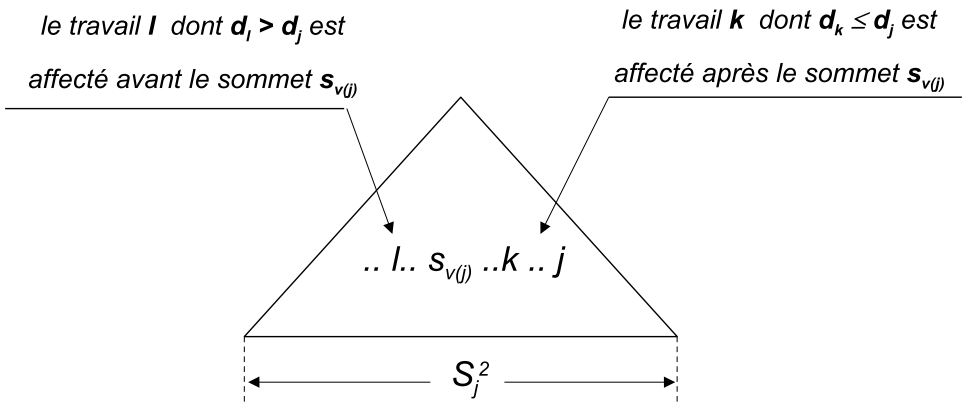


FIG. 4.12: La sous-séquence S^j_2 de la pyramide d'indice $v(j)$

Notons $Seq^{\max}_j$, la juxtaposition des deux sous-séquences S^1_j et S^2_j précédemment déterminées (*i.e.* $Seq^{\max}_j = S^1_j \prec S^2_j$), et $C^{Seq^{\max}_j}_{\max}$, le makespan correspondant. Nous avons alors :

$$L_j^{\max} = \max(C_{\max}^{Seq_j^{\max}}, r_j) + p_j - d_j$$

L'algorithme pour calculer $L_j^{\max}$ est décrit sur la figure 4.13. Sa complexité temporelle est en $O(n \log n)$.

Pour illustration, nous reprenons à nouveau l'exemple proposé dans le tableau 4.1 en nous intéressant au calcul du pire retard du travail 6, $L_6^{\max}$. Le travail 6 appartient aux pyramides caractérisées par les sommets 2 et 4, d'où $v(6) = 2$. L'ensemble $Pred_6^{\max}$ contient les travaux k tels que $u(k) \leq v(6) = 2$. Donc $Pred_6^{\max} = \{1, 2, 3, 4\}$.

Pour maximiser le $C_{\max}$ des travaux dans $Pred_6^{\max}$, nous affectons d'abord les travaux k possédant $v(k) < v(6) = 2$ dans les pyramides d'indice $v(k)$. Dans cet exemple, ces travaux sont les travaux 2 et 3 puisque $v(2) = v(3) = 1$. La séquence $S_1^{\max}$ est $S_1^{\max} = 2 \prec 3$.

Les travaux k tels que $u(k) \leq v(6) \leq v(k)$ sont affectés à la pyramide d'indice $v(6) = 2$, avec le travail 6. Ces travaux k sont les travaux 1 et 4. Pour maximiser le $C_{\max}$ de ces travaux, nous appliquons la règle de la proposition 4.3.2 qui produit la séquence $S_2^{\max} = 1 \prec 4 \prec 6$, maximisant le $C_{\max}$ du travail 6.

En concaténant les deux sous-séquences S_j^1 et S_j^2, nous obtenons la séquence $2 \prec 3 \prec 1 \prec 4 \prec 6$, qui produit le plus grand retard pour le travail 6, et $L_6^{\max} = 7$.

4.3.3 Le diagramme de retards

Étant donné un problème V et son ensemble de séquences dominantes S_{dom}, il est intéressant de disposer d'une représentation visuelle permettant de mettre en évidence les performances de S_{dom} pour l'ensemble des travaux. Dans ce but, nous introduisons ici le diagramme de retards associant à chaque travail j de V un intervalle $[L_j^{\min}, L_j^{\max}]$ calculé comme décrit dans les paragraphes précédents. La détermination des $L_j^{\min}$ et $L_j^{\max}$ de tous les travaux a une complexité temporelle en $O(n^2 \log n)$. Étant donnée la structure d'intervalles associée au problème du tableau 4.1, la figure 4.14 présente un exemple de diagramme de retards obtenu. Les séquences partielles, au mieux et au pire, ayant permis le calcul de chaque valeur $L_j^{\min}$ et $L_j^{\max}$ sont indiquées au dessus de chaque borne. On remarque sur ce diagramme que, quelle que soit la séquence de S_{dom} considérée, les travaux 1 et 7 ne seront jamais en retard.

La connaissance des retards au mieux et au pire de chaque travail permet de déduire des bornes inférieure et supérieure pour le retard algébrique $L_{\max}^*$ optimal :

$$\max(L_j^{\min}) \leq L_{\max}^* \leq \max(L_j^{\max}), \forall j \in T$$

Proc CalculerLjMax(j, T) $(*)$
$Pred_j^{S_1} = \emptyset$;
$Pred_j^{S_2} = \emptyset$;
$A = \emptyset$;
$B = \emptyset$;

Pour chaque $i \in T$ faire
Si $v(i) < v(j)$ alors $Pred_j^{S_1} \leftarrow Pred_j^{S_1} \cup i$; FinSi
Si $u(i) \leq v(j) \leq v(i)$ alors $Pred_j^{S_2} \leftarrow Pred_j^{S_2} \cup i$; FinSi
FinPour

Pour chaque $i \in Pred_j^{S_1}$ faire
Affecter i à la pyramide d'indice $v(i)$;
FinPour

Pour chaque pyramide $P_k, k < v(j)$ faire
$Seq_{P_k}^{\max} \leftarrow$ travaux séquencés par ordre croissant des d_i ;
FinPour

$S_1 \leftarrow Seq_{P_0}^{\max} \prec \cdots \prec Seq_{P_{v(j)-1}}^{\max}$

Pour chaque $i \in Pred_j^{S_2} - \{s_{v(j)}\}$ faire
Si $d_i > d_j$ alors
$A \leftarrow A + \{i\}$;
SiNon
$B \leftarrow B + \{i\}$;
FinSi
FinPour

Séquencer A par ordre croissant des r_i ;
Séquencer B par ordre croissant des d_i ;
$S_2 \leftarrow A \prec s \prec B \prec j$

$Seq_j^{\max} \leftarrow S_1 \prec S_2$;
calculer $L_j^{\max}$ sur $Seq_j^{\max}$;
FinProc
$(*)$ Dans cet algorithme, T est l'ensemble de travaux à ordonnancer. j est le travail dont le retard au pire sera calculé par l'algorithme. $Pred_j^{S_1}$ est l'ensemble de tous les travaux devant être placés avant la pyramide $P_{v(j)}$. $Pred_j^{S_2}$ est l'ensemble de travaux placés dans la pyramide $P_{v(j)}$. A (resp. B) est l'ensemble des travaux de $Pred_j^{S_2}$ placés avant (resp. après) $s_{v(j)}$.

FIG. 4.13: Algorithme de calcul de $L_j^{\max}$

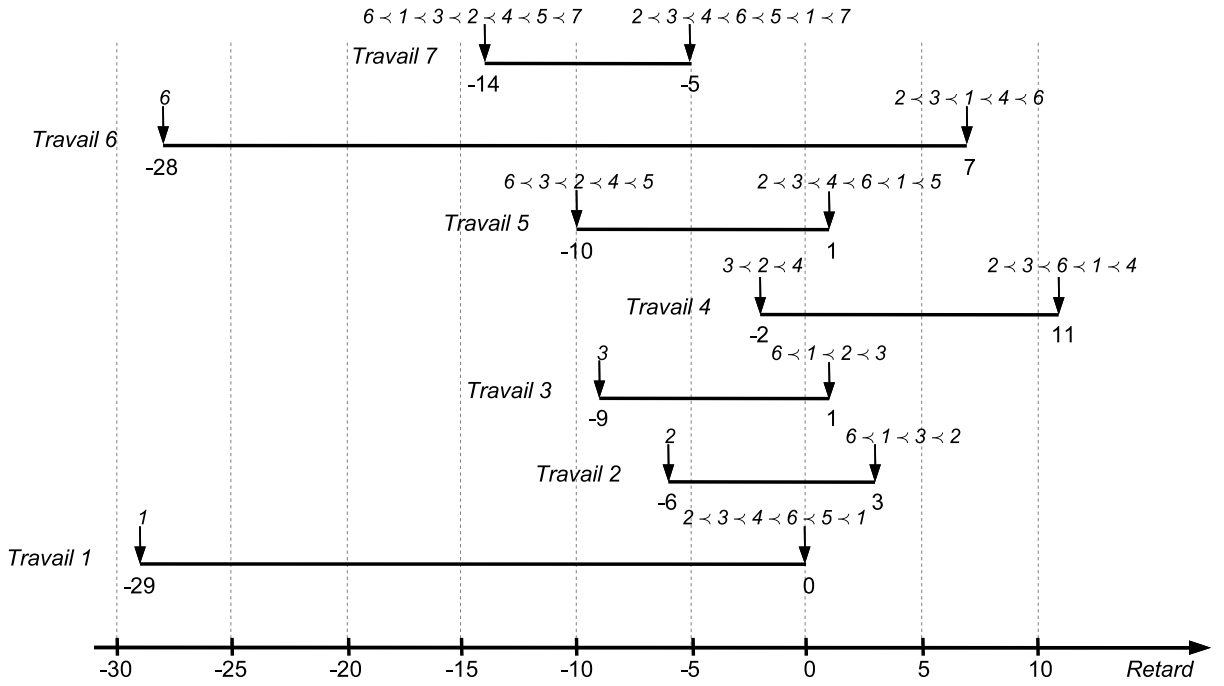


FIG. 4.14: Diagramme de retards obtenu pour l'exemple du tableau 4.1

Par exemple, sur le diagramme de la figure 4.14, nous avons $-2 \leq L^*_{\max} \leq 11$. Ces valeurs sont à comparer au retard optimal de -1, établi grâce à la PSEP de Carlier [Carlier 82].

Un diagramme de retards permet à un décideur de visualiser les performances associées à l'ensemble de séquences dominantes S_{dom} d'un problème, du point de vue du retard algébrique. Les séquences partielles au mieux et au pire de chaque travail n'étant pas nécessairement cohérentes entre elles, remarquons qu'il n'existe évidemment pas de séquences permettant de garantir l'obtention simultanée pour tous les travaux de leur retard algébrique au mieux ou au pire. Autrement dit, imposer une séquence partielle permettant à un travail de se finir au plus tôt se traduit nécessairement par une augmentation des retards au mieux de certains autres travaux. Inversement, imposer une séquence partielle, la pire pour un travail donné, tend à réduire les retards algébriques au pire des autres travaux.

Dans l'hypothèse où les dates de fin des travaux correspondent à des délais de livraison (cas de la production à la commande), le diagramme de retards permet au décideur de juger si un retard "au pire" est acceptable ou non. Dans la négative, une interaction serait possible, via cette représentation graphique, dans laquelle le décideur demanderait une réduction des retards au pire qui ne lui conviendraient pas (ou une augmentation des retards au mieux).

Une procédure d'ordonnancement aurait alors pour objectif d'éliminer, au détriment de la flexibilité, les séquences de l'ensemble dominant qui produisent les retards non acceptables. Deux approches possibles d'une telle procédure sont respectivement décrites dans les chapitres 5 et 6.

4.4 Réflexions sur l'insensibilité d'un ensemble dominant

4.4.1 Vers un modèle d'incertitudes par intervalles

Rappelons qu'une propriété intéressante du théorème des pyramides est sa relative insensibilité aux variations des dates de début et de fin de travaux dans la mesure où l'ordre relatif entre les dates est conservé. En effet, étant donné un problème V, S_{dom} étant l'ensemble de séquences dominantes défini par le théorème des pyramides, l'ensemble S_{dom} reste dominant tant que l'ordre relatif des r_i et d_i est conservé, cela quelles que soient les valeurs des durées opératoires.

Nous avons de plus montré dans la partie 4.3 qu'il était possible de déterminer pour chaque travail, en temps polynomial, la séquence la plus favorable et celle la plus défavorable vis-à-vis du retard algébrique. La détermination de ces séquences ne requiert également que la connaissance de l'ordre relatif des dates de début au plus tôt et de fin au plus tard des travaux (c'est-à-dire un corps d'hypothèses restreint). Cependant, le calcul des valeurs de $L_j^{\min}$ et $L_j^{\max}$ considère quant à lui le corps d'hypothèses complet.

La propriété suivante peut donc être énoncée :

Proposition 4.4.1. *Considérons un problème à une machine V dont les scénarios potentiels de réalisation sont caractérisés par un modèle par intervalles de type $r_i \in [\underline{r}_i, \overline{r}_i]$, $p_i \in [\underline{p}_i, \overline{p}_i]$ et $d_i \in [\underline{r}_i, \overline{d}_i]$ et tel que tous les intervalles associés aux r_i et d_i sont disjoints. Le théorème des pyramides caractérise un ensemble dominant de séquences S_{dom} pour lequel il est possible de déterminer pour chaque travail, en temps polynomial, les performances au mieux et au pire vis-à-vis du retard algébrique sur l'ensemble des scénarios de réalisation. Quel que soit le scénario de réalisation considéré, compatible avec le modèle par intervalles, l'ensemble S_{dom} contient toujours une solution optimale.*

Démonstration. La preuve de cette propriété est relativement évidente. Faire l'hypothèse que tous les intervalles associés aux r_i et d_i sont disjoints permer de garantir qu'il existe une relation d'ordre total entre ces paramètres et que le théorème des pyramides peut être appliqué. Soit S_{dom} l'ensemble dominant obtenu. Étant dominant, il contient bien entendu une solution optimale quel que soit le scénario de réalisation considéré dans la mesure où l'ordre relatif des dates de début au plus tôt et de fin au plus tard est inchangé. De plus, il est possible de déterminer dans S_{dom}, pour chaque travail j, la séquence la plus

favorable $Seq_j^{\min}$ et celle la plus défavorable $Seq_j^{\max}$ vis-à-vis du retard algébrique, ainsi que nous l'avons décrit dans la partie 4.3, cela indépendamment des valeurs de r_i, p_i et d_i. Connaissant $Seq_j^{\min}$, la valeur de $L_j^{\min}$ peut être calculée grâce à l'algorithme de la figure 4.10, en fixant pour chaque travail : $r_j = \underline{r}_j$, $p_j = \underline{p}_j$ et $d_j = \overline{d}_j$. Similairement, la valeur de $L_j^{\max}$, connaissant $Seq_j^{\max}$, peut être calculée grâce à l'algorithme de la figure 4.13, en fixant pour chaque travail $r_j = \overline{r}_j$, $p_j = \overline{p}_j$ et $d_j = \underline{d}_j$. □

Cette propriété est importante car elle permet de garantir que les performances d'un ensemble de séquences dominantes, caractérisé grâce au théorème des pyramides, sont robustes vis-à-vis d'un ensemble de scénarios potentiels de réalisation d'un ordonnancement. Nous nous proposons d'illustrer cette propriété dans la partie suivante en montrant comment déduire, à partir d'un problème V déterministe, un modèle par intervalles, capturant un ensemble de scénarios de réalisation, vis-à-vis duquel on peut garantir une performance robuste.

4.4.2 Un modèle d'incertitudes par intervalles

Afin d'illustrer la propriété précédente, nous considérons un problème $1|r_i|L_{\max}$ déterministe V dans lequel tous les paramètres r_i, p_i et d_i des travaux sont supposés connus. Afin de simplifier, nous supposons de plus que les valeurs des r_i et d_i sont toutes différentes. Nous montrons dans la suite du texte comment déduire à partir des valeurs nominales des r_i et d_i, que nous indicerons r_i^0 et d_i^0, un modèle de scénarios caractérisé par des intervalles $r_i \in [\underline{r}_i, \overline{r}_i]$ et $d_i \in [\underline{r}_i, \overline{d}_i]$ sur lequel il est possible de garantir une performance.

On suppose tout d'abord que les dates de début et de fin des n travaux sont classées par ordre croissant. On obtient un ordre total $x(1)_0 < x(2)_0 < \cdots < x(2n)_0$ contenant $2n$ éléments, où $x(i)_0$ désigne la date de début au plus tôt ou de fin au plus tard nominale d'un travail et $i \in [1, 2n]$ indique son numéro d'ordre. On pose alors arbitrairement la contrainte suivante :

$$\underline{x}(i) \leq x(i) \leq \overline{x}(i)$$

où $\forall i \in [2, 2n-1]$:

$$\begin{cases} \underline{x}(i) &= x(i)_0 - \frac{x(i)_0 - x(i-1)_0}{2} & \text{si } x(i)_0 - x(i-1)_0 \text{ est impair} \\ &= x(i)_0 - \frac{x(i)_0 - x(i-1)_0}{2} + 1 & \text{si } x(i)_0 - x(i-1)_0 \text{ est pair} \\ \overline{x}(i) &= x(i)_0 + \frac{x(i+1)_0 - x(i)_0}{2} & \end{cases}$$

Les divisions sont entières et imposent que la "marge" disponible entre $x(i)_0$ et $x(i+1)_0$ est également partagée entre $x(i)$ et $x(i+1)$ (sauf dans le cas où cette marge est paire). Pour les valeurs extrêmes de i, on impose $\underline{x}(1) = x(1)_0$ et $\overline{x}(2n) = x(2n)_0$. Ainsi, en respectant ces contraintes, la relation $precedes(x(i), x(i+1))$ est toujours vérifiée $\forall i \in 2n$, sans modification de la structure d'intervalles nominale.

Ce modèle par intervalles caractérise un ensemble de scénarios de réalisation ES possédant une cardinalité :

$$card(ES) = \prod_{i=1}^{2n} (\overline{x}(i) - \underline{x}(i))$$

De plus, $x(i) \in [\underline{x}(i), \overline{x}(i)]$ correspondant soit à la date de début au plus tôt r_j, soit à la date de fin au plus tard d_j d'un travail j, on peut déduire pour chaque travail les intervalles $[\underline{r}_j, \overline{r}_j]$ et $[\underline{d}_j, \overline{d}_j]$.

Les intervalles associés aux r_j et d_j étant disjoints, le théorème des pyramides peut être appliqué. Un ensemble dominant de séquences peut alors être déduit possédant des performances au mieux et au pire connues, calculées ainsi que défini dans la proposition 4.4.1. Remarquons de plus qu'un intervalle de durée $[\underline{p}_j, \overline{p}_j]$ quelconque peut éventuellement être pris en compte à ce stade, sans perte de généralité. Les performances obtenues pour l'ensemble dominant sont alors robustes vis-à-vis de tous les scénarios capturés par le modèle par intervalles.

Considérons par exemple le problème de 4 travaux décrit par le tableau 4.2. L'ensemble des travaux $T = \{1, 2, 3, 4\}$ caractérise la structure d'intervalles mono-pyramidale représentée sur la figure 4.15, le travail 1 étant l'unique sommet.

Travail	r_j	d_j	p_j
1	10	15	3
2	8	19	5
3	2	17	6
4	4	23	7

TAB. 4.2: Une instance de quatre travaux

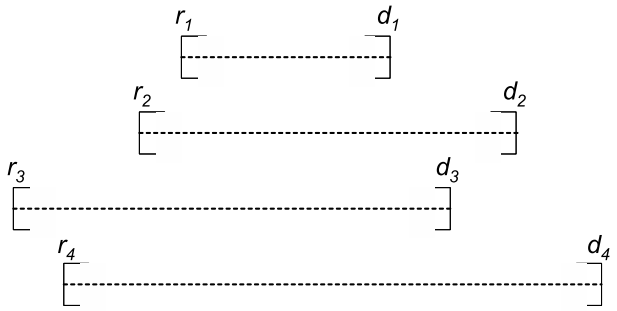


FIG. 4.15: Structure d'intervalles du problème considéré

On a :

$$(x(1) = r_3) < (x(2) = r_4) < (x(3) = r_2) < (x(4) = r_1)$$
$$< (x(5) = d_1) < (x(6) = d_3) < (x(7) = d_2) < (x(8) = d_4)$$

Selon le principe de répartition de la marge décrit précédemment, on déduit :

i	$\underline{x}(i)$	$\overline{x}(i)$
1	2	3
2	4	6
3	7	9
4	10	12
5	13	16
6	17	18
7	19	21
8	22	23

Si ces valeurs sont projetées sur l'ensemble des travaux, on obtient :

j	$[\underline{r}_j, \overline{r}_j]$	$[\underline{d}_j, \overline{d}_j]$
1	[10,12]	[13,16]
2	[7,9]	[19,21]
3	[2,3]	[17,18]
4	[4,6]	[22,23]

Ce modèle par intervalles caractérise $card(ES) = 2592$ scénarios de réalisation. Le théorème des pyramides caractérise quant à lui $(1+1)^3 = 8$ séquences dominantes. Le calcul des performances au mieux et au pire donne le diagramme de retards de la figure 4.16. Rappelons que le calcul de $L_j^{\min}$ utilise les valeurs $\underline{r}_i$, p_i et $\overline{d}_i$, tandis que celui de $L_j^{\max}$ utilise les valeurs $\overline{r}_i$, p_i et $\underline{d}_i$. On peut assurer que ces performances sont robustes vis-à-vis de tout scénario de réalisation possible, respectant le modèle par intervalles.

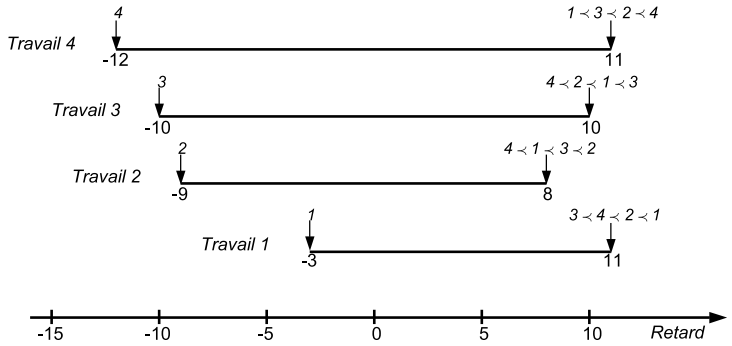


FIG. 4.16: Diagramme de retards du problème considéré

Chapitre 5

Une PSEP pour l'ordonnancement robuste

En utilisant les algorithmes de calcul de $L_i^{\min}$ et de $L_i^{\max}$ présentés dans le chapitre 4, nous décrivons dans ce chapitre une procédure par séparation et évaluation progressive (PSEP) pour l'ordonnancement robuste. Nous considérons ici un problème déterministe. L'objectif est de trouver un ordre partiel dominant qui soit aussi suffisant vis-à-vis de la minimisation du retard algébrique $L_{\max}$, c'est-à-dire tel que le théorème des pyramides caractérise des séquences dominantes qui soient toutes exclusivement optimales. Des résultats expérimentaux sont présentés illustrant les performances de l'algorithme.

5.1 Une première procédure par séparation et évaluation

5.1.1 Généralités

Dans cette partie, nous présentons une procédure θ d'ordonnancement robuste transformant un problème initial $V = \{j|r_j, d_j, p_j\}$ avec $j \in T$ - l'ensemble des travaux à ordonnancer, en un ensemble de nouveaux problèmes $V' = \{j'|r_{j'} \geq r_j, p_{j'} = p_j, d_{j'} \leq d_j\}$, tels que l'ensemble de séquences dominantes $S_{\text{dom}}^{V'}$, associé à chaque problème V', soit inclus dans l'ensemble dominant S_{dom}^{V} et optimal vis-à-vis de la minimisation du retard $L_{\max}$. Cette procédure a donc pour but de modifier la structure d'intervalles associée à un problème, en réduisant certaines fenêtres d'exécution de travail, de sorte que l'ensemble dominant caractérisé par le théorème des pyramides soit optimal pour chaque structure d'intervalles déterminée.

Nous nous proposons de construire θ en adoptant un schéma classique de PSEP. Une arborescence de recherche est donc progressivement construite dont la racine est le problème initial V. Chaque nœud de l'arborescence est séparé en un ensemble de nœuds fils, chaque fils correspondant à un nouveau problème $V' = \{j'|r_{j'} \geq r_j, p_{j'} = p_j, d_{j'} \leq d_j\}$.

Un nœud est étiqueté par une borne inférieure et une borne supérieure du critère $L_{\max}$, correspondant respectivement aux valeurs de $\max_{j \in T} L_j^{\min}$ et $\max_{j \in T} L_j^{\max}$, calculées comme décrit dans le chapitre 4. À un niveau de l'arborescence, le branchement sur un nœud fils est effectué en choisissant celui possédant la meilleure borne supérieure.

Lorsqu'un nœud ne peut plus être séparé, la procédure considère le nœud fils en amont, le plus proche, non encore exploré. Un nœud, et éventuellement la sous-arborescence correspondante, est éliminé dès que sa borne inférieure est supérieure à la meilleure borne supérieure connue. À la fin de la procédure, toutes les feuilles restantes de l'arborescence correspondent à des structures d'intervalles optimales.

Remarquons que les structures d'intervalles optimales caractérisent un ensemble flexible de solutions, ces dernières étant optimales vis-à-vis du problème déterministe initial. La méthode est donc qualifiée de robuste dans le sens où elle met en évidence une flexibilité séquentielle, potentiellement exploitable au sein d'une procédure d'ordonnancement en ligne. Pour une structure d'intervalles optimale, il est toutefois également envisageable de déterminer, ainsi que cela a été décrit dans le chapitre 4, un ensemble de scénarios de réalisation de l'ordonnancement vis-à-vis duquel une performance robuste peut être garantie. Dans ce cas, on peut assurer que toute séquence caractérisée est optimale pour les valeurs nominales de r_i, p_i et d_i et que, si on s'écarte de ces valeurs nominales, dans le respect du modèle par intervalles, alors :

1. la détérioration maximale du retard algébrique de chaque travail est connue ;
2. il existe une solution optimale dans l'ensemble de séquences caractérisé quel que soit le scénario considéré.

5.1.2 Principe de séparation et d'évaluation

Le nœud racine de l'arborescence de recherche représente la structure d'intervalles initiale caractérisant un ensemble de séquences dominantes grâce au théorème des pyramides. Nous utilisons deux valeurs $\max_{j \in T}(L_j^{\min})$ et $\max_{j \in T}(L_j^{\max})$, notées respectivement $L^{\min}$ et $L^{\max}$, comme les deux bornes inférieure et supérieure de ce nœud.

Le mécanisme de séparation que nous proposons permet de séparer un ensemble de séquences dominantes en une bipartition sans créer de nouvelles séquences. Considérons en un nœud de l'arborescence, le chemin critique C associé à la séquence produisant la borne supérieure du retard algébrique $L^{\max}$. À partir de ce chemin, sont déterminés un travail *pivot*, correspondant à un sommet de ce chemin, et un travail *libre*, non-sommet, que nous séquencerons soit à droite, soit à gauche du pivot. Pour déterminer le travail pivot et le travail libre, nous distinguons trois cas (notons j le travail engendrant la borne supérieure $L^{\max}$) :

- si j est un travail non-sommet, alors le travail pivot est le sommet de la pyramide d'indice $v(j)$ à laquelle j appartient, celui-ci étant nécessairement sur le chemin critique C ; le travail libre choisi pour séparer est celui le plus proche à droite sur C de ce pivot ;
- si j est le sommet de la dernière pyramide du chemin critique, et si cette dernière pyramide contient d'autres travaux, alors le travail pivot est j et le travail libre choisi est le premier travail sur C à gauche de j ;
- si j est le sommet de la dernière pyramide, et si cette pyramide ne contient aucun autre travail, alors le travail pivot est le prochain sommet rencontré en parcourant de droite à gauche le chemin critique C dont la pyramide contient encore des travaux non-sommets ; le travail libre choisi est celui le plus proche à droite du pivot.

Notons respectivement i et α le travail libre à pivoter et le travail pivot, déterminés comme décrit précédemment. Deux branchements sont alors envisagés : soit α précède i, soit i précède α. Compte tenu de l'ordre partiel fixé par le théorème des pyramides, ces deux précédences peuvent être imposées de la façon suivante :

- $\alpha \prec i$ peut être imposé en actualisant r_i de sorte que $r_i \leftarrow r_\alpha$. En effet, α étant un sommet, le théorème des pyramides garantit que si $r_j \geq r_\alpha$ alors α précède j dans toute séquence dominante qu'il caractérise (cf. figure 5.1) ;

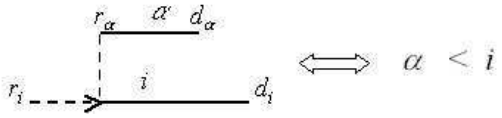


FIG. 5.1: Actualisation de r_i par r_α

- similairement, $i \prec \alpha$ peut être imposé en actualisant d_i de sorte que $d_i \leftarrow d_\alpha$. En effet, α étant un sommet, le théorème des pyramides garantit que si $d_j \leq d_\alpha$ alors j précède α dans toute séquence dominante qu'il caractérise (cf. figure 5.2).

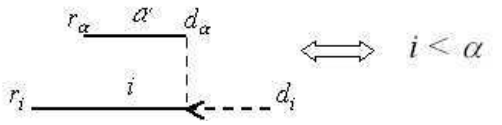


FIG. 5.2: Actualisation de d_i par d_α

L'algorithme pour déterminer le travail pivot α et le travail libre i est décrit dans la figure 5.3.

L'évaluation de chaque branchement est ensuite effectuée par le calcul des bornes inférieure et supérieure du retard, $L^{\min}$ et $L^{\max}$. Le branchement possédant la meilleure borne

```
Proc SetPivotLibre(travail_pivot, travail_libre, T)
    mettre le pointeur du chemin critique à la fin ;
    sommet_i ← dernier sommet sur le chemin critique ;

    TantQue la pyramide du sommet_i ne contient pas de travaux non-sommets
        sommet_i ← sommet précédent du sommet_i ;
    FinTantQue

    travail_pivot ← sommet_i ;

    Si travail i à gauche du travail pivot n'est pas un sommet alors
        travail_libre ← ce travail i ;
    SiNon
        chercher le travail i à droite du travail pivot, qui est le plus proche
                                                    et qui n'est pas un sommet ;
        travail_libre ← travail i ;
    FinSi
FinProc
```

FIG. 5.3: Algorithme pour déterminer le travail pivot et le travail libre

supérieure est ensuite choisi pour la séparation suivante.

La séparation d'un nœud de l'arborescence s'arrête lorsque le chemin critique associé ne contient plus que des sommets ou lorsque les deux bornes $L^{\min}$ et $L^{\max}$ sont égales. Dans ce dernier cas, la valeur des bornes est la valeur optimale locale de ce branchement.

Ce mécanisme de séparation et d'évaluation est représenté sur l'algorithme nommé TCOR décrit sur la figure 5.4.

5.1.3 Illustration de l'algorithme TCOR

Reprenons l'exemple du tableau 4.1 (cf. chapitre 4), caractérisé initialement par quatre sommets : $s_1 = 2$, $s_2 = 4$, $s_3 = 5$ et $s_4 = 7$; et quatre pyramides correspondantes $P_1 = \{1, 3, 6\}$, $P_2 = \{1, 6\}$, $P_3 = \{1\}$ et $P_4 = \emptyset$. L'application de l'algorithme TCOR passe par les étapes suivantes :

Étape 1 : La racine N_0 de l'arborescence est évaluée par le couple $\max_{i \in T} L_i^{\min} = L_4^{\min} = -2$ et $\max_{i \in T} L_i^{\max} = L_4^{\max} = 11$. Le chemin critique produisant $\max_{i \in T} L_i^{\max}$ est $2 \prec 3 \prec 6 \prec 1 \prec 4$. Le travail 4, étant sommet de P_2 et contenant des travaux non-sommets, est choisi comme pivot. Le travail 1, appartenant à P_2, étant le plus proche du pivot, est choisi comme travail libre pour pivoter. Deux branchements sont envisagés : celui

```
Proc TCOR(Solution, L_max^optimal, T)(*)
    L_max^optimal ← +∞ ;
    Solution ← ∅ ; /* solution est une liste de SIs optimales */
    A_traiter ← ∅ ; /* A_traiter est la liste de SIs non encore traitées */
    Calculer les bornes pour Noeud_racine ;
    A_traiter ← A_traiter ∪ Noeud_racine ;
    TantQue A_traiter ≠ ∅ faire
        Noeud_courant ← dépiler A_traiter ;
        Si L^min = L^max alors        /* une SI optimale locale */
            Si L^min < L_max^optimal alors
                L_max^optimal ← L^min ;
                Solution ← ∅ ;
                Solution ← Solution ∪ Noeud_courant ;
            FinSi
            Si L^min = L_max^optimal alors
                Solution ← Solution ∪ Noeud_courant ;
            FinSi
            Si L^min > L_max^optimal alors
                Couper Noeud_courant ;
            FinSi

        SiNon
            SetPivotLibre(Travail_pivot, Travail_libre, T) ;
            Séparer Noeud_courant en deux fils : Fils_gauche et Fils_droit ;
            Évaluer Fils_gauche et Fils_droit ;
            /* Traiter Fils_gauche possédant la plus grande borne supérieure */
            Si L_fils_gauche^min ≤ L_max^optimal alors
                A_traiter ← A_traiter ∪ Fils_gauche ;
            SiNon
                Couper Fils_gauche ;
            FinSi
            /* Traiter Fils_droit possédant la plus petite borne supérieure */
            Si L_Fils_droit^min ≤ L_max^optimal alors
                A_traiter ← A_traiter ∪ Fils_droit ;
            SiNon
                Couper Fils_droit ;
            FinSi
        FinSi
    FinTantQue
FinProc
```

(∗) SI : structure d'intervalles. $L^{\min}$, $L^{\max}$ sont les retards au mieux et au pire du nœud courant. $L_{\max}^{\text{optimal}}$ est le meilleur $L_{\max}$ connu, et à la fin de la PSEP, nous obtenons la valeur optimale.

FIG. 5.4: Algorithme TCOR

où $r_1 \leftarrow r_4$ $(4 \prec 1)$ et celui où $d_1 \leftarrow d_4$ $(1 \prec 4)$, qui conduisent à créer les deux nouveaux nœuds N_1 et N_2.

Les nouvelles bornes $\max_{i \in T} L_i^{\min}$ et $\max_{i \in T} L_i^{\max}$ pour ces deux nœuds sont indiquées sur la figure ainsi que le chemin critique associé (elles restent fixées par le travail 4). À la prochaine itération, le branchement est effectué sur le nœud N_1 car sa borne supérieure est meilleure que celle de N_2 (figure 5.5).

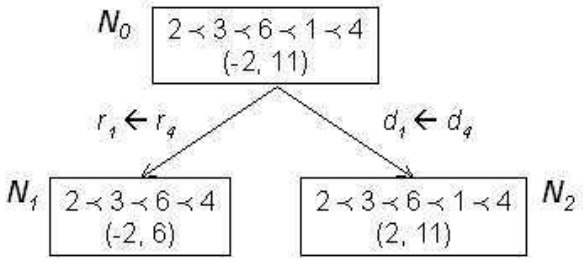


FIG. 5.5: Séparation du nœud racine N_0

Étape 2 : Considérons à présent la séparation du nœud N_1. Le chemin critique du nœud est $2 \prec 3 \prec 6 \prec 4$. Le travail 4, le dernier sommet sur le chemin critique, est choisi comme le travail pivot. Dans la pyramide du sommet 4, il y a un seul travail non sommet, le travail 6, qui est donc choisi comme le travail libre à pivoter.

Deux nouveaux branchements N_3 et N_4 sont envisagés : celui où $r_6 \leftarrow r_4$ et celui où $d_6 \leftarrow d_4$. Les nouvelles bornes sont à nouveau calculées pour N_3 et N_4. Dans l'étape suivante, le branchement est effectué sur N_3 qui possède la meilleure borne supérieure.

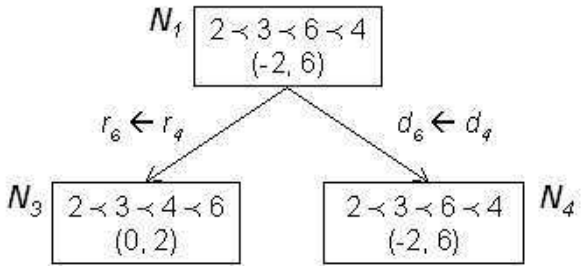


FIG. 5.6: Séparation du nœud N_1

Étape 3 : Considérons le nœud N_3. Le travail 3 est le seul travail non sommet sur le chemin critique $2 \prec 3 \prec 4 \prec 6$. Il est donc choisi pour le branchement et le travail 2, étant le sommet de la pyramide à laquelle 3 appartient, est choisi comme le travail pivot. Nous obtenons donc deux nouveaux nœuds N_5 et N_6 correspondant aux actualisations $r_3 \leftarrow r_2$

et $d_3 \leftarrow d_2$.

Les deux branchements N_5 et N_6 ont leurs deux bornes égales, à 2 pour N_5, à 0 pour N_6, la séparation est donc arrêtée. De plus, la borne maximale de N_6 étant égale à 0, les deux nœuds N_2 et N_5 sont coupés puisque leurs bornes minimales sont supérieures à cette valeur. Il ne reste donc plus que N_4 à développer, et le meilleur retard optimal connu est 0.

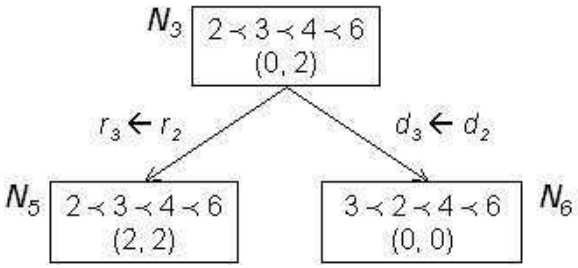


FIG. 5.7: Séparation du nœud N_3

Étape 4 : On considère N_4. Le chemin critique produisant la borne supérieure est $2 \prec 3 \prec 6 \prec 4$. Le travail 4 est un sommet dont la pyramide ne contient aucun autre travail. Le sommet précédent est le travail 2. Sa pyramide étant non vide, il est choisi comme pivot. Le pivotement concerne le travail 3 (le plus proche de 2). Deux nouveaux nœuds N_7 et N_8 sont donc envisagés : celui où $r_3 \leftarrow r_2$ et celui où $d_3 \leftarrow d_2$. Après avoir calculé les bornes, N_8 est choisi pour continuer la séparation.

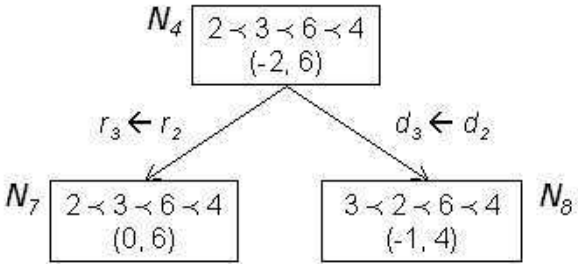


FIG. 5.8: Séparation du nœud N_4

Étape 5 : On considère le nœud N_8. Sur le chemin critique, il ne reste qu'un seul travail non sommet, 6, et le sommet 2 correspondant est alors choisi comme le travail pivot. Deux nouveaux nœuds N_9 et N_{10} sont envisagés : celui où $r_6 \leftarrow r_2$ et celui où $d_6 \leftarrow d_2$.

Les deux bornes de N_{10} sont égales à -1, on arrête donc la séparation. De plus, les nœuds N_6, N_7 et N_9 peuvent être coupés, leurs bornes minimales étant supérieures à cette

valeur. Il ne reste donc plus aucun nœud à développer et seule la feuille N_{10} est une solution optimale du problème.

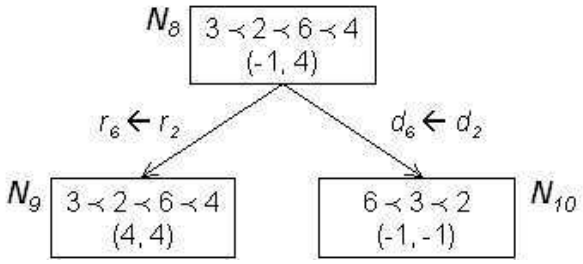


FIG. 5.9: Séparation du nœud N_8

Les séquences optimales de l'ensemble dominant associé à ce problème sont donc toutes caractérisées par le nœud N_{10}. Le retard optimal est donc égal à -1. Le chemin critique est $6 \prec 3 \prec 2$ et l'ensemble dominant de solutions associé à la structure d'intervalles obtenue pour le nœud N_{10} est $6 \prec 3 \prec 2 \prec 4 \prec (5-1) \prec 7$ où $(5-1)$ est un groupe de travaux permutables.

Nous remarquons que dans cet algorithme, la borne supérieure étant une borne "au pire" du retard algébrique, elle est évidemment peu efficace pour éliminer rapidement les nœuds de l'arborescence n'amenant à aucun optimum.

Il est donc intéressant d'avoir, en complément d'évaluation, une borne supérieure "au mieux", inférieure à $\max_{i \in T} L_i^{\max}$, qui permettrait de couper plus tôt l'arborescence de recherche, et donc d'améliorer les temps de calcul. Bien entendu, la borne $\max_{i \in T} L_i^{\max}$ doit être conservée puisqu'elle est utilisée lors des branchements.

Pour calculer cette borne supérieure "au mieux" du retard, nous avons utilisé la méthode de Carlier (cf. chapitre 4) qui permet de connaître directement la valeur optimale du retard. Dans la partie suivante, nous décrivons comment l'algorithme précédent peut être amélioré en tirant partie de la solution optimale déterminée par la PSEP de Carlier.

5.2 Une deuxième procédure par séparation et évaluation

5.2.1 Principes généraux

Nous proposons dans cette partie une version améliorée de l'algorithme TCOR, utilisant la solution de Carlier, pour déterminer un ensemble flexible de solutions optimales, dans une optique de robustesse. Nous montrons en particulier comment la séquence optimale

de Carlier peut être utilisée comme "guide" dans notre algorithme afin de converger plus rapidement vers une première structure d'intervalles optimale.

Au début de l'algorithme, la PSEP de Carlier est appliquée pour trouver la valeur optimale du retard $L_{\max}^{\text{optimal}}$ et une séquence optimale. Comme nous le verrons, cette séquence intervient dans les choix du travail pivot et du travail libre. De plus, le retard optimal est utilisé pour couper plus efficacement les nœuds de l'arborescence de recherche. Les deux évaluations associées à chaque nœud sont toujours la borne inférieure $\max_{i \in T} L_i^{\min}$ et la borne supérieure "au pire" $\max_{i \in T} L_i^{\max}$.

Pour chaque nœud de l'arborescence, nous déterminons le chemin critique produisant $\max_{i \in T} L_i^{\max}$. Sur ce chemin critique, nous cherchons un travail sommet s et un travail non sommet i, appartenant à la pyramide de sommet s, tels que l'ordre entre s et i sur le chemin critique est différent de celui sur la séquence optimale de Carlier. Par exemple, si $s \prec i$ dans la séquence de Carlier, alors i doit précéder s sur le chemin critique produisant $\max_{i \in T} L_i^{\max}$. Le travail pivot est alors le sommet s, et le travail libre est i. Le branchement adopté pour continuer la recherche après une séparation est celui où l'ordre entre s et i est le même que celui de la séquence de Carlier. Notons que s'il existe plusieurs choix possibles, le travail pivot s et le travail libre i sont choisis de sorte à privilégier le cas où la distance entre s et i est la plus grande possible sur le chemin critique.

Cette stratégie permet d'assurer que l'algorithme converge directement et rapidement vers une première structure d'intervalles optimale proche de la solution optimale de Carlier. En effet, au pire des cas, on obtient une première structure d'intervalles ne caractérisant que la séquence de Carlier. De plus, on remarque que privilégier le choix où un travail libre est très éloigné du pivot produit de plus fortes actualisations sur les dates de début au plus tôt ou de fin au plus tard de ce travail, ce qui permet de conclure plus vite quant à l'intérêt d'un nœud. Après avoir trouvé une première structure d'intervalles optimale, il est possible qu'en explorant les nœuds restants, il n'existe aucun couple pivot, tâche libre, dont l'ordre sur le chemin critique soit différent de celui de la séquence de Carlier. Dans ce cas, pivot et tâche libre sont déterminés en utilisant la règle indiquée dans la partie précédente.

L'algorithme de sélection du travail pivot et du travail libre est décrit sur la figure 5.10.

La nouvelle version de l'algorithme TCOR, nommée TCOR-2 est décrite dans la figure 5.11.

5.2.2 Illustration de l'algorithme TCOR-2

Nous reprenons l'exemple défini dans le tableau 4.1 pour illustrer l'application de l'algorithme TCOR-2, et discuter de sa performance relativement à TCOR. L'application de l'algorithme TCOR-2 conduit à passer par les étapes suivantes :

```
Proc SetPivotLibreAmeliore(TravailPivot, TravailLibre, Sequence_Carlier, T)
    TravailPivot ← NULL ; TravailLibre ← NULL ;
    Arret ← false ;
    Sommet_i ← Premier sommet du chemin critique ;
    TantQue Arret = false et Sommet_i ≠ NULL faire
        Initialiser i au dernier travail sur le chemin critique appartenant à la
                                                    pyramide de Sommet_i ;
        Ordre_chemin_critique ← s ≺ i ;
        TantQue Arret = false et i ≠ Sommet_i faire
            Déterminer Ordre_Sequence_Carlier entre s et i ;
            Si Ordre_Sequence_Carlier ≠ Ordre_chemin_critique alors
                Arret ← true ;
                TravailPivot ← Sommet_i ;
                TravailLibre ← i ;
            SiNon
                i ← Prec(i) ;   /* Prec(i) retourne le premier travail situé avant i sur
                      le chemin critique et appartenant à la pyramide de Sommet_i */
            FinSi
        FinTantQue
        Si Arret = false alors
            Initialiser i au premier travail sur le chemin critique appartenant à la
                                                    pyramide de Sommet_i ;
            Ordre_chemin_critique ← i ≺ s ;
            TantQue Arret = false et i ≠ Sommet_i faire
                Déterminer Ordre_Sequence_Carlier entre s et i ;
                Si Ordre_Sequence_Carlier ≠ Ordre_chemin_critique alors
                    Arret ← true ;
                    TravailPivot ← Sommet_i ;
                    TravailLibre ← i ;
                SiNon
                    i ← Suiv(i) ;   /* Suiv(i) retourne le premier travail situé après i
                     sur le chemin critique et appartenant à la pyramide de Sommet_i */
                FinSi
            FinTantQue
        FinSi
        Si Arret = false alors
            Sommet_i ← Prochain sommet à droite de Sommet_i
                                sur le chemin critique ;
        FinSi
    FinTantQue
    Si Arret = false alors
        SetPivotLibre(TravailPivot,TravailLibre,T) ;  /* utiliser la version 1 */
    FinSi
FinProc
```

FIG. 5.10: Algorithme amélioré pour choisir le travail pivot et le travail libre

```
Proc TCOR-2(Solution, L_max^optimal, T)(*)
    Solution ← ∅ ;
    A_traiter ← ∅ ;
    Calculer les bornes L^max, L^min pour le Noeud_racine ;
    A_traiter ← A_traiter ∪ Noeud_racine ;

    CARLIER(Sequence_Carlier, L_max^optimale, T) ;

    TantQue A_traiter ≠ ∅ faire
        Noeud_courant ← dépiler A_traiter ;
        Si L^max = L_max^optimale alors
            Solution ← Solution ∪ Noeud_courant ;
        SiNon
            Si L^min > L_max^optimal alors
                Couper le Noeud_courant ;
            SiNon
                SetPivotLibreAmeliore(TravailPivot, TravailLibre, Sequence_Carlier, T) ;
                Séparer Noeud_courant en deux fils : Fils_gauche et Fils_droit ;
                Évaluer Fils_gauche et Fils_droit ;

                /* Traiter Fils_gauche qui ne respecte pas la séquence de Carlier */
                Si L^min_Fils_gauche ≤ L_max^optimal alors
                    A_traiter ← A_traiter ∪ Fils_gauche ;
                SiNon
                    Couper Fils_gauche ;
                FinSi

                /* Traiter Fils_droit qui respecte si possible la séquence de Carlier */
                Si L^min_Fils_droit ≤ L_max^optimal alors
                    A_traiter ← A_traiter ∪ Fils_droit ;
                SiNon
                    Couper Fils_droit ;
                FinSi
            FinSi
        FinSi
    FinTantQue
FinProc
```

FIG. 5.11: Algorithme amélioré TCOR-2

Étape 1 : En appliquant, tout d'abord, l'algorithme de Carlier, nous obtenons la valeur optimale du retard $L_{\max}^{\text{optimal}} = -1$, et une solution optimale est $6 \prec 3 \prec 2 \prec 4 \prec 1 \prec 5 \prec 7$. Le nœud racine N_0, ayant la borne supérieure $\max_{i \in T} L_i^{\max} = 11 > L_{\max}^{\text{optimal}}$, nous séparons donc ce nœud. En utilisant l'algorithme *SetPivotLibreAmeliore* pour le chemin critique $2 \prec 3 \prec 6 \prec 1 \prec 4$, le travail 2 est choisi comme le travail pivot, et 3 est le travail libre à pivoter. Deux branchements sont alors envisagés : celui où $r_3 \leftarrow r_2$ et celui où $d_3 \leftarrow d_2$; qui conduisent à créer les nouveaux nœuds N_1 et N_2. (figure 5.12)

Les nouvelles bornes pour ces deux nœuds sont indiquées sur la figure 5.12. Le nœud N_1, ayant sa borne inférieure égale à 0, supérieure au $L_{\max}^{\text{optimal}}$, est coupé. Seul le nœud N_2 est donc à développer.

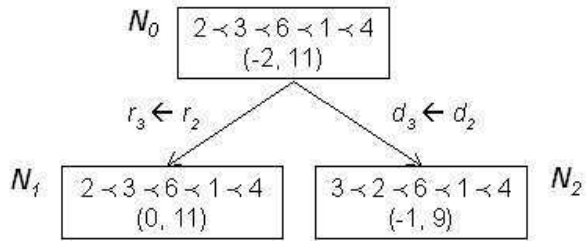


FIG. 5.12: Séparation du nœud N_0

Étape 2 : Le chemin critique de N_2 est $3 \prec 2 \prec 6 \prec 1 \prec 4$. Selon l'algorithme *SetPivot-LibreAmeliore*, on choisit le travail 3 comme le pivot, et le travail 6 comme le travail libre à pivoter. Deux nouveaux branchements N_3 et N_4 sont envisagés (cf. figure 5.13) : celui où $r_6 \leftarrow r_3$ et celui où $d_6 \leftarrow d_3$. La borne inférieure de ces deux nœuds est égale à -1, on ne peut donc couper aucun nœud. Le branchement est effectué sur N_4, dont l'ordre entre le pivot et la tâche libre est le même que celui de la séquence de Carlier.

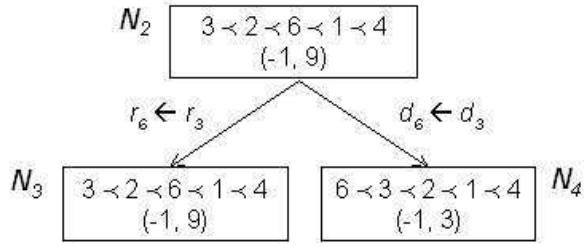


FIG. 5.13: Séparation du nœud N_2

Étape 3 : Le chemin critique du nœud N_4 est $6 \prec 3 \prec 2 \prec 1 \prec 4$. Le travail 4 est le nouveau pivot et le travail 1 est choisi pour pivoter. Deux nouveaux nœuds N_5 et N_6 sont créés correspondant aux actualisations $r_1 \leftarrow r_4$ et $d_1 \leftarrow d_4$. La borne minimale de N_6 est

égale à 2, qui est supérieure au retard optimal. N_6 est alors coupé de l'arborescence. De plus, N_5 ayant ses deux bornes minimale et maximale égales à -1, est un nœud optimal. Sa structure d'intervalles caractérise deux séquences optimales : $6 \prec 3 \prec 2 \prec 4 \prec (1-5) \prec 7$, où $(1-5)$ est un groupe de travaux permutables. Il reste le nœud N_3 à développer (figure 5.14).

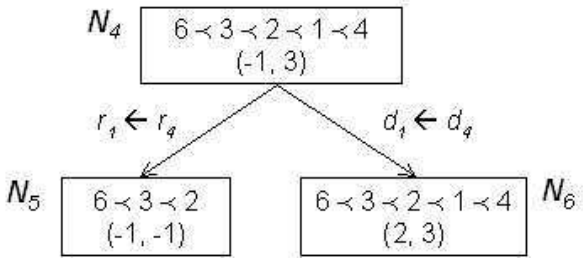


FIG. 5.14: Séparation du nœud N_4

Étape 4 : Pour le nœud N_3, le chemin critique est $3 \prec 2 \prec 6 \prec 1 \prec 4$. Et deux travaux 2 et 6 sont choisis respectivement comme le travail pivot et celui libre à pivoter. Deux nouveaux nœuds N_7 et N_8 sont à considérer : celui où $r_6 \leftarrow r_2$ et celui où $d_6 \leftarrow d_2$. N_8, possédant sa borne inférieure égale à 5 est éliminé. Nous continuons avec N_7 (figure 5.15).

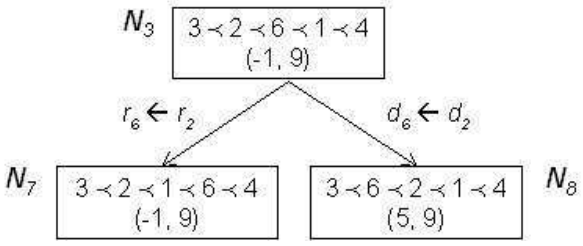


FIG. 5.15: Séparation du nœud N_3

Étape 5 : Pour le nœud N_7, le travail pivot choisi est le travail 4 et le travail libre est 1. Deux nouveaux nœuds N_9 et N_{10} sont créés pour deux actualisations : $r_1 \leftarrow r_4$ et $d_1 \leftarrow d_4$. Le nœud N_{10} est éliminé parce qu'il possède une borne minimale égale à 2. Nous continuons avec le nœud N_9 (figure 5.16).

Étape 6 : Pour le nœud N_9, le chemin critique est $3 \prec 2 \prec 6 \prec 4$. Les travaux 6 et 4 sont chosis respectivement comme le pivot et le travail libre à pivoter. Nous obtenons deux nœuds : N_{11} et N_{12}. N_{11} ayant ses deux bornes égales à 0 et N_{12} ayant ses deux bornes égales à 4, ils sont donc coupés, et l'exploration est achevée (figure 5.17).

Bien que le nombre d'étapes développées pour TCOR et TCOR-2 soit similaire, il est clair que TCOR-2 converge plus vite puisqu'une première structure d'intervalles optimale

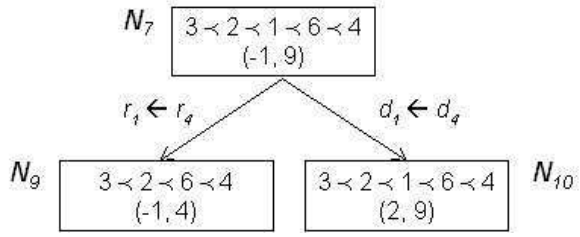


FIG. 5.16: Séparation du nœud N_7

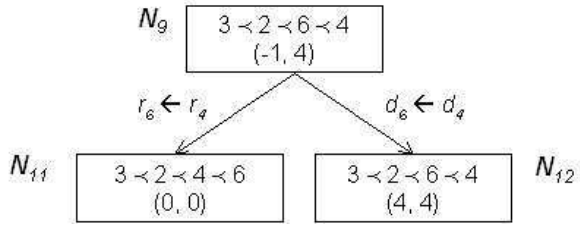


FIG. 5.17: Séparation du nœud N_9

est trouvée après trois étapes, au lieu de cinq pour TCOR. De plus le nombre de coupes directes est augmenté. Nous illustrons dans la suite les performances de TCOR-2 pour des instances de problème comptant de 10 à 500 travaux.

5.3 Expérimentations

Dans cette partie, nous présentons les résultats expérimentaux obtenus pour l'algorithme TCOR-2.

5.3.1 Principe de génération des instances de problème

Concernant la génération des instances du problème, nous avons repris la méthode utilisée dans [Esswein 03] dans laquelle les durées opératoires correspondent à des variables aléatoires uniformes choisies dans l'intervalle [1,100] et les dates de disponibilité sont déterminées en utilisant la procédure de génération de Hariri et Potts [Hariri & Potts 83]. Celle-ci consiste à modéliser chaque r_i par la réalisation d'une variable aléatoire uniforme dans l'intervalle $[0, \alpha \times \sum_{i=1}^{n} p_i]$ où α est un paramètre permettant de régler l'étalement des dates de disponibilité à partir de la date 0. Quatre valeurs de α ont été prises : $\{0,25; 0,5; 0,75; 1\}$. La génération des dates de fin au plus tard utilise le même principe en fixant chaque d_i à la réalisation d'une variable aléatoire uniforme dans l'intervalle $[(1-\beta) \times a \times \sum_{i=1}^{n} p_i, a \times \sum_{i=1}^{n} p_i]$. Cette fois, l'étalement se fait par la droite à partir de la date $a \times \sum_{i=1}^{n} p_i$, où $a \in \{100\%, 110\%\}$, permet de régler la marge temporelle maximale associée aux travaux. Le paramètre β contrôle quant à lui la dispersion des dates de fin. Il

est choisi tel que $\beta \in \{0,25; 0,5; 0,75; 1\}$. Afin d'assurer, pour chaque travail, la cohérence des dates de début au plus tôt et de fin au plus tard avec les durées opératoires, tout d_i dont la valeur serait inférieure à $r_i + p_i$ est modifié afin d'être égal à cette valeur.

En appliquant ce principe de génération, nous générons des instances de problème à une machine comptant 10, 50, 100 ou 500 travaux. Pour chacun de ces cas, nous avons générés 32 instances de problème, une pour chaque combinaison possible de valeur de α, β et a.

5.3.2 Résultats et analyse

Les résultats donnés par application de l'algorithme TCOR-2 sont indiqués dans les tableaux 5.1, 5.2, 5.3, 5.4. Les tests ont été menés sur une station Sun Blade 150 possédant 640Mo de RAM. On s'intéresse pour chaque problème à la valeur optimale du retard algébrique, au temps CPU nécessaire pour trouver la première structure d'intervalles optimale (et au nombre de séquences optimales qu'elle caractérise), au temps CPU pour trouver toutes les structures d'intervalles optimales (et au nombre de séquences optimales qu'elles caractérisent). Remarquons que les temps CPU indiqués incluent les temps de calcul liés à l'application de la PSEP de Carlier.

Pour les problèmes de **10** travaux, nous trouvons rapidement toutes les structures d'intervalles optimales, ces dernières caractérisant toutes les solutions optimales contenues dans l'ensemble initial de séquences dominantes. Le temps moyen mis pour trouver la première structure d'intervalles optimale est de 0.006 seconde. Pour trouver toutes les solutions optimales dans l'ensemble dominant initial, il faut en moyenne 0.016 seconde. Le nombre moyen de solutions optimales caractérisées par la première structure optimale est compris entre 8 et 25.

Pour les problèmes de **50** travaux, nous tentons aussi de trouver toutes les structures d'intervalles optimales en fixant cependant un temps limite de 1000 secondes. Les résultats montrent que, seules les arborescences de recherche des deux premiers jeux de données sont exhaustivement parcourues. Pour les autres jeux de données, à la fin de 1000 secondes, il reste toujours des structures à explorer. Le temps moyen pour trouver la première structure optimale est de 0.26 seconde et le nombre moyen de solutions optimales caractérisées est 40.10^9 !

Remarquons que le nombre moyen de solutions optimales caractérisées par la première structure d'intervalles optimale est souvent très important et qu'il n'est pas très "payant" de rechercher d'autres structures optimales, l'énumération de celles-ci nécessitant un temps CPU conséquent et ne caractérisant cependant pas énormément plus de solutions supplémentaires. C'est pourquoi dans les expérimentations à 100 et 500 travaux, nous avons volontairement stoppé la PSEP dès l'obtention de la première structure optimale.

Pour les problèmes de **100** travaux, le temps CPU moyen pour trouver la première structure d'intervalles optimale est de 1.96 seconde, et le nombre moyen de solutions optimales est 10^{12} !

Pour les problèmes de **500** travaux, le temps CPU pour trouver la première structure optimale varie entre 174 à 402 secondes (300 secondes en moyenne). On constate donc une augmentation importante du temps de calcul lorsque le nombre de travaux croît (même si cette augmentation est encore ici acceptable). Le nombre de solutions optimales caractérisées par cette première structure optimale est très important : 10^{270} au plus ! La flexibilité séquentielle mise en évidence par la première structure optimale est donc particulièrement remarquable.

Notons qu'il aurait été intéressant d'étudier la corrélation entre le nombre de sommets contenu dans la structure d'intervalles initiale et le temps de calcul. En effet, plus il y a de sommets dans la structure initiale et moins la flexibilité séquentielle est importante, et donc moins l'espace de recherche est grand. De ce fait, le temps de calcul devrait être plus faible lorqu'il y a beaucoup de sommets que dans le cas contraire. Nous n'avons cependant pas vérifier cette hypothèse lors des expérimentations.

Notons aussi que nous n'avons pas détecté d'influence particulière des choix des paramètres α, β et a quant à la flexibilité séquentielle mise en évidence. Nous pensons que cela est du au fait que chaque instance de problème est différente et qu'il n'est donc pas significatif de comparer les flexibilités séquentielles entre elles. C'est pourquoi il nous semblerait intéressant d'étudier des problèmes possédant des r_i et p_i fixés et de faire ensuite varier a entre 50% et 110% de sorte à étudier comment varie la flexibilité séquentielle produite selon que les dates de fin au plus tard sont plus ou moins proches de la valeur $r_i + p_i$.

Fichier de données	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*
jeu10_1	10	0.00	108	0.02	9	380
jeu10_2	-13	0.01	6	0.02	1	6
jeu10_3	57	0.01	2	0.05	13	58
jeu10_4	-18	0.00	9	0.00	1	9
jeu10_5	139	0.01	3	0.03	8	25
jeu10_6	34	0.00	4	0.01	1	4
jeu10_7	63	0.01	1	0.01	1	1
jeu10_8	81	0.01	1	0.01	1	1
jeu10_9	25	0.01	4	0.05	16	45
jeu10_10	-11	0.01	1	0.03	3	3
jeu10_11	63	0.01	4	0.01	1	4
jeu10_12	54	0.01	2	0.02	5	8
jeu10_13	77	0.00	1	0.00	2	2
jeu10_14	74	0.00	2	0.01	4	6
jeu10_15	20	0.00	1	0.01	2	3
jeu10_16	195	0.00	27	0.00	1	27
jeu10_17	94	0.01	4	0.02	4	40
jeu10_18	97	0.01	6	0.01	2	10
jeu10_19	20	0.01	1	0.04	1	1
jeu10_20	14	0.00	1	0.00	2	2
jeu10_21	196	0.01	6	0.01	2	8
jeu10_22	62	0.01	9	0.02	9	31
jeu10_23	76	0.01	2	0.01	1	2
jeu10_24	144	0.00	3	0.00	2	6
jeu10_25	82	0.01	16	0.02	7	44
jeu10_26	109	0.00	12	0.00	1	12
jeu10_27	26	0.00	1	0.05	2	2
jeu10_28	199	0.00	12	0.01	12	55
jeu10_29	32	0.01	1	0.01	1	1
jeu10_30	14	0.01	4	0.02	2	6
jeu10_31	67	0.00	4	0.00	1	4
jeu10_32	57	0.00	2	0.00	1	2

(1) : valeur optimale du $L_{\max}$;
(2) : temps CPU pour trouver la première structure d'intervalles optimale (en sec.) ;
(3) : nombre de solutions optimales caractérisées par la première structure d'intervalles optimale ;
(4) : temps CPU total (en sec.) ;
(5) : nombre de structures optimales trouvées ;
(6) : nombre total de solutions optimales trouvées ;

TAB. 5.1: Résultat pour les problèmes de 10 travaux

Fichier de données	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*
jeu50_1	106	0,32	24	242,85	10450	118994
jeu50_2	273	0,30	252	638,43	20856	46420
jeu50_3	242	0,29	1344	1000	53527	2.57e+07
jeu50_4	98	0,37	3072	1000	27677	4.96e+07
jeu50_5	430	0,17	4.9e+09	1000	57003	1.50e+12
jeu50_6	143	0,23	30240	1000	12581	2.26e+06
jeu50_7	519	0,11	8.71e+08	1000	58596	1.84e+11
jeu50_8	355	0,18	8.77e+11	1000	90582	3.31e+14
jeu50_9	92	0,27	1.51e+08	1000	52052	2.18e+10
jeu50_10	-173	0,27	8.67e+09	1000	52531	5.93e+11
jeu50_11	127	0,31	1344	1000	55222	942277
jeu50_12	-115	0,31	5.17e+07	1000	53699	1.09e+08
jeu50_13	61	0,33	16632	1000	32382	1.44e+07
jeu50_14	23	0,30	17280	1000	65365	3.58e+07
jeu50_15	396	0,201	1.10e+06	1000	83308	7.40e+06
jeu50_16	212	0,25	2160	1000	64777	1.04e+07
jeu50_17	29	0,35	12288	1000	70954	397186
jeu50_18	-186	0,30	3.02e+10	1000	40501	2.23e+12
jeu50_19	204	0,28	1024	1000	29033	1.50e+06
jeu50_20	46	0,30	2	1000	336	968
jeu50_21	213	0,33	10800	1000	24718	2.62e+08
jeu50_22	92	0,35	156800	1000	63242	3.45e+08
jeu50_23	273	0,25	3.62e+09	1000	85454	1.28e+12
jeu50_24	106	0,28	2880	1000	38740	7.04e+06
jeu50_25	230	0,33	34560	1000	51756	2.86e+08
jeu50_26	92	0,28	6.08e+06	1000	43584	8.74e+08
jeu50_27	255	0,22	1.03e+07	1000	87586	2.09e+09
jeu50_28	261	0,20	32256	1000	90401	1.63e+07
jeu50_29	406	0,20	5.45e+07	1000	61841	4.52e+11
jeu50_30	329	0,15	3.58e+11	1000	58516	1.17e+13
jeu50_31	228	0,20	8.25e+08	1000	66093	8.31e+11
jeu50_32	331	0,19	1.03e+06	1000	51032	8.21e+09

(1) : valeur optimale du $L_{\max}$;
(2) : temps CPU pour trouver la première structure d'intervalles optimale (en sec.) ;
(3) : nombre de solutions optimales contenues dans la première structure d'intervalles optimale ;
(4) : temps CPU total (en sec.) ;
(5) : nombre de structures optimales trouvées après 1000 secondes ;
(6) : nombre total de solutions optimales trouvées après 1000 secondes ;

TAB. 5.2: Résultat pour les problèmes de 50 travaux

Fichier de données	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*
jeu100_1	280	2,47	1.81809e+25	75	154	6.78996e+25
jeu100_2	240	1,80	1.44461e+08	85	193	1.48593e+08
jeu100_3	436	1,72	8.49506e+17	62	236	2.26021e+19
jeu100_4	573	1,46	4.05504e+06	56	40	1.97075e+08
jeu100_5	194	2,15	2.35996e+28	76	314	5.55741e+28
jeu100_6	215	2,14	2.06391e+12	100	240	5.47517e+14
jeu100_7	359	1,60	6.65736e+18	70	247	6.67072e+20
jeu100_8	651	1,62	3.59251e+06	28	69	8.21687e+08
jeu100_9	78	1,78	1.25415e+23	71	288	3.70878e+25
jeu100_10	-453	2,02	1.20545e+25	72	342	5.44152e+25
jeu100_11	58	2,12	2.78453e+23	95	279	1.80497e+24
jeu100_12	-434	2,15	6.63704e+24	101	382	1.42421e+25
jeu100_13	198	2,23	1.7636e+09	104	165	1.07801e+11
jeu100_14	103	2,33	4.74321e+16	104	144	1.09148e+17
jeu100_15	302	2,19	1.61741e+06	49	66	7.1139e+09
jeu100_16	217	1,91	1.91756e+25	42	268	6.28381e+25
jeu100_17	78	2,09	1.04744e+13	79	325	6.21506e+14
jeu100_18	-386	2,16	108000	102	156	177696
jeu100_19	169	2,48	5.09608e+08	97	180	2.95229e+10
jeu100_20	546	1,68	1.88968e+19	41	130	9.41278e+20
jeu100_21	223	2,13	2.18494e+13	72	172	7.24607e+15
jeu100_22	-463	1,36	1.63217e+57	50	317	5.97892e+57
jeu100_23	162	2,06	3.87072e+07	91	243	7.18658e+09
jeu100_24	97	2,46	3.38146e+10	94	108	8.82937e+11
jeu100_25	196	1,96	1.63002e+14	130	111	1.2327e+15
jeu100_26	5	2,63	6.9673e+08	103	3	4.18038e+09
jeu100_27	394	1,59	2.83262e+17	89	385	3.41536e+17
jeu100_28	114	1,83	8.448e+06	111	128	1.94684e+07
jeu100_29	307	1,36	3.932e+16	65	232	7.33485e+17
jeu100_30	364	1,78	2.29771e+12	66	182	2.66273e+13
jeu100_31	292	1,43	1.76664e+13	59	147	5.74937e+13
jeu100_32	155	2,15	7.1092e+15	105	168	5.85114e+16

(1) : valeur optimale du $L_{\max}$;
(2) : temps CPU pour trouver la première structure d'intervalles optimale (en sec.) ;
(3) : nombre de solutions optimales contenues dans la première structure d'intervalles optimale ;
(4) : nombre de structures d'intervalles restant à explorer après 10 secondes ;
(5) : nombre de structures optimales trouvées après 10 secondes ;
(6) : nombre total de solutions optimales trouvées après 10 secondes ;

TAB. 5.3: Résultat pour les problèmes de 100 travaux

Fichier de données	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*
jeu500_1	681	316,98	6.32369e+70	504	140	6.58095e+70
jeu500_2	228	296,93	1.43496e+32	694	38	8.40474e+32
jeu500_3	855	284,2	7.26098e+102	436	99	9.54448e+103
jeu500_4	171	341,62	3.12159e+62	587	30	2.15184e+63
jeu500_5	1491	178,96	7.73155e+178	391	379	2.23981e+179
jeu500_6	449	306,69	2.17282e+91	631	108	1.21638e+94
jeu500_7	897	173,58	2.74648e+178	358	384	5.09776e+178
jeu500_8	511	220,30	9.08958e+113	642	138	4.34929e+114
jeu500_9	74	314,33	1.00271e+271	379	137	1.35993e+271
jeu500_10	-2430	300,20	3.00558e+270	453	29	1.66541e+271
jeu500_11	108	402,88	7.86449e+174	492	14	1.35986e+175
jeu500_12	-1259	323,84	4.52119e+92	524	196	1.50082e+94
jeu500_13	116	395,48	3.76963e+35	589	4	9.67209e+35
jeu500_14	136	340,99	1.97291e+21	581	57	3.90273e+22
jeu500_15	523	346,02	7.28501e+71	435	55	2.90602e+72
jeu500_16	340	330,46	5.82611e+52	416	143	8.7669e+53
jeu500_17	91	324,02	9.60562e+85	506	198	1.40724e+86
jeu500_18	371	268,85	2.41948e+43	552	191	1.91018e+44
jeu500_19	168	369,01	1.32068e+28	570	18	1.48576e+29
jeu500_20	165	357,64	5.15065e+158	524	66	4.24793e+160
jeu500_21	318	310,11	2.47433e+64	595	155	3.8487e+65
jeu500_22	568	278,53	3.9056e+88	617	187	8.20968e+88
jeu500_23	1421	247,71	4.05917e+133	326	272	3.53363e+134
jeu500_24	1038	258,90	6.34166e+112	539	328	2.10417e+113
jeu500_25	1091	318,38	1.24867e+72	254	62	6.89845e+75
jeu500_26	79	382,60	1.63996e+41	661	6	4.5099e+41
jeu500_27	1096	223,94	1.58122e+83	424	164	4.23249e+85
jeu500_28	250	302,79	5.08119e+75	591	48	7.14543e+76
jeu500_29	249	281,60	5.78165e+70	653	96	2.85903e+72
jeu500_30	196	324,17	8.96471e+73	709	91	1.45078e+75
jeu500_31	588	219,92	1.34289e+120	481	224	3.59432e+121
jeu500_32	359	277,10	6.8407e+88	668	171	5.29501e+89

(1) : valeur optimale du $L_{\max}$;
(2) : temps CPU pour trouver la première structure d'intervalles optimale (en sec.) ;
(3) : nombre de solutions optimales contenues dans la première structure d'intervalles optimale ;
(4) : nombre de structures d'intervalles restant à explorer après 420 secondes ;
(5) : nombre de structures optimales trouvées après 420 secondes ;
(6) : nombre total de solutions optimales trouvées après 420 secondes ;

TAB. 5.4: Résultat pour les problèmes de 500 travaux

Chapitre 6

Une aide à la décision pour l'ordonnancement robuste

Ce chapitre s'intéresse à l'application des résultats présentés dans les chapitres 4 et 5 en vue de produire une aide à la décision pour l'ordonnancement robuste. L'aide à la décision a pour but d'aider un décideur à trouver un compromis flexibilité / performance satisfaisant. Dans un premier temps, une adaptation de l'algorithme TCOR-2 est présentée permettant de trouver un ensemble de solutions ε-optimales, c'est-à-dire un ensemble de solutions le plus vaste possible, n'induisant pas une variation du critère supérieure à un pourcentage fixé de sa valeur optimale. Dans un deuxième temps, un outil logiciel d'Aide à la Décision pour l'Ordonnancement Robuste (ADOR) est proposé. L'aide à la décision est ici basée sur une interaction avec un décideur, via un diagramme de retards utilisé ici en tant qu'objet d'interaction.

6.1 ε-optimalité

6.1.1 Principes de l'aide à la décision

En pratique, l'optimalité d'une solution n'est pas toujours requise, un décideur pouvant accepter une dégradation de qualité pour obtenir davantage de flexibilité séquentielle afin d'augmenter les chances pour que, lors de la mise en œuvre de la solution, les perturbations soient absorbées sans remise en cause des performances (cf. chapitre 1). Dans cette partie, une aide à la décision est proposée basée sur l'augmentation progressive par un décideur d'un coefficient ε, permettant de contrôler la dégradation de qualité. Pour chaque valeur de ε, une procédure de résolution est lancée dont l'objectif est de caractériser un ensemble de solutions ε-optimales, de cardinalité la plus grande possible. L'objectif de l'aide à la décision est de permettre au décideur de trouver la valeur de ε la plus petite possible, garantissant un niveau de flexibilité séquentielle satisfaisant.

Une solution est dite ε-optimale si l'écart relatif entre sa performance et la perfor-

mance optimale est inférieur ou égal à un seuil ε prédéterminé. Cette contrainte peut être représentée par la formule suivante :

$$\frac{|L_{\varepsilon-optimal} - L_{optimal}|}{L_{optimal}} \leq \varepsilon$$

6.1.2 Expérimentation

Dans le chapitre précédent, l'algorithme TCOR-2, permettant de caractériser un ensemble de solutions optimales pour un problème à une machine déterministe, à partir de la connaissance de sa structure d'intervalles initiale, a été décrit. Cet algorithme, qui nécessite la connaissance de la valeur optimale du retard algébrique (établie grâce à la PSEP de Carlier), peut facilement être adapté pour produire une solution ε-optimale. Il suffit en effet d'arrêter le développement d'un nœud de l'arborescence de recherche dès que sa borne supérieure du retard est inférieure ou égale à $L_{optimal} + \varepsilon L_{optimal}$ (au lieu de $L_{optimal}$ dans TCOR-2).

Dans la suite, les résultats expérimentaux pour les problèmes de 10, 50, 100 et 500 travaux sont présentés en fonction de différents niveaux de dégradation de la qualité : $\varepsilon = 10\%$, $\varepsilon = 30\%$ et $\varepsilon = 50\%$.

Remarquons qu'une diminution de performance (c'est-à-dire une augmentation de ε) s'accompagne généralement d'une diminution du temps de calcul. Toutefois, ce temps de calcul peut parfois augmenter. En effet, les coupes étant moins fréquentes du fait de l'augmentation de la valeur seuil de coupe (i.e. $L_{optimal} + \varepsilon L_{optimal}$), les évaluations de $L^{\min}$, $L^{\max}$ sont faites pour chaque nouveau nœud fils généré. Or, lorsqu'un nœud fils peut être coupé, seul le calcul de $L^{\min}$ est réalisé. Donc, une augmentation du nombre de nœuds à conserver conduit à une augmentation du nombre d'opérations d'évaluation, ce qui peut parfois augmenter le temps de calcul lorsque le nombre de séparations produites pour arriver à une première structure d'intervalles ε-optimale est proche de celui nécessaire pour parvenir à une première structure optimale.

Pour chaque type de problème, le tableau 6.1 indique la valeur moyenne des coefficients multiplicatifs s'appliquant au nombre de solutions caractérisées intialement (c'est-à-dire dans le cas où $\varepsilon = 0$), en fonction de ε. On constate que dans le cas où le nombre de travaux est faible, la flexibilité séquentielle supplémentaire est relativement faible. Ceci s'explique en partie par le fait que la variation autorisée sur le critère est plus faible en valeur absolue quand le nombre de travaux est petit que dans le cas contraire. Or, dans l'algorithme TCOR-2, la diminution du critère entre deux nœuds successifs de l'arborescence est au plus égale à la durée opératoire du travail libre choisi pour pivoter. Par conséquent, si les durées opératoires sont toutes supérieures à la variation autorisée, alors il n'est pas possible d'obtenir un ensemble optimal plus grand que celui obtenu initialement. Une autre

	$\varepsilon = 10\%$	$\varepsilon = 30\%$	$\varepsilon = 50\%$
10 travaux	1	2	4
50 travaux	21	257	41e+3
100 travaux	71	155e+3	38e+9
500 travaux	7e+7	7e+26	5e+54

TAB. 6.1: Variation moyenne du nombre initial de solutions

raison réside dans le fait que si la structure d'intervalles initiale contient beaucoup de sommets, alors il est difficile de produire beaucoup de flexibilité séquentielle (une structure ne comptant que des sommets ne caractérise qu'une seule solution quelle que soit la valeur de ε).

Par contre, dans le cas où le nombre de travaux devient conséquent, alors le nombre de solutions augmente exponentiellement avec la valeur de ε. En effet, dans ce cas, la variation en valeur absolue du critère est très fréquemment supérieure aux durées opératoires des travaux, ce qui induit qu'il n'est pas nécessaire d'avoir beaucoup de pivotements pour trouver une solution ε-optimale (la recherche est moins profonde). De ce fait, les travaux ont tendance à appartenir à plusieurs pyramides, ce qui augmente la cardinalité de l'ensemble dominant de façon exponentielle.

Les résultats complets des évaluations sont détaillés dans les tableaux suivants.

	$\varepsilon = 0\%$		$\varepsilon = 10\%$		$\varepsilon = 30\%$		$\varepsilon = 50\%$	
Fichier	temps	nb de sols	temps	nb de sols	temps	nb de sols	temps	nb de sols
jeu10_1	0.00	108	0,01	108	0,02	432	0,01	432
jeu10_2	0.01	6	0,01	6	0,02	6	0,02	6
jeu10_3	0.01	2	0,01	2	0,03	2	0,02	10
jeu10_4	0.00	9	0,01	9	0,01	9	0,02	9
jeu10_5	0.01	3	0,01	9	0,01	144	0,01	144
jeu10_6	0.00	4	0,01	8	0,02	8	0,01	24
jeu10_7	0.01	1	0,01	1	0,04	1	0,04	1
jeu10_8	0.01	1	0,01	1	0,03	2	0,03	2
jeu10_9	0.01	4	0,02	4	0,04	4	0,03	4
jeu10_10	0.01	1	0,01	1	0,03	1	0,03	1
jeu10_11	0.01	4	0,01	4	0,01	8	0,02	8
jeu10_12	0.01	2	0,02	2	0,04	8	0,03	8
jeu10_13	0.00	1	0,01	1	0,03	4	0,02	27
jeu10_14	0.00	2	0,01	2	0,02	10	0,02	10
jeu10_15	0.00	1	0,02	1	0,04	1	0,03	1
jeu10_16	0.00	27	0,00	27	0,02	27	0,01	81
jeu10_17	0.01	4	0,01	12	0,03	24	0,02	80
jeu10_18	0.01	6	0,01	6	0,03	6	0,03	6
jeu10_19	0.01	1	0,01	1	0,03	1	0,03	1
jeu10_20	0.00	1	0,02	1	0,04	1	0,03	1
jeu10_21	0.01	6	0,01	6	0,01	96	0,01	192
jeu10_22	0.01	9	0,01	12	0,01	36	0,01	36
jeu10_23	0.01	2	0,01	3	0,02	6	0,01	27
jeu10_24	0.00	3	0,01	3	0,02	3	0,01	6
jeu10_25	0.01	16	0,01	16	0,02	16	0,02	24
jeu10_26	0.00	12	0,01	12	0,01	12	0,01	12
jeu10_27	0.00	1	0,01	1	0,03	1	0,03	1
jeu10_28	0.00	12	0,01	18	0,02	32	0,00	64
jeu10_29	0.01	1	0,01	1	0,02	1	0,02	1
jeu10_30	0.01	4	0,01	4	0,03	4	0,02	4
jeu10_31	0.00	4	0,00	4	0,01	4	0,01	4
jeu10_32	0.00	2	0,01	2	0,02	4	0,01	8

- ε : pourcentage de déviation par rapport à l'optimalité.
- Fichier : fichier de données.
- temps : temps (en sec.) pour trouver la première structure d'intervalles optimale.
- nb de sols : nombre de solutions optimales contenues dans la première structure optimale.

TAB. 6.2: ε-optimalité pour les problèmes de 10 travaux

	$\varepsilon = 0\%$		$\varepsilon = 10\%$		$\varepsilon = 30\%$		$\varepsilon = 50\%$	
Fichier	temps	nb de sols	temps	nb de sols	temps	nb de sols	temps	nb de sols
jeu50_1	0,32	24	0,33	24	0,33	120	0,30	720
jeu50_2	0,30	252	0,32	756	0,30	7392	0,28	458304
jeu50_3	0,29	1344	0,28	36288	0,27	907200	0,23	3.886e+08
jeu50_4	0,37	3072	0,38	6144	0,39	21504	0,38	483840
jeu50_5	0,17	4.9e+09	0,16	1.377e+10	0,14	3.703e+12	0,11	7.110e+13
jeu50_6	0,23	30240	0,23	60480	0,24	120960	0,23	483840
jeu50_7	0,11	8.71e+08	0,08	4.354e+09	0,06	6.370e+11	0,03	5.255e+13
jeu50_8	0,18	8.77e+11	0,15	1.053e+13	0,14	5.477e+14	0,14	5.477e+14
jeu50_9	0,27	1.51e+08	0,27	1.517e+08	0,27	1.973e+09	0,25	5.031e+09
jeu50_10	0,27	8.67e+09	0,28	8.670e+09	0,24	7.511e+11	0,24	7.511e+11
jeu50_11	0,31	1344	0,34	1344	0,32	9152	0,32	64896
jeu50_12	0,31	5.17e+07	0,31	5.174e+07	0,31	8.279e+08	0,32	8.279e+08
jeu50_13	0,33	16632	0,34	465696	0,33	465696	0,32	465696
jeu50_14	0,30	17280	0,30	34560	0,31	34560	0,31	34560
jeu50_15	0,201	1.10e+06	0,22	2.322e+07	0,24	2.654e+08	0,22	3.214e+11
jeu50_16	0,25	2160	0,27	2160	0,25	144000	0,28	2.448e+06
jeu50_17	0,35	12288	0,37	12288	0,38	12288	0,37	21504
jeu50_18	0,30	3.02e+10	0,32	3.029e+10	0,32	5.149e+11	0,29	4.721e+13
jeu50_19	0,28	1024	0,29	3072	0,28	3072	0,29	82944
jeu50_20	0,30	2	0,31	2	0,31	4	0,31	4
jeu50_21	0,33	10800	0,32	4.147e+06	0,31	2.488e+07	0,29	5.334e+09
jeu50_22	0,35	156800	0,35	156800	0,34	5.644e+06	0,34	1.075e+07
jeu50_23	0,25	3.62e+09	0,26	8.888e+10	0,24	5.333e+11	0,23	7.999e+12
jeu50_24	0,28	2880	0,29	34560	0,29	92160	0,26	737280
jeu50_25	0,33	34560	0,32	995328	0,34	995328	0,33	1.791e+07
jeu50_26	0,28	6.08e+06	0,27	1.216e+07	0,26	1.216e+07	0,28	1.216e+07
jeu50_27	0,22	1.03e+07	0,22	2.985e+08	0,18	1.576e+11	0,16	5.016e+12
jeu50_28	0,20	32256	0,23	32256	0,22	1.892e+06	0,22	2.081e+07
jeu50_29	0,20	5.45e+07	0,17	8.734e+08	0,12	2.330e+12	0,06	7.847e+16
jeu50_30	0,15	3.58e+11	0,13	7.166e+11	0,13	1.044e+13	0,08	4.171e+15
jeu50_31	0,20	8.25e+08	0,18	3.540e+11	0,16	1.027e+12	0,15	5.016e+12
jeu50_32	0,19	1.03e+06	0,19	1.036e+06	0,19	8.294e+06	0,16	1.866e+09

- ε : pourcentage de déviation par rapport à l'optimalité.
- Fichier : fichier de données.
- temps : temps (en sec.) pour trouver la première structure d'intervalles optimale.
- nb de sols : nombre de solutions optimales contenues dans la première structure optimale.

TAB. 6.3: ε-optimalité pour les problèmes de 50 travaux

	$\varepsilon = 0\%$		$\varepsilon = 10\%$		$\varepsilon = 30\%$		$\varepsilon = 50\%$	
Fichier	temps	nb de sols	temps	nb de sols	temps	nb de sols	temps	nb de sols
jeu100_1	2,47	1.818e+25	2,16	1.222e+27	2,08	3.374e+28	2,00	1.411e+31
jeu100_2	1,80	1.444e+08	1,63	1.444e+08	1,60	7.223e+08	1,53	6.500e+09
jeu100_3	1,72	8.495e+17	1,50	6.116e+19	1,28	8.672e+21	1,09	3.283e+26
jeu100_4	1,46	4.055e+06	1,32	1.459e+08	1,24	4.230e+11	1,19	2.514e+14
jeu100_5	2,15	2.359e+28	2,07	2.359e+28	2,00	3.096e+30	1,92	3.721e+32
jeu100_6	2,14	2.063e+12	1,81	1.506e+15	1,69	1.799e+18	1,40	6.258e+23
jeu100_7	1,60	6.657e+18	1,33	4.077e+21	1,20	8.834e+23	1,18	7.951e+24
jeu100_8	1,62	3.592e+06	1,73	1.077e+08	1,74	2.479e+11	1,71	2.225e+17
jeu100_9	1,78	1.254e+23	1,62	1.254e+23	1,57	4.389e+24	1,55	1.364e+26
jeu100_10	2,02	1.205e+25	1,82	3.254e+26	1,73	1.739e+29	1,69	3.956e+30
jeu100_11	2,12	2.784e+23	1,92	2.784e+23	1,88	1.069e+24	1,83	1.279e+26
jeu100_12	2,15	6.637e+24	2,00	3.534e+26	1,84	2.025e+31	1,72	6.711e+33
jeu100_13	2,23	1.763e+09	2,06	1.763e+09	2,01	3.950e+11	2,03	3.389e+14
jeu100_14	2,33	4.743e+16	2,13	4.743e+16	2,11	7.114e+16	2,10	1.280e+18
jeu100_15	2,19	1.617e+06	2,05	1.617e+06	2,24	7.472e+08	2,18	1.280e+18
jeu100_16	1,91	1.917e+25	1,77	1.046e+26	1,85	3.350e+29	1,85	1.396e+31
jeu100_17	2,09	1.047e+13	1,92	1.047e+13	1,89	1.047e+13	1,88	2.094e+14
jeu100_18	2,16	108000	1,94	3.024e+06	1,86	1.679e+07	1,83	4.976e+08
jeu100_19	2,48	5.096e+08	2,27	6.370e+08	2,22	1.605e+11	2,17	1.748e+15
jeu100_20	1,68	1.889e+19	1,57	2.902e+22	1,60	4.517e+24	1,54	1.143e+27
jeu100_21	2,13	2.184e+13	1,91	4.719e+15	1,81	3.470e+20	1,80	2.890e+23
jeu100_22	1,36	1.632e+57	1,24	2.285e+58	1,13	9.659e+59	1,10	9.142e+60
jeu100_23	2,06	3.870e+07	1,85	1.741e+08	1,77	9.510e+10	1,73	4.970e+12
jeu100_24	2,46	3.381e+10	2,25	3.381e+10	2,22	2.028e+11	2,16	5.477e+13
jeu100_25	1,96	1.630e+14	1,76	1.630e+14	1,71	1.222e+15	1,71	1.222e+15
jeu100_26	2,63	6.967e+08	2,35	6.967e+08	2,34	6.967e+08	2,34	6.967e+08
jeu100_27	1,59	2.832e+17	1,33	2.427e+21	1,19	3.506e+24	1,05	8.053e+29
jeu100_28	1,83	8.448e+06	1,65	8.448e+06	1,62	3.168e+07	1,58	6.336e+07
jeu100_29	1,36	3.932e+16	1,21	1.179e+17	1,17	5.662e+18	1,04	6.470e+20
jeu100_30	1,78	2.297e+12	1,60	4.595e+12	1,56	6.893e+13	1,37	2.685e+21
jeu100_31	1,43	1.766e+13	1,25	6.477e+13	1,12	2.293e+16	0,99	5.012e+19
jeu100_32	2,15	7.109e+15	1,88	5.971e+17	1,80	2.752e+20	1,59	1.157e+25

- ε : pourcentage de déviation par rapport à l'optimalité.
- Fichier : fichier de données.
- temps : temps (en sec.) pour trouver la première structure d'intervalles optimale.
- nb de sols : nombre de solutions optimales contenues dans la première structure optimale.

TAB. 6.4: ε-optimalité pour les problèmes de 100 travaux

	$\varepsilon = 0\%$		$\varepsilon = 10\%$		$\varepsilon = 30\%$		$\varepsilon = 50\%$	
Fichier	temps	nb de sols	temps	nb de sols	temps	nb de sols	temps	nb de sols
jeu500_1	316,98	6.323e+70	312,10	8.726e+72	307,01	1.661e+90	300,04	4.731e+105
jeu500_2	296,93	1.434e+32	296,82	1.434e+32	292,73	6.887e+33	293,11	3.702e+36
jeu500_3	284,2	7.261e+102	278,54	8.294e+108	252,91	5.373e+137	237,87	1.090e+160
jeu500_4	341,62	3.121e+62	339,00	4.994e+63	334,18	4.880e+67	328,73	3.301e+72
jeu500_5	178,96	7.731e+178	164,66	1.661e+195	140,73	1.013e+202	121,37	3.645e+258
jeu500_6	306,69	2.172e+91	300,98	1.254e+98	289,60	7.865e+105	284,19	3.994e+114
jeu500_7	173,58	2.746e+178	172,738	9.844e+180	166,76	4.587e+188	144,02	1.094e+216
jeu500_8	220,30	9.089e+113	214,19	1.284e+119	202,36	1.082e+129	190,85	1.533e+142
jeu500_9	314,33	1.002e+271	314,33	1.002e+271	311,91	1.177e+271	312,13	3.554e+273
jeu500_10	300,20	3.005e+270	291,07	6.038e+278	279,18	8.034e+296	270,20	1.043e+300
jeu500_11	402,88	7.864e+174	402,88	7.864e+174	395,95	1.289e+177	400,84	3.822e+179
jeu500_12	323,84	4.521e+92	321,11	5.150e+101	312,51	1.001e+121	305,95	1.088e+135
jeu500_13	395,48	3.769e+35	395,12	2.035e+39	392,98	6.871e+41	392,44	6.184e+43
jeu500_14	340,99	1.972e+21	340,72	1.165e+23	338,10	8.454e+24	336,32	5.991e+27
jeu500_15	346,02	7.285e+71	345,41	4.460e+80	364,42	7.079e+86	360,45	7.778e+97
jeu500_16	330,46	5.826e+52	328,89	2.099e+54	380,18	4.644e+60	379,63	3.328e+63
jeu500_17	324,02	9.605e+85	322,61	9.605e+85	321,85	9.605e+85	322,58	9.605e+85
jeu500_18	268,85	2.419e+43	277.98	2.477e+45	295.18	9.699e+48	293.65	1.445e+53
jeu500_19	369,01	1.321e+28	364,17	3.169e+29	359,37	1.562e+36	356,43	4.367e+39
jeu500_20	357,64	5.151e+158	356,52	6.014e+161	369,92	4.619e+164	367,39	4.041e+165
jeu500_21	310,11	2.474e+64	306,47	4.165e+65	300,76	7.188e+74	296,46	1.676e+80
jeu500_22	278,53	3.905e+88	277,42	2.997e+92	275,97	8.258e+99	273,51	7.437e+104
jeu500_23	247,71	4.059e+133	292,68	5.489e+143	280,41	3.406e+167	268,98	2.748e+186
jeu500_24	258,90	6.341e+112	259,77	1.779e+116	257,74	4.589e+122	254,66	4.829e+128
jeu500_25	318,38	1.248e+72	317,89	2.503e+78	292,60	3.531e+109	289,53	2.316e+132
jeu500_26	382,60	1.639e+41	377,88	1.639e+41	375,56	3.935e+42	374,22	4.048e+43
jeu500_27	223,94	1.581e+83	219,99	5.495e+88	216,02	8.788e+93	210,74	1.127e+100
jeu500_28	302,79	5.081e+75	293,09	8.966e+81	285,70	1.198e+93	277,37	4.045e+104
jeu500_29	281,60	5.781e+70	269,71	2.164e+77	261,14	1.849e+86	251,74	2.653e+92
jeu500_30	324,17	8.964e+73	320,62	3.480e+76	311,28	2.964e+83	298,72	4.886e+90
jeu500_31	219,92	1.342e+120	213,66	8.160e+123	211,05	2.843e+129	205,66	1.092e+136
jeu500_32	277,10	6.840e+88	265,52	5.678e+95	253,92	7.417e+109	247,15	1.223e+118

- ε : pourcentage de déviation par rapport à l'optimalité.
- Fichier : fichier de données.
- temps : temps (en sec.) pour trouver la première structure d'intervalles optimale.
- nb de sols : nombre de solutions optimales contenues dans la première structure optimale.

TAB. 6.5: ε-optimalité pour les problèmes de 500 travaux

6.2 Une aide à la décision basée sur une interaction via le diagramme de retards

6.2.1 Principes

L'aide à la décision a ici aussi pour but de guider le décideur dans son exploration de l'ensemble de séquences dominantes initial, de sorte qu'il trouve un compromis flexibilité / performance acceptable. À la différence de l'approche précédente, le diagramme de retards (cf. partie 4.3) est choisi comme objet d'interaction. En effet, celui-ci permet de juger l'acceptabilité d'un compromis flexibilité / performance puisqu'il représente, pour chaque travail, les indicateurs de performance, $L_j^{\min}$ et $L_j^{\max}$, et qu'un indicateur de flexibilité, précisant le nombre de séquences contenues dans l'ensemble dominant, peut être ajouté.

L'aide à la décision consiste, dans le cas où le compromis ne satisfait pas le décideur, à lui donner la possibilité de sélectionner une borne, $L_j^{\min}$ ou $L_j^{\max}$ d'un travail, pour indiquer qu'il désire la réduire ou l'augmenter, en précisant la nouvelle valeur souhaitée (appelée *consigne* dans la suite du texte). Le principe de l'aide à la décision consiste à coupler une telle action à une modification de la structure d'intervalles, comme indiqué dans les paragraphes suivants [La *et al.* 05].

6.2.2 Un méthode de réduction / d'augmentation du retard d'un travail

La réduction ou l'augmentation du retard $L_j^{\max}$ peut être respectivement réalisée en décrémentant ou en incrémentant progressivement la valeur de $v(j)$, en vérifiant toujours l'inégalité $v_0(j) \geq v(j) \geq u(j)$. Notons que $v_0(j)$ correspond à l'indice de la dernière pyramide à laquelle le travail j appartient initialement. Garantir que $v_0(j) \geq v(j) \geq u(j)$ permet donc d'assurer que l'ensemble dominant caractérisé est toujours inclus dans l'ensemble dominant initial. Toute modification de la valeur $v(j)$ peut être traduite en termes d'actualisation de la date de fin au plus tard d_j du travail j. Ainsi, la décrémentation de $v(j) \leftarrow v(j) - 1$ conduit à la mise à jour de la valeur de d_j de sorte que $d_j \leftarrow \max(d_{s_{v(j)-1}}, d_{s_{u(j)}})$ (la contrainte $d_j \geq d_{s_{u(j)}}$ permet d'assurer que la travail j ne peut pas être affecté à une pyramide d'indice inférieur au $u(j)$ courant). De façon semblable, l'incrémentation de $v(j)$ est réalisée en mettant à jour la valeur de d_j de sorte que $d_j \leftarrow \min(d_{s_{v(j)+1}}, d_j^0)$, où d_j^0 est la valeur initiale de d_j.

Similairement, la réduction ou l'augmentation du retard $L_j^{\min}$ est respectivement réalisée en décrémentant ou en incrémentant progressivement la valeur de $u(j)$, en vérifiant l'inégalité $u_0(j) \leq u(j) \leq v(j)$. Notons que $u_0(j)$ correspond à l'indice de la première pyramide à laquelle le travail j appartient initialement. L'incrémentation de $u(j)$ conduit à la mise à jour de la valeur de r_j de sorte que $r_j \leftarrow \min(r_{s_{u(j)+1}}, r_{s_{v(j)}})$. La décrémentation de $u(j)$

est elle aussi réalisée en mettant à jour la valeur de r_j de sorte que $r_j \leftarrow \max(r_{s_{u(j)-1}}, r_j^0)$, où r_j^0 est la valeur initiale de r_j.

L'incrémentation ou la décrémentation des valeurs de $u(j)$ ou $v(j)$ est automatiquement réitérée jusqu'à ce que la valeur de $L_j^{\min}$ ou $L_j^{\max}$, pour la nouvelle structure d'intervalles obtenue après actualisation de r_j ou d_j, soit la plus proche possible de la consigne donnée par le décideur. Remarquons que, quelle que soit l'action envisagée, elle se traduit toujours par une augmentation ou une diminution des bornes de r_j ou d_j associées au travail j considéré. Une nouvelle structure d'intervalles est donc obtenue telle que l'ensemble de séquences dominantes correspondant est inclus dans l'ensemble de séquences dominantes initial (du fait de le façon d'actualiser les valeurs de r_j et d_j). Notons également que l'augmentation ou la réduction d'un retard au pire ou au mieux d'un travail j peut avoir des conséquences sur les retards des autres travaux du fait de la méthode de calcul des $L_j^{\min}$ et $L_j^{\max}$ (voir chapitre 4). De façon générale, le resserrement de l'intervalle de retard d'un travail j réduit la flexibilité séquentielle et peut entraîner le resserrement d'intervalles de retard d'autres travaux. Inversement, l'élargissement d'un intervalle de retard augmente la flexibilité séquentielle et peut entraîner l'élargissement d'intervalles de retard d'autres travaux.

Chaque décision conduit donc à une nouvelle structure d'intervalles, un nouveau diagramme de retards et à une nouvelle valeur de l'indicateur de flexibilité. Les décisions peuvent être prises en séquence ou en concurrence, de sorte que le décideur puisse élaborer plusieurs stratégies. Comme décrit dans la partie suivante, un *arbre de décision* peut permettre de représenter l'ensemble des stratégies explorées par le décideur.

6.2.3 Algorithmes

Notons i le travail dont le retard doit être modifié. $L_i^{\min}$, $L_i^{\max}$ correspondent respectivement aux retards au mieux et au pire, et L_{consigne}, à la nouvelle valeur souhaitée du retard. L'algorithme de modification des $L_i^{\min}$ et $L_i^{\max}$ est décrit dans la figure 6.1.

Afin de valider les principes de l'aide à la décision, une interface logicielle a été élaborée, baptisée ADOR (Aide à la Décision pour l'Ordonnancement Robuste). Le langage de programmation utilisé pour cette application est Ada, interfacé avec les fonctions de calcul en C++ (dédiées à la manipulation des structures d'intervalles). Pour la partie graphique, la bibliothèque gratuite GtkAda a été adoptée.

6.2.4 Description du logiciel ADOR

Au lancement, la fenêtre d'accueil, représentée sur la figure 6.2, apparaît. Elle est munie de deux régions intitulées *Lateness Diagram* et *Decision Tree* dont les rôles sont explicités dans la suite du texte. Pour commencer, il est nécessaire soit de saisir un nouveau problème, ce qui crée un projet, soit d'ouvrir un projet existant (utilisation du menu *project*).

```
Proc ModifierRetard(i, L_consigne, type) (*)
    calculer L^min, L^max de noeud_courant ;
    Si type = 1 alors        /* modifier L^min */
        Si L_consigne > L^min alors
            TantQue L_consigne > L^min et u(i) ≤ v(i) faire
                r_i ← r_{u(i)} ;          /* u(i) ← u(i) + 1 */
                mettre à jour L^min et u(i) de noeud_courant ;
            FinTantQue
            Si L^min ≥ L_consigne alors  afficher "Modification réussite !" ;
            SiNon  afficher "La nouvelle valeur ne peut pas être atteinte !" ;  FinSi
        SiNon
            TantQue L_consigne < L^min et u(i) > u(i)_0 faire
                r_i ← r_{u(i)−1} ;          /* u(i) ← u(i) − 1 */
                mettre à jour L^min et u(i) de noeud_courant ;
            FinTantQue
            Si L^min ≤ L_consigne alors  afficher "Modification réussite !" ;
            SiNon  afficher "La nouvelle valeur ne peut pas être atteinte !" ;  FinSi
        FinSi

    SiNon        /* modifier L^max */
        Si L_consigne < L^max alors
            TantQue L_consigne < L^max et v(i) ≥ u(i) faire
                d_i ← d_{v(i)} ;          /* v(i) ← v(i) − 1 */
                mettre à jour L^max et v(i) de noeud_courant ;
            FinTantQue
            Si L^max ≤ L_consigne alors  afficher "Modification réussite !" ;
            SiNon  afficher "La nouvelle valeur ne peut pas être atteinte !" ;  FinSi
        SiNon
            TantQue L_consigne > L^max et v(i) < v(i)_0 faire
                d_i ← d_{v(i)+1} ;          /* v(i) ← v(i) + 1 */
                mettre à jour L^max et v(i) de noeud_courant ;
            FinTantQue
            Si L^max ≥ L_consigne alors  afficher "Modification réussite !" ;
            SiNon  afficher "La nouvelle valeur ne peut pas être atteinte !" ;  FinSi
        FinSi
    FinSi
FinProc
```

(∗) Dans cet algorithme, *type* indique le type de retard à modifier, $type = 1 \rightarrow$ modifier $L^{\min}$, $type = 0 \rightarrow$ modifier $L^{\max}$. $u(i)_0$ et $v(i)_0$ sont respectivement les valeurs initiales de $u(i)$ et $v(i)$.

FIG. 6.1: Algorithme de modification de $L_i^{\min}$ et $L_i^{\max}$

La figure 6.3 représente l'interface graphique associée à la saisie d'un problème, ici celui correspondant au tableau 6.6 traité également dans le chapitre 4.

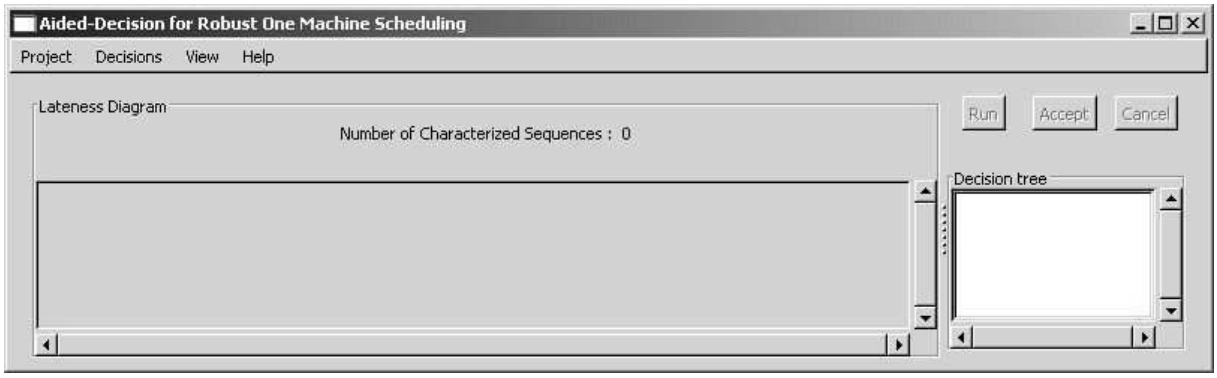


FIG. 6.2: Fenêtre principale d'accueil

Travaux	1	2	3	4	5
r_j	6	1	21	24	4
d_j	13	37	33	31	17
p_j	4	5	8	6	7

TAB. 6.6: Une instance de problème à une machine

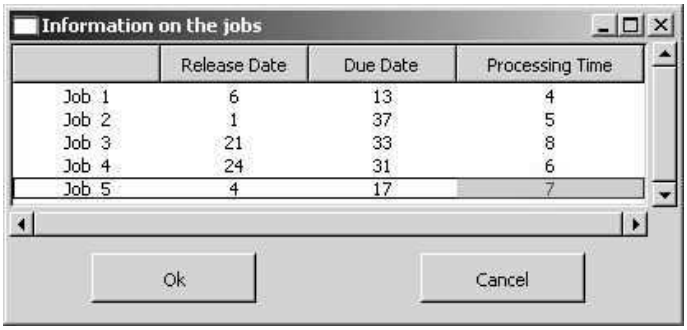


FIG. 6.3: Saisie des données du projet

Après validation de la saisie, le diagramme de retards initial est automatiquement calculé et affiché ainsi qu'illustré sur la figure 6.4 . Le nombre de séquences contenues dans l'ensemble dominant initial est également affiché au dessus du diagramme de retards. Pour représenter les retards, une règle graduée, munie de deux curseurs positionnés à ces extrémités, est associée à chaque travail. Les curseurs gauche et droit d'un travail j matérialisent respectivement les valeurs de $L_j^{\min}$ et $L_j^{\max}$. Une région ombrée, commune à l'ensemble des règles, indique l'intervalle dans lequel est situé le retard optimal

($L^* \in [\max(L_j^{\min}), \max(L_j^{\max})]$), ici l'intervalle $[-1, 6]$.

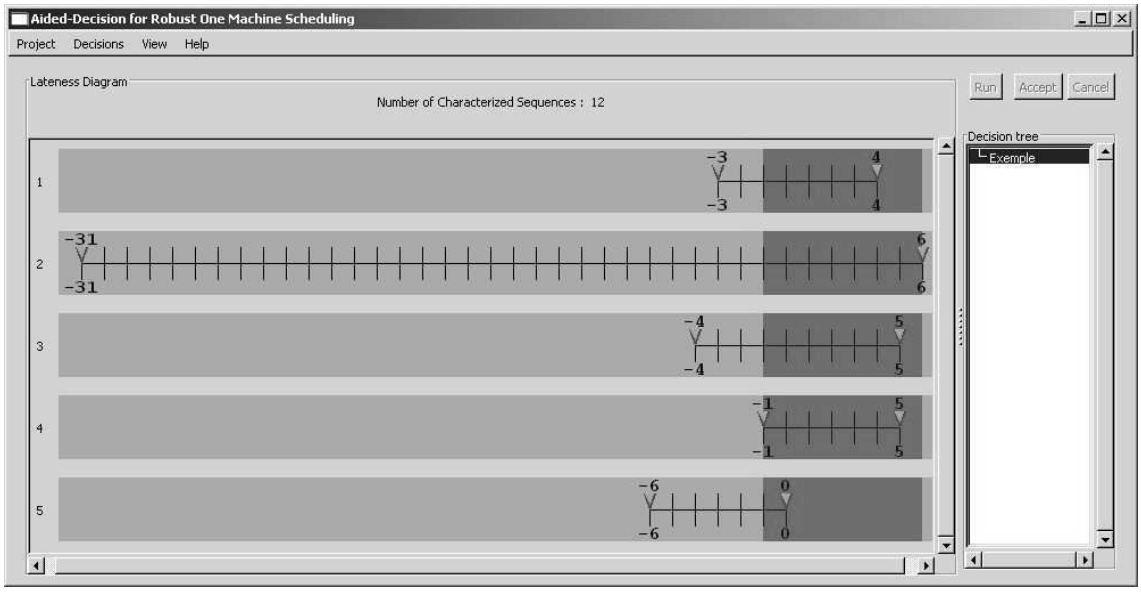


FIG. 6.4: Affichage du diagramme de retards initial

Le sous-menu *search optimal L^** du menu *Decisions* permet, si l'utilisateur le souhaite, de déterminer la valeur exacte du retard optimal (utilisation de la procédure de Carlier [Carlier 82]). Dans ce cas, la région est réduite à une seule valeur, 4 dans notre cas, comme l'indique la figure 6.5.

Afin de permettre l'interaction, les curseurs de chaque règle peuvent être déplacés à l'aide de la souris. Cette interaction permet au décideur d'indiquer qu'il souhaite modifier un retard au mieux ou au pire associé à un travail. Supposons par exemple que le décideur désire modifier le retard au mieux du travail 2 de sorte qu'il soit le plus proche possible de la valeur 12. Dans ce cas, il doit sélectionner le curseur gauche de la règle associée à ce travail, puis le faire glisser à la valeur 12, puis enfin cliquer sur le bouton *run*. La fenêtre est alors actualisée (voir figure 6.6) de sorte à afficher pour chaque travail les anciennes et nouvelles valeurs des retards (un curseur grisé indique la valeur initiale d'un retard lorsqu'elle a été modifiée). On constate que, conformément à la procédure de mise à jour de $u(2)$ (voir section précédente), la valeur la plus proche de la consigne 12 (représentée par un marqueur à cette abscisse sur la figure) est 3 (nouvelle position du curseur) et il ne reste plus que 4 séquences dans l'ensemble dominant sur les 12 initiales. De plus, les retards au pire des travaux 1 et 4 ont été réduits.

L'utilisateur doit alors indiquer s'il souhaite ou non mémoriser la structure d'intervalles correspondant à ce nouveau diagramme de retards, en cliquant sur les boutons *Cancel* ou *Accept*. Ainsi qu'illustré sur la figure 6.7, l'appui sur le bouton *Accept* provoque d'une part

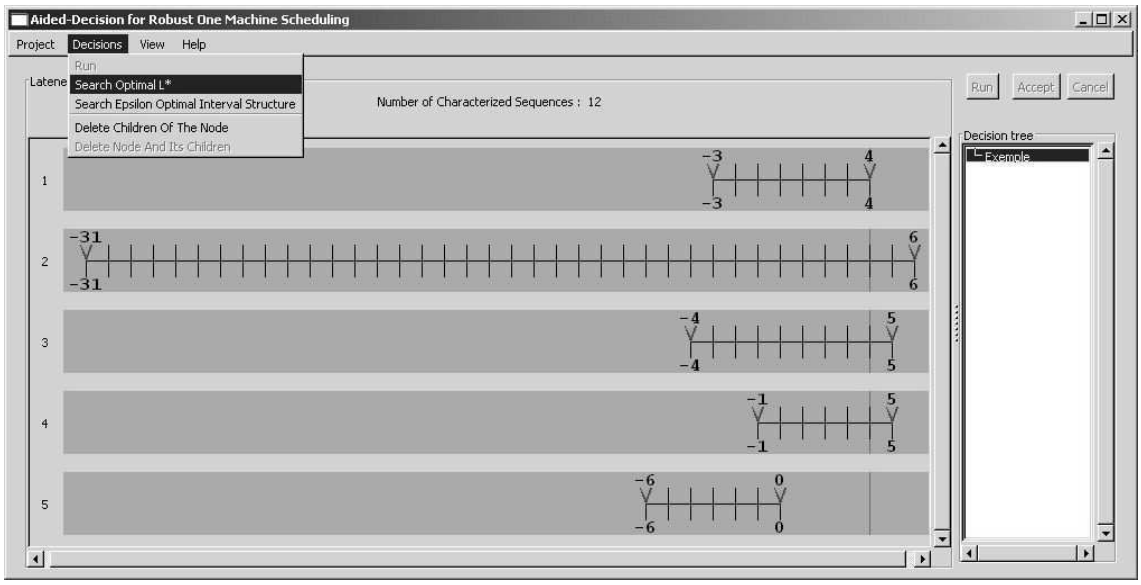


FIG. 6.5: Détermination du retard optimal

la création d'un nouveau nœud dans l'arbre de décision, et d'autre part la validation des nouvelles positions des curseurs.

En se plaçant sur un nœud quelconque de l'arbre et en modifiant un retard, l'utilisateur peut répéter l'opération, et affiner ainsi sa stratégie de compromis flexibilité / performance, en développant soit en séquence soit en concurrence un ensemble de décisions (cf. figure 6.8). Le diagramme de retards affiché correspond toujours à celui associé au nœud sélectionné dans l'arbre de décision. L'arborescence de l'arbre de décision peut être gérée (suppression de nœuds ou de branches) en utilisant les sous-menus du menu *Decisions*.

6.3 Discussion

Dans cette partie, deux types d'aide à la décision pour un compromis flexibilité / performance ont été proposés dans le cadre du problème à une machine. Cette aide à la décision est fondée sur l'utilisation de l'ordre partiel dominant, défini par le théorème des pyramides, permettant de caractériser une famille de solutions. L'intérêt de cet ordre partiel réside dans le fait qu'il permet de déterminer, sans énumération des solutions, à la fois un indicateur de flexibilité (nombre de solutions) et un indicateur de performance (retard algébrique au mieux et au pire). Les vœux du décideur relatifs à la performance pour chaque travail sont pris en compte en agissant sur la structure d'intervalles associée au problème de façon à explorer l'ensemble dominant initial.

Au delà de cette contribution, ce type d'approche montre l'intérêt d'ordres partiels

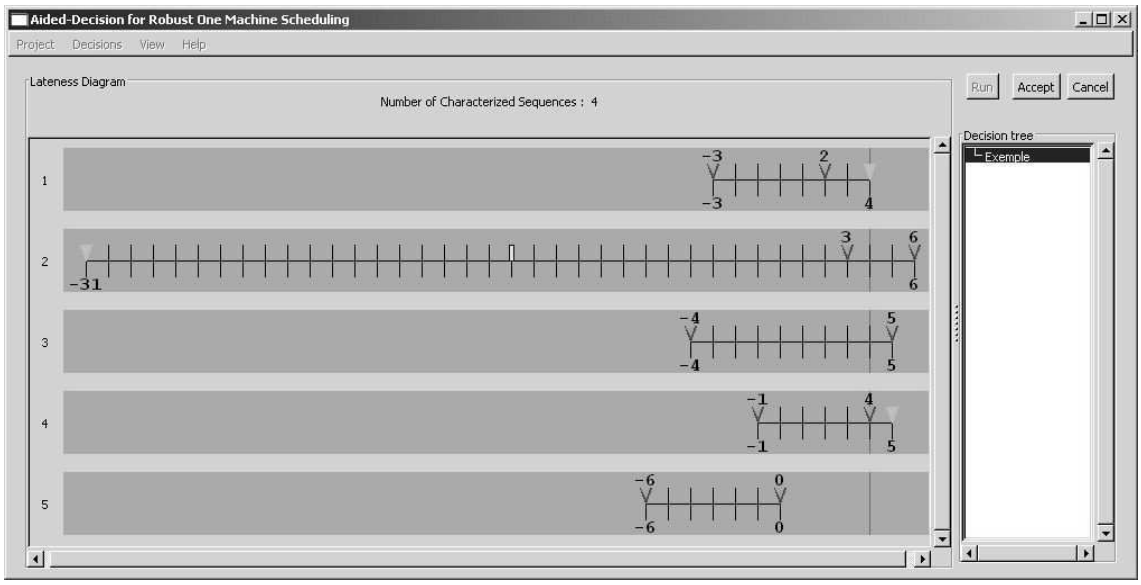


FIG. 6.6: Exemple d'interaction

dominants en ordonnancement robuste, en particulier dans le cadre d'une aide à la décision. Dans notre cas, cet ordre est construit à partir d'une information relativement pauvre : l'ordre relatif des dates de disponibilité et des dates d'échéance des travaux. Il est donc relativement insensible aux variations des données du problème, dans la mesure où l'ordre relatif reste inchangé. Néanmoins, si cette dernière propriété est intéressante du point de vue de la robustesse, elle constitue aussi un obstacle pour pouvoir produire une aide à la décision plus riche. Dans la pratique, il serait en effet plus approprié de permettre à un décideur de dimensionner ses incertitudes, en associant par exemple des fenêtres de valeurs aux données r_j, d_j et p_j, ainsi que discuté au chapitre 4, en lui montrant l'effet qu'aurait une augmentation ou une diminution de ces fenêtres sur le compromis flexibilité / performance, et vice-versa. Une adaptation de l'outil ADOR est actuellement à l'étude pour permettre ce type d'aide à la décision.

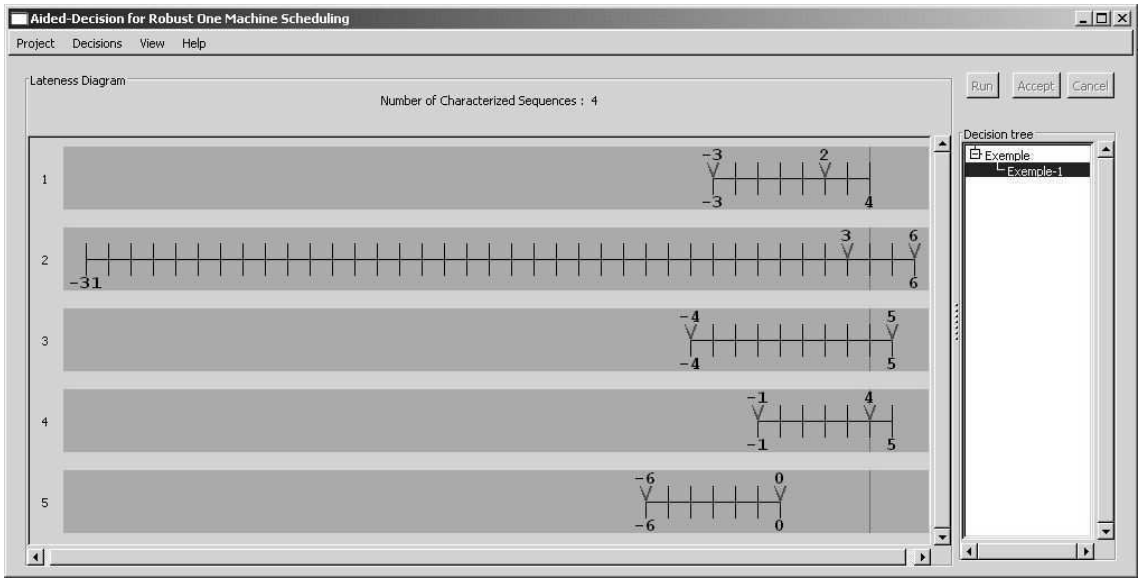


FIG. 6.7: Validation d'une décision

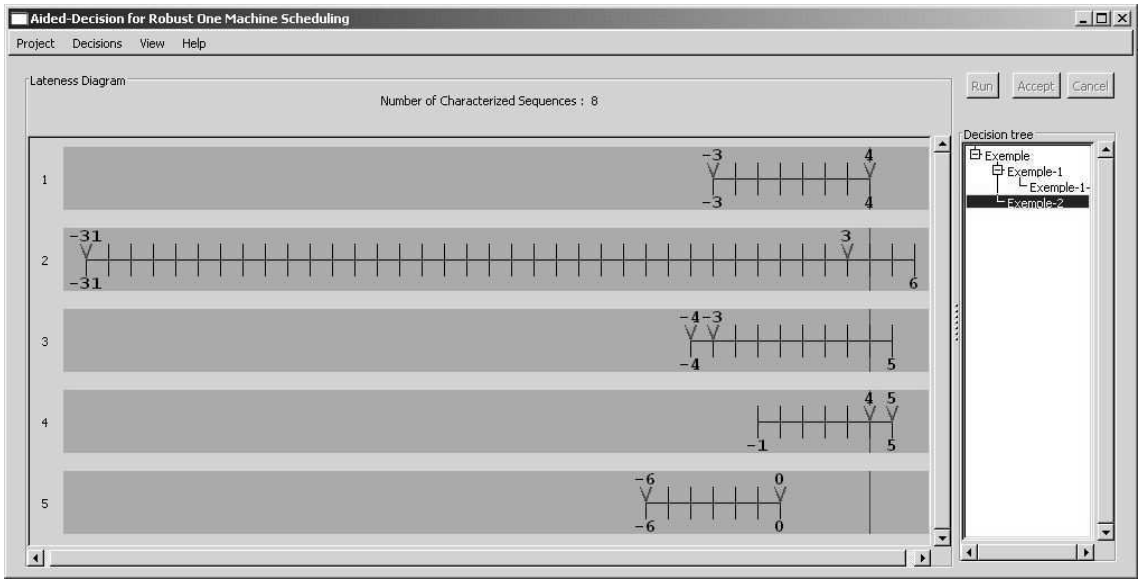


FIG. 6.8: Exemple de décisions concurrentes

Troisième partie

PROBLÈMES À PLUSIEURS MACHINES

Chapitre 7

Problèmes de flow shop à deux machines

Dans la même optique que dans la partie précédente, ce chapitre s'intéresse à l'exploitation d'ordres partiels pour le problème de flow shop à deux machines. Le critère considéré est la minimisation du makespan. Après une définition du problème, deux structures d'intervalles particulières sont définies pour le problème $F2|prmu|C_{\max}$. Sur la base d'une analyse de ces structures d'intervalles, un ordre partiel suffisant vis-à-vis de l'optimalité (i.e. caractérisant exclusivement des séquences optimales) est progressivement présenté. Un exemple illustratif ainsi que quelques résultats expérimentaux sont présentés à la fin du chapitre.

7.1 Définition du problème

Dans ce chapitre, nous abordons le problème de flow shop à deux machines avec minimisation du makespan $C_{\max}$. Comme dans le cas du problème à une machine, notre but est de caractériser un ensemble de solutions optimales le plus grand possible. Classiquement, nous notons J l'ensemble des n travaux à ordonnancer sur les deux machines M_1 et M_2, p_{j1} et p_{j2} correspondant respectivement aux durées opératoires de chaque travail j sur M_1 et M_2. Seules les séquences de permutation sont considérées (*ie.* $F2|prmu|C_{\max}$), car elles sont dominantes pour $Fm||C_{\max}, \forall m \leq 3$ [Conway *et al.* 67]. Nous abordons quelques résultats bien connus pour ce problème dans les paragraphes suivants.

Le résultat le plus connu pour le problème de flow shop de permutation à deux machines a été énoncé par Johnson [Johnson 54]. Celui-ci a montré que toute séquence respectant la règle $\min(p_{i1}, p_{j2}) \leq \min(p_{j1}, p_{i2}) \Leftrightarrow i \prec j$ est optimale, une séquence optimale unique de ce type pouvant être calculée en $O(n \log n)$.

Sur la base de ce résultat, plusieurs auteurs se sont intéressés à déterminer un ensemble de solutions optimales pour $F2|prmu|C_{\max}$. Dans [Belman *et al.* 82] est par exemple proposé un algorithme permettant d'énumérer toutes les séquences satisfaisant la règle de Johnson. Billaut et Lopez, dans [Billaut & Lopez 98], proposent aussi un algorithme qui

énumère, en permutant des travaux des séquences optimales de Johnson, l'ensemble complet des séquences optimales. Une approche connexe basée sur la notion de *séquence maximale* est aussi proposée dans [Benoît & Billaut 00]. Pour les deux derniers cas, notons que seuls des problèmes de faible taille, d'une dizaine de travaux au plus, peuvent être résolus du fait de la complexité sous-jacente à l'énumération de toutes les solutions optimales.

Un problème $F2|prmu|C_{\max}$ ayant généralement un ensemble très grand de solutions optimales, l'énumération complète de cet ensemble doit être évitée. Par conséquent, les approches privilégiant la caractérisation d'un sous-ensemble de solutions optimales, et non leur énumération, semblent plus appropriées. De telles aproches manipulent classiquement un ordre partiel. Dans [Baptiste 96] et [Esswein & Billaut 02], les auteurs proposent par exemple de réutiliser la notion de séquence de groupes (cf. chapitre 3) pour le problème $F2|prmu|C_{\max}$. L'algorithme de Baptiste donne une séquence de groupes caractérisant un ensemble de séquences sous-optimales dont la performance au pire est bornée. À partir d'une séquence optimale initiale, Esswein et al. proposent l'algorithme *Greedy Forward Grouping*, de complexité temporelle $O(n \log n)$, déterminant une séquence de groupes optimale qui maximise la flexibilité (*i.e.* le nombre de groupes est minimisé). Dans ces deux travaux, les auteurs ont mis en évidence que le nombre de solutions optimales caractérisées est souvent colossal.

L'approche présentée dans cette partie a aussi pour but la caractérisation d'un grand ensemble de séquences optimales pour le problème $F2|prmu|C_{\max}$. Néanmoins, elle se distingue des approches précédentes par les caractéristiques suivantes. Tout d'abord, un ordre partiel nouveau est défini qui n'utilise pas la notion de séquence de groupes. De plus, notre approche n'utilise pas de séquence optimale initiale, toute séquence caractérisée par notre ordre partiel étant optimale par nature (l'ordre partiel est suffisant vis-à-vis du critère $C_{\max}$). Enfin, notre ordre partiel étant défini sur la base de l'ordre relatif des durées opératoires p_{i1} et p_{i2}, il reste suffisant même si les durées opératoires changent, tant que l'ordre relatif des durées reste inchangé, une telle caractéristique étant évidemment très intéressante dans le contexte de l'ordonnancement robuste. Une autre caractéristique intéressante réside dans le fait que toute séquence de Johnson est nécessairement incluse dans l'ensemble de séquences caractérisées.

Dans les parties suivantes, nous décrivons deux types de structures d'intervalles particulières et nous montrons comment ces structures peuvent être exploitées pour définir un ordre partiel suffisant pour $F2|prmu|C_{\max}$.

7.2 Structures d'intervalles considérées

Rappelons qu'une structure d'intervalles est définie par un couple $< I, C >$, avec $I = \{i_1, ..., i_n\}$ un ensemble d'intervalles et C un ensemble de contraintes sur $I \times I$. Chaque intervalle i_j est défini par ses bornes inférieure x_j et supérieure y_j. Pour défi-

nir les contraintes entre deux intervalles i_j et i_k, nous utilisons l'algèbre de Allen (cf. 3.3, chapitre 3).

Pour le problème $F2|prmu|C_{\max}$, deux structures d'intervalles particulières, I_1 et I_2, sont définies :

- I_1 est la structure d'intervalles associée aux travaux $j \in J$ tels que $p_{j1} \leq p_{j2}$;
- I_2 est la structure d'intervalles associée aux travaux $j \in J$ tels que $p_{j2} \leq p_{j1}$.

Un intervalle $i_j = [p_{j1}, p_{j2}]$ est associé à chaque travail $j \in I_1$ et similairement, un intervalle $i_j = [p_{j2}, p_{j1}]$ est associé à chaque travail $j \in I_2$. Un travail j tel que $p_{j1} = p_{j2}$ appartient donc aux deux structures d'intervalles I_1 et I_2 et son intervalle est un point.

Dans la suite du texte, nous nous concentrons sur les bases de I_1 et I_2 et sur les b-pyramides qu'elles caractérisent. On suppose que les n_1 bases de I_1 sont indexées selon l'ordre croissant de leurs durées opératoires sur la première machine (dans un ordre arbitraire en cas d'égalité). Similairement, les n_2 bases de I_2 sont indexées, à partir de l'index $n_1 + 1$, selon l'ordre décroissant de leurs durées opératoires sur la deuxième machine (dans un ordre arbitraire en cas d'égalité).

Pour l'illustration, considérons l'exemple à 9 travaux, défini dans le tableau 7.1.

Travail j	1	2	3	4	5	6	7	8	9
p_{j1}	1	3	8	8	2	4	7	9	5
p_{j2}	6	8	12	2	4	5	7	11	3

TAB. 7.1: Une instance de problème $F2|prmu|C_{\max}$

Pour cet exemple, les structures d'intervalles I_1 et I_2 sont représentées sur la figure 7.1. La structure I_1 contient trois bases, $b_1 = 1$, $b_2 = 2$ et $b_3 = 3$, définissant les trois b-pyramides : $P_1 = \{5, 6\}$, $P_2 = \{6, 7\}$ et $P_3 = \{8\}$. La structure d'intervalles I_2 contient seulement une base, $b_4 = 4$, qui caractérise la b-pyramide $P_4 = \{7, 9\}$.

Avant de présenter le résultat général, nous considérons deux cas particuliers : le cas où le problème $F2|prmu|C_{\max}$ considéré ne possède qu'une seule b-pyramide, et celui où il en possède deux.

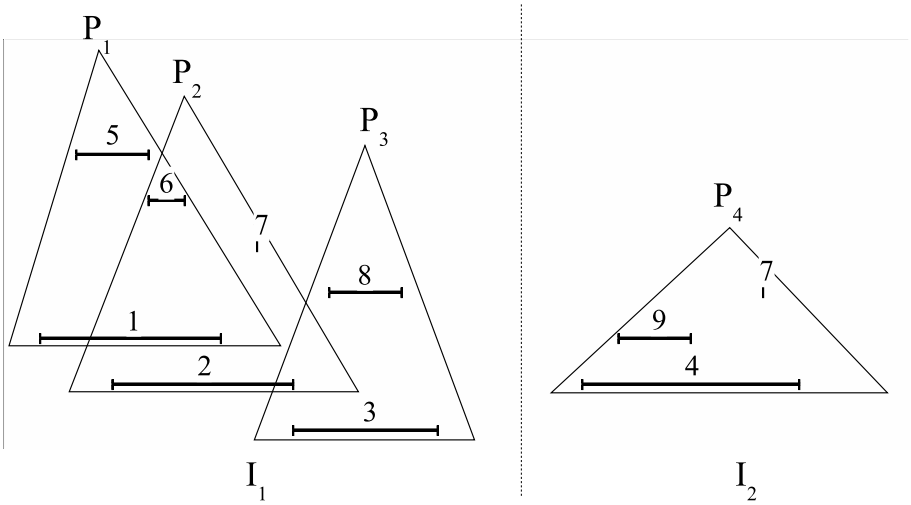


FIG. 7.1: Structures d'intervalles I_1 et I_2 pour l'exemple

7.3 Analyse de structures d'intervalles mono-pyramidales et bi-pyramidales

7.3.1 Le cas mono-pyramidal

Soit un problème $F2|prmu|C_{\max}$ caractérisé par une seule structure d'intervalles, celle-ci ne possédant qu'une seule base b et une seule b-pyramide P_b. Deux cas sont à distinguer : soit P_b est une b-pyramide de I_1, soit P_b est une b-pyramide de I_2.

7.3.1.1 P_b est une b-pyramide de I_1

Dans ce cas, le théorème suivant peut être énoncé.

Théorème 1. *Toute séquence plaçant la base b en première position puis les autres travaux dans n'importe quel ordre est optimale avec :*

$$C_{\max} = p_{b1} + \sum_{j \in J} p_{j2}$$

Démonstration. Nous prouvons qu'il n'y a nécessairement pas de temps mort sur M_2 quand la base b est séquencée en première position. D'abord, comme b est la seule base, tout travail $j \in J - \{b\}$ de I_1 respecte clairement l'inégalité suivante : $p_{b1} < p_{j1} \leq p_{j2} < p_{b2}$. Comme illustré sur la figure 7.2, notons σ la sous-séquence de n travaux déjà séquencés après la base b, et Δt_n la distance entre la fin des travaux de σ sur M_1 et la fin des travaux de σ sur M_2 (*i.e.* $\Delta t_n = C_{\sigma+\{b\}} - \sum_{j \in \sigma+\{b\}} p_{j1}$). Démontrer qu'il n'y a pas de temps mort sur M_2 est équivalent à démontrer que $\Delta t_n \geq p_{j1}\ \forall n$ et $\forall j \in J - \sigma - \{b\}$. Le cas initial où seulement la

base b est séquencée (*i.e.* $\sigma = \emptyset$) est évident car, $\Delta t_1 = p_{b2}$ et donc, $\Delta t_1 > p_{j1}\ \forall j \in J - \{b\}$. Si nous considérons maintenant le cas général, on a : $\Delta t_n = \Delta t_{n-1} + p_{j2} - p_{j1}$. Comme $p_{j2} - p_{j1} \geq 0$ par définition, nous pouvons déduire que $\Delta t_n \geq \Delta t_{n-1} \geq \cdots \geq \Delta t_1 > p_{j1}$. Il n'y a donc pas de temps mort sur M_2 et $C_{\max} = p_{b1} + \sum_{j \in J} p_{j2}$. □

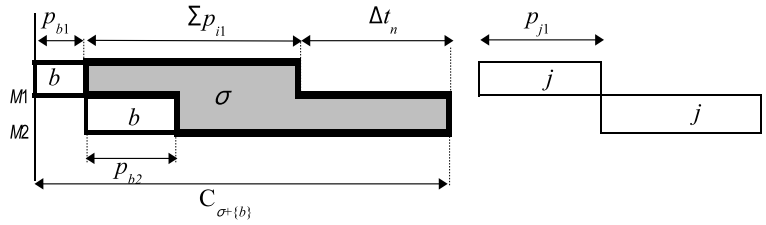


FIG. 7.2: Séquences optimales pour le cas mono-pyramidal (P_b appartient à I_1)

7.3.1.2 P_b est une b-pyramide de I_2

Dans le cas où b est l'unique base de I_2, un théorème dual peut être formulé.

Théorème 2. *Toute séquence plaçant la base b en dernière position et les autres travaux dans n'importe quel ordre est optimale avec :*

$$C_{\max} = p_{b2} + \sum_{j \in J} p_{j1}$$

Démonstration. La preuve est similaire à la précédente en permutant les durées opératoires p_{j1} et p_{j2} (formulation inverse du problème). Les séquences optimales sont donc les mêmes, mais dans l'ordre inverse. □

7.3.2 Le cas bi-pyramidal

Dans cette partie, nous supposons que le problème est caractérisé par deux bases b_1 et b_2. Nous considérons deux cas : soit b_1 et b_2 appartiennent à la même structure d'intervalles (I_1 ou I_2), soit b_1 appartient à I_1 et b_2 appartient à I_2.

7.3.2.1 Les bases b_1 et b_2 appartiennent à la même structure d'intervalles I_1

Supposons que b_1 et b_2 sont deux bases appartenant à I_1 telles que $p_{b1} \leq p_{b2}$. Comme indiqué sur la figure 7.3, nous prenons en compte les séquences de la forme $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2$. Deux cas sont à considérer selon que la relation $precedes(b_1, b_2)$ est vérifiée, ou non.

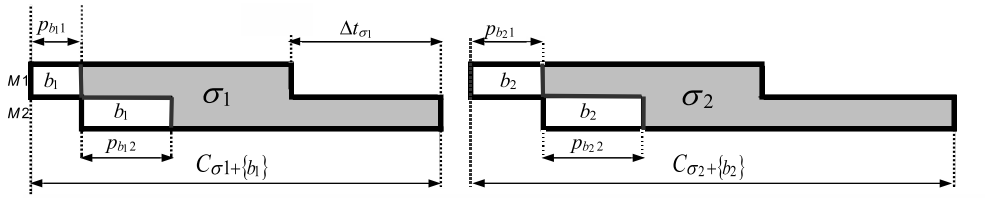


FIG. 7.3: Séquences optimales pour la cas bi-pyramidal (b_1 et b_2 appartiennent à I_1)

• *La relation precedes(b_1, b_2) est vérifiée*

Puisque *precedes*(b_1, b_2) est vrai, alors $P_{b_1} \cap P_{b_2} = \emptyset$ et tout travail $j \in J - \{b_1, b_2\}$ appartient soit à P_{b_1} soit à P_{b_2}. Nous démontrons alors le théorème suivant :

Théorème 3. *Si P_{b_1} et P_{b_2} sont deux b-pyramides de I_1 avec precedes(b_1, b_2), alors toute séquence telle que :*

- *b_1 précède b_2 ;*
- *tous les travaux appartenant à P_{b_1} sont séquencés à l'intérieur de σ_1 dans n'importe quel ordre ;*
- *tous les travaux appartenant à P_{b_2} sont séquencés à l'intérieur de σ_2 dans n'importe quel ordre ;*

est optimale avec :

$$C_{\max} = \sum_{j \in \sigma_2 + \{b_2\}} p_{j2} + \max(p_{b_1 1} + \sum_{j \in \sigma_1 + \{b_1\}} p_{j2}, \sum_{j \in \sigma_1 + \{b_1\}} p_{j1} + p_{b_2 1}).$$

Démonstration. Considérons la séquence optimale de Johnson qui range les travaux dans l'ordre croissant de leur p_{j1}. Cette séquence optimale est sous la forme $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2$ et respecte naturellement les trois conditions du théorème 3 puisque b_1 précède b_2, tout travail $j \in P_{b_1}$ est affecté à σ_1 et tout travail $j \in P_{b_2}$ est affecté à σ_2.

À partir du théorème 1, il est clair que les valeurs de $C_{\sigma_1+\{b_1\}}$ et de $C_{\sigma_2+\{b_2\}}$ (donc la valeur du makespan optimal), ne changent pas quels que soient les ordres des travaux au sein de σ_1 et de σ_2. Selon la valeur de Δt_{σ_1} par rapport à $p_{b_2 1}$, le makespan optimal $C_{\max}$ peut alors avoir les deux valeurs suivantes :

– Si $\Delta t_{\sigma_1} \geq p_{b_2 1}$ alors :

$$C_{\max} = p_{b_1 1} + \sum_{j \in \sigma_1 + \{b_1\}} p_{j2} + \sum_{j \in \sigma_2 + \{b_2\}} p_{j2}$$

– Sinon,

$$C_{\max} = \sum_{j \in \sigma_1 + \{b_1\}} p_{j1} + p_{b_2 1} + \sum_{j \in \sigma_2 + \{b_2\}} p_{j2}$$

La valeur du makespan est la plus grande de ces deux valeurs, d'où l'expression de $C_{\max}$ donnée dans le théorème 3. □

• *La relation precedes*(b_1, b_2) *n'est pas vérifiée*

Supposons que b_1 et b_2 sont deux bases de la structure d'intervalles I_1 telles que $p_{b_1 1} \leq p_{b_2 1}$. Puisque la relation $precedes(b_1, b_2)$ est fausse, six relations de Allen restent possibles : $equals(b_2, b_1)$, $starts(b_2, b_1)$, $starts(b_1, b_2)$, $ends(b_2, b_1)$, $overlaps(b_1, b_2)$ et $meets(b_1, b_2)$. Dans tous ces cas, notons que plusieurs travaux peuvent appartenir à la fois aux deux b-pyramides P_{b_1} et P_{b_2}.

Théorème 4. *Si P_{b_1} et P_{b_2} sont deux b-pyramides de I_1 telles que la relation precedes(b_1, b_2) est fausse, alors toute séquence telle que :*

- *b_1 précède b_2 ;*
- *tous les travaux appartenant seulement à P_{b_1} sont séquencés à l'intérieur de σ_1 dans n'importe quel ordre ;*
- *tous les travaux appartenant seulement à P_{b_2} sont séquencés à l'intérieur de σ_2 dans n'importe quel ordre ;*
- *tout travail appartenant aux deux b-pyramides P_{b_1} et P_{b_2} est séquencé soit dans σ_1, soit dans σ_2, dans n'importe quel ordre ;*

est optimale avec :

$$C_{\max} = p_{b_1 1} + \sum_{j \in J} p_{j2}.$$

Démonstration. La preuve est similaire à la précédente. En effet, considérons à nouveau la séquence optimale de Johnson rangeant les travaux dans l'ordre croissant de leur p_{j1}. Cette séquence optimale est de la forme $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2$ (voir figure 7.3) et respecte clairement les conditions du théorème 4 puisque, b_1 précède b_2, les travaux appartenant seulement à P_{b_1} sont affectés à σ_1 et tous les autres travaux (appartenant soit à $P_{b_1} \cap P_{b_2}$ soit à P_{b_2} seul) sont affectés à σ_2. Puisque b_1 et b_2 sont deux bases telles que $precedes(b_1, b_2)$ est faux, alors la relation $p_{b_1 1} \leq p_{b_2 1} \leq p_{b_1 2}$ est valide. Selon la preuve du théorème 1, on peut déduire que $\Delta t_{\sigma_1} \geq p_{b_2 1}$. La valeur optimale du makespan est donc :

$$C_{\max} = C_{\sigma_1 + \{b_1\}} + C_{\sigma_2 + \{b_2\}} - p_{b_2 1} = p_{b_1 1} + \sum_{j \in \sigma_1 + \{b_1\}} p_{j2} + \sum_{j \in \sigma_2 + \{b_2\}} p_{j2}$$

Selon le théorème 1, les valeurs de $C_{\sigma_1+\{b_1\}}$ et $C_{\sigma_2+\{b_2\}}$ (donc celle du makespan optimal) ne changent pas quels que soient les ordres de travaux à l'intérieur de σ_1 et σ_2. De plus, ordonnancer un travail appartenant à la fois à P_{b_1} et P_{b_2} dans σ_1 (au lieu de σ_2 comme dans la séquence de Johnson) ne diminue pas Δt_{σ_1}. Donc la relation $\Delta t_{\sigma_1} \geq p_{b_2 1}$ reste toujours valide et la valeur de $C_{\max}$ ne change pas. Enfin, dans le cas particulier où $p_{b_1 1} = p_{b_2 1}$, les index de bases peuvent être permutés dans le théorème. □

7.3.2.2 Les bases b_1 et b_2 appartiennent à la même structure d'intervalles I_2

Nous supposons ici que b_1 et b_2 sont deux bases appartenant à I_2 telles que $p_{b_1 2} \geq p_{b_2 2}$. Les théorèmes suivants sont alors directement énoncés puisque leur preuve est similaire aux deux preuves précédentes, en considérant la formulation inverse du problème. Dans ce cas, la structure de séquences optimales est $\sigma_1 \prec b_1 \prec \sigma_2 \prec b_2$.

Théorème 5. *Si P_{b_1} et P_{b_2} sont deux b-pyramides de I_2 telles que precedes(b_2, b_1), alors toute séquence telle que :*

- *b_1 précède b_2 ;*
- *tous les travaux appartenant à P_{b_1} sont séquencés à l'intérieur de σ_1 dans n'importe quel ordre ;*
- *tous les travaux appartenant à P_{b_2} sont séquencés à l'intérieur de σ_2 dans n'importe quel ordre ;*

est optimale avec :

$$C_{\max} = \sum_{j \in \sigma_1 + \{b_1\}} p_{j1} + \max(p_{b_2 2} + \sum_{j \in \sigma_2 + \{b_2\}} p_{j1}, \sum_{j \in \sigma_2 + \{b_2\}} p_{j2} + p_{b_1 2}).$$

Théorème 6. *Si P_{b_1} et P_{b_2} sont deux b-pyramides de I_2 telles que la relation precedes(b_2, b_1) n'est pas vérifiée, alors toute séquence telle que :*

- *b_1 précède b_2 ;*
- *tous les travaux appartenant seulement à P_{b_1} sont séquencés à l'intérieur de σ_1 dans n'importe quel ordre ;*
- *tous les travaux appartenant seulement à P_{b_2} sont séquencés à l'intérieur de σ_2 dans n'importe quel ordre ;*
- *tout travail appartenant à P_{b_1} et P_{b_2} est séquencé soit dans σ_1, soit dans σ_2, dans n'importe quel ordre ;*

est optimale avec :

$$C_{\max} = p_{b_2 2} + \sum_{j \in J} p_{j1}.$$

7.3.2.3 Les bases b_1 et b_2 n'appartiennent pas à la même structure d'intervalles

Dans cette partie, nous supposons que b_1 est la base de I_1 et b_2 est la base de I_2. Notons que seuls les travaux j tels que $p_{j1} = p_{j2}$ peuvent appartenir aux deux b-pyramides P_{b_1} et P_{b_2} (leur intervalle étant un point). Comme illustré sur la figure 7.4, nous nous concentrons sur les séquences de la forme $b_1 \prec \sigma_1 \prec \sigma_2 \prec b_2$. Nous énonçons alors le théorème suivant :

Théorème 7. *Si P_{b_1} est une b-pyramide de I_1 et P_{b_2} est une b-pyramide de I_2, alors toute séquence telle que :*

- *b_1 précède b_2 ;*
- *tous les travaux appartenant seulement à P_{b_1} sont séquencés à l'intérieur de σ_1 dans n'importe quel ordre ;*
- *tous les travaux appartenant seulement à P_{b_2} sont séquencés à l'intérieur de σ_2 dans n'importe quel ordre ;*
- *tout travail appartenant à P_{b_1} et P_{b_2} est séquencé soit dans σ_1, soit dans σ_2 dans n'importe quel ordre ;*

est optimale avec :

$$C_{\max} = p_{b_1 1} + p_{b_2 2} + \max\Big(\sum_{j \in \sigma_1 \cup \sigma_2} p_{j1} + p_{b_2 1}, \sum_{j \in \sigma_1 \cup \sigma_2} p_{j2} + p_{b_1 2}\Big).$$

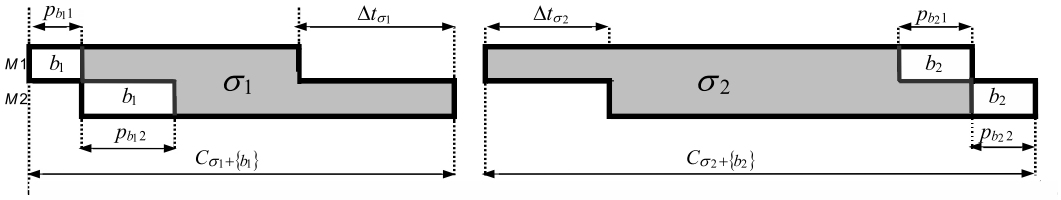


FIG. 7.4: Séquences optimales pour le cas bi-pyramidal (b_1 appartient à I_1 et b_2 appartient à I_2)

Démonstration. À nouveau, nous considérons la séquence optimale de Johnson rangeant les travaux appartenant à I_1 dans l'ordre croissant de leur p_{j1} et les travaux appartenant à I_2 dans l'ordre décroissant de leur p_{j2}. Cette séquence optimale respecte évidemment les trois conditions du théorème 7 car, b_1 précède b_2, tout travail $j \in P_{b_1}$ est affecté à σ_1 et tout travail $j \in P_{b_2}$ est affecté à σ_2. À partir du théorème 1, on constate que les valeurs de $C_{\sigma_1+\{b_1\}}$ et $C_{\sigma_2+\{b_2\}}$ (donc la valeur du makespan optimal) ne changent pas quels que soient les ordres de travaux à l'intérieur de σ_1 et σ_2. De plus, si un travail j appartient aux deux pyramides P_{b_1} et P_{b_2} ($p_{j1} = p_{j2}$), alors il peut être affecté soit à σ_1 soit à σ_2 car, quelle que soit l'affectation, cette affectation ne modifie ni les valeurs de Δt_{σ_1} et Δt_{σ_2} ni la valeur du makespan. Selon la valeur de Δt_{σ_1} par rapport à Δt_{σ_2}, deux cas sont distingués :

– Si $\Delta t_{\sigma_1} \geq \Delta t_{\sigma_2}$ alors :

$$C_{\max} = p_{b_1 1} + p_{b_1 2} + \sum_{j \in \sigma_1 \cup \sigma_2} p_{j2} + p_{b_2 2}$$

– Sinon,

$$C_{\max} = p_{b_1 1} + \sum_{j \in \sigma_1 \cup \sigma_2} p_{j1} + p_{b_2 1} + p_{b_2 2}$$

D'où l'expression générale de $C_{\max}$ du théorème 7. □

7.4 Analyse de structures pyramidales quelconques

7.4.1 Une condition suffisante d'optimalité pour $F2|prmu|C_{\max}$

Considérons à présent le cas général où les structures d'intervalles I_1 et I_2 sont respectivement caractérisée par n_1 et n_2 b-pyramides. Comme illustré sur la figure 7.5, notons σ_j, avec $j \in [1, n_1]$, la sous-séquence de travaux se situant entre les bases b_j et b_{j+1}, les bases de I_1 étant indexées selon l'ordre croissant de leur p_{j1} (dans un ordre arbitraire en cas d'égalité). Similairement, notons σ_k, avec $k \in [n_1+1, n_1+n_2]$, la sous-séquence de travaux se situant entre les bases b_{k-1} et b_k, les bases de I_2 étant indexées selon l'ordre décroissant de leur p_{j2} (dans un ordre arbitraire en cas d'égalité). Intéressons-nous aux séquences de la forme : $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2 \prec \cdots \prec b_{n_1} \prec \sigma_{n_1} \prec \sigma_{n_1+1} \prec b_{n_1+1} \prec \cdots \prec \sigma_{n_1+n_2} \prec b_{n_1+n_2}$, comme illustré sur la figure 7.5.

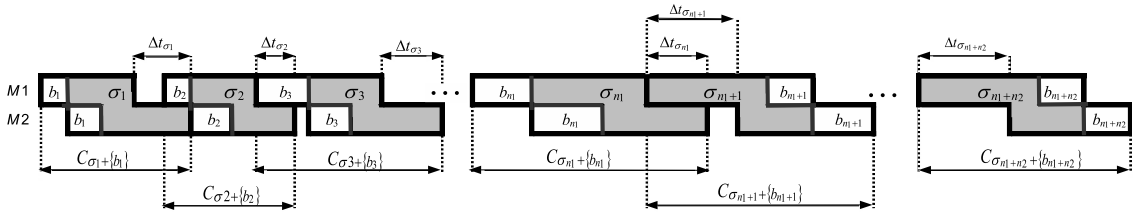


FIG. 7.5: Séquences optimales pour le cas général

Pour chaque travail j, notons $u(j)$ (resp $v(j)$) l'index de la base qui caractérise la première (resp. la dernière) b-pyramide à laquelle le travail j appartient. Pour la base b_i, nous imposons $u(b_i) = v(b_i) = i$, le théorème général suivant peut alors être énoncé :

Théorème 8. *Toute séquence telle que :*
- *les bases de I_1 et I_2 sont séquencées selon l'ordre croissant de leur index ;*
- *tout travail j est séquencé à l'intérieur de n'importe quelle sous-séquence, de $\sigma_{u(j)}$ à $\sigma_{v(j)}$, dans n'importe quel ordre ;*

est optimale avec :

$$\begin{aligned}
C_{\max} \;=\; & C_{\sigma_1+\{b_1\}} + \\
& \max(C_{\sigma_2+\{b_2\}} - p_{b_2 1}, C_{\sigma_2+\{b_2\}} - \Delta t_{\sigma_1}) + \\
& \max(C_{\sigma_3+\{b_3\}} - p_{b_3 1}, C_{\sigma_3+\{b_3\}} - \Delta t_{\sigma_2}) + \\
& \cdots + \\
& \max(C_{\sigma_{n_1}+\{b_{n_1}\}} - p_{b_{n_1} 1}, C_{\sigma_{n_1}+\{b_{n_1}\}} - \Delta t_{\sigma_{n_1-1}}) + \\
& \max(C_{\sigma_{n_1+1}+\{b_{n_1+1}\}} - \Delta t_{\sigma_{n_1+1}}, C_{\sigma_{n_1+1}+\{b_{n_1+1}\}} - \Delta t_{\sigma_{n_1}}) + \\
& \max(C_{\sigma_{n_1+2}+\{b_{n_1+2}\}} - \Delta t_{\sigma_{n_1+2}}, C_{\sigma_{n_1+2}+\{b_{n_1+2}\}} - p_{b_{n_1+1} 2}) + \\
& \cdots + \\
& \max(C_{\sigma_{n_1+n_2-1}+\{b_{n_1+n_2-1}\}} - \Delta t_{\sigma_{n_1+n_2-1}}, C_{\sigma_{n_1+n_2-1}+\{b_{n_1+n_2-1}\}} - p_{b_{n_1+n_2-2} 2}) + \\
& \max(C_{\sigma_{n_1+n_2}+\{b_{n_1+n_2}\}} - \Delta t_{\sigma_{n_1+n_2}}, C_{\sigma_{n_1+n_2}+\{b_{n_1+n_2}\}} - p_{b_{n_1+n_2-1} 2})
\end{aligned}$$

avec :

$$C_{\sigma_i+\{b_i\}} = \begin{cases} p_{b_i 1} + \sum_{j \in \sigma_i+\{b_i\}} p_{j2} & si\ i \in [1, n_1] \\ p_{b_i 2} + \sum_{j \in \sigma_i+\{b_i\}} p_{j1} & si\ i \in [n_1+1, n_1+n_2] \end{cases}$$

$$\Delta t_{\sigma_i} = \begin{cases} C_{\sigma_i+\{b_i\}} - \sum_{j \in \sigma_i+\{b_i\}} p_{j1} & si\ i \in [1, n_1] \\ C_{\sigma_i+\{b_i\}} - \sum_{j \in \sigma_i+\{b_i\}} p_{j2} & si\ i \in [n_1+1, n_1+n_2] \end{cases}$$

Démonstration. Nous nous intéressons à la séquence de Johnson qui range d'abord les travaux de I_1, selon l'ordre croissant de leur p_{j1}, puis les travaux de I_2, selon l'ordre décroissant de leur p_{j2}. Cette séquence optimale respecte le théorème 8 car l'ordre de bases suit l'ordre croissant de leur index, tout travail $j \in I_1$ est affecté à $\sigma_{v(j)}$ et tout travail $j \in I_2$ est affecté à $\sigma_{u(j)}$. Selon les théorèmes 3, 4, 5, 6 et 7, on peut déduire l'expression du makespan optimal fourni dans le théorème 8. De plus, cette expression donnant la valeur optimale du makespan pour la séquence particulière de Johnson, on peut déduire que cette valeur ne changera pas pour toute affectation d'un travail non-base j entre $\sigma_{u(j)}$ et $\sigma_{v(j)}$ (comme indiqué dans la preuve du théorème 4). □

Nous allons prouver maintenant que l'ordre partiel défini par la règle de Johnson est un extension de celui de notre théorème 8.

Corollaire 1. *Toute séquence de Johnson est nécessairement incluse dans l'ensemble de séquences caractérisé par le théorème 8.*

Démonstration. À partir du théorème 8, nous déduisons qu'un travail i ne peut pas précéder un travail j dans toute séquence optimale si et seulement si $u(i) > v(j)$. En effet, si $u(i) \leq v(j)$, alors soit $u(i) \leq v(i) < u(j) \leq v(j)$ et i précède j dans toute séquence, soit $u(i) \leq u(j) \leq v(i) \leq v(j)$ et il existe des séquences telles que i et j sont affectés à la même sous-séquence σ_k avec $u(j) \leq k \leq v(i)$ (alors i et j peuvent être séquencés dans n'importe quel ordre). La relation $u(i) > v(j)$ (*i.e.* $i \not\prec j$) est vraie seulement dans les cas suivants :

- j appartient à I_1 (mais pas à I_2) et i appartient à I_2 (mais pas à I_1), donc $p_{j2} > p_{j1}$ et $p_{i1} > p_{i2}$;

- i et j appartiennent tous à I_1 et soit $p_{j1} < p_{i1} \leq p_{j2} < p_{i2}$ (*i.e.* $overlaps(j,i)$ ou $meets(j,i)$) soit $p_{j1} < p_{j2} < p_{i1} < p_{i2}$ (*i.e.* $precedes(j,i)$), et dans ce cas il n'existe aucune base b telle que les deux relations $during(i,b)$ et $during(j,b)$ se vérifient en même temps (*i.e.* i et j n'appartiennent à aucune b-pyramide commune) ;

- i et j appartiennent tous à I_2 et soit $p_{i2} < p_{j2} \leq p_{i1} < p_{j1}$ (*i.e.* $overlaps(i,j)$ ou $meets(i,j)$) soit $p_{i2} < p_{i1} < p_{j2} < p_{j1}$ (*i.e.* $precedes(i,j)$), et dans ce cas il n'existe aucune base b telle que les deux relations $during(i,b)$ et $during(j,b)$ se vérifient en même temps (*i.e.* i et j n'appartiennent à aucune b-pyramide commune) ;

On constate donc que dans tous les cas listés ci-dessus, l'inégalité $\min(p_{i1}, p_{j2}) > \min(p_{i2}, p_{j1})$ est toujours vérifiée. Donc, le théorème 8 n'élimine pas les séquences satisfaisant la règle de Johnson. Les séquences de Johnson sont alors conservées. □

Une propriété intéressante de ce théorème réside dans sa capacité à caractériser un nombre de séquences très impressionnant. Notons S l'ensemble de toutes les affectations possibles des travaux aux sous-séquences σ. La cardinalité de S est la suivante :

$$S = \prod_{q=1}^{m} q^{n_q}$$

où m est le nombre total de b-pyramides, et n_q est le nombre de travaux appartenant exactement à q b-pyramides. Pour une affectation donnée des travaux $s \in S$ aux sous-séquences σ, nous notons $n_i(s)$ le nombre de travaux affectés à σ_i en respectant le théorème. Alors le nombre total de permutations est $\prod_{i=1}^{m}(n_i(s)!)$. On déduit donc que le nombre total de séquences caractérisées est $\sum_{s \in S}(\prod_{i=1}^{m} n_i(s)!)$.

Par exemple, pour un problème comptant 20 travaux tel que $m = 4, u(j) = v(j)\ \forall j \in J$ et $n_i = 4$, le théorème caractérise alors $(4!)^4 = 331776$ séquences optimales. Maintenant, pour le même problème, si nous supposons qu'un travail et un seul appartient à deux b-pyramides, alors le nombre de séquences optimales caractérisées devient $(4!)^4 + (3! * 5! * 4! * 4!) = 746496$.

7.4.2 Exemple illustratif

Pour illustrer le théorème 8, reprenons l'exemple de la figure 7.1. Les valeurs des fonctions $u(j)$ et $v(j)$ sont données pour chaque travail j sur la table de la figure 7.2.

Travail j	1	2	3	4	5	6	7	8	9
u_j	1	2	3	4	1	1	2	3	4
v_j	1	2	3	4	1	2	4	3	4

TAB. 7.2: Les valeurs de $u(j)$ et $v(j)$

Selon le théorème, nous nous intéressons aux séquences de la forme $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2 \prec b_3 \prec \sigma_3 \prec \sigma_4 \prec b_4$. À partir du tableau sur la figure 7.2, les affectations possibles de travaux sont : $5 \in \sigma_1$, $6 \in \sigma_1$ ou $6 \in \sigma_2$, $7 \in \sigma_2$ ou $7 \in \sigma_3$ ou $7 \in \sigma_4$, $8 \in \sigma_3$ et $9 \in \sigma_4$. Rappelons que l'ordre des travaux à l'intérieur de chaque sous-séquence σ_i n'influence pas la valeur du makespan. Alors, par énumération de toutes les possibilités, nous obtenons treize séquences optimales dont le makespan optimal est $C_{\max} = 59$.

$1 \prec 5 \prec 6 \prec 2 \prec 7 \prec 3 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 6 \prec 2 \prec 3 \prec 8 \prec 7 \prec 9 \prec 4$,
$1 \prec 5 \prec 6 \prec 2 \prec 3 \prec 7 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 6 \prec 2 \prec 3 \prec 8 \prec 9 \prec 7 \prec 4$,
$1 \prec 5 \prec 2 \prec 6 \prec 7 \prec 3 \prec 8 \prec 9 \prec 4(*)$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 7 \prec 8 \prec 9 \prec 4(*)$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 8 \prec 7 \prec 9 \prec 4(*)$.

$1 \prec 6 \prec 5 \prec 2 \prec 7 \prec 3 \prec 8 \prec 9 \prec 4$,
$1 \prec 6 \prec 5 \prec 2 \prec 3 \prec 8 \prec 7 \prec 9 \prec 4$,
$1 \prec 6 \prec 5 \prec 2 \prec 3 \prec 7 \prec 8 \prec 9 \prec 4$,
$1 \prec 6 \prec 5 \prec 2 \prec 3 \prec 8 \prec 9 \prec 7 \prec 4$,
$1 \prec 5 \prec 2 \prec 7 \prec 6 \prec 3 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 8 \prec 9 \prec 7 \prec 4$,

Pour cette instance du problème, notons qu'il n'y a que trois séquences (indiquées par une étoile) satisfaisant la règle de Johnson.

7.4.3 Discussion sur l'insensibilité

L'ordre partiel suffisant défini dans le théorème 8 utilise l'ordre relatif des durées opératoires p_{j1} et p_{j2}. L'ensemble de solutions optimales caractérisé est donc insensible aux variatations de durées opératoires dans la mesure où l'ordre relatif de ces durées reste inchangé. Comme nous l'avons fait dans le chapitre 4 pour le problème à une machine, il est ici aussi possible d'associer aux durées des intervalles potentiels de réalisation (i.e. $p_{j1} \in [\underline{p}_{j1}, \overline{p}_{j1}]$ et $p_{j2} \in [\underline{p}_{j2}, \overline{p}_{j2}]$, dans la mesure où les intervalles restent disjoints, c'est-à-dire qu'il existe un ordre total entre les durées. Des valeurs au mieux et au pire du makespan optimal peuvent alors être déterminées en utilisant respectivement soit les valeurs $\underline{p}$, soit les valeurs $\overline{p}$, dans la formule de calcul du $C_{\max}$ indiquée dans le théorème 8. La performance obtenue est alors robuste vis-à-vis de l'ensemble des scénarios caratérisés par le modèle par intervalles

Pour illustrer cette robustesse, nous montrons dans les paragraphes suivants, en considérant l'exemple précédent, comment l'augmentation de quelques durées opératoires influence l'ensemble optimal précédent.

D'abord, supposons que, à cause d'une perturbation, p_{51} augmente de sorte que $p_{51} = 3$ (au lieu de 2). Cette modification ne change pas les structures d'intervalles I_1 et I_2 car, toutes les relations de Allen les couples d'intervalles de travail sont conservées. Donc, l'ensemble de séquences optimales ci-dessus reste inchangé. La valeur de $C_{\max}$ reste également inchangée (car la tâche 1 du travail 5 n'appartient pas au chemin critique engendrant la valeur de $C_{\max}$.

Supposons à présent que $p_{52} = 5$ (au lieu de 4). Les structures d'intervalles I_1 et I_2 restent encore inchangées, ainsi que l'ensemble de séquences optimales. Néanmoins, la valeur optimale du makespan est modifiée et passe à $C_{\max} = 60$ (au lieu de 59) pour toute séquence optimale de travaux de l'ensemble.

Supposons que p_{62} augmente de sorte que $p_{62} = 7$ (au lieu de 5). Dans ce cas, la structure d'intervalles I_1 est modifiée parce que le travail 6 n'appartient plus à la b-pyramide P_{b_1}. Donc, toute séquence affectant le travail 6 à σ_1 doit être supprimée de l'ensemble de séquences optimales initial. Ainsi, il ne reste que cinq séquences optimales possédant le makespan optimal $C_{\max} = 62$.

$1 \prec 5 \prec 2 \prec 6 \prec 7 \prec 3 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 2 \prec 7 \prec 6 \prec 3 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 7 \prec 8 \prec 9 \prec 4$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 8 \prec 7 \prec 9 \prec 4$,
$1 \prec 5 \prec 2 \prec 6 \prec 3 \prec 8 \prec 9 \prec 7 \prec 4$.

À travers ce scénario simple, nous constatons que notre approche est relativement insensible à des variations de données, à condition que ces variations ne modifient pas les bases des structures d'intervalles. Évidemment, si cette condition n'est plus respectée, alors une nouvelle analyse des structures d'intervalles est nécessaire pour recalculer dynamiquement les nouvelles b-pyramides.

7.5 Résultats expérimentaux

Dans ce paragraphe, nous présentons les résultats expérimentaux obtenus pour le problème $F2|prmu|C_{\max}$. Nous considérons des instances de n travaux où $n \in \{10, 50, 100, 500\}$. Pour chaque valeur de n, 32 instances de problème ont été générées en tirant uniformément les durées opératoires entre 1 et 100. Les tests ont été menés sur une station Sun Blade 150 ayant 640 Mo de RAM. Les résultats obtenus sont présentés dans les tableaux suivants.

Nous constatons que le nombre de solutions optimales trouvées augmente exponentiel-

lement en fonction du nombre de travaux, le nombre de solutions pouvant même dépasser la valeur maximale représentable par une variable de type *double* dans le cas où on a 500 travaux ! Le nombre moyen de solutions est de 6, pour les problèmes à 10 travaux, de 13.10^{14} pour les problèmes à 50 travaux, de 10^{30} pour les problèmes à 100 travaux et est incalculable dès que l'on considère plus de 500 travaux.

Fichier	Temps (s)	$C_{\max}$	Nb de solutions
fs10_1.txt	0,00	604	4
fs10_2.txt	0,00	649	2
fs10_3.txt	0,00	590	1
fs10_4.txt	0,01	526	2
fs10_5.txt	0,00	499	1
fs10_6.txt	0,00	471	4
fs10_7.txt	0,01	567	2
fs10_8.txt	0,01	487	4
fs10_9.txt	0,00	571	24
fs10_10.txt	0,00	578	24
fs10_11.txt	0,00	612	1
fs10_12.txt	0,00	555	1
fs10_13.txt	0,00	755	1
fs10_14.txt	0,00	567	1
fs10_15.txt	0,01	586	32
fs10_16.txt	0,01	662	1
fs10_17.txt	0,00	484	2
fs10_18.txt	0,00	585	4
fs10_19.txt	0,00	454	1
fs10_20.txt	0,00	364	1
fs10_21.txt	0,00	651	16
fs10_22.txt	0,01	627	2
fs10_23.txt	0,01	504	4
fs10_24.txt	0,00	596	2
fs10_25.txt	0,00	555	2
fs10_26.txt	0,00	454	1
fs10_27.txt	0,00	568	2
fs10_28.txt	0,00	632	16
fs10_29.txt	0,01	557	8
fs10_30.txt	0,01	475	1
fs10_31.txt	0,00	613	12
fs10_32.txt	0,00	481	1

TAB. 7.3: Les problèmes de 10 travaux

Fichier	Temps (s)	$C_{\max}$	Nb de solutions
fs50_1.txt	0,00	2567	5.00965e+13
fs50_2.txt	0,00	2411	4.03108e+08
fs50_3.txt	0,00	2690	4.12782e+10
fs50_4.txt	0,00	2628	4.1943e+06
fs50_5.txt	0,00	2528	2.93601e+07
fs50_6.txt	0,00	3236	1.51165e+18
fs50_7.txt	0,01	2329	1.86624e+06
fs50_8.txt	0,00	2548	4.73946e+14
fs50_9.txt	0,00	2370	5.15978e+09
fs50_10.txt	0,00	2422	5.05658e+14
fs50_11.txt	0,01	2800	8.06216e+11
fs50_12.txt	0,00	2562	6.60452e+11
fs50_13.txt	0,00	2431	5.80475e+10
fs50_14.txt	0,01	2746	1.01445e+15
fs50_15.txt	0,00	2587	1.69869e+08
fs50_16.txt	0,00	2675	2.44612e+11
fs50_17.txt	0,00	2699	2.34827e+12
fs50_18.txt	0,00	2810	3.38151e+16
fs50_19.txt	0,01	2586	8.35884e+11
fs50_20.txt	0,00	2375	1.91319e+13
fs50_21.txt	0,01	2847	9.17294e+09
fs50_22.txt	0,01	2675	2.539e+17
fs50_23.txt	0,00	2419	5.44196e+10
fs50_24.txt	0,00	2420	4.12782e+11
fs50_25.txt	0,00	2978	4.70716e+14
fs50_26.txt	0,00	2409	8.51364e+11
fs50_27.txt	0,00	2591	4.29982e+08
fs50_28.txt	0,01	2466	5.28362e+13
fs50_29.txt	0,00	2630	5.94407e+12
fs50_30.txt	0,00	2647	5.30842e+07
fs50_31.txt	0,00	2593	5.87068e+12
fs50_32.txt	0,00	2617	4.40402e+07

TAB. 7.4: Les problèmes de 50 travaux

Fichier	Temps (s)	$C_{\max}$	Nb de solutions
fs100_1.txt	0,01	4900	1.35346e+33
fs100_2.txt	0,01	5271	1.17773e+32
fs100_3.txt	0,00	5328	2.43469e+27
fs100_4.txt	0,00	5139	2.3934e+24
fs100_5.txt	0,00	5287	1.33555e+28
fs100_6.txt	0,00	5368	1.31061e+45
fs100_7.txt	0,01	5418	1.22209e+42
fs100_8.txt	0,01	5393	1.09772e+31
fs100_9.txt	0,00	4907	2.43515e+28
fs100_10.txt	0,01	5355	1.1346e+35
fs100_11.txt	0,01	5515	4.47777e+31
fs100_12.txt	0,00	5136	2.98148e+42
fs100_13.txt	0,01	4971	4.30794e+46
fs100_14.txt	0,01	5176	8.7038e+33
fs100_15.txt	0,00	5534	9.49505e+39
fs100_16.txt	0,00	4852	3.76874e+35
fs100_17.txt	0,01	4954	1.33429e+38
fs100_18.txt	0,01	5128	5.18142e+34
fs100_19.txt	0,00	5001	5.48603e+37
fs100_20.txt	0,01	5016	2.99219e+42
fs100_21.txt	0,01	5323	2.53285e+43
fs100_22.txt	0,00	4899	6.49732e+43
fs100_23.txt	0,01	5149	1.38201e+38
fs100_24.txt	0,01	5370	9.47676e+21
fs100_25.txt	0,00	5082	2.42605e+21
fs100_26.txt	0,00	5230	3.66321e+29
fs100_27.txt	0,00	5502	4.04342e+36
fs100_28.txt	0,00	5137	1.48888e+32
fs100_29.txt	0,01	5274	3.68457e+31
fs100_30.txt	0,00	4997	1.25241e+22
fs100_31.txt	0,00	4939	2.56191e+34
fs100_32.txt	0,00	5445	1.22383e+43

TAB. 7.5: Les problèmes de 100 travaux

Fichier	Temps (s)	$C_{\max}$	Nb de solutions
fs500_1.txt	0,02	24874	Infinity
fs500_2.txt	0,03	24823	Infinity
fs500_3.txt	0,03	25564	Infinity
fs500_4.txt	0,03	24688	Infinity
fs500_5.txt	0,03	25048	Infinity
fs500_6.txt	0,02	26411	9.90876e+302
fs500_7.txt	0,03	24645	2.75414e+296
fs500_8.txt	0,03	25272	Infinity
fs500_9.txt	0,03	25931	8.17601e+304
fs500_10.txt	0,02	25447	1.36351e+302
fs500_11.txt	0,03	25575	Infinity
fs500_12.txt	0,02	26371	1.82286e+190
fs500_13.txt	0,03	24879	Infinity
fs500_14.txt	0,03	25222	Infinity
fs500_15.txt	0,02	25497	1.23614e+276
fs500_16.txt	0,03	26542	Infinity
fs500_17.txt	0,02	23948	Infinity
fs500_18.txt	0,03	23861	Infinity
fs500_19.txt	0,02	25123	Infinity
fs500_20.txt	0,02	24925	Infinity
fs500_21.txt	0,03	25011	Infinity
fs500_22.txt	0,02	25315	5.99583e+297
fs500_23.txt	0,03	25977	Infinity
fs500_24.txt	0,03	25581	Infinity
fs500_25.txt	0,02	25054	1.06696e+288
fs500_26.txt	0,03	25539	Infinity
fs500_27.txt	0,03	25163	5.38859e+299
fs500_28.txt	0,03	26370	Infinity
fs500_29.txt	0,03	25545	Infinity
fs500_30.txt	0,03	26302	Infinity
fs500_31.txt	0,03	25459	Infinity
fs500_32.txt	0,02	25221	1.74131e+258

TAB. 7.6: Les problèmes de 500 travaux

Chapitre 8

Problèmes de job shop

Dans ce chapitre, nous montrons comment les ordres partiels présentés aux chapitres 4 et 7 peuvent être utilisés pour le problème job shop. Un résultat trivial pour le problème $J2||C_{\max}$, basé sur le résultat obtenu pour $F2|prmu|C_{\max}$, est tout d'abord présenté. Quelques algorithmes pour le problème plus général, $Jn||C_{\max}$ sont ensuite décrits. Notons que ces algorithmes sont encore au stade de développement à l'heure où ce mémoire est rédigé et qu'ils n'ont donc pas encore été évalués. Toutefois, nous pensons qu'ils méritent d'être exposés dans ce manuscrit.

8.1 Le job shop à deux machines $J2||C_{\max}$

8.1.1 Un résultat trivial

Dans cette section, nous présentons un résultat trivial pour le problème de job shop à deux machines M_1 et M_2 (chaque travail étant constitué d'au plus deux tâches). Ce type de problème est résolu optimalement en temps polynomial par l'algorithme de Jackson [Jackson 56], qui utilise la règle de Johnson. Dans cet algorithme, l'ensemble des travaux est partitionné en quatre sous-ensembles en fonction des gammes opératoires :

- le sous-ensemble $\{O_1\}$ composé des travaux n'utilisant que M_1 ;
- le sous-ensemble $\{O_2\}$ composé des travaux n'utilisant que M_2 ;
- le sous-ensemble $\{O_{12}\}$ composé des travaux passant sur M_1, puis sur M_2 ;
- le sous-ensemble $\{O_{21}\}$ composé des travaux passant sur M_2, puis sur M_1 ;

Les travaux des sous-ensembles $\{O_{12}\}$ et $\{O_{21}\}$ sont séquencés selon la règle de Johnson et ceux de $\{O_1\}$ et $\{O_2\}$ dans un ordre quelconque. Un ordonnancement optimal est obtenu en séquençant :

- sur M_1 : $\{O_{12}\} < \{O_1\} < \{O_{21}\}$;

- sur M_2 : $\{O_{21}\} < \{O_2\} < \{O_{12}\}$;

En adoptant la même notation, la propriété suivante peut être énoncée.

Propriété 8.1.1. *Étant donné un problème job shop à deux machines $J2||C_{\max}$ et ses quatre sous ensembles $\{O_1\}$, $\{O_2\}$, $\{O_{12}\}$ et $\{O_{21}\}$, constitués comme décrit précédemment, les travaux de $\{O_{12}\}$ et $\{O_{21}\}$ étant séquencés dans le respect du théorème 8 présenté au chapitre 7 et ceux de $\{O_1\}$ et $\{O_2\}$ étant ordonnés de façon quelconque, tout ordonnancement séquençant sur M_1, les travaux de $\{O_{12}\}$ puis les travaux de $\{O_1\}$ suivis des travaux de $\{O_{21}\}$, et sur M_2, les travaux de $\{O_{21}\}$ puis les travaux de $\{O_2\}$ suivis des travaux de $\{O_{12}\}$ est optimal.*

Démonstration. La preuve est triviale. Dans la règle de Jackson, il suffit d'assurer que les ordres choisis pour les ensembles $\{O_{12}\}$ et $\{O_{21}\}$ soient optimaux relativement aux deux problèmes de flow shop $F2|prmu|C_{\max}$ où les travaux de $\{O_{12}\}$ passent sur M_1 puis M_2 et les travaux de $\{O_{21}\}$ passent sur M_2 puis M_1, pour que l'ordonnancement soit optimal. Or, le théorème 8 garantit cette optimalité. □

8.1.2 Exemple

À titre d'exemple, considérons une instance du problème $J2||C_{\max}$ définie dans le tableau 8.1.

Dans cet exemple, le partitionnement des travaux dans les quatre sous-ensembles $\{O_1\}$, $\{O_2\}$, $\{O_{12}\}$ et $\{O_{21}\}$ est le suivant :

- $\{O_1\} = \{2, 3\}$;
- $\{O_2\} = \{6, 8\}$;
- $\{O_{12}\} = \{1, 4, 10, 12\}$;
- $\{O_{21}\} = \{5, 7, 9, 11\}$;

Pour les deux sous-ensembles $\{O_1\}$ et $\{O_2\}$, les travaux peuvent être séquencés dans un ordre arbitraire.

Pour $\{O_{12}\}$, les travaux définissent une structure d'intervalles I_1 (il n'y a aucun travail dans I_2) mono-pyramidale décrite sur la figure 8.1. La seule base de cette structure est le travail 10, et trois travaux non bases 1, 4, 12 appartiennent à la b-pyramide de base 10. Selon le théorème 8, l'ensemble de solutions optimales pour $\{O_{12}\}$ est de la forme $10 \prec (1 - 4 - 12)$, où $(1 - 4 - 12)$ est un groupe de travaux permutables.

	Opération	
Travail j	1	2
1	$4, M_1$	$7, M_2$
2	$3, M_1$	–
3	$4, M_1$	–
4	$3, M_1$	$5, M_2$
5	$1, M_2$	$7, M_1$
6	$1, M_2$	–
7	$5, M_2$	$6, M_1$
8	$3, M_2$	–
9	$4, M_2$	$9, M_1$
10	$2, M_1$	$10, M_2$
11	$6, M_2$	$2, M_1$
12	$6, M_1$	$9, M_2$

TAB. 8.1: Une instance du problème $J2||C_{\max}$

Pour le sous-ensemble $\{O_{21}\}$, la structure d'intervalles I_1 définie par les travaux (voir la figure 8.2) contient trois bases, le travail 9, le travail 5 et le travail 11 (cf. section 7.2 pour la définition de I_1 et I_2). Le travail 7, non base, appartient aux deux b-pyramides caractérisées par les bases 5 et 9. Selon le théorème 8, l'ensemble de solutions optimales contient deux séquences $5 \prec 7 \prec 9 \prec 11$ et $5 \prec 9 \prec 7 \prec 11$, que l'on représente par la séquence de groupes $5 \prec (7-9) \prec 11$, où (7-9) est un groupe de travaux permutables.

Les ordonnancements optimaux du problème sont au nombre de 48 et respectent les séquencements suivants, donnant une valeur optimale du makespan $C^*_{\max} = 51$.

- sur M_1 : $\Big(10 \prec (1-4-12)\Big) \prec (2-3) \prec \Big(5 \prec (7-9) \prec 11\Big)$;
- sur M_2 : $\Big(5 \prec (7-9) \prec 11\Big) \prec (6-8) \prec \Big(10 \prec (1-4-12)\Big)$;

8.2 Le job shop à plusieurs machines $Jn||C_{\max}$

8.2.1 Idée générale

Dans cette section, nous proposons deux algorithmes pour la résolution du problème de job shop à plusieurs machines en utilisant le résultat de dominance obtenu pour le problème à une machine, présenté dans la partie 2. Notons que ces algorithmes sont présentés et illustrés, mais qu'ils n'ont pas encore été évalués finement sur de grosses instances de problème.

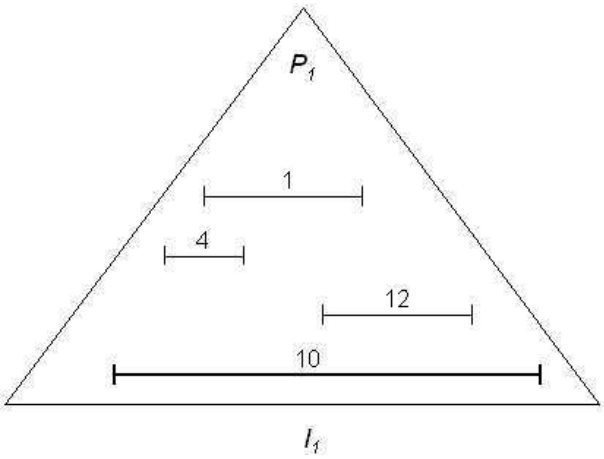


FIG. 8.1: Structure d'intervalles pour le sous-ensemble $\{O_{12}\}$

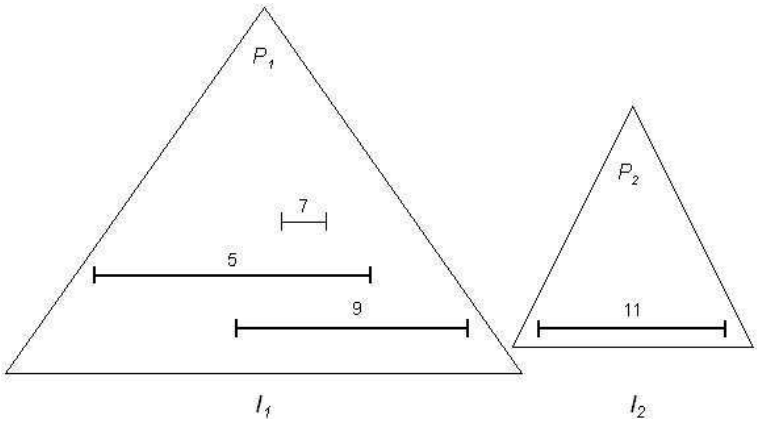


FIG. 8.2: Structure d'intervalles pour le sous-ensemble $\{O_{21}\}$

Le problème $Jn||C_{\max}$ consiste à ordonnancer un ensemble T de travaux sur un ensemble M de m machines $M = \{M_1, \ldots M_m\}$. Chaque travail est constitué d'un ensemble de tâches devant s'exécuter chacune sur une des m machines de M dans un ordre quelconque. L'objectif est de trouver un ordonnancement minimisant le makespan $C_{\max}$. Notons $(1, \ldots, n)$ l'ensemble de tâches du problème, p_i la durée opératoire de la tâche i. Notons aussi que 0 et $n+1$ désignent deux tâches fictives représentant respectivement le début et la fin de l'ordonnancement.

Ce problème a été démontré NP-difficile [Lenstra *et al.* 77]. De nombreuses méthodes existent dans la littérature pour le résoudre de façon approchée ou exacte comme la méthode proposée par Adams et al. dans [Adams *et al.* 88], la méthode proposée par Carlier et al. dans [Carlier & Pinson 89]. L'objectif de ces méthodes est de trouver une solution d'ordonnancement proche de l'optimalité ou optimale dans un temps acceptable.

Toujours dans l'optique de produire de la flexibilité séquentielle, nous cherchons comment construire des algorithmes permettant de déterminer un ensemble flexible de solutions possédant des performances au mieux et au pire connues. Deux algorithmes sont décrits dont les principes sont présentés dans les paragraphes suivants.

Classiquement, un problème $Jn||C_{\max}$ peut être décomposé en m problèmes à une machine interdépendants $1|r_i|L_{\max}$. Les dates de début au plus tôt et de fin au plus tard des travaux pour chaque machine peuvent être définis ainsi que précisé dans [Adams *et al.* 88] :

$$r_i = L(0, i)$$

et

$$d_i = L(0, n+1) - L(i, n+1) + p_i$$

où $L(i, j)$ est la longueur du plus long chemin de i à j sur le graphe conjonctif du problème, qui peut être déterminée grâce à l'algorithme de Bellman-Ford (cf. [Esquirol & Lopez 99]).

Pour chaque problème à une machine, l'application du théorème des pyramides permet de déterminer sur chaque machine un ensemble de séquences dominantes, la dominance étant ici relative au problème à une machine et non au problème job shop (puisque les problèmes à une machine sont interdépendants). En utilisant les méthodes de calcul de $L_i^{\min}$ et de $L_i^{\max}$ (cf. chapitre 4), nous pouvons déduire pour chaque travail i le retard au mieux $L_i^{\min}$ et le retard au pire $L_i^{\max}$. Nous notons $s_i^{\min}$ et $s_i^{\max}$ les dates de début au mieux et au pire de la tâche i, calculées de la façon suivante :

$$s_i^{\min} = L_i^{\min} + d_i - p_i \qquad (8.1)$$

$$s_i^{\max} = L_i^{\max} + d_i - p_i \qquad (8.2)$$

Les algorithmes que nous proposons visent à trouver sur chaque machine une structure d'intervalles caractérisant un ensemble de séquences dominantes respectant la condition suivante :

$$s_i^{\max} + p_i \leq s_{i+1}^{\min}$$

où $(i, i+1)$ est un couple de deux tâches tel que i doit précéder $i+1$, selon la gamme opératoire.

En effet, en respectant cette contrainte, on peut assurer que tous les ensembles dominants de séquences constitués sur les machines sont compatibles entre eux (i.e. la date de fin au pire de i sur une machine est inférieure ou égale à la date de début au mieux de $i+1$ sur une autre machine ce qui garantit le respect de la gamme opératoire). L'ensemble de toutes les combinaisons possibles des séquences dominantes sur chaque machine forme

alors l'ensemble de solutions du problème global.

L'évaluation des performances au mieux et au pire de l'ensemble de solutions du problème peut être réalisée de la façon suivante.

Notons ED_i l'ensemble de séquences dominantes obtenu sur la machine M_i et $C_{ED_i}^{\min}$ et $C_{ED_i}^{\max}$ respectivement la durée au mieux et au pire de l'ordonnancement sur chaque machine. On a :

$$C_{ED_i}^{\min} = \max(s_i^{\min} + p_i), \ \ \forall i \in ED_i \qquad (8.3)$$

$$C_{ED_i}^{\max} = \max(s_i^{\max} + p_i), \ \ \forall i \in ED_i \qquad (8.4)$$

Notons alors $\underline{C}_{\max}$ et $\overline{C}_{\max}$ respectivement les makespan au mieux et au pire du job shop. On a alors :

$$\underline{C}_{\max} = \max(C_{ED_i}^{\min}) \ \ \forall \ ED_i \qquad (8.5)$$

et

$$\overline{C}_{\max} = \max(C_{ED_i}^{\max}) \ \ \forall \ ED_i \qquad (8.6)$$

Le makespan $C_{\max}$ du problème est donc tel que :

$$\underline{C}_{\max} \leq C_{\max} \leq \overline{C}_{\max}$$

8.2.2 Un premier algorithme TCJ-1

8.2.2.1 Principes

Nous supposons que le problème est décomposé en m problèmes à une machine dont les dates de début au plus tôt et de fin au plus tard sont calculées par application de l'algorithme de Bellman-Ford sur le graphe conjonctif associé au problème, ne prenant pas en compte les contraintes de ressource. Pour chaque problème à une machine, nous utilisons ensuite le théorème des pyramides pour déterminer l'ensemble de séquences dominantes correspondant. Puis, pour chaque tâche i, nous calculons les retards au mieux $L_i^{\min}$ et au pire $L_i^{\max}$ selon les algorithmes décrits dans le chapitre 4. Une fois $L_i^{\min}$ et $L_i^{\max}$ établis, les dates de début au mieux $s_i^{\min}$ et au pire $s_i^{\max}$ peuvent être déduites par l'utilisation des formules (8.1) et (8.2).

Comme décrit dans la partie précédente, le but de notre algorithme TCJ-1 est de trouver sur chaque machine une structure d'intervalles caractérisant un ensemble de séquences dominantes tel que :

$$s_i^{\max} + p_i \leq s_{i+1}^{\min} \qquad (*)$$

Pour parvenir à cet objectif, nous pouvons agir sur $s_i^{\max}$ ou sur $s_{i+1}^{\min}$. Plus précisément, si deux tâches i et $i+1$ ne respectent pas cette condition, nous allons alors soit réduire $s_i^{\max}$ en agissant sur la structure d'intervalles dont i fait partie (diminution de d_i), soit augmenter $s_{i+1}^{\min}$ en agissant sur la structure d'intervalles dont $i+1$ fait partie (augmentation de r_{i+1}). Notons que dans les deux cas, la flexibilité séquentielle est réduite. Cependant, actualiser $s_i^{\max}$ tend à réduire la flexibilité disponible au début de l'ordonnancement alors qu'agir sur $s_{i+1}^{\min}$ tend à réduire la flexibilité disponible en fin d'ordonnancement, quitte à détériorer la qualité au mieux de l'ordonnancement (voire à perdre l'optimalité).

Notons que s'il existe plusieurs couples de tâches $(i, i+1)$ ne respectant pas la condition $(*)$, nous choisissons alors celui produisant la valeur $\Delta_i = s_i^{\max} - s_{i+1}^{\min}$ la plus grande afin de privilégier les plus fortes actualisations d'abord. D'autres régles pourraient cependant être adoptées comme : choisir le couple $(i, i+1)$ produisant le Δ_i le plus petit, ou choisir d'abord la machine critique M_j et ensuite le couple $(i, i+1)$ sur cette machine produisant le Δ_i le plus grand.

Pour réduire $s_i^{\max}$, il est possible de diminuer la valeur de d_i en imposant cependant que la nouvelle valeur reste supérieure ou égale à $r_i + p_i$:

$$d_i \leftarrow \max(r_i + p_i, d_i - (s_i^{\max} - s_{i+1}^{\min}))$$

Cette actualisation peut être faite de façon itérative dans le but de parvenir au respect de la condition $*$.

Symétriquement, l'augmentation de $s_{i+1}^{\max}$ peut être réalisée en augmentant la valeur de r_{i+1} de la façon suivante :

$$r_{i+1} \leftarrow \min(d_{i+1} - p_{i+1}, r_{i+1} + (s_i^{\max} - s_{i+1}^{\min}))$$

Notons que, pour l'algorithme TCJ-1, la stratégie adoptée est de d'abord diminuer d_i autant que possible pour parvenir au respect de la condition $*$, puis si la condition n'est toujours pas vérifiée et que $d_i = r_i + p_i$, à augmenter r_{i+1} autant que nécessaire. Cette stratégie tend à préserver la flexibilité à la fin de l'ordonnancement qui n'est réduite que dans un deuxième temps.

Après une actualisation de d_i ou de r_{i+1}, l'actualisation est propagée sur toutes les machines. Les paramètres de chaque problème à une machine (les r_i et d_i) et les dates de début au mieux et au pire $s_i^{\min}$, $s_i^{\max}$ sont alors récalculés et les couples ne satisfaisant pas

la condition $(*)$ sont à nouveau recherchés. Le processus d'actualisation de d_i ou de r_{i+1} est réitéré jusqu'à ce que la condition $(*)$ soit vérifiée pour tout couple de tâches reliées par une contrainte de gamme.

Si à un moment donné, nous arrivons à l'état où $d_i = r_i + p_i$ et $r_{i+1} = d_{i+1} - p_{i+1}$ et la condition $(*)$ n'est toujours pas vérifiée, alors la technique de décalage à droite est appliquée. Celle-ci consiste à augmenter en même temps r_{i+1} et d_{i+1} de la valeur $\Delta_i = s_i^{\max} - s_{i+1}^{\min}$. Cette actualisation conduit biensûr à détériorer la performance au mieux de l'ensemble d'ordonnancements caractérisés.

Remarquons que les principes d'actualisation adoptés ne garantissent pas que chaque nouvelle structure d'intervalles ait un ensemble dominant de séquences inclus dans celui caractérisé par la structure d'intervalles initiale. En effet, il ne sert à rien de vouloir rester dans l'ensemble dominant initial puisque rien ne garantit que celui-ci contienne une solution optimale pour le problème général (les problèmes à une machine sont interdépendants).

L'algorithme TCJ-1 est décrit sur la figure 8.3.

8.2.2.2 Illustration de l'algorithme TCJ-1

Pour illustrer l'algorithme TCJ-1, nous considérons un problème académique de job shop à 3 travaux et 3 machines. La gamme opératoire et les durées des tâches sont données dans le tableau 8.2. Le graphe conjonctif associé à ce problème est représenté sur la figure 8.4. L'application de l'algorithme TCJ-1 passe par les étapes suivantes :

Étape 1 : Le problème de job shop est décomposé en trois problèmes à une machine sur M_1, M_2 et M_3.

Sur M_1 s'exécutent trois tâches 1, 5 et 9 dont les dates de début au mieux et au pire sont :

$$r_1 = L(0,1) = 0, \qquad d_1 = L(0,10) - L(1,10) + p_1 = 19 - 19 + 3 = 3$$

où $L(i,j)$ est la longueur du plus long chemin de i à j dans le graphe de la figure 8.4. Les mêmes calculs étant appliqués pour les autres tâches, nous obtenons pour M_1 les

Proc TCJ-1($Solution[M_1,\ldots,M_m], \underline{C}_{\max}, \overline{C}_{\max}$)(*)
  Pour chaque $M_i \in M$ faire
    $Solution[M_i] \leftarrow \emptyset$;
  FinPour
  $TachesIncoherentes \leftarrow \emptyset$;
  $TachesCritiques \leftarrow NULL$;

  /* Décomposer le job shop en m problèmes à une machine */
  Calculer r_i et d_i de chaque tâche par l'application de l'algorithme de Bellman-Ford sur le graphe conjonctif ;
  Calculer $L_i^{\min}$, $L_i^{\max}$ des tâches par les algorithmes CalculerLjMin, CalculerLjMax appliqués sur chaque machine M_i.
  Calculer $s_i^{\min}$ et $s_i^{\max}$ à partir de $L_i^{\min}$ et $L_i^{\max}$; /* utiliser (8.1) et 8.2) */

  /* Déterminer les tâches non cohérentes */
  Pour chaque couple $(i, i+1)$ selon la gamme opératoire faire
    Si $s_i^{\max} + p_i > s_{i+1}^{\min}$ alors
      $TachesIncoherentes \leftarrow TachesIncoherentes \cup (i, i+1)$;
    FinSi
  FinPour

  TantQue $TachesIncoherentes \neq \emptyset$ faire
    $TachesCritiques \leftarrow$ couple $(i, i+1)$ produisant $\max(\Delta_i = s_i^{\max} - s_{i+1}^{\min})$ $\forall$ couple $(i, i+1) \in TachesIncoherentes$;
    /* Actualiser d_i et r_{i+1} */
    $d_i \leftarrow d_i - \Delta_i$;
    Si $d_i < r_i + p_i$ alors
      $d_i \leftarrow r_i + p_i$;
      $r_{i+1} \leftarrow r_{i+1} + \Delta_i$;
      Si $r_{i+1} > d_{i+1} - p_{i+1}$ alors
        $d_{i+1} \leftarrow d_{i+1} + \Delta_i$;
      FinSi ;
    FinSi ;

    Mettre à jour tous les r_i, d_i ;
    Mettre à jour $TachesIncoherentes$;
  FinTantQue ;
  Pour chaque M_i faire
    $Solution[M_i] \leftarrow$ structure d'intervalles actuelle sur M_i ;
  FinPour ;
  Calculer $\underline{C}_{\max}$ et $\overline{C}_{\max}$ par (8.3),(8.4),(8.5),(8.6) ;
FinProc
(*) $Solution[M_1,\ldots,M_m]$ est la liste de structures d'intervalles, chacune correspondant à celle sur une machine ; $TachesIncohererntes$ est l'ensemble de couples de tâches $(i, i+1)$ ne respectant pas la condition (*) ; $TachesCritiques$ est la couple de tâches choisi pour actualiser leur r_i et d_i ;

FIG. 8.3: Algorithme TCJ-1

Travail	*Tâche*	M_i	$p_{i,j}$
1	1	1	3
1	2	2	7
1	3	3	9
2	4	2	7
2	5	1	10
2	6	3	2
3	7	3	3
3	8	2	3
3	9	1	4

TAB. 8.2: Une instance de job shop à trois machines

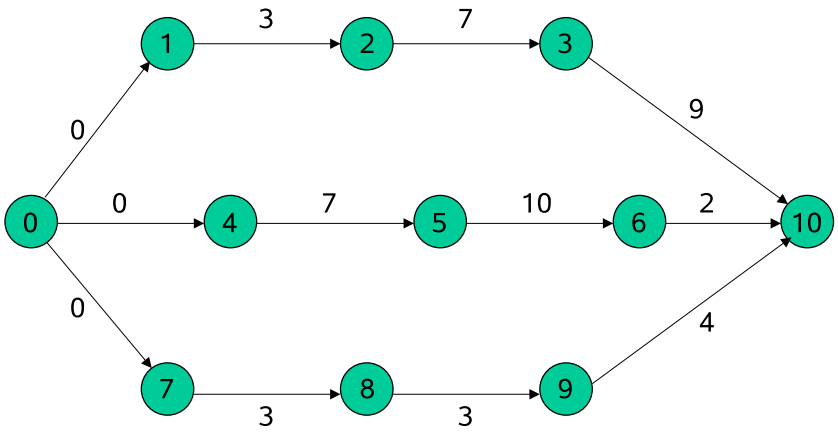


FIG. 8.4: Graphe conjonctif du problème dans le tableau 8.2

données définies dans le tableau 8.3 et la structure d'intervalles correspondante illustrée sur la figure 8.3.

Tâche	r_j	d_j	p_j
1	0	3	3
5	7	17	10
9	6	19	4

TAB. 8.3: Problème à une machine sur M_1 - Étape 1

Cette structure d'intervalles caractérise un ensemble dominant de deux séquences : $1 \prec 5 \prec 9$ et $1 \prec 9 \prec 5$. Les dates de début au mieux et au pire de chaque tâche sont :

$$s_1^{\min} = 0, \;\; s_1^{\max} = 0, \qquad s_5^{\min} = 7, \;\; s_5^{\max} = 10, \qquad s_9^{\min} = 6, \;\; s_9^{\max} = 17$$

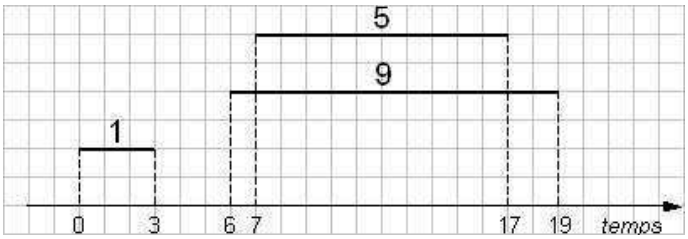


FIG. 8.5: Structure d'intervalles du problème sur M_1 - Étape 1

Les calculs similaires appliqués aux machines M_2 et M_3, sont également réalisés.

Sur M_2, les données du problème à une machine sont décrites dans le tableau 8.4 et la structure d'intervalles est représentée sur la figure 8.6.

Tâche	r_j	d_j	p_j
2	3	10	7
4	0	7	7
8	3	15	3

TAB. 8.4: Problème à une machine sur M_2 - Étape 1

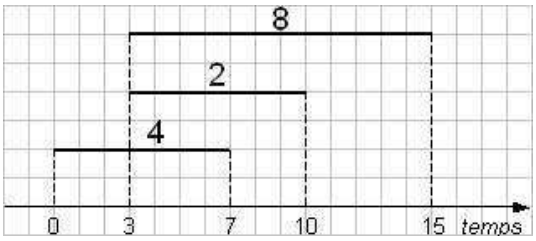


FIG. 8.6: Structure d'intervalles du problème sur M_2 - Étape 1

L'ensemble de séquences dominantes sur M_2 contient une seule séquence : $4 \prec 2 \prec 8$, d'où :

$$s_4^{\min} = 0, \;\; s_4^{\max} = 0, \qquad s_2^{\min} = 7, \;\; s_2^{\max} = 7, \qquad s_8^{\min} = 14, \;\; s_8^{\max} = 14$$

Sur M_3, les données du problème à une machine sont indiquées dans le tableau 8.5 et la structure d'intervalles est représentée sur la figure 8.7.

L'ensemble de séquences dominantes sur M_3 contient une seule séquence : $7 \prec 3 \prec 6$, d'où :

Tâche	r_j	d_j	p_j
3	10	19	9
6	17	19	2
7	0	12	3

TAB. 8.5: Problème à une machine sur M_3 - Étape 1

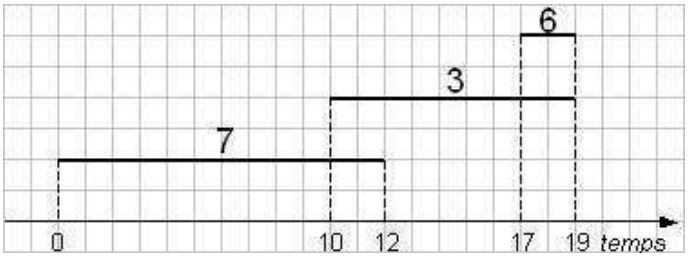


FIG. 8.7: Structure d'intervalles du problème sur M_3 - Étape 1

$s_7^{\min} = 0,\ s_7^{\max} = 0, \qquad s_3^{\min} = 10,\ s_3^{\max} = 10, \qquad s_6^{\min} = 19,\ s_6^{\max} = 19$

Passons maintenant à la vérification de cohérence de la condition $*$ entre les couples de tâches reliées par une contrainte de gamme opératoire. Le résultat est donné sur le tableau 8.6.

Couple (i,j)	Vérification	État	Δ_i
(1,2)	$s_1^{\max} + p_1 = 3 < s_2^{\min} = 7$	cohérent	
(2,3)	$s_2^{\max} + p_2 = 14 > s_3^{\min} = 10$	incohérent	4
(4,5)	$s_4^{\max} + p_4 = 7 = s_5^{\min}$	cohérent	
(5,6)	$s_5^{\max} + p_5 = 20 > s_6^{\min} = 19$	incohérent	1
(7,8)	$s_7^{\max} + p_7 = 3 < s_8^{\min} = 14$	cohérent	
(8,9)	$s_8^{\max} + p_8 = 17 > s_9^{\min} = 6$	incohérent	11

TAB. 8.6: Vérification de cohérence entre tâches

La liste *TachesIncoherentes* comprend trois couples de tâches (2,3), (5,6) et (8,9) et $\Delta_8 = 11 = \max \Delta_i$. Le couple (8,9) est donc choisi pour actualiser les dates. Deux manipulations sont à considérer : soit $d_8 \leftarrow d_8 - \Delta_8$, soit $r_9 \leftarrow r_9 + \Delta_8$.

Comme $d_8 - \Delta_8 = 4 < r_8 + p_8 = 6$, on ne peut pas diminuer d_8. De plus, comme $r_9 + \Delta_8 = 17 > d_9 - p_9 = 15$, on ne peut non plus augmenter r_9. Par conséquent, il faut décaler la tâche 9 à droite pour que le couple (8,9) soit cohérent. Actualisons donc $r_9 \leftarrow r_9 + \Delta_8 = 17$ et $d_9 \leftarrow d_9 + \Delta_8 = 30$.

La structure d'intervalles actuelle du problème à une machine sur M_1 est donnée sur la fi-

gure 8.8. Cette structure d'intervalles caractérise une seule séquence dominante : $1 \prec 5 \prec 9$.

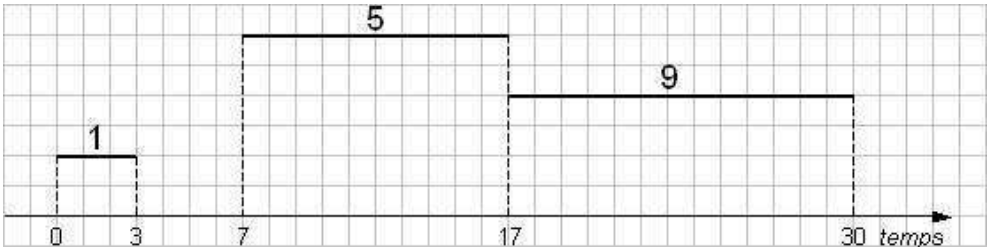


FIG. 8.8: Structure d'intervalles du problème sur M_1 - Étape 2

Étape 2 : Récalculons maintenant les dates de début au mieux et au pire pour les tâches sur la machine M_1, les nouvelles valeurs sont :

$$s_1^{\min} = 0,\ \ s_1^{\max} = 0, \qquad s_5^{\min} = 7,\ \ s_5^{\max} = 7, \qquad s_9^{\min} = 17,\ \ s_9^{\max} = 17$$

La vérification de cohérence étant effectuée, la liste $TachesIncoherences$ ne contient plus qu'un seule couple de tâches (2,3) tel que $s_2^{\max} + p_2 = 14 > s_3^{\min} = 10$. Deux manipulations sont envisagées : soit $d_2 \leftarrow d_2 - \Delta_2$, soit $r_3 \leftarrow r_3 + \Delta_2$.

Comme $d_2 - \Delta_2 = 6 < r_2 + d_2$ et $r_3 + \Delta_2 = 14 > d_3 - p_3$, il faut à nouveau décaler la tâche 3 à droite en imposant : $r_3 \leftarrow r_3 + \Delta_2 = 14$ et $d_3 \leftarrow d_3 + \Delta_2 = 23$. La nouvelle structure d'intervalles sur la machine 3 est représentée sur la figure 8.9 et caractérise deux séquences dominantes : $7 \prec 3 \prec 6$ et $7 \prec 6 \prec 3$.

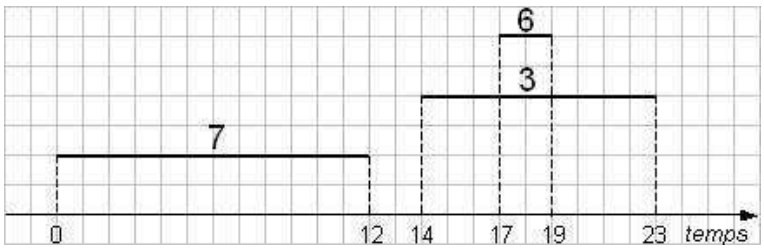


FIG. 8.9: Structure d'intervalles du problème sur M_3 - Étape 3

Étape 3 : La mise à jour des dates de début au mieux et au pire des tâches concernées donne le résultat suivant :

$$s_7^{\min} = 0,\ \ s_7^{\max} = 0, \qquad s_3^{\min} = 14,\ \ s_3^{\max} = 19, \qquad s_6^{\min} = 17,\ \ s_6^{\max} = 23$$

La vérification de cohérence ne détectant aucune incohérence, l'algorithme s'arrête. Nous déterminons alors les bornes du makespan des solutions d'ordonnancement trouvées :

Sur M_1 : $\max(\underline{C}_i) = \underline{C_3} = s_3^{\min} + p_3 = 23$ et $\max(\overline{C}_i) = \overline{C_3} = s_3^{\max} + p_3 = 28$.

Sur M_2 : $\max(\underline{C}_i) = \underline{C_6} = s_6^{\min} + p_6 = 19$ et $\max(\overline{C}_i) = \overline{C_6} = s_6^{\max} + p_6 = 25$.

Sur M_3 : $\max(\underline{C}_i) = \underline{C_9} = s_9^{\min} + p_9 = 21$ et $\max(\overline{C}_i) = \overline{C_3} = s_9^{\max} + p_9 = 21$.

Nous déduisons donc : $\underline{C}_{\max} = \max(\underline{C}_3, \underline{C}_6, \underline{C}_9) = 23$ et $\overline{C}_{\max} = \max(\overline{C}_3, \overline{C}_6, \overline{C}_9) = 28$. D'où nous avons : $23 \leq C_{\max} \leq 28$.

L'ensemble de solutions trouvé contient les deux ordonnancements suivants :

- O1 : sur M_1 : $1 \prec 5 \prec 9$, sur M_2 : $4 \prec 2 \prec 8$ et sur M_3 : $7 \prec 6 \prec 3$.
  Le makespan de O1, $C_{\max}^{O1} = 28$.

- O2 : sur M_1 : $1 \prec 5 \prec 9$, sur M_2 : $4 \prec 2 \prec 8$ et sur M_3 : $7 \prec 3 \prec 6$.
  Le makespan de O2, $C_{\max}^{O2} = 25$.

Notons également que l'ordonnancement O2 est optimal et que $C_{\max}^{\text{optimal}} = 25$.

8.2.3 Un deuxième algorithme - TCJ-2

8.2.3.1 Principes

Nous supposons à présent qu'un ordonnancement initial, éventuellement optimal est disponible. Nous proposons un algorithme dont le but est de définir un ensemble d'ordonnancements, contenant l'ordonnancement initial.

À partir de l'ordonnancement initial, nous construisons le graphe conjonctif associé au problème en ajoutant les arcs disjonctifs sélectionnés dans la solution initiale. Nous déterminons ensuite le chemin critique de l'ordonnancement initial, puis nous supprimons tous les arcs ressource ne se situant pas sur ce chemin.

En décomposant le problème job shop en des problèmes à une machine en prenant cette fois en compte les arcs ressource figurant dans le chemin critique, nous obtenons une structure d'intervalles (les tâches figurant sur le chemin critique sont les sommets de la structure). Comme on est sûr qu'il existe pour chaque problème à une machine au moins une séquence satisfaisant la contrainte $s_i^{\max} + p_i \leq s_i^{\min}$, celle de la solution initiale, nous recherchons toutes les structures d'intervalles garantissant que $L_j^{\max} \leq 0$ grâce à l'algorithme TCOR-2. En effet, si la condition suivante est satisfaite :

$$L_i^{\max} \leq 0, \quad \forall i, \ \forall M_i,$$

alors nous pouvons garantir la cohérence entre toutes les structures d'intervalles et obtenons alors un ensemble d'ordonnancements cohérents.

L'algorithme TCJ-2 est décrit sur la figure 8.10.

8.2.3.2 Illustration de l'algorithme TCJ-2

Pour illustrer les principes de TCJ-2, reprenons l'exemple du tableau 8.2. Supposons que l'ordonnancement initial d'entrée est optimal :

- sur M_1 : $1 \prec 5 \prec 9$;
- sur M_2 : $4 \prec 2 \prec 8$;
- sur M_3 : $7 \prec 3 \prec 6$;

Le graphe conjonctif correspondant à cet ordonnancement est représenté sur la figure 8.11 et le chemin critique sur ce graphe est $0 \prec 4 \prec 2 \prec 3 \prec 6 \prec 10$ (en trait gras).

En supprimant les arcs ressource ne se situant pas sur le chemin critique, nous obtenons le graphe conjonctif simplifié de la figure 8.12.

Établissons maintenant les données des problèmes à une machine en se basant sur le graphe simplifié.

Sur M_1 s'exécutent trois tâches 1, 5 et 9 dont les dates de début au plus tôt et de fin au plus tard sont données dans le tableau 8.7.

Tâche	r_j	d_j	p_j
1	0	7	3
5	7	23	10
9	6	25	4

TAB. 8.7: Problème à une machine sur M_1

La structure d'intervalles pour M_1 est représentée sur la figure 8.13, qui caractérise 2 séquences dominantes : $1 \prec 5 \prec 9$ et $1 \prec 9 \prec 5$. D'où les retards au pire des tâches suivants : $\max(L_i^{\max})_{i \in M_1} = \max(L_1^{\max}, L_5^{\max}, L_9^{\max}) = -3 < 0$. La structure d'intervalles sur M_1 est donc cohérente.

Sur M_2 s'exécutent trois tâches 2, 4 et 8 dont les dates de début au plus tôt et de fin au plus tard sont données dans le tableau 8.8.

Proc TCJ-2($SolutionInitiale, Solution[M_1, \ldots, M_m]$) $(*)$
  Pour chaque $M_i \in M$ faire
    $Solution[M_i] \leftarrow \emptyset$;
  FinPour

  Construire le graphe conjontif du problème ;
  Ajouter les arcs disjonctifs arbitrés correspondant à l'ordonnancement initial et déterminer son chemin critique ;
  Enlever les arcs disjonctifs ne se situant pas sur le chemin critique ;

  /* Décomposer le job shop en m problèmes à une machine */
  Calculer r_i et d_i de chaque tâche par l'application l'algorithme de Bellman-Ford sur le graphe conjontif ;
  Calculer $L_i^{\max}$ sur chaque machine par l'algorithme CalculerLjMax ;

  Pour chaque machine M_i faire
    TantQue $\max(L_i^{\max}) > 0$ faire
      Utiliser l'algorithme TCOR-2 pour diminuer $\max(L_i^{\max})$;
    FinTantQue
  FinPour

  Pour chaque M_i faire
    $Solution[M_i] \leftarrow$ structure d'intervalles actuelle sur M_i ;
  FinPour ;
FinProc

$(*)$ *SolutionInitiale* est l'ordonnancement initial fourni par une autre méthode ; $Solution[M_1, \ldots, M_m]$ est la liste de structures d'intervalles, chacune correspondant à celle sur une machine ;

FIG. 8.10: Algorithme TCJ-2

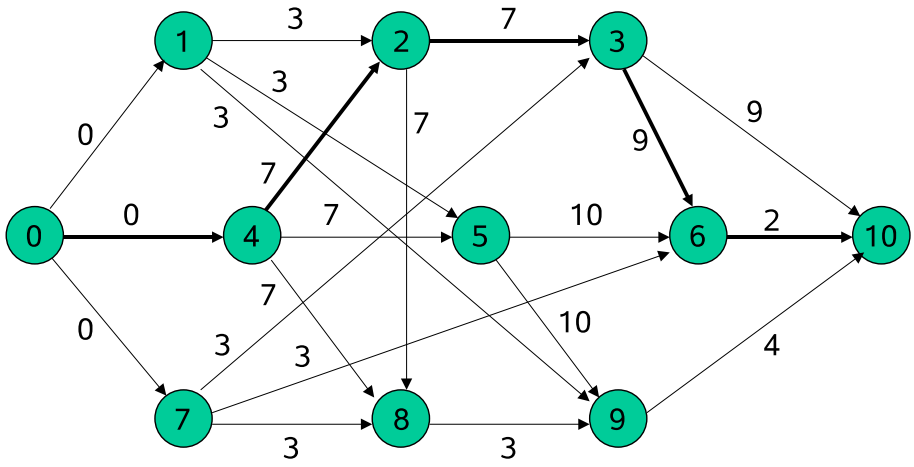


FIG. 8.11: Graphe conjonctif de l'ordonnancement initial

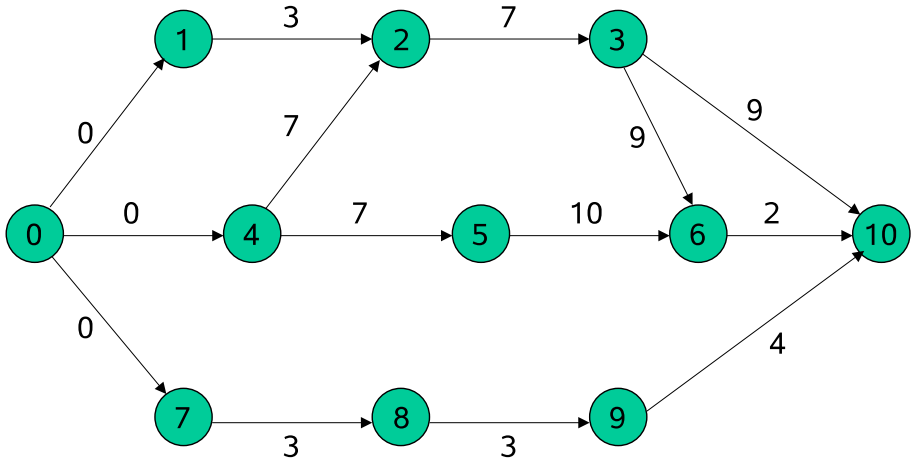


FIG. 8.12: Graphe conjonctif simplifié de l'ordonnancement initial

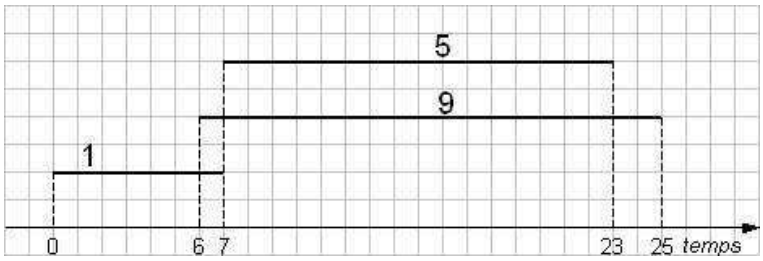


FIG. 8.13: Structure d'intervalles du problème sur M_1

Tâche	r_j	d_j	p_j
2	7	14	7
4	0	7	7
8	3	21	3

TAB. 8.8: Problème à une machine sur M_2

La structure d'intervalles du problème sur M_2 est représentée sur la figure 8.14, qui caractérise 2 séquences dominantes : $4 \prec 2 \prec 8$ et $4 \prec 8 \prec 2$. D'où les retards au pire des tâches : $\max(L_i^{\max})_{i \in M_2} = \max(L_2^{\max}, L_4^{\max}, L_8^{\max}) = L_2^{\max} = 3 > 0$. La cohérence n'est pas vérifiée puisque le retard au pire est positif. L'algorithme TCOR-2 est alors utilisé pour réduire $\max(L_i^{\max})$ sur M_2. La tâche pivot est 2 et le travail libre est 8. Deux actualisations sont alors considérées : soit $2 \prec 8$ (équivalent à $r_8 \leftarrow r_2$), soit $8 \prec 2$ (équivalent à $d_8 \leftarrow d_2$). Comme le deuxième choix, produisant $\max(L_i^{\max}) = 3$, ne correspond pas à une solution, le premier est alors choisi. Il ne reste alors sur la machine M_2 qu'une seule séquence dominante : $4 \prec 2 \prec 8$ avec $\max(L_i^{\max}) = 0$.

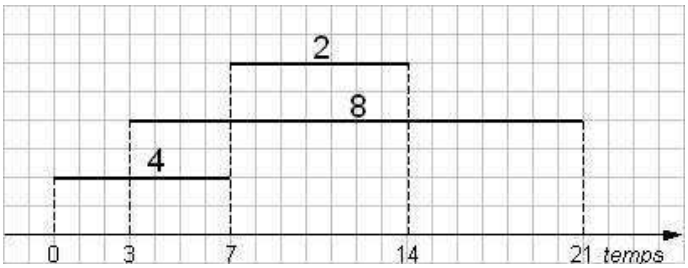


FIG. 8.14: Structure d'intervalles du problème sur M_2

Sur M_3 s'exécutent trois tâches 3, 6 et 7 dont les dates de début au plus tôt et de fin au plus tard sont données dans le tableau 8.9.

La structure d'intervalles sur M_3 est représentée sur la figure 8.15, qui caractérise une seule séquence dominante : $7 \prec 3 \prec 6$. D'où les retards au pire des tâches suivants :

$Tâche$	r_j	d_j	p_j
3	14	23	9
6	23	25	2
7	0	18	3

TAB. 8.9: Problème à une machine sur M_3

$\max(L_i^{\max})_{i\in M_1} = \max(L_7^{\max}, L_3^{\max}, L_6^{\max}) = L_3^{\max} = 0$. La structure d'intervalles sur M_3 est donc cohérente.

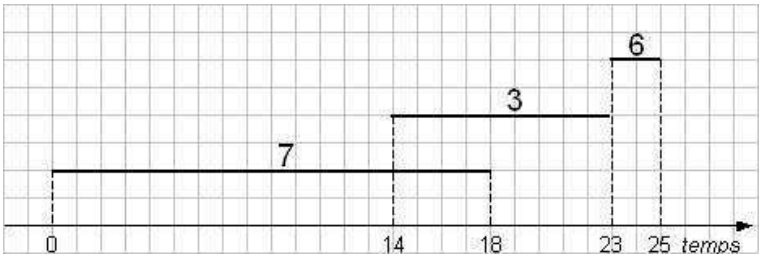


FIG. 8.15: Structure d'intervalles du problème sur M_3

L'algorithme TCJ-2 s'arrête alors et l'ensemble d'ordonnancements trouvé contient 2 solutions :

- sur $M_1 : 1 \prec 5 \prec 9$, sur $M_2 : 4 \prec 2 \prec 8$, sur $M_3 : 7 \prec 3 \prec 6$.

- sur $M_1 : 1 \prec 9 \prec 5$, sur $M_2 : 4 \prec 2 \prec 8$, sur $M_3 : 7 \prec 3 \prec 6$.

Parmi ces deux solutions, la première est l'ordonnancement optimal initial. La deuxième solution produit $C_{\max} = 33$.

L'avantage principal de TCJ-2 par rapport à TCJ-1 est la garantie de la performance au mieux de l'ensemble d'ordonnancements trouvé, qui est dans les pires de cas égale à la performance de l'ordonnancement initial. Un autre intérêt de TCJ-2 est sa capacité d'adaptation pour trouver les solutions ε-optimales comme l'algorithme TCOR-2 dans le cas à une machine.

En effet, si on accepte sur M_2 la valeur $\max(L_i^{\max}) = 3$, on peut bénéficier de deux ordonnancements en plus, correspondant à la séquence $4 \prec 8 \prec 2$ sur M_2. Et ces deux nouvelles solutions produisent le même makespan $C_{\max} = 28$. Notons bien que la dégradation du makespan de ces deux nouvelles solutions par rapport à l'optimalité (28-25=3) correspond bien à l'augmentation de $\max(L_i^{\max})$ sur M_2 (de 0 à 3).

CONCLUSION ET PERSPECTIVES

Plusieurs études prospectives consacrées à l'ordonnancement ont montré l'écart important entre la recherche théorique et les besoins pratiques, en pointant notamment du doigt le fait que, les méthodes d'ordonnancement étant pour la plupart déterministes, ne prennent pas suffisamment en compte le caractère incertain de leur environnement de mise en œuvre. De ce fait, une solution déterministe, bonne à un instant donné, peut s'avérer rapidement obsolète, suite à une perturbation même mineure. Pour pallier ce problème, des méthodes d'ordonnancement, dites robustes, ont été développées, ayant la capacité de "résister" aux perturbations de l'environnement tout en assurant une stabilité de la performance. Le travail présenté dans ce mémoire s'inscrit dans ce champ de recherche.

Une nouvelle façon de classifier les approches d'ordonnancement robuste a été proposée dans la première partie de ce travail, privilégiant le point de vue automaticien, n'occultant pas le caractère réactif qu'un processus d'ordonnancement se doit d'intégrer. Selon que les approches manipulent ou non un ordonnancement de référence, et qu'elles tentent d'anticiper ou non les perturbations, trois catégories ont été distinguées : les approches réactives, les approches prédictives-réactives et les approches proactives-réactives. En adoptant ce point de vue, un panorama relativement complet des méthodes d'ordonnancement robuste a été proposé. Celui-ci illustre en particulier l'intérêt des méthodes proactives-réactives, vis-à-vis des autres, ces dernières tentant en effet d'anticiper les perturbations de sorte qu'un ordonnancement robuste ou flexible, présentant une performance stable relativement à un ensemble de perturbations possibles, puisse être construit.

Certaines méthodes d'ordonnancement proactives-réactives proposent d'ajouter une flexibilité soit temporelle, soit séquentielle, à une solution d'ordonnancement afin de faire face aux incertitudes. En effet, la flexibilité peut être utilisée pour caractériser hors-ligne un ensemble de solutions, exploitables en ligne, en choisissant la solution la plus adaptée étant donné un scénario de réalisation réel de l'ordonnancement. Dans ce cadre, la nécessité d'un compromis flexibilité / performance a été mise en évidence, augmenter la flexibilité d'un ensemble de solutions se traduisant généralement par une perte de performance, et vice-versa. Notre travail s'est principalement focalisé sur les méthodes produisant de la flexibilité séquentielle, dans une optique de recherche de compromis, ce type de flexibilité induisant en effet de la flexibilité temporelle et semblant avantageux pour traiter un panel très large de perturbations.

Plusieurs méthodes d'ordonnancement produisent de la flexibilité séquentielle en exploitant le concept de groupes de tâches permutables. Ce concept est intéressant à plusieurs titres. Tout d'abord, il permet de caractériser un ensemble de solutions très vaste, en associant à chaque ressource une séquence de groupes telle que toutes les tâches d'un groupe sont totalement permutables. D'autre part, il définit un ordre partiel dont la structure permet de déterminer, en temps polynomial, la performance de la pire des solutions contenue dans l'ensemble caractérisé, vis-à-vis d'un critère minmax. Les solutions caractérisées étant dénombrables, il est également possible de rechercher un compromis flexibilité / performance satisfaisant vis-à-vis d'un décideur.

Toutefois, le concept de groupes de tâches présente quelques limites. Tout d'abord, seules des tâches placées de façon contiguë sur la même ressource peuvent être permutées, ce qui limite quelque peu la flexibilité séquentielle pouvant potentiellement être générée. D'autre part, la constitution de groupes de tâches ne tire pas partie d'une modélisation a priori des incertitudes. Autrement dit, si la flexibilité séquentielle produite grâce aux groupes permet de faire face à un vaste ensemble de perturbations, il n'est pas possible de garantir a priori la robustesse de l'ensemble de solutions généré vis-à-vis d'un ensemble de scénarios potentiels de réalisation d'un ordonnancement. Ce sont ces deux constats qui ont motivé nos recherches.

Dans un premier temps, nous nous sommes focalisés sur l'étude du problème à une machine $1|r_j|L_{\max}$. Pour ce problème, nous avons montré l'intérêt d'un théorème de dominance démontré dans les années quatre-vingt. L'avantage principal de ce théorème réside dans sa capacité à caractériser un ensemble dominant de séquences, sans les énumérer, grâce à un ordre partiel construit sur un corps d'hypothèses restreint, ne considérant exclusivement que les relations d'ordre entre les dates de début au plus tôt et de fin au plus tard des travaux. Un intérêt de cet ordre partiel est qu'il offre potentiellement davantage de flexibilité séquentielle que l'ordre partiel relatif au concept de groupe de tâches permutables. De plus, nous avons montré qu'il était possible d'évaluer les performances au mieux et au pire de l'ensemble caractérisé en temps polynomial. D'autre part, la cardinalité de l'ensemble pouvant facilement être calculée, nous avons montré que cet ordre partiel pouvait être utilisé dans le cadre d'une recherche de compromis flexibilité / performance. Enfin, nous avons montré qu'il était possible de caractériser un modèle par intervalles des incertitudes tel qu'une performance robuste pour tous les scénarios de réalisation que ce modèle caractérise, puisse être garantie. Même si ce dernier résultat n'est valide que dans le cas où les intervalles de réalisation des dates de début au plus tôt et de fin au plus tard des travaux sont disjoints, il nous semble particulièrement intéressant dans le contexte de l'ordonnancement robuste.

Sur la base des résultats précédents, une procédure par séparation et évaluation progressive a été proposée, visant à caractériser un large ensemble de solutions optimales pour le problème $1|r_j|L_{\max}$. Cet ensemble est caractérisé par l'énumération de structures d'in-

tervalles, telles que l'application du théorème de dominance à ces structures ne caractérise que des solutions optimales. L'ensemble de ces solutions optimales est strictement inclus dans l'ensemble dominant caractérisé par la structure d'intervalles initiale du problème. Les expérimentations ont confirmé l'efficacité de la procédure qui fournit des ensembles extrêmement vastes de solutions, pour des problèmes comptant jusqu'à 500 travaux, en un temps relativement court.

Nous avons ensuite montré comment la procédure précédente pouvait être utilisée dans le cadre d'une aide à la décision ayant pour but de guider un décideur dans sa recherche d'un compromis flexibilité / performance satisfaisant. L'idée consiste cette fois à ne plus contraindre la procédure à ne fournir que des solution optimales, en se contentant des solutions ε-optimales, le coefficient ε permettant au décideur de contrôler la détérioration de performance. Nous avons alors montré que si on accepte une détérioration de la performance, on peut augmenter considérablement la flexibilité, ceci d'autant plus que le nombre de travaux est important. Une autre approche d'aide à la décision, fondée sur une interaction homme-machine utilisant le diagramme de retards, a également été proposée. Cette approche permet au décideur de contrôler la performance de l'ensemble dominant vis-à-vis des retards de chaque travail, tout en maintenant une flexibilité séquentielle satisfaisante.

Forts des résultats précédents, nous nous sommes alors focalisés sur l'étude de problèmes d'ordonnancement à plusieurs machines. Nous avons tout d'abord étudié le problème flow shop de permutation à deux machines $F2|prmu|C_{\max}$ pour lequel nous avons proposé un ordre partiel suffisant d'optimalité, défini à partir des relations d'ordre entre les durées opératoires des tâches. Un résultat intéressant est que cet ordre partiel reste optimal quelles que soient les valeurs explicites des durées opératoires, dans la mesure où les relations d'ordre restent conservées. De plus, cet ordre partiel autorise également la prise en compte d'intervalles de réalisation des durées opératoires, dans la mesure où ces intervalles sont disjoints. Une performance peut alors être garantie a priori vis-à-vis de tous les scénarios de réalisation caractérisés par le modèle. Sur la base de ce dernier résultat, un ordre partiel suffisant d'optimalité pour le problème de job shop à deux machines a également été proposé.

Le cas des problèmes job shop a été abordé dans un dernier temps. Deux algorithmes visant à trouver un ensemble d'ordonnancements dont les performances au mieux et au pire sont calculables ont été décrits. Le premier algorithme est une heuristique qui vise à construire un ensemble de solutions en décomposant le problème initial en un ensemble de problèmes à une machine pour lesquels un ordre partiel dominant peut être défini. Des ajustements portant sur les structures d'intervalles relatives à chaque machine sont alors opérés, dans le but d'obtenir un ensemble de solutions cohérentes entre elles, c'est-à-dire telles que les dates de fin au pire des tâches soient compatibles avec les dates de début au mieux des tâches leur succédant selon la gamme opératoire. Le deuxième algorithme, basé sur la procédure par séparation et évaluation présentée précédemment, utilise une solution initiale pour déterminer les structures d'intervalles sur chaque machine. L'algo-

rithme resserre alors chaque structure d'intervalles de sorte que les tâches ne soient pas en retard, c'est-à-dire de sorte que la longueur du chemin critique initial reste inchangée. L'implémentation et l'évaluation de ces algorithmes n'ont pas encore été effectuées, ceci constituant une perspective immédiate de ce travail.

Dans le cadre de la recherche d'ordonnancements flexibles, l'utilisation d'ordres partiels dominants ou suffisants vis-à-vis de l'optimalité constitue, ainsi que nous l'avons montré, une voie de recherche intéressante, encore relativement peu explorée. Une première perspective envisageable pour ce travail serait alors de tenter de définir de nouveaux ordres partiels dominants, se basant sur des corps d'hypothèses restreints plus riches que ceux considérés dans ce mémoire de thèse. En effet, plus le corps d'hypothèses est restreint et plus l'ordre partiel est insensible aux variations des données du problème, mais moins les propriétés de dominance sont fortes. Il serait par exemple intéressant de considérer, pour le problème à une machine, un corps d'hypothèses intégrant les relations d'ordre existant entre les durées opératoires des travaux. Pour les problèmes flow shop, il serait également intéressant de considérer des corps d'hypothèses intégrant des relations d'ordre entre les dates de début au plus tôt des travaux, et entre les durées opératoires, afin de pouvoir traiter des problèmes plus complexes de type $Fm|r_i|C_{\max}$.

Une autre perspective de travail, pouvant éventuellement tirer partie de la précédente, consisterait à déterminer des conditions de dominance nécessaires qui puissent être appliquées de façon générique à tout problème d'ordonnancement. On rejoint ici la logique des approches par contraintes qui visent à caractériser, par des conditions nécessaires d'admissibilité, un ensemble de solutions possédant la meilleure borne inférieure possible. En effet, disposer de conditions nécessaires de dominance permettrait de caractériser un ensemble de solutions dont on peut espérer qu'il possède une meilleure borne inférieure, puisqu'une condition nécessaire de dominance élimine davantage de solutions qu'une condition nécessaire d'admissibilité.

Enfin, un dernier axe de recherche concerne l'exploitation d'ordres partiels dominants au sein d'une procédure d'ordonnancement en ligne. Il s'agit de montrer comment exploiter au mieux la flexibilité séquentielle fournie hors ligne, et comment maintenir l'ensemble de solutions caractérisées de sorte à éliminer progressivement les solutions devenues obsolètes. Un autre sujet d'intérêt concerne la "réparation" en ligne d'un ensemble de séquences lorsque, suite à une perturbation trop forte, plus aucune solution n'est valide.

Bibliographie

[Adams *et al.* 88] J. Adams, E. Balas & D. Zawack. *The shifting bottleneck procedure for job shop scheduling.* Management Science, vol. 34, no. 3, pages 391–401, 1988.

[Akturk & Gorgulu 99] M.S. Akturk & E. Gorgulu. *Match-up scheduling under a machine breakdown.* European Journal of Operational Research, vol. 112, pages 81–97, 1999.

[Allen 91] J.F. Allen. *Time and time again : The many ways to represent time.* International Journal of Intelligent Systems, vol. 6, no. 4, pages 341–355, 1991.

[Aloulou *et al.* 02] M.A. Aloulou, M-C. Portmann & A. Vignier. *Predictive-Reactive Scheduling for the Single Machine Problem.* Dans Eighth International Workshop on Project Management and Scheduling, pages 39–42, Valencia, Spain, 2002.

[Aloulou 02] M.A. Aloulou. *Structure flexible d'ordonnancements à performances contrôlées pour le pilotage d'atelier en présence des perturbations.* Thèse de Doctorat, Institut National Polytechnique de Lorraine, Nancy, France, Décembre 2002.

[Artigues & Roubellat 00] C. Artigues & F. Roubellat. *A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes.* European Journal of Operational Research, vol. 127, pages 297–216, 2000.

[Artigues & Roubellat 01] C. Artigues & F. Roubellat. *Ordonnancement d'atelier en temps réel.* Dans Ordonnancement de la production sous la direction de P. Lopez et F. Roubellat, pages 393–425. Hermès, 2001.

[Artigues *et al.* 02] C. Artigues, C. Briand, M-C. Portmann & F. Roubellat. *Pilotage d'atelier basé sur un ordonnancement flexible.* Dans Méthodes du pilotage des systèmes de production sous la direction de P. Pujo et J-P. Kiefer, pages 61–97. Hermès, 2002.

[Artigues 97] C. Artigues. *Ordonnancement en temps réel d'ateliers avec temps de préparation des ressources.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 1997.

[Aytug *et al.* 03] H. Aytug, M.A. Lawley, K. McKay, S. Mohan & R. Uzsoy. *Executing production schedules in the face of uncertainties : A review and some future directions.* European Journal of Operational Research, vol. 161, no. 1, pages 86–110, 2003.

[Baptiste *et al.* 01] P. Baptiste, C. Le Pape & W. Nuijten. Constraint-based scheduling. Kluwer Academic Publishers, Boston/Dordrecht/London, 2001.

[Baptiste 96] P. Baptiste. *Sub-optimal groups of sequences in the classical flow shop F2//Cmax.* Dans Fifth International Workshop on Project Management and Scheduling, pages 27–30, Poznan, Poland, 1996.

[Bartush *et al.* 88] M. Bartush, R.H. Möhring & F.J. Radermacher. *Scheduling project networks with resource constraints and time windows.* Annals of Operations Research, vol. 16, pages 201–240, 1988.

[Belman *et al.* 82] R. Belman, A.O. Esogbue & I. Nabeshima. Mathematical aspects of scheduling and applications. Pergamon press, Oxford, 1982.

[Benoît & Billaut 00] M. Benoît & J-C. Billaut. *Characterization of the optimal solutions of the F2//Cmax scheduling problem.* Dans Second Conference on Management and Control of Production and Logistic (MCPL 2000), Grenoble, France, 2000.

[Bertsimas & Sim 04] D. Bertsimas & M. Sim. *The price of robustness.* Operations Research, vol. 52, no. 1, pages 35–53, 2004.

[Bidot 03] J. Bidot. *Planification et ordonnancement sous incertitudes. Application à la gestion de projet.* Dans 4ème Congrès des doctorants de l'Ecole Doctorale Systèmes, Toulouse, France, 2003.

[Billaut & Lopez 98] J-C. Billaut & P. Lopez. *Enumeration of all optimal sequences in the two-machine flowshop.* Dans Computational Engineering in Systems Application (CESA'98), Symposium on Industrial and Manufacturing Systems, IEEE-SMC/IMACS, pages 378–382, Nabeul-Hammamet, Tunisie, 1998.

[Billaut *et al.* 05] J-C. Billaut, A. Moukrim & E. Sanlaville. *Introduction à la flexibilité et à la robustesse en ordonnancement.* Dans Flexibilité et Robustesse en Ordonnancement sous la direction de J-C. Billaut et A. Moukrim et E. Sanlaville, pages 13–32. Hèrmes, 2005.

[Billaut 93] J-C. Billaut. *Prise en compte de ressources multiples et des temps de préparation dans les problèmes d'ordonnancement en temps réel.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 1993.

[Billaut 99] J-C. Billaut. *Recherche Opérationnelle et aide à la décision pour les problèmes d'ordonnancement.* Habilitation à diriger des recherche, Laboratoire d'Informatique, E3i, Université François Rabelais, Tours, France, 1999.

[Blazewicz *et al.* 96] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt & J. Weglarz. Scheduling in computer and manufacturing processes. Springer Verlag, 1996.

[Briand & La 03] C. Briand & H.T. La. *Une procédure par séparation et évaluation progressive pour l'ordonnancement robuste de problèmes à une machine.* Dans Recherche Informatique Vietnam & Francophonie 2003 (RIVF2003), pages 11–16, Hanoi, Vietnam, 2003.

[Briand *et al.* 03a] C. Briand, H.T. La & J. Erschler. *Ordonnancement de problèmes à une machine : une aide à la décision pour un compromis flexibilité vs performance.* Dans 5ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'03), pages 146–147, Avignon, France, 2003.

[Briand *et al.* 03b] C. Briand, H.T. La & J. Erschler. *Une approche pour l'ordonnancement robuste de tâches sur une machine.* Dans 4ème Conférence Francophone de MOdélisation et SIMulation (MOSIM'03), pages 205–211, Toulouse, France, 2003.

[Briand *et al.* 04] C. Briand, H.T. La & J. Erschler. *Robust scheduling for the two-machine flowshop problems.* Dans 9th International Workshop on Project Management and Scheduling (PMS'04), pages 371–374, Nancy, France, 2004.

[Briand *et al.* 05a] C. Briand, M-J. Huguet, H.T. La & P. Lopez. *Approches par contraintes pour l'ordonnancement robuste.* Dans Flexibilité et Robustesse en Ordonnancement sous la direction de J-C. Billaut et A. Moukrim et E. Sanlaville, pages 181–208. Hermès, 2005.

[Briand *et al.* 05b] C. Briand, H.T. La & J. Erschler. *A new sufficient condition of optimality for the two-machine flowshop problems.* European Journal of Operational Research, à paraître, 2005.

[Brucker *et al.* 99] P. Brucker, A. Drexl, R. Möhring, K. Neumann & E. Pesch. *Resource-constrained project scheduling : Notation, classification, models, and methods.* European Journal of Operational Research, vol. 112, pages 3–41, 1999.

[Burns *et al.* 99] A. Burns, S. Punnekkat, L. Strigini & D.R. Wright. *Probabilistic Scheduling Guarantees for Fault-Tolerant Real-Time Systems.* Dans Conference on Dependable Computing for Critical Application (DCCA'99), pages 361–378, California, USA, 1999.

[Byeon *et al.* 93] E.S. Byeon, S.D. Wu & R.H. Storer. *Decomposition heuristics for robust job-shop scheduling.* 93t-008, Departement of Industrial Engineering, Lehigh University, Bethlehem, USA, 1993.

[Carlier & Chrétienne 88] J. Carlier & P. Chrétienne. Problèmes d'ordonnancement : modélisation/complexité/algorithmes. Masson, 1988.

[Carlier & Pinson 89] J. Carlier & E. Pinson. *An algorithm for solving the job-shop problem.* Management Science, vol. 35, no. 2, pages 164–176, 1989.

[Carlier & Pinson 94] J. Carlier & E. Pinson. *Adjustment of heads and tails for the job-shop problem.* European Journal of Operational Research, vol. 78, pages 146–161, 1994.

[Carlier & Pinson 04] J. Carlier & E. Pinson. *Jackson pseudo-preemptive schedule and cumulative scheduling problems.* Discrete Applied Mathematics, vol. 145, pages 80–94, 2004.

[Carlier 75] J. Carlier. *Ordonnancement à contraintes disjonctives.* Thèse de troisième cycle, Université Paris VI, 1975.

[Carlier 82] J. Carlier. *The one-machine sequencing problem.* European Journal of Operational Research, vol. 11, pages 42–47, 1982.

[Chen & Yih 96] C.C. Chen & Y. Yih. *Identifying attributes for knowledge-based development in dynamic scheduling environments.* International Journal of Production Research, vol. 34, no. 6, pages 1739–1755, 1996.

[Cheng *et al.* 02] J. Cheng, G. Steiner & P. Stephenson. *Fast algorithms to minimize makespan or maximum lateness in the two-machine flow shop with release times.* Journal of Scheduling, vol. 5, pages 71–92, 2002.

[Chiang & Fox 90] W-Y. Chiang & M.S. Fox. *Protection againts uncertainty in a deterministic schedule.* Dans Fourth International Conference on Expert Systems in Production and Operations Management, South California, USA, 1990.

[Chong *et al.* 03] C.S. Chong, A.I. Sivakuma & R. Gay. *Simulation-based scheduling for dynamic discrete manufacturing.* Dans Winter Simulation Conference, pages 1465–1473, New Orleans, USA, 2003.

[Coffman 76] E.G. Coffman. Computer and job-shop scheduling theory. John Wiley and Sons, 1976.

[Conway *et al.* 67] R.W. Conway, W.L. Maxwel & L.W. Miller. Theory of scheduling. Addison-Wesley, Reading, Massachusetts, 1967.

[Couzinet-Mercé 79] C. Couzinet-Mercé. *Étude de l'existence de solutions pour certains problèmes d'ordonnancement.* Thèse de Doctorat, Université Paul Sabatier, Toulouse, France, 1979.

[Daniels & Carrillo 97] R.L. Daniels & J.E. Carrillo. *beta-Robust scheduling for single-machine systems with uncertain processing times.* IIE Transactions, vol. 29, pages 977–985, 1997.

[Daniels & Kouvelis 95] R.L. Daniels & P. Kouvelis. *Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production.* Management Science, vol. 41, no. 2, pages 363–376, 1995.

[Davenport & Beck 00] A.J. Davenport & J.C. Beck. *A survey of techniques for scheduling with uncertainty*, http ://www.eil.utoronto.ca/chris/chris.papers.html, 2000.

[Davenport *et al.* 01] A.J. Davenport, C. Gefflot & J.C. Beck. *Slack-based Techniques for Robust Schedules.* Dans Six European Conference on Planning (ECP-2001), Toledo, Spain, 2001.

[Dechter *et al.* 91] R. Dechter, I. Meiri & J. Pearl. *Temporal constraint networks.* Artificial Intelligence, vol. 49, pages 61–95, 1991.

[Demmou 77] R. Demmou. *Etude de familles remarquables d'ordonnancements en vue d'une aide à la décision.* Thèse de troisième cycle, Université Paul Sabatier, Toulouse, France, 1977.

[Dorndorf *et al.* 00] U. Dorndorf, E. Pesch & T. Phan Huy. *Constraint propagation techniques for the disjunctive scheduling problem.* Artificial Intelligence, vol. 122, pages 189–240, 2000.

[Drummond *et al.* 94] M. Drummond, J. Bresina & K. Swanson. *Just-In-Case Scheduling.* Dans Twelfth National Conference on Artificial Intelligence (AAAI-94), Seattle, USA, 1994.

[Durand & Trentesaux 00] S. Durand & D. Trentesaux. *Des indices de robustesse pour la méthode prudente et pour la fonction de choix de Borda.* Journal of Decision Systems, vol. 9, pages 269–288, 2000.

[Erschler & Terssac 88] J. Erschler & G. De Terssac. *Flexibilité et rôle de l'opérateur humain dans l'automatisation intégrée de production.* Rapport laas no 88137, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, France, 1988.

[Erschler *et al.* 83] J. Erschler, G. Fontan, C. Merce & F. Roubellat. *A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates.* Operations Research, vol. 31, no. 1, pages 114–127, 1983.

[Erschler *et al.* 85] J. Erschler, G. Fontan & C. Merce. *Un nouveau concept de dominance pour l'ordonnancement des travaux sur une machine.* RAIRO Recherche Opérationnelle/Operations Research, vol. 19, no. 1, pages 1–13, 1985.

[Erschler 76] J. Erschler. *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement.* Thèse de Doctorat d'Etat, Université Paul Sabatier, Toulouse, 1976.

[Esquirol & Lopez 99] P. Esquirol & P. Lopez. L'ordonnancement. Economica, 1999.

[Esquirol & Lopez 02] P. Esquirol & P. Lopez. Structures temporelles pour le problème d'ordonnancement à une machine. 2002.

[Esquirol *et al.* 92] P. Esquirol, M.-J. Huguet & P. Lopez. *Time bounds on node graph for scheduling.* Dans 3^{rd} International Workshop on Project Management and Scheduling, Como, Italy, 1992.

[Esquirol *et al.* 95] P. Esquirol, M.-J. Huguet & P. Lopez. *Modeling and managing disjunctions in scheduling problems.* Journal of Intelligent Manufacturing, vol. 6, pages 133–144, 1995.

[Esquirol *et al.* 01] P. Esquirol, P. Lopez & M.-J. Huguet. *Propagation de contraintes en ordonnancement.* Dans P. Lopez & F. Roubellat, editeurs, Ordonnancement de la production, pages 131–167. Hermès Science Publications, Paris, 2001.

[Esswein & Billaut 02] C. Esswein & J-C. Billaut. *Trade-off between flexibility and maximum completion time in the two-machine flowshop scheduling problem.* Dans International Symposium on Combinatorial Optimisation, Paris, France, 2002.

[Esswein *et al.* 05] C. Esswein, J.-C. Billaut & C. Artigues. *Une approche multi-critères pour un apport de flexibilité séquentielle.* Dans Flexibilité et robustesse en ordonnancement sous la direction de J-C. Billaut et A. Moukrim et E. Sanlaville, pages 209–232. Hermès, 2005.

[Esswein 03] C. Esswein. *Un apport de flexibilité séquentielle pour l'ordonnancement robuste.* Thèse de Doctorat, Université François Rabelais, Tours, France, Décembre 2003.

[Fisher & Thompson 63] H. Fisher & G. L. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules.* Dans J. F. Muth & G. L. Thompson, editeurs, Industrial Scheduling, pages 225–251. Prentice Hall, Englewood Cliffs, NJ, 1963.

[Fontan 80] G. Fontan. *Notion de dominance et son application à l'étude de certains problèmes d'ordonnancement.* Thèse de doctorat d'etat, Université Paul Sabatier, Toulouse, France, 1980.

[Fortemps 97] P. Fortemps. *Job-shop scheduling with imprecise durations : a fuzzy approach.* IEEE Transactions on Fuzzy Systems, vol. 5, pages 557–569, 1997.

[Gao *et al.* 95] H. Gao, M.S. Fox, W-Y. Chiang & S. Hikita. Building robust schedules - an empirical study of single machine scheduling with uncertainty. 1995.

[Ghosh *et al.* 95] S. Ghosh, R. Melhem & D. Mossé. *Enhancing Real-Time Schedules to Tolerate Transient Faults.* Dans 16th IEEE Real-Time Systems Symposium (RTSS'95), Pisa, Italy, 1995.

[Giard 91] V. Giard. Gestion de projet. Economica, Paris, 1991.

[Goldratt 97] E. Goldratt. Critical chain. Great Barrington : The North River Press, 1997.

[GOThA 02] GOThA. *Flexibilité et Robustesse en Ordonnancement.* Le bulletin de la ROADEF, vol. 8, pages 10–12, 2002.

[Graham *et al.* 79] R.L. Graham, E.L. Lawler, J.K. Lenstra & A.H.G. Rinnooy Kan. *Optimization and approximation in deterministic sequencing and scheduling : a survey.* Annals of Discrete Mathematics, vol. 5, pages 287–326, 1979.

[Hall & Posner 04] N.G. Hall & M.E. Posner. *Sensitivity analysis for scheduling problems.* Journal of Scheduling, vol. 7, pages 49–83, 2004.

[Hariri & Potts 83] A.M. Hariri & C.N. Potts. *An algorithm for single machine sequencing with release dates to minimize total weighted completion time.* Discrete Applied Mathematics, vol. 5, pages 99–109, 1983.

[Herroelen & Leus 01] W. Herroelen & R. Leus. *On the merits and pitfalls of critical chain scheduling.* Journal of Operations Management, vol. 19, pages 559–577, 2001.

[Herroelen & Leus 02] W. Herroelen & R. Leus. *Project scheduling under uncertainty. Survey and Research Potentials.* Dans Eighth International Workshop on Project Management and Scheduling (PMS'2002), Valencia, Spain, 2002.

[Herroelen & Leus 04a] W. Herroelen & R. Leus. *The construction of stable project baseline schedules.* European Journal of Operational Research, vol. 156, pages 550–565, 2004.

[Herroelen & Leus 04b] W. Herroelen & R. Leus. *Robust and reactive project scheduling : a review and classification of procedures.* International Journal of Production Research, vol. 42, no. 8, pages 1599–1620, 2004.

[Herroelen *et al.* 99] W. Herroelen, E. Demeulemeester & B. De Reyck. *A classification scheme for project scheduling.* Dans Project Scheduling - Recent Models, Algorithms and Applications sous la direction de J. Weglarz, pages 1–26. Kluwer's International Series, 1999.

[Holhaus & Rajendran 97] O. Holhaus & C. Rajendran. *Efficient dispatching rules for scheduling in a job shop.* International Journal of Production Economics, vol. 48, pages 87–105, 1997.

[Holhaus & Rajendran 00] O. Holhaus & C. Rajendran. *Efficient jobshop dispatching rules : further developments.* Production Planning & Control, vol. 11, no. 2, pages 171–178, 2000.

[Jackson 56] J.R. Jackson. *An extension of Johnson's results on job lot scheduling.* Naval Research Logistics Quaterly, vol. 1, pages 61–68, 1956.

[Jensen 01] M.T. Jensen. *Robust and Flexible Scheduling with Evolutionary Computation.* Thèse de Doctorat, Department of Computer Science, University of Aarhus, Danemark, 2001.

[Johnson 54] S.M. Johnson. *Optimal two and three stage production schedules with set-up times included.* Naval research Logistics Quaterly, vol. 1, pages 61–68, 1954.

[Kouvelis *et al.* 00] P. Kouvelis, R.L. Daniels & G. Vairaktarakis. *Robust scheduling of a two-machine flow shop with uncertain processing times.* IIE Transactions, vol. 32, pages 421–432, 2000.

[La *et al.* 05] H.T. La, J.L. Santamaria & C. Briand. *Une aide à la décision pour l'ordonnancement robuste à une machine : un compromis flexibilité/performance, à paraître.* Dans 6ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'05), Tours, France, 2005.

[Le Gall 89] A. Le Gall. *Un système interactif d'aide à la décision pour l'ordonnancement et le pilotage en temps réel d'atelier.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 1989.

[Le Pape 91] C. Le Pape. *Constraint propagation in Planning and Scheduling.* Rapport technique, Robotics Laboratory, Department of Computer Science, Stanford, 1991.

[Lenstra *et al.* 77] J.K. Lenstra, A.H.G. Rinnooy Kan & P. Brucker. *Complexity of machine scheduling problems.* Annals of Discrete Mathematics, vol. 1, pages 343–362, 1977.

[Leon *et al.* 94] V.J. Leon, S.D. Wu & R.H. Storer. *Robustness measures and robust scheduling for job shops.* IIE Transactions, vol. 26, no. 5, pages 32–43, 1994.

[Levy 96] M-L. Levy. *Méthodes par décomposition temporelle et problèmes d'ordonnancement.* Thèse de Doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, Mars 1996.

[Lim & Zhang 02] M.K. Lim & Z. Zhang. *Iterative multi-agent bidding and coordination based on genetic algorithm.* Dans 3 Complex Systems, and E-Businesses, pages 682–689, Erfurt, Germany, 2002.

[Lopez 91] P. Lopez. *Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources.* Thèse de Doctorat, Université Paul Sabatier, Toulouse, France, 1991.

[MacCathy & Liu 93] B.L. MacCathy & J. Liu. *Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling.* International Journal of Production Research, vol. 31, no. 1, pages 59–79, 1993.

[Mackay *et al.* 89] K.N. Mackay, J.A. Buzacott & F.R. Safayeni. *The scheduler's knowledge of uncertainty : The missing link.* Dans J. Browne, editeur, Knowledge based production management systems. Elsevier Science Publishers, 1989.

[McMahon & Florian 75] G.B. McMahon & M. Florian. *On scheduling with ready times and due dates to minimize maximum lateness.* Operations Research, vol. 23, pages 475–482, 1975.

[Mehta & Uzsoy 98] S.V. Mehta & R.H. Uzsoy. *Predictable Scheduling of a Job Shop Subject to Breakdowns.* IEEE Transactions on Robotics and Automation, vol. 14, no. 3, pages 365–378, 1998.

[Mehta & Uzsoy 99] S.V. Mehta & R.H. Uzsoy. *Predictive scheduling of a single machine subject to breakdowns.* International Journal of Computer Integrated Manufacturing, vol. 12, no. 1, pages 15–38, 1999.

[Morris *et al.* 01] P. Morris, N. Muscettola & T. Vidal. *Dynamic Control Of Plans With Temporal Uncertainty.* Dans International Joint Conference on A.I (IJCAI-01), Seattle, USA, 2001.

[Moukrim *et al.* 03] A. Moukrim, E. Sanlaville & F. Guinand. *Parallel machine scheduling with uncertain communication delays.* RAIRO Operations Reseach, vol. 37, pages 1–16, 2003.

[Nabeshima 73] I. Nabeshima. *Algorithms and reliable heuristics programs for multi-projects scheduling with resource constraints and related parallel scheduling.* Thèse de Doctorat, University of Electrocommunication, Chofu, Tokyo (Japan), 1973.

[Nuijten 94] W.P.M. Nuijten. *Time and resource constrained scheduling - A constraint satisfaction approach.* Thèse de Doctorat, Eindhoven University of Technology, 1994.

[Penz *et al.* 01] B. Penz, C. Rapine & D. Tristam. *Sensitivity analysis od Scheduling Algorithms.* European Journal of Operational Research, vol. 134, pages 606–615, 2001.

[Phan Huy 00] T. Phan Huy. Constraint propagation in flexible manufacturing. LNEMS, volume 492, Springer Verlag, M. Beckmann, H. P. Künzi (éditeurs), 2000.

[Pinedo 95] M. Pinedo. Scheduling : Theory, algorithms and systems. Prentice Hall, 1995.

[Priore *et al.* 98] P. Priore, D.D. Garcia & I.F. Quesada. *Manufacturing systems scheduling through machine learning.* Dans Neural Computation, NC'98, pages 914–917, Vienna, Austria, 1998.

[Priore *et al.* 01] P. Priore, De la Fuente, A. Gomez & J. Puente. *A review of machine learning in dynamic scheduling of flexible manufacturing systems.* Dans AI EDAM Artificial Intelligence for Engineering Design Analysis and Manufacturing, pages 251–263, Cambrige University Press, USA, 2001.

[Pujo & Brun-Picard 02] P. Pujo & D. Brun-Picard. *Pilotage sans plan prévisionnel ni ordonnancement préalable.* Dans Méthodes du pilotage des systèmes de production sous la direction de P. Pujo et J-P. Kiefer, pages 129–162. Hèrmes, 2002.

[Rajendran & Holhaus 99] C. Rajendran & O. Holhaus. *A comparative study of dispatching rules in dynamic flowshops and jobshops.* European Journal of Operational Research, vol. 116, pages 156–170, 99.

[Rossi 02] A. Rossi. *Ordonnancement en mileu incertain, mise en œuvre d'une démarche robuste.* Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 2002.

[Roy 70] B. Roy. Algèbre moderne et théorie des graphes, volume 2. Dunod, Paris, 1970.

[Roy 02] B. Roy. *Robustesse de quoi et vis-à-vis de quoi mais aussi robustesse pourquoi en aide à la décision ?* Newsletter of the European Working Group - Multicriteria Aid for Decisions, vol. 3, no. 6, pages 1–6, 2002.

[Sabuncuoglu & Bayiz 00] I. Sabuncuoglu & M. Bayiz. *Analyse of reactive scheduling problems in a job shop environment.* European Journal of Operational Research, vol. 126, pages 567–586, 2000.

[Sakkout *et al.* 97] El Sakkout, E.T. Richards & M.G. Wallace. *Unimodular Probing for Minimal Perturbance in Dynamic Resource Feasibility Problems.* Dans CP97 Workshop on the Theory and Practice of Dynamic Constraint Satisfaction, Linz, Austria, 1997.

[Sakkout *et al.* 98] El Sakkout, M.G. Wallace & E.T. Richards. *Minimal Perturbation in Dynamic Scheduling.* Dans 13th European Conference on Artificial Intelligence (ECAI-98), Brighton, UK, 1998.

[Sevaux & Sörensen 04] M. Sevaux & K. Sörensen. *A genetic algorithm for robust schedules in a just-in-time environment.* 4OR - Quaterly journal of the Belgian, French and Italian Operations Research Societies, vol. 2, no. 2, pages 129–147, 2004.

[Smith 95] S.F. Smith. Intelligent scheduling systems, chapitre Reactive scheduling systems. Kluwer Press, 1995.

[Snoek 01] M. Snoek. *Anticipation Optimization in Dynamic Job Shops.* Dans Genetic and Evolutionary Computation Conference 2001 (GECCO-2001), pages 43–46, San Francisco, USA, 2001.

[Sotskov 98] Y.N. Sotskov. *On the calculation of the stability radius of an optimal or an approximate schedule.* Annals of Operational Research, vol. 83, pages 213–225, 1998.

[Tadei *et al.* 98] R. Tadei, JND. Gupta, F. Della Croce & M. Cortesi. *Minimising makespan in the two-machine flow-shop with release times.* Journal of Operational Research Society, vol. 49, pages 77–85, 1998.

[Tavares *et al.* 98] L.V. Tavares, J.A.A. Ferreira & J.S. Coelho. *On the optimal management of project risk.* European Journal of Operational Research, vol. 107, pages 451–469, 1998.

[Thomas 80] V. Thomas. *Aide à la décision pour l'ordonnancement d'atelier en temps réel.* Thèse de troisième cycle, Université Paul Sabatier, Toulouse, France, 1980.

[T'kindt & Billaut 02] V. T'kindt & J-C. Billaut. Multicriteria scheduling. Springer-Verlag, 2002.

[Torres & Lopez 00] P. Torres & P. Lopez. *On not-first/not-last conditions in disjunctive scheduling.* European Journal of Operational Research, vol. 127, pages 332–343, 2000.

[Tsai & Gemmill 98] Y-W. Tsai & D.D. Gemmill. *Using tabu search to schedule activities of stochastic resource-constrained projects.* European Journal of Operational Research, vol. 111, pages 129–141, 1998.

[Vepsalainen & Morton 87] A.P.J. Vepsalainen & T.E. Morton. *Priority rules for job shops with weighted tardiness costs.* Management Science, vol. 33, pages 1035–1047, 1987.

[Vidal *et al.* 03] T. Vidal, J. Bidot, J.C. Beck & P. Laborie. *Gestion de projets sous incertitudes : un modèle de génération de plans flexibles en horizon glissant.* Dans 5ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'03), pages 302–303, Avignon, France, 2003.

[Vieira *et al.* 03] G.E. Vieira, J.W. Herrmann & E. Lin. *Rescheduling manufacturing systems : a framework of strategies, policies, and methods.* Journal of Scheduling, vol. 6, no. 1, pages 35–58, 2003.

[Vincke 99] P. Vincke. *Robust solutions and Methods in Decision-Aid.* Journal of Multi-Criteria Decision Analysis, vol. 8, pages 181–187, 1999.

[Wu *et al.* 99] S.D. Wu, E. Byeon & R.H. Storer. *A graph theoretic decomposition of the job shop scheduling problem to achieve robustness.* Operations Research, vol. 47, no. 1, pages 1–14, 1999.

[Yang & Yu 02] J. Yang & G. Yu. *On the robust single machine scheduling problem.* Journal of Combinatorial Optimization, vol. 6, no. 2, pages 18–33, 2002.