THESE

présentée à

l' **UNIVERSITE DE NANCY 1**

pour l'obtention du

DOCTORAT D'ETAT ES SCIENCES

Spécialité : **MATHEMATIQUES**
Mention : Informatique

par Marie-Claude **PORTMANN**

Sujet :

Méthodes de décomposition spatiale et temporelle en ordonnancement de la production

Soutenue le 18 septembre 1987

Jury composé de:
Jacques **CARLIER**
Jean-Bernard **CAVAILLE**
Michel **DEPAIX**
Jean-Claude **DERNIAME**
Ludo **GELDERS**
Jean-Marie **PIERREL**
Jean-Marie **PROTH**
François **ROUBELLAT**
Bernard **ROY**

THESE

présentée à

l' **UNIVERSITE DE NANCY 1**

pour l'obtention du

DOCTORAT D'ETAT ES SCIENCES

Spécialité : **MATHEMATIQUES**
Mention : Informatique

par Marie-Claude **PORTMANN**

Sujet :

Méthodes de décomposition spatiale et temporelle en ordonnancement de la production

Soutenue le 18 septembre 1987

Jury composé de:
Jacques **CARLIER**
Jean-Bernard **CAVAILLE**
Michel **DEPAIX**
Jean-Claude **DERNIAME**
Ludo **GELDERS**
Jean-Marie **PIERREL**
Jean-Marie **PROTH**
François **ROUBELLAT**
Bernard **ROY**

Mes remerciements s'adressent tout particulièrement à :

- mes parents, qui m'ont appris le goût de l'effort et du travail soigné,
- Monsieur le Professeur Jean-Claude **DERNIAME** (Université de Nancy 1), qui m'a enseigné les premiers rudiments d'informatique et a toujours suivi mes travaux de recherche avec intérêt,
- Monsieur le Professeur Michel **DEPAIX** (Université de Nancy 1), qui m'a initiée aux statistiques et à l'analyse des données,
- Monsieur le Professeur Jean-Marie **PIERREL** (Université de Nancy 1), qui a gentiment accepté le rôle de non spécialiste dans ce jury,
- Monsieur le Professeur Jean LEGRAS (professeur honoraire de l'Université de Nancy 1), qui m'a fait connaître et aimer la recherche opérationnelle et qui, avec Madame le Professeur Colette ROLLAND (Université de Paris 1), m'a dirigée pendant mes premières années de recherche,
- Monsieur le Professeur Bernard **ROY** (Université de Paris 9), qui m'a accueillie pendant dix années au laboratoire LAMSADE et m'a fait profiter de ses connaissances pratiques et théoriques sur les problèmes d'ordonnancements,
- Monsieur le Professeur Jacques **CARLIER** (Université de COMPIEGNE), qui m'a apporté ses conseils pour la réalisation de ce travail et a accepté la responsabilité de rapporteur,
- Monsieur le Professeur Ludo **GELDERS** (Université catholique de LEUVEN), connu pour ces travaux en ordonnancement et qui a accepté la responsabilité de rapporteur,
- François **ROUBELLAT**, directeur de recherche au LAAS (CNRS Toulouse), et Jean-Bernard **CAVAILLE**, ingénieur de recherche au CERT-DERA (Toulouse), que j'ai souvent rencontrés lors de réunions de travail ou de congrès (en particulier dans le cadre des activités du pôle "Ateliers flexibles" de ARA) et qui m'ont toujours apporté soutien et encouragements,
- Jean-Marie **PROTH**, directeur de recherche à l'INRIA-LORRAINE, qui, depuis plus de deux ans, m'a accueillie dans l'équipe SAGEP, m'a suggéré de nouvelles voies de recherche, m'a poussée à un travail intensif et a suivi mes travaux dans les moindres détails,
- tous ceux qui m'ont entourée d'une ambiance chaleureuse dans le travail et m'ont apporté leur aide, en particulier, Brigitte PIERRARD et Catherine VICHARD qui ont apporté leur précieuse collaboration à la frappe de ce document.

A tous et à toutes : MERCI

SOMMAIRE

Remarques générales :

Les chapitres sont réalisés de manière indépendante :
la numérotation est propre à chaque chapitre,
la bibliographie associée termine chaque chapitre.

Le sommaire se limite aux paragraphes principaux.

2. METHODES DE DECOMPOSITION EN ORDONNANCEMENT

2.1. INTRODUCTION

2.2. DECOMPOSITION TEMPORELLE

2.3. DECOMPOSITION SPATIALE

2.4. DECOMPOSITION SPATIALE ET TEMPORELLE

2.5. CONCLUSIONS

3. EVALUATION DES METHODES DE DECOMPOSITION TEMPORELLE POUR LES PROBLEMES A UNE MACHINE

3.1. INTRODUCTION GENERALE

3.2. CAS PARTICULIER : DUREES OPERATOIRES UNITAIRES

3.3. CAS PARTICULIER : DUREES OPERATOIRES EGALES ET NON UNITAIRES

3.4. AUTRES CAS PARTICULIERS

3.5. CAS GENERAL ET CONCLUSIONS

ANNEXES

INTRODUCTION

INTRODUCTION

La fonction "ordonnancement" est une fonction essentielle en gestion de production. Pourtant, très peu de progiciels de gestion de production intégrée ou de gestion de production assistée par ordinateur contiennent un module d'ordonnancement pouvant prendre en compte des ressources limitées. Tout chercheur en ordonnancement connaît les raisons de cette pauvreté : sauf dans des cas très particuliers, le nombre de calculs à effectuer pour obtenir un ordonnancement qui optimise un critère donné croît comme l'exponentielle du nombre d'entités (produits et machines) concernés.

En outre, il n'existe pas un problème d'ordonnancement, mais de nombreux problèmes d'ordonnancement en fonction du système de production considéré, de son fonctionnement, et du contexte économique. Par exemple, l'estimation d'un bon ordonnancement n'est pas identique pour une entreprise dont le carnet de commandes sature les machines et qui doit essentiellement maximiser la charge, quitte à ne pas respecter certains délais-clients, et pour une entreprise qui est suréquipée par rapport à son carnet de commandes, qui est dans un marché concurrentiel, et qui doit avant tout respecter les délais imposés par les clients tout en cherchant à produire au moindre coût avec, par exemple, des stocks minimaux. Cette variété se traduit par des contraintes et des critères différents qui conduisent à des modèles d'ordonnancement différents.

Pour chaque modèle, des méthodes exactes et/ou des méthodes approchées ont été proposées par les chercheurs. Il est possible de les regrouper par approches de résolution.

La contribution de cette thèse est de proposer de nouvelles approches par décomposition et de les évaluer.

Le chapitre 1 est un chapitre introductif. Nous y rappelons d'abord la définition des problèmes d'ordonnancement ainsi qu'une modélisation très générale du problème. Nous insistons ensuite sur la diversité des domaines d'application et des systèmes de production. Toutes les notions sont définies et les hypothèses retenues dans le cadre de ce travail sont précisées. Une classification générale propre aux problèmes d'ordonnancement est rappelée.
Après avoir présenté brièvement quelques notions sur l'évaluation des algorithmes (les explications plus détaillées faisant l'objet de l'annexe A), nous proposons une classification générale des approches de résolution qui inclut nos méthodes par décomposition.
Nous terminons par un état de l'art des méthodes exactes existantes, dans le cas des tâches non interruptibles de durées non unitaires. Ces méthodes exactes sont utiles dans le cadre de notre travail où nous découpons le problème en sous-problèmes de taille plus petite qu'elles peuvent éventuellement résoudre (la méthode utilisée dans nos algorithmes est détaillée dans l'annexe B).

Nos méthodes de décomposition spatiale et/ou temporelle font l'objet du chapitre 2.
Dans les méthodes de décomposition temporelle, nous construisons la solution de manière itérative. Nous introduisons des sous-ensembles de produits au fur et à mesure de leur arrivée dans l'atelier et nous optimisons localement les nouvelles opérations avec les opérations qui entrent en conflit avec elles pour l'utilisation des machines.
Les méthodes de décomposition spatiale nécessitent l'utilisation de méthodes de classification non hiérarchique pour découper l'atelier en îlots de fabrication. Nous présentons une brève bibliographie sur les méthodes de technologie de groupe utilisées dans les systèmes de production et nous décrivons la méthode de classification croisée ou méthode par sériation de blocs diagonaux non empiétants élaborée dans notre équipe de recherche. Après le découpage en îlots, les produits sont découpés en sous-produits de telle sorte qu'un sous-produit utilise les machines d'un seul îlot.
La méthode de décomposition spatiale consiste à ordonnancer les sous-produits, îlot par îlot, puis à gérer les liens résiduels entre les îlots.
Il est possible de croiser de différentes façons la décomposition spatiale et la décomposition temporelle. On obtient ainsi des méthodes de décomposition spatiale et temporelle.

Nos méthodes de décomposition sont des méthodes approchées. En effet, la réunion de solutions optimales locales ne donne pas, en général, un optimum global. Elles sont du même ordre de complexité que la méthode utilisée pour résoudre les sous-problèmes obtenus.

Dans le chapitre 3, nous évaluons les méthodes de décomposition temporelle dans le cas particulier où l'atelier ne comporte qu'une machine. Nous montrons que, lorsque les durées sont unitaires, une méthode de décomposition temporelle est identique à l'algorithme de Jackson : elle est polynômiale et minimise le critère "somme des retards". Dans le cas où les durées sont égales, mais non unitaires, c'est-à-dire que les instants d'arrivée des tâches ne sont pas multiples de la durée opératoire unique (tâches longues par rapport à l'unité de temps), une méthode de décomposition temporelle est toujours polynômiale, mais c'est une méthode approchée. Nous fournissons une majoration de l'erreur commise en fonction de la taille du problème.
Une nouvelle notion est introduite dans ce chapitre 3 : c'est celle d'ordonnancement u-actif. Il est démontré que dans le cas des durées égales, ce sous-ensemble est dominant pour le critère "somme des retards", mais il ne l'est pas dans le cas général.

Dans le chapitre 4, les méthodes de décomposition introduites précédemment sont comparées entre elles et à d'autres méthodes approchées. Les comparaisons sont réalisées à partir d'expériences sur des ensembles générés de manière aléatoire. Elles sont présentées essentiellement sous forme de tableaux. Les conclusions sur le comportement moyen des méthodes ne sont que des tendances indicatives, limitées aux particularités des exemples générés.

D'autres résultats peuvent être obtenus sur l'évaluation de cette nouvelle famille de méthodes qui peut s'enrichir de multiples variantes. Nous proposons pour terminer quelques extensions et perspectives.

Ces méthodes, bien que développées hors contrat en laboratoire, intéressent les industriels. Il ne manque que l'interface pour les intégrer dans les outils de gestion existants (l'annexe D rappelle les expériences pratiques antérieures de l'auteur en milieu industriel). Des contacts ont été pris pour des tests en situation réelle.

CHAPITRE 1

LES PROBLEMES D'ORDONNANCEMENT

1. LES PROBLEMES D'ORDONNANCEMENT

1.1. INTRODUCTION, DEFINITION ET MODELISATION

1.1.1. Introduction

Ce travail propose une nouvelle approche de résolution pour les problèmes d'ordonnancement.

Dans un chapitre introductif, nous rappelons les approches et les méthodes existantes. Mais auparavant il est nécessaire :

- de définir ce que l'on appelle un problème d'ordonnancement,
- d'en présenter une formulation générale,
- d'insister sur la grande diversité des problèmes rencontrés,
- de constater les liens entre les hypothèses différentes et la complexité des problèmes posés,
- puis de souligner l'influence de la complexité sur la qualité des solutions obtenues,
- et enfin, d'introduire les différentes approches de résolution.

1.1.2. Définition

Il y a une dizaine d'années, en collaboration avec Bernard ROY et Gilles CHAUVIN, nous avons rédigé un document interne au laboratoire LAMSADE sur la définition des "modèles d'ordonnancement" ([B. ROY / G. CHAUVIN / M.C. PORTMANN 1978]). La tâche fut complexe. Trop générale et conçue dans le cadre des problématiques d'aide à la décision, la définition englobe d'autres problèmes de décision (par exemple, suivi de processus en présence d'aléas); trop restrictive, elle exclut certains aspects des problèmes d'ordonnancement comme, par exemple, le pilotage et le suivi en présence de perturbations.

La définition présentée ici est très générale. Par la suite, nous la particulariserons en ajoutant des hypothèses ou des caractéristiques nouvelles.

* **Définition générale des problèmes d'ordonancement** ([B. ROY 1970])

Il y a problème d'ordonnancement :

- quand un ensemble de travaux est à réaliser,
- que cette réalisation est décomposable en tâches,
- que le problème consiste à définir la localisation temporelle des tâches et/ou la manière de leur affecter les moyens nécessaires.

Les contraintes et les critères du problème ne doivent concerner que les tâches, leur localisation temporelle, et les moyens nécessaires à leur réalisation.

Résoudre un problème d'ordonnancement, c'est donc fournir la réponse à deux questions : Quand ? et Comment ?

La réponse à ces questions est fournie lorsqu'on a obtenu :

- des valeurs qui optimisent un critère donné,
- ou des valeurs jugées satisfaisantes face à plusieurs critères éventuellement antagonistes,
- ou des propositions de choix entre plusieurs valeurs qui satisfont des contraintes,
- ou encore un outil conversationnel d'aide à la décision qui intègre l'avis du décideur pour certains choix.

1.1.3. Modélisation

Modéliser un système de fabrication, c'est définir :

- l'ensemble des paramètres représentatifs du système de fabrication étudié,
- les liens entre ces paramètres,
- les critères qui permettront d'évaluer les solutions obtenues.

En ce qui concerne les problèmes d'ordonnancement, il est rare qu'ils soient clairement définis en début d'analyse. Cela nécessite donc une première phase de négociation et de collaboration avec les utilisateurs.

La décomposition en tâches n'est pas toujours immédiate. Il n'y a pas unicité de cette décomposition, et un même problème concret peut conduire à plusieurs problèmes d'ordonnancement selon la précision du découpage en tâches et les critères d'évaluation retenus en fonction de cette précision. Et pour chaque problème d'ordonnancement mis en évidence, il y a différentes possibilités de modélisation.

Pour décrire un modèle, on est amené à choisir un langage de description (formules mathématiques, description graphique, Réseau de Pétri,...). Quel que soit le langage utilisé, il manipule des informations connues (les données et les hypothèses du problème) et des informations inconnues (les décisions à prendre). Les informations inconnues peuvent avoir de multiples significations suivant le modèle choisi. Par exemple, on peut avoir à choisir entre :

$y_{i,j}$: entier désignant la machine qui exécute l'opération j du produit i

et

$x_{i,j,k}$: entier valant

* 1 si la machine k exécute l'opération j du produit i
* 0 sinon

ou encore entre :

t_i : date de début de la tâche i

et

$u_{i,t}$: entier valant

* 1 si la tâche i commence à l'instant t
* 0 sinon.

Aussi est-il difficile de présenter un modèle pour les problèmes d'ordonnancement.

En supposant que la modélisation des informations puisse être adaptée ultérieurement aux problèmes particuliers, nous proposons la modélisation trés générale suivante ([B. ROY 1970]) :

*** Définition générale d'un ordonnancement**

Une solution d'un problème d'ordonnancement est fournie par un triplet (T,D,W) appelé ordonnancement où (i étant l'indice des tâches) :

$T = \{t_i\}$ est l'ensemble des informations fournissant les dates de début des tâches (ou les dates de début de chaque tronçon, pour le cas particulier où les tâches sont interruptibles) ;

$D = \{d_i\}$ est l'ensemble des informations fournissant les durées des tâches (ainsi que les durées de chaque tronçon si les tâches sont interruptibles) ;

$W = \{(w_{i,k}) \ \forall \ k\}$ est l'ensemble des informations concernant les modes d'utilisation des moyens pour chaque moyen k.

T et D précisent la localisation temporelle des tâches et W la manière d'allouer les moyens aux tâches.

Les ordonnancements (T,D,W) fournis par une méthode de résolution dépendent de la problématique retenue pour le modèle et de la qualité de la méthode de résolution dans le cadre de cette problématique.

Bernard ROY ([1977] ,par exemple) distingue quatre problématiques :

- la problématique du choix ou de la sélection,
- la problématique du tri,
- la problématique du classement ou du rangement,
- la problématique descriptive ou cognitive.

En ordonnancement, les problématiques utilisées sont essentiellement la première et la dernière.

Dans la problématique du choix ou de la sélection, on retient, parmi les solutions réalisables, un sous-ensemble de solutions sélectionnées que l'on fournit au décideur. Celui-ci n'a pas la connaissance des solutions rejetées.

La recherche d'une solution optimale ou de toutes les solutions optimales, en présence d'un seul critère, est un cas particulier de cette problématique. La recherche d'une solution à moins de 90 % de l'optimum en est une autre. La recherche de toutes les solutions "efficaces" relativement à plusieurs critères ou d'un compromis entre ces solutions efficaces ([R.S. STAINTON 1978], [R.T. NELSON / R.K. SARIN / R.L. DANIELS 1986], [L.N. VAN WASSENHOVE / K.R. BAKER 1982], [R. SLOWINSKI 1981]) appartient également à cette problématique. (En analyse multicritère, on appelle solution efficace une solution telle que l'on ne peut pas gagner sur un critère quelconque sans perdre sur un ou plusieurs autres).

Dans la problématique du tri, on informe le décideur sur l'ensemble des solutions en distinguant dans cet ensemble un sous-ensemble certainement bon et un sous ensemble certainement mauvais. Ceci est surtout valable lorsque les données sont évolutives ou incertaines.

Dans la problématique du classement ou du rangement, on recherche un préordre entre certaines solutions à partir des différents ordres founis par les critères antagonistes.

En choisissant une problématique descriptive ou cognitive, le problème est posé en termes de description de la solution et de ses conséquences. On étudie, par exemple, l'influence des conditions de pilotage sur le déroulement futur de l'ordonnancement.

Il est évident que le choix de la problématique retenue pour poser le problème a une influence importante sur la modélisation et sur les méthodes de résolution.

Dans cette thèse, nous nous intéressons à la problématique la plus courante : celle du choix ou de la sélection en présence d'un ou plusieurs critères. Nous les utiliserons de manière hiérarchique : parmi les solutions optimales pour un critère donné, nous retenons celles qui optimisent le critère suivant.

Il n'est pas possible de détailler davantage la modélisation si on ne précise pas les hypothèses particulières des problèmes posés. Ceci fait l'objet du paragraphe suivant.

1.2. DIVERSITE ET ESSAIS DE CLASSIFICATION

1.2.1 Diversité des problèmes et des domaines d'application

Souvent, lorsque l'on parle d'ordonnancement, on pense uniquement aux ordonnancements d'atelier. En réalité, les domaines d'application sont nombreux :

- ordonnancement de projets, avec éventuellement limitation de certains moyens (construction de maisons individuelles, d'immeubles, d'autoroutes, d'usines..., planification des tâches d'études et de conception, de tâches administratives...) ;

- organisation d'emplois du temps (établissements scolaires, sessions de stages, sessions d'examens...) ;

- problèmes de tournées, avec éventuellement des contraintes supplémentaires dûes aux clients ou au personnel (ramassage de lait, distribution de courrier, de béton, de colis...) ;

- ordonnancement des services publics de transport (horaires du service rendu au client, tournée des véhicules, affectation du personnel de chaque catégorie) ;

- problèmes de découpes (à une dimension, en surface, en volume) ....

Le problème abordé dans ce travail concerne les ordonnancements d'atelier. C'est pourquoi nous n'avons pas cité la littérature abondante sur les autres problèmes d'ordonnancement.

La grande diversité des cas possibles ([CESTA 1983], [J. BERNAD / M. PAKER 1985]) montre qu'il est impossible de concevoir des logiciels universels d'ordonnancement quel que soit leur niveau de paramètrage.

Cette diversité est liée aux systèmes de production et aux hypothèses différentes de leur fonctionnement. Elle est illustrée par la figure 1.2.1. .

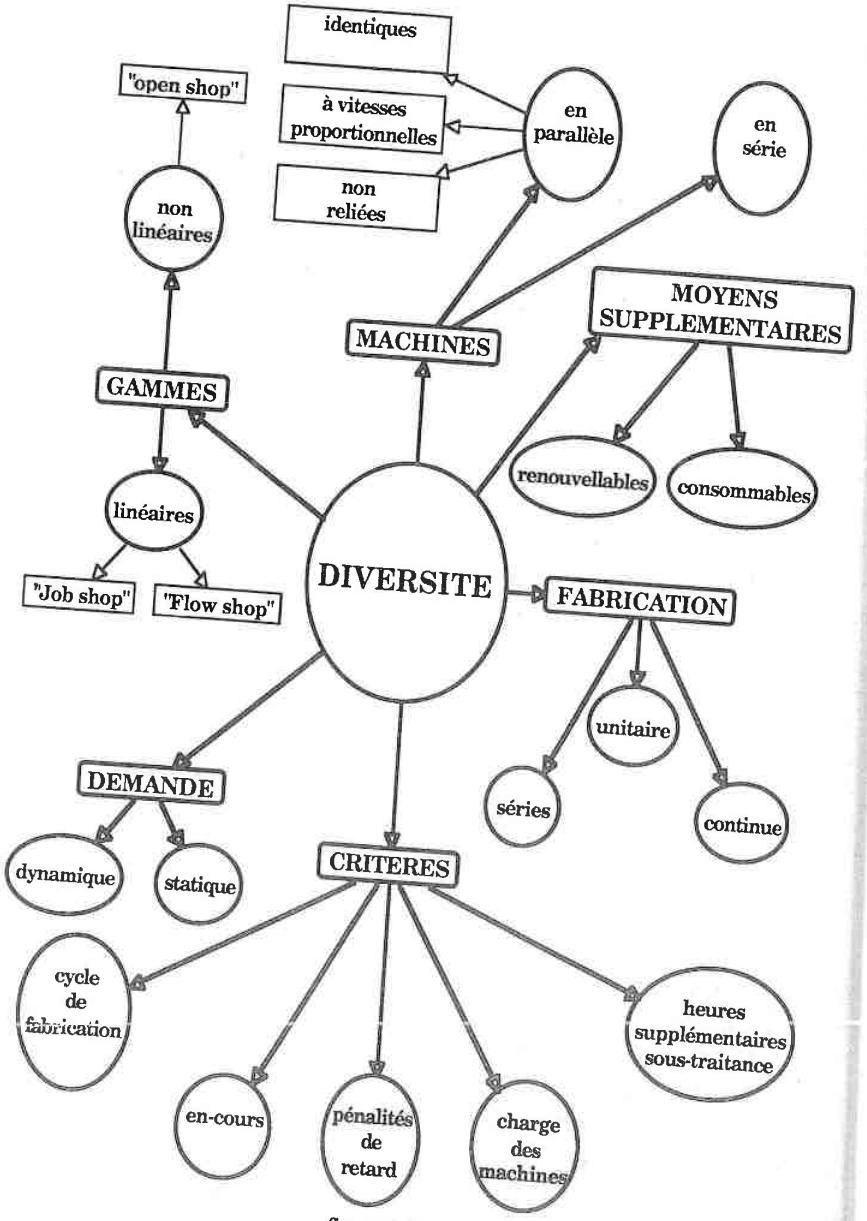


figure 1.2.1.

1.2.2. Diversité des systèmes de production

1.2.2.1. Présentation générale

En avant-propos, signalons la grande similitude entre l'ordonnancement des tâches à l'intérieur d'un ordinateur et celui des opérations dans un atelier de production. Ces deux problèmes seront appelés ordonnancements d'atelier.

Toutefois, ils diffèrent par une hypothèse fondamentale, l'interruptibilité des tâches d'un ordinateur qui rend les problèmes d'ordonnancement plus aisés à résoudre.

Rappelons les hypothèses de composition et de fonctionnement des ateliers qui sont à la source de la grande diversité des problèmes.

a) Les machines

Dans le cas le plus simple, on considère une seule machine ([J. CARLIER 1982]).

C'est un cas rare dans une entreprise, plus fréquent pour les ordinateurs mono-processeur mais multi-usagers. Toutefois, il ne manque pas d'intérêt même pour les entreprises. En effet, si une machine constitue un goulot d'étranglement et ralentit le débit général de la production, alors que les délais sont serrés, il y a deux solutions envisageables :

* la solution "matériel" consiste :
  - soit à acheter une machine supplémentaire,
  - soit à échanger la machine goulot contre une autre plus rapide,
  - soit à modifier les gammes de production de manière à éviter cette machine ;
* la solution "logiciel" consiste à améliorer l'ordonnancement de cette machine. La présence des autres machines est alors traduite par des contraintes dans un modèle simplifié. Cette approche est bien sûr utilisée lorsque l'ordonnancement "optimal" de tout l'atelier nécessite des coûts de calcul trop élevés.

Dans la mesure où la solution "matériel" n'est pas toujours possible ou immédiatement envisageable, les résultats développés pour les problèmes à une machine sont loin d'être inutiles.

Lorsque l'on considère plus d'une machine, plusieurs cas sont à envisager. Ils correspondent aux notions, bien connues en électricité par exemple, de parallèlisme, série, et série-et-parallèlisme :

* dans le premier cas, les machines sont utilisées en parallèle. Elles ont la même fonction et sont susceptibles d'effectuer les mêmes opérations (par exemple une unité centrale multi-processeurs ou une batterie de trois fraiseuses).
Dans ce cadre, on note trois possibilités :

  - les machines sont *identiques* et nécessitent le même temps pour exécuter la même opération,

  - les machines diffèrent uniquement par leur rapidité (par exemple une imprimante 600 lignes/minute et une imprimante 1200 lignes/minute). La durée des opérations dépend de l'opération et du facteur de rapidité de la machine. Il s'agit de machines à *vitesses proportionnelles* ou encore, en expression anglaise, de machine *uniformes* ([J. LABETOULLE / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979]),

  - les machines sont différentes. Deux machines ou plus peuvent effectuer certaines opérations, mais il n'y a aucun lien entre les opérations, les durées opératoires et les machines. Simplement, la durée d'une opération est fonction de la machine choisie. On parle de machines *non reliées* ([E. LAWLER / J. LABETOULLE 1978]), mais aussi parfois de machines à *opérations privilégiées* dans la mesure où chacune est plus rapide que les autres pour une famille d'opérations qui lui sont destinées, mais tolère d'autres opérations avec des durées moins avantageuses pour soulager ou remplacer d'autres machines ;

* dans le deuxième cas, les machines sont utilisées en série. Il ne s'agit plus de choisir, pour une opération, quelle machine va l'effectuer, mais d'assurer une suite d'opérations sur des machines différentes (par exemple, une fraiseuse, puis un tour, puis une machine à usiner) ;

* dans le cas le plus général, les produits subissent une série d'opérations et pour certaines de ces opérations, on a le choix entre plusieurs machines en parallèle.

La description des opérations à effectuer sur le produit, ainsi que les machines nécessaires, constitue la gamme de fabrication du produit dans l'atelier.

Nous reviendrons plus tard sur l'influence des hypothèses correspondantes sur la diversité des problèmes d'ordonnancement. Nous allons auparavant nous intéresser à l'organisation générale de la fabrication.

b) L'organisation de la fabrication

La fabrication peut se limiter à quelques produits, voire un seul de chaque type, fournis à la demande. Il s'agit alors de fabrication unitaire.

La fabrication peut se faire sous forme de séries non fractionnables, ni pour la fabrication, ni pour les transports. Dans ce cas, une série de produits élémentaires est considérée comme un seul produit et les problèmes à résoudre ressemblent à ceux de la fabrication unitaire.
Tout en fabricant les produits en séries, celles-ci peuvent être découpées en lots ou rafales pour l'usinage ou, à cause de la taille limitée des containers utilisés, pour le transport. Cela induit des contraintes particulières de succession (voir par exemple [B. ROY 1970]).

Dans le cas des ateliers dits flexibles, on fabrique souvent plusieurs produits en grande quantité dans des proportions imposées ; on alterne alors leur passage sur les machines sans constituer de séries, de manière à mieux occuper les machines. Il est possible d'envisager une fabrication périodique où les mêmes ordonnancements se répètent ([C. THURIOT 1985], [H. HILLION 1987]).

Le processus de fabrication se complique encore davantage s'il fait intervenir des opérations d'assemblage. Cela induit des contraintes de succession entre les opérations et de synchronisation des rafales.

Pour être complet, il faut signaler ici la fabrication par des processus continus. Elle pose également des problèmes d'ordonnancement. Ils sont très spécifiques de la fabrication considérée et ne feront l'objet d'aucun développement dans le cadre de ce travail.

c) Les gammes

Nous les avons introduites en a). Une gamme concerne un produit ou un ensemble de produits élémentaires identiques dans le cas de la fabrication en séries non fractionnables ou par rafales. Elle décrit le processus de fabrication pour tout produit dans l'atelier considéré.

Dans le cas le moins souple, mais le plus simple, les gammes sont linéaires. On suppose qu'il n'y a pas de machines en parallèle, ou, ce qui revient au même, qu'à toute opération on a affecté une seule machine. Alors chaque produit visite les machines selon un ordre imposé.
Si l'ordre de visite des machines est le même pour tous les produits et si l'ensemble des produits passent toujours dans le même ordre sur chacune des machines (pas de dépassement autorisé), le problème d'ordonnancement d'atelier correspondant est connu sous le terme anglais de "flow-shop" de permutation.
Sous les mêmes hypothèses, mais en autorisant que les produits se dépassent (c'est-à-dire qu'un produit traité avant un autre sur la $k^{ième}$ machine puisse passer après ce dernier sur la $(k+1)^{ième}$ machine), on obtient un problème de "flow-shop" général.

Toujours en utilisant des gammes linéaires, lorsque l'ordre de passage sur les différentes machines dépend du produit, on obtient un problème de "job-shop".

Les gammes ne sont plus linéaires lorsque l'ordre des opérations sur les machines est partiel et non total. Cela laisse des possibilités d'inversion plus ou moins limitées. On parle parfois d' "open shop" lorsqu'il n'y a aucune contrainte de succession entre les opérations ([R. SLOWINSKI 1979]).

D'autres caractéristiques viennent compléter la description des gammes.

En cas de fabrication en séries ou en rafales, du recouvrement peut être autorisé entre deux opérations successives sur le même produit (L.G. MITTEN 1958]). En effet, l'opération suivante dans la gamme est susceptible de commencer sur les premiers produits élémentaires de la série alors que l'opération en cours se termine sur les derniers produits élémentaires. Cette hypothèse de recouvrement entre opération est d'autant plus utile et nécessaire que les places de stockage entre les deux opérations sur la série sont limitées, voire inexistantes (flow-shop de permutation sans en-cours ([K.R. BAKER 1974]).

Les opérations peuvent nécessiter des outils spécifiques. Nous aborderons cette question en d). Nous nous intéressons ici non pas à la disponibilité de moyens, mais aux conséquences induites sur les gammes. Les opérations de la gamme sont souvent précédées de temps de réglage : simples réglages ou changement d'outils et réglages avec, dans le cas le plus défavorable, acheminement des outils vers la machine.
Si les temps de réglage ne dépendent que du produit traité et ont lieu alors que le produit est déjà sur la machine, on en tient compte en allongeant simplement la durée opératoire. Si les réglages ne dépendent que du produit traité et peuvent avoir lieu avant l'arrivée du produit, on les ajoute à la durée opératoire, mais on autorise un recouvrement du temps de réglage avec l'opération précédente. Dans le cas le plus général et le plus difficile, la durée des temps de réglage dépend de l'ordre de passage des opérations associées aux différents produits sur la machine.

d) Les moyens

Il s'agit ici des ressources complémentaires autres que les machines, en particulier le personnel qualifié, les outils et les ressources en énergie. Chacun d'eux possède trois caractéristiques essentielles : disponibilité, quantité, nature. L'idéal est de disposer de moyens toujours disponibles et fonctionnels. Cela est, en général, impossible pour les machines qui ont besoin d'être entretenues. Par ailleurs, on ne peut pas toujours organiser les 3x8 de manière à avoir en permanence du personnel de toute qualification, et il arrive que le personnel s'absente. On dispose alors d'un calendrier de disponibilités des moyens.
La quantité de moyen peut être constante, au moins pendant les périodes de disponibilité, ou au contraire variable avec le calendrier.

En utilisant la terminologie de l'école polonaise ([J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI / J. WEGLARZ 1986]), nous distinguons trois types de moyens selon leur nature.

Les moyens "renouvelables", comme par exemple le personnel, sont les moyens inusables. Le fait de les utiliser à un instant donné les laisse identiques pour la suite. Les contraintes exprimant qu'ils existent en quantité limitée sont des contraintes instantanées qui doivent être vérifiées à chaque unité de temps.

Les moyens "consommables", comme par exemple certains outils ou une alimentation auxiliaire en pièces secondaires (vis, boulons, écrous...), disparaissent au fur et à mesure de la production avec des livraisons à certains moments. Il s'agit de stocks de matières auxiliaires. Les contraintes exprimant qu'ils existent en quantité suffisante pour pouvoir produire sont des limitations de consommations (diminuées par des livraisons) cumulées sur des intervalles de temps.

Enfin, les moyens "doublement limités" juxtaposent les deux types de contraintes précédentes : la consommation instantanée doit rester dans une fourchette de valeurs et il en est de même pour la consommation cumulée sur des intervalles. C'est le cas en particulier pour les sources d'énergie, comme le gaz ou l'électricité, si on a signé des contrats spécifiques avec les fournisseurs.

Pour en terminer avec la diversité des systèmes de production et leur lien avec celle des problèmes d'ordonnancement, nous allons aborder deux points qui concernent les liens entre les systèmes de production et leur environnement , à savoir la connaissance du carnet de commandes et celle des objectifs de l'entreprise, de manière à définir des critères d'évaluation des ordonnancements.

e) Connaissance de la demande

L'homme d'étude aime que son problème soit parfaitement défini. Cela implique la connaissance des commandes, de leurs délais, des gammes, de la disponibilité des produits et des moyens etc... C'est, bien sûr, le cas idéal. Souvent, cependant, ces informations sont soumises à des aléas, ou sont prévisionnelles, ou encore sont connues en probabilité. En outre, la production à réaliser n'est pas toujours connue a priori.

Un problème d'ordonnancement est dit statique si l'on connaît a priori toutes les données le concernant. Il est dit dynamique si de nouveaux produits à faire surgissent au fur et à mesure de son déroulement.

f) Les critères d'appréciation

Dans la mesure où il y a rarement un seul ordonnancement compatible avec les contraintes du problème posé, il est nécessaire de disposer de critères d'évaluation qui aideront à faire un choix. Ces critères doivent traduire les objectifs de l'entreprise. On peut retenir de nombreux objectifs :

- minimiser les cycles de fabrication,

- minimiser le volume d'en-cours,

- minimiser la variance des durées totales de fabrication des produits ([S. CHAKRAVARTHY 1986]) de manière à régulariser les délais internes,

- respecter au mieux les délais, ce qui peut se traduire par :
  * minimiser le plus grand retard,
  * minimiser le nombre de retards,
  * minimiser la somme des retards,
  * minimiser une somme de pénalités dûes aux retards,

- maximiser la charge des machines,

- minimiser les heures supplémentaires ([C.A. HOLLOWAY / R.T. NELSON 1974], [L. GELDERS / P.R. KLEINDORFER 1974]) et la sous-traitance,

- minimiser les temps de réglages,

mais aussi des objectifs moins quantifiables tels que :
- maximiser la sécurité,
- minimiser l'absentéisme.

Certains objectifs vont dans le même sens, sans avoir toutefois leur optimum pour la même solution. D'autres, par contre, sont antagonistes. Pour les prendre en considération on peut :

- ou bien en retenir un seul, le plus important pour l'atelier considéré et les objectifs actuels de l'entreprise,
- ou bien les utiliser hiérarchiquement (on retient toutes les meilleures solutions pour le premier critère, parmi lesquelles on retient seulement les meilleures pour le deuxième...),
- ou bien en agréger plusieurs par combinaison linéaire (moyenne pondérée),
- ou bien ne pas retenir de critère du tout et souhaiter des variantes de solutions réalisables ([J.ERSCHLER /G. FONTAN / C. MERCE / F. ROUBELLAT 1983],[F. ROUBELLAT / V. THOMAS 1987]).

Dans ces quatre cas, il est possible de garantir une qualité minimale pour certains objectifs au moyen de contraintes supplémentaires.

Le décideur peut intervenir en cours de résolution dans une procédure interactive multicritère, ou en temps réel ou différé, pour faire des choix entre plusieurs variantes réalisables.

1.2.2.2. Hypothèses envisagées

Face à la diversité des systèmes de production voilà, point par point, les hypothèses que nous avons retenues dans ce travail.

a) Les machines sont utilisées en série et non en parallèle. S'il existe plusieurs machines susceptibles d'effectuer une même opération, nous supposons qu'un choix précis a été fait au préalable dans une étape antérieure de la gestion de production, dont l'ordonnancement considéré n'est qu'un maillon. Cette étape antérieure, qui consiste à fixer le cheminement des pièces dans l'atelier, est appelée routage. Cela n'interdit pas, a posteriori, de réintroduire d'autres cheminements pour obtenir des solutions encore meilleures. Simplement, cette souplesse ne sera pas prise en compte dans le modèle proposé.

b) Il s'agit de fabrication unitaire (ou en séries non fractionnables sans recouvrement). On peut noter que :

- autoriser l'interruptibilité des tâches rendrait le problème plus facile,
- gérer des séries fractionnables compliquerait le modèle sans remettre en cause la méthode choisie.

Il n'y a pas de contraintes de succession entre les produits dûes à des assemblages. On peut noter que cette contrainte s'ajouterait sans difficulté au modèle choisi, elle compliquerait simplement les notations et demanderait un effort supplémentaire de programmation pour les expérimentations.

c) Les gammes sont linéaires avec un ordre de passage qui dépend du produit. Il s'agit d'un "Job-Shop". On peut noter que :

- le "flow-shop" est pris en compte par le modèle, puisque c'est un cas particulier de "job-shop",
- autoriser des gammes non linéaires demanderait la même modification des notations du modèle et du programme que pour intégrer des contraintes de succession dûes aux assemblages de produits.

Le recouvrement entre opérations n'a pas été intégré au modèle, mais pourrait l'être sans difficulté.

Les places de stockage sont supposées en quantité suffisante pour permettre des en-cours quasi illimités.

Les temps de réglage sont supposés indépendants de la séquence et inclus dans la durée opératoire. Il est possible d'ajouter à l'ordonnancement des durées dépendant de la séquence, mais il serait coûteux en durée des calculs de chercher à minimiser la somme des durées de réglage (ou à optimiser un autre critère en essayant toutes les séquences).

d) Nous ne considérons pas de ressources supplémentaires et les machines sont supposées toujours en état de fonctionner. Toutefois, la possibilité d'interdire l'utilisation d'une machine pendant un intervalle de temps est très facile à obtenir par une astuce qui s'intègre bien au modèle :

prévoir une tâche "panne" ou "maintenance" dont :

- la durée est celle de l'intervalle de temps sur lequel on veut immobiliser la machine,
- la date de disponibilité est celle du début de l'intervalle,
- le délai est la date de fin de l'intervalle,
- la pénalité dûe au retard est infiniment grande.

e) Les produits à fabriquer et leurs gammes sont supposés connus avant l'arrivée du produit dans l'atelier, mais pas obligatoirement dès le début de la période sur laquelle on ordonnance. Les ordonnancements considérés pourront être statiques (hypothèse obligatoire pour la décomposition spatiale) ou dynamiques.

f) Nous utiliserons une hiérarchie de critères. En principe, tous ceux qui sont quantifiables sont utilisables dans nos méthodes de décomposition. La méthode de décomposition spatiale impose l'existence de délais pour les produits.

Pour nos résultats, nous insistons tout particulièrement sur le critère "somme des pénalités de retard", qui est un des critères les plus difficiles à optimiser.

1.2.3. Une classification propre aux problèmes d'ordonnancement

A partir des paragraphes 1.2.1. et 1.2.2., on pourrait construire une arborescence de tous les problèmes d'ordonnancement possibles à partir des différentes hypothèses. Les branches en seraient multiples et cela n'apporterait rien à la connaissance des problèmes d'ordonnancement et leurs structures particulières.

Si l'on souhaite une typologie des problèmes d'ordonnancement pour les classer et les situer les uns par rapport aux autres, mieux vaut revenir à la modélisation générale présentée en 1.1.3. On distingue, en allant du plus simple au plus complexe (voir [B. ROY 1970]) :

* Les problèmes de type T, pour lesquels les seules informations inconnues sont les dates de début des tâches. Les durées sont fixées et connues, et le mode d'affectation des moyens aux tâches est ou bien fixé a priori, ou bien ignoré car inutile dans le modèle.

Dans cette classe de problèmes, une première sous-classe, appelée dans ([B. ROY 1970]) "problème central d'ordonnancement", suppose les moyens en quantité suffisante pour qu'ils ne créent pas de conflit entre les tâches. Ce sont les problèmes d'ordonnancement à moyens illimités encore appelés "ordonnancements de projet". Ce furent les premiers résolus ([P.E.R.T. 1958], [J.E. KELLEY / M.R. WALKER 1959], [B. ROY 1960]). La modélisation correspondante utilise des graphes où les arcs traduisent des "contraintes de potentiel" ou contraintes de succession chronologique entre des évènements.

Une deuxième sous-classe, où les moyens existent en exemplaire unique, comporte des contraintes disjonctives entre les tâches ([B. ROY 1966], [Y. TABOURIER 1969]). Lorsque au moins un moyen (renouvelable) existe en plusieurs exemplaires identiques en quantité limitée, on parle de contraintes cumulatives ([J.F. BOSS 1966], [T.L. PASCOE 1966], [E. BALAS 1968].

* Pour les problèmes de type W, la localisation temporelle des tâches est supposée fixée et connue. Les seules inconnues du problème sont les modes d'utilisation des moyens. Il s'agit d'affecter des moyens à des tâches déjà planifiées. Ces problèmes sont des problèmes d'affectation avec ou sans réemploi des moyens. La difficulté de ces problèmes est liée à la forme du critère d'appréciation : ils sont faciles si le critère est linéaire ([H.W. KÜHN 1955], [L.R. FORD / D.R. FULKERSON 1956 et 1957]) et difficiles s'il est quadratique ([M. BEGHIN-PICAVET / P. HANSEN 1982]).

* Dans les problèmes de type (T,D(T)), la durée des tâches dépend de leur localisation temporelle. Ils se posent, par exemple, lorsqu'il y a des tâches de transport dans une région où le trafic, important, comporte des heures de pointes ([R. TREMOLIERES / J.C. AUBERT 1975]).

* Dans les problème de type (T,D(W),W), la durée des tâches dépend du mode d'affectation des moyens. Deux possibilités essentiellement :

- la durée et le coût de la tâche varient avec la quantité de moyens qu'on lui attribue (voir [D.R. FULKERSON 1961] et [BAKER 1974] lorsque la variation est linéaire),
- la durée de la tâche dépend de l'exemplaire du moyen qu'on lui attribue (voir [E.L. LAWLER / J. LABETOULLE 1978] et [J. LABETOULLE / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979] pour les machines non identiques en parallèle).

Dans ce travail, comme nous supposons les moyens déjà affectés et en exemplaire unique, nous avons un problème de type T avec contraintes disjonctives.

1.3. COMPLEXITE ET APPROCHES DE RESOLUTION

1.3.1. Algorithmes exacts et heuristiques

Etant donné un problème, une procédure qui, pour toute application numérique, fournit une solution optimale, est appelée algorithme optimal ou algorithme exact.

Etant donné un problème, une procédure telle que,
- ou bien, il existe des exemples pour lesquels la solution fournie n'est pas optimale,
- ou bien, l'on n'arrive pas à démontrer que la solution fournie est optimale quelle que soit l'application numérique,
- ou bien l'algorithme conduit à la solution optimale, mais est interrompu avant d'y parvenir ,

est appelée heuristique ou algorithme approché.

Il existe d'autres définitions des heuristiques, par exemple, Nicholson ([T. NICHOLSON 1971]) définit une méthode heuristique comme une procédure qui "... résout des problèmes par une approche intuitive dans laquelle la structure du problème peut être interprétée et exploitée intelligemment afin d'obtenir une solution raisonnable".

Cette dernière définition correspond à un souhait plus qu'à une réalité :
- comment démontrer qu'une procédure utilise "intelligemment" la structure du problème ?
- comment prouver qu'une solution obtenue est raisonnable ?

Nous avons retenu les définitions fournies au début de ce paragraphe car elles ont le mérite d'être mutuellement exclusives, bien qu'évolutives (une heuristique devenant, grâce à une nouvelle démonstration, un algorithme exact). On peut noter, en particulier, que s'il existe des conditions suffisantes pour qu'une procédure fournisse à coup sûr une solution optimale, alors une procédure est algorithme exact ou heuristique selon que les conditions suffisantes sont satisfaites ou non.

Par ailleurs, en cas de choix multicritère, la notion de solution optimale n'ayant plus de sens, il n'y a plus d'algorithme exact ni d'heuristique, mais des procédures interactives multicritères où la notion de solution efficace remplace celle de solution optimale ([R. SLOWINSKI / J. WEGLARZ 1982]).

Le problème lié aux algorithmes exacts est leur coût souvent prohibitif. Celui des heuristiques est la qualité des solutions obtenues. Nous allons revenir sur ces deux problèmes au paragraphe suivant.

1.3.2. Notions de complexité et d'évaluation

La première évaluation intéressante d'un algorithme exact ou approché est son coût en nombre d'opérations élémentaires (additions et comparaisons). Ce coût, exprimé en fonction de la "taille" du problème (nombre de bits occupés par les données), est appelé complexité de l'algorithme.

Si la fonction complexité peut être majorée par un polynôme fonction de la taille du problème, alors l'algorithme est dit polynômial. Si, au contraire, la fonction complexité peut être minorée par une fonction exponentielle de la taille du problème, alors l'algorithme est dit exponentiel.

Dans le cas polynômial, la propriété est universelle : quel que soit l'exemple, la fonction complexité est bornée par le polynôme. Dans le cas exponentiel, la propriété est existentielle : il existe au moins un exemple pour lequel la fonction complexité est supérieure à une fonction exponentielle. Les notions d'algorithmes exponentiels et polynômiaux sont donc des analyses dans le pire des cas (Worst Case Analysis).

Il est également possible de définir la complexité moyenne d'un algorithme. Pour cela, deux possibilités existent :

- la génération d'exemples aléatoires et des mesures expérimentales sur ces échantillons,
- le choix de lois de probabilité pour les données du problème et le calcul théorique de l'espérance mathématique de la fonction complexité.

Ces mesures ont toutes un inconvénient : elles "biaisent" l'ensemble des données du problème. En effet, pour la complexité dans le pire des cas, les exemples qui sont minorés par une fonction complexité exponentielle peuvent être rares et entièrement artificiels, sans aucun rapport avec les exemples de problèmes réels. Au contraire, pour la complexité en moyenne, les lois de probabilité choisies pour la génération d'échantillons ou pour les calculs théoriques réduisent considérablement l'ensemble des données du problème en fournissant des valeurs plus "régulières". Ces mesures ne sont donc que des indicateurs particuliers du coût des algorithmes.

Depuis 1970, toute une théorie a été élaborée sur les analyses dans le pire des cas. Elle montre en particulier que la complexité des meilleurs algorithmes construits pour résoudre un problème est étroitement liée à la structure même du problème. Elle permet d'établir des frontières entre les problèmes polynômiaux et les problèmes NP-complets, NP-difficiles et NP-difficiles au sens fort. Ces frontières évoluent chaque fois que, pour un problème ouvert (c'est-à-dire non classé), on détermine sa complexité ([M.R. GAREY / D.S. JOHNSON 1979]).Une présentation plus détaillée de ces notions est proposée en annexe A.

Le problème d'ordonnancement abordé dans ce travail est NP-difficile au sens fort, c'est-à-dire qu'il fait partie de la classe des problèmes les plus difficiles.

La deuxième évaluation d'un algorithme est la valeur du ou des critères de la solution fournie. Comme pour le coût en calcul, l'évaluation peut être faite dans le pire des cas ou en moyenne. Dans le pire des cas, il est possible de définir des majorations absolues ou relatives ainsi que des ratios de performance.

Cette deuxième évaluation n'a d'intérêt que pour les problèmes multicritères et pour les algorithmes approchés. Les mesures correspondantes ont les mêmes défauts que celles qui sont basées sur le coût en calcul. Ce ne sont donc que des indicateurs qui permettent partiellement d'apprécier des algorithmes.

Pour E.A. SILVER, R.V.V. VIDAL et D. DE WERRA ([1980]) une heuristique peut être qualifiée de bonne si :
(1) la durée des calculs nécessaires est "réaliste",
(2) la valeur du critère est en moyenne proche de l'optimum,
(3) la probabilité d'avoir une très mauvaise valeur pour le critère est faible,
(4) elle est aussi simple que possible et donc facile à comprendre par l'utilisateur.

Le terme "réaliste" dépend de la taille maximale des problèmes abordés et du délai autorisé pour fournir la solution : par exemple, les algorithmes exponentiels sont inutilisables en temps réel pour des problèmes de taille industrielle.

Si les problèmes sont NP-difficiles, on est obligé de se contenter d'algorithmes approchés et d'utiliser des heuristiques polynômiales. Leurs performances respectives sont comparées par simulation.

1.3.3. Approches de résolution

Que l'on cherche des algorithmes exacts ou approchés, une question se pose : comment trouver ou construire ces algorithmes ou un ensemble d'algorithmes qui fourniront les informations nécessaires à la prise de décision?

Nous allons présenter dans la suite de ce paragraphe quelques grandes approches qui sont à l'origine de nombreux algorithmes, c'est-à-dire :

- l'approche par analogie des modèles,

- l'approche par modification des modèles :
  * réduction
  * extension,
  * linéarisation,
  ...

- l'approche par décomposition
  * hiérarchique,
  * structurelle,
  * spatiale,
  * temporelle,
  * de l'ensemble des solutions,
  ...

- l'approche par construction et exploration :
  * aléatoire,
  * dirigée par des empirismes,
  * utilisant le recuit simulé,
  * arborescente,

- l'approche par enrichissement des connaissances sur le problème :
  * analyse sous contraintes,
  * simulation et études analytiques,
  * systèmes experts et intelligence artificielle.

1.3.3.1. Approche par analogie des modèles

Les problèmes d'ordonnancement ne sont que des cas particuliers des problèmes de combinatoires. En modélisant certains d'entre eux, on retrouve des modèles déjà examinés par les chercheurs opérationnels. Il est alors possible d'utiliser les algorithmes exacts ou approchés, polynômiaux ou exponentiels, déjà développés.

C'est le cas en particulier pour :

- les problèmes d'ordonnancement à moyens illimités qui sont résolus depuis près de 30 ans par des algorithmes polynômiaux de cheminement ([P.E.R.T. 1958], [J.E. KELLEY / M.R. WALKER 1959], [B. ROY 1960],[M.L. DIBON 1970]) ;

- les problèmes d'affectation et de transport, ainsi que les problèmes où la durée des tâches varie linéairement avec le coût des moyens affectés, qui sont résolus efficacement par les méthodes primal-dual de la programmation linéaire ([H.W. KÜHN 1955], [L.R. FORD / D.R. FULKERSON 1956 et 1957], [D.R. FULKERSON 1961], [K.R. BAKER 1974]).
Quelques problèmes d'ordonnancement d'atelier se ramènent également à la recherche de flots dans un graphe et utilisent les méthodes primal-dual ([E.L. LAWLER 1964],[W.A. HORN 1973]);

- les problèmes d'ordonnancement, où les temps de réglage ou les coûts d'interruption dépendent de la séquence ([R.G. PARKER / R.H. DEANE / R.A. HOLMES 1977]), ou bien où les en-cours sont totalement interdits ([K.R. BAKER 1974]), qui se ramènent à un problème de voyageur de commerce ;

- des problèmes particuliers d'emploi du temps qui se ramènent à des problèmes de coloration dans un graphe [J.C. HERZ 1966],[D.C. WOOD 1969]) ; problèmes de coloration qu'on peut traiter par des heuristiques de classification automatique ([J. THEPOT 1980] ,[J. THEPOT / G. LECHENAULT 1980 et 1981]);

- plus généralement, en ajoutant des inconnues et des contraintes, il est souvent possible de modéliser un problème d'ordonnancement sous forme de programmation linéaire en nombres entiers ou P.L.E. ([H. MÜLLER-MERBACH 1974], [R.S. STAINTON 1978]). Il suffit alors d'utiliser des méthodes générales de résolution des P.L.E. ([R.E. GOMORY 1958], [A.M. GEOFFRION / R.E. MARSTEN 1972]). Certains problèmes conduisent à des modèles de P.L.E. comprenant des contraintes disjonctives ([E. BALAS 1976 et 1977]).

Malheureusement, dans ces trois derniers cas, le problème est NP-difficile et on retrouve pour sa résolution des approches analogues à celles développées pour les problèmes d'ordonnancement (et de combinatoire) en général.

Une approche un peu plus générale présentée par Heiner MULLER-MERBACH ([1974]) propose de s'intéresser aux sous-modèles standards inclus dans le modèle. Comme sous-modèle standard, il distingue :

- les problèmes de <u>séquencement</u> où l'on recherche un ordre total ou partiel ;

- les problèmes de <u>sélection</u> où l'on cherche un meilleur sous-ensemble dans un ensemble ;

- les problèmes d'<u>affectation</u> où l'on met en correspondance biunivoque deux sous-ensembles.

La présence de ces sous-modèles informe sur la complexité des problèmes posés et suggère des pistes pour des approches de résolution par décomposition.

1.3.3.2. Approche par modification des modèles

- Réduction

Pour diminuer le coût des algorithmes, on ajoute des contraintes supplémentaires de manière à réduire artificiellement le domaine des réalisables.

- Extension

On plonge le problème posé dans un problème plus général qui se prête mieux à certaines méthodes de résolution : c'est une des bases de la programmation dynamique que nous verrons plus en détail en 1.4. Ou encore, on utilise une méthode, optimale sous certaines hypothèses, pour des problèmes plus généraux où ces hypothèses ne sont pas vérifiées.

- Linéarisation

On approche, par exemple, la fonction objectif par une fonction linéaire, ou par une fonction linéaire par morceaux ([L.N. VAN WASSENHOVE / K.R. BAKER 1982]) ce qui nécessite des changements de variables pour se ramener totalement à un problème linéaire.

On peut encore changer la nature des lois de distribution des variables aléatoires pour se ramener à des modèles pour lesquels il existe des résultats théoriques, ou encore agréger des variables, ou des contraintes,...

1.3.3.3. Approche par décomposition

- Décomposition hiérarchique

C'est une méthode traditionnelle en gestion de production. Elle consiste à lier la finesse du découpage en tâches et de l'agrégation des données à la longueur de l'horizon de planification considéré.

On obtient alors quatre niveaux d'ordonnancement ([V. GIARD 1981]) :
- ordonnancement à long terme,
- ordonnancement à moyen terme,
- ordonnancement à court terme,
- ordonnancement à très court terme.

Bien que désigné dans certains ouvrages sous le terme d'ordonnancement, les problèmes du long terme ne sont pas des problèmes d'ordonnancement au sens où on les a décrit dans ce travail. Plutôt que de répondre aux questions Quand ? et Comment ?, on répond aux questions Avec quels moyens ? et Pour faire quoi ? En quelle quantité ?

Au niveau moyen terme, on répond bien aux questions Quand ? et Comment ? afin de prendre des décisions tactiques d'équilibrage de la charge par rapport aux limites de capacité.
A ce niveau, on lisse les flux de production de manière à respecter les contraintes de moyens, mais en travaillant sur des entités agrégées : familles de produits (sans distinguer les variantes) et sous-systèmes de production. On n'entre pas dans les détails techniques de la réalisation précise de la fabrication ou des tâches de transport.

Au niveau du court terme, on s'intéresse aux informations élémentaires : produits, avec toutes les variantes possibles, opérations et machines. A ce niveau, la réponse aux questions Quand ? et Comment ? peut prendre des formes diverses et ne pas être complète de manière à laisser une marge de décision au très court terme ; ce peut être :

* des règles à appliquer en temps réel pour piloter la fabrication ;
* des solutions partielles précisant avec quels moyens, mais laissant des marges de liberté pour les décisions temporelles ;
* des solutions complètes précisant quand et avec quel moyen chaque opération sera effectuée ;
* des solutions complètes enrichies par des choix possibles de variantes compatibles avec les objectifs.

Au niveau du très court terme, on suit la réalisation en prenant les dernières décisions dans le cadre des contraintes techniques et des contraintes issues des niveaux précédents. C'est à ce niveau qu'il faut réagir face aux perturbations aléatoires, quitte à remettre en cause les décisions des niveaux supérieurs.

Cette décomposition est communément utilisée dans les entreprises. Elle est intégrée dans les progiciels de gestion de production construits suivant la méthode M.R.P. "Matérial Requirement Planning" ([V. GIARD 1981], [M. CROUHY 1983], [J. ORLICKY 1975]). Mais tous les niveaux ne sont pas toujours développés dans ces progiciels ([G. DOUMEINGTS / D. BREUIL / L. PUN 1983],[J.P. KIEFFER /S. HAMICHI 1986]).

Dans cette décomposition hiérarchique, les décisions de chaque niveau deviennent des contraintes pour les niveaux inférieurs, les impossibilités à un niveau donné remettent en cause les décisions du niveau supérieur ([J. MELESE 1972]).
En outre, comme chaque niveau possède un horizon différent, les problèmes des niveaux inférieurs apparaissent plusieurs fois avant que l'horizon du niveau supérieur ne change. Pour mieux ajuster les niveaux de la hiérarchie, on utilise la technique du "plan glissant", qui consiste à recalculer les décisions sur l'horizon supérieur chaque fois que l'on termine un horizon du niveau inférieur.

Dans ce travail, nous nous intéressons aux problèmes d'ordonnancement qui se posent au niveau du court terme, et même du très court terme, dans la mesure où, s'il existe plusieurs machines capables d'effectuer une opération donnée, nous supposons que le choix de la machine a été fait au préalable.

- Décomposition structurelle

Elle consiste à utiliser la structure du problème pour décomposer le problème en sous-problèmes liés entre eux.

Par exemple ([B. ROY 1970]), on part d'un problème d'ordonnancement de type (T,W). Dans un premier temps, on suppose que les moyens sont en quantité suffisante et on résout un problème de type (T). On suppose alors la localisation temporelle connue et on résout un problème de type (W). S'il n'y a pas de solutions réalisables, on résout à nouveau un problème de type (T) comportant des contraintes supplémentaires, et ainsi de suite.

Un autre exemple, concernant l'emploi du temps d'une session d'examens, utilise l'existence de sous-modèles inclus. Si l'on néglige les moyens en salles, le problème de type (T,W_1) est un problème de coloration. Si l'on suppose la localisation temporelle T connue, le problème de type (T,W_2) est un problème d'affectation. Si l'on n'est pas satisfait, on modifie les contraintes du premier problème, et on recommence.

- Décomposition spatiale

Elle consiste à découper le problème en sous-problèmes par la création de groupes d'exemplaires de moyens, groupes de machines ou îlots de fabrication pour les problèmes d'ordonnancement d'ateliers, et de groupes de tâches ou groupes de produits.

Si à chaque groupe de machines correspond un groupe de produits et un seul, alors on a mis en évidence des sous-problèmes indépendants plus petits et donc de résolution moins coûteuse.
Si non, on découpe chaque produit en autant de sous-produits que de groupes de machines. On ajoute des contraintes pour rendre les sous-produits indépendants. On peut alors ordonnancer indépendamment chaque groupe de machines.
Si la solution globale obtenue n'est pas réalisable, on modifie les contraintes et on reprend.

Dans ce travail, nous développons des méthodes utilisant cette décomposition, elles seront longuement développées au cours des chapitres suivants.

- Décomposition temporelle

Elle consiste à découper le problème en sous-problèmes par la création de groupes de tâches que l'on ordonnance successivement en progressant le long de l'axe du temps.

M. YAMAMOTO [1977] utilise une telle méthode pour minimiser la durée totale dans un "job-shop". Il ordonnance des paquets de travaux qu'il place les uns après les autres à partir des dates de disponibilité des machines à la fin du paquet précédent.

Dans ce travail, nous développons des méthodes utilisant une décomposition temporelle différente. Nous les combinons avec les méthodes de décomposition spatiale. Ces méthodes seront présentées et évaluées dans les chapitres suivants.

- Décomposition de l'ensemble des solutions

Il s'agit d'une approche générale dans laquelle on considère l'ensemble des solutions d'un problème et où l'on décompose cet ensemble en sous-ensembles plus petits plus faciles à explorer. Cela conduit aux procédures par séparation et évaluation ou "branch and bound". Nous les présentons plus longuement en 1.4. Presque tous les articles dont le titre annonce une méthode de décomposition pour des problèmes d'ordonnancement développent ce type de décomposition, d'autres présentent ainsi des méthodes utilisant la programmation dynamique.

1.3.3.4. Approche par construction et exploration

Pour cette approche, deux familles de méthodes existent :

* les méthodes de construction progressive de l'ordonnancement: à chaque étape, on prend une décision pour le placement d'une nouvelle tâche ;

* les méthodes par voisinage: à chaque étape, on passe d'un ordonnancement réalisable à un autre ordonnancement réalisable "voisin", le voisinage pouvant se traduire, par exemple, par la permutation de deux tâches.

Et dans chaque famille, les méthodes se distinguent par la manière de prendre des décisions.

- A chaque étape, la décision peut être aléatoire. On tire au sort la nouvelle tâche à placer parmi les tâches accessibles, ou la permutation à retenir parmi celles qui améliorent le critère.

- A chaque étape, la décision peut être dirigée par des règles empiriques. On choisit, par exemple parmi les tâches accessibles, celles de durée minimale, ou la permutation qui améliore le plus le critère (méthode de plus forte pente).

- Dans le cas des méthodes par voisinage, il est possible de mêler aléatoire et empirisme de la manière suivante :
en début d'itération, on dispose d'un ordonnancement réalisable, ou solution actuelle ; on tire au sort parmi des ordonnancements voisins obtenus par permutation de tâches (à partir de la solution actuelle), qu'ils améliorent ou non le critère par rapport à la solution actuelle ; dans le cas où le critère est dégradé, on accepte de prendre l'ordonnancement tiré au sort comme nouvelle solution actuelle avec une probabilité qui dépend de l'importance de la dégradation et d'une quantité T appelée "température". La probabilité d'accepter une dégradation du critère est d'autant plus grande que la dégradation est faible et que la température est élevée. La méthode fait baisser progressivement la température jusqu'à 0 où la probabilité d'accepter une dégradation est nulle. Ainsi, en début d'algorithme, la probabilité est faible de s'arrêter sur un optimum local (non optimum global) et, en fin d'algorithme, on termine par une "méthode de descente".
Ces méthodes ont été imaginées par analogie avec des phénomènes de thermodynamique et portent le nom de "recuit simulé" ou "simulated anealing" ([S. KIRKPATRICK / C.D. GELATT / M.P. VECCHI 1982]). Elles ont été utilisées en ordonnancement ([L. DUPONT 1986].

Enfin, pour ce type d'approche par construction ou par voisinage, on peut construire des "algorithmes gloutons", où toute décision prise est définitive, et des "algorithmes avec retour arrière", ou "backtracking", plus ou moins limités dans leur souplesse. Dans ce dernier cas, on accepte de revenir à un point antérieur où l'on a pris une décision, et de modifier cette décision. L'ensemble des chemins représentant les décisions successives prend la forme d'une arborescence qui permet d'explorer par construction ou par voisinage un sous-ensemble plus grand de solutions. Si, pour toute décision, on essaie tous les choix possibles, on obtient une exploration explicite de toutes les solutions. En général, on limite l'ensemble des choix et on obtient une exploration partielle explicite de solutions. Dans ce cas, on obtient des méthodes heuristiques.

1.3.3.5. Approche par enrichissement des connaissances sur le problème

** Analyse sous contrainte*

Cette méthode d'approche a été développée par le laboratoire LAAS de Toulouse depuis 1976 ([J. ERSCHLER 1976]). Partant du constat que les problèmes d'ordonnancement sont par essence dynamiques, avec des données incertaines et des produits nouveaux qui surviennent à chaque instant, les auteurs considèrent qu'il est vain d'optimiser au sens habituel du terme. Par contre, pour éviter que certains critères ne prennent des valeurs trop mauvaises, on s'arrange pour encadrer la fonction objectif. De cette façon, le modèle ne comporte que des contraintes et pas de critère, du moins de explicitement. Souvent, les contraintes sont issues des niveaux supérieurs de la décomposition hiérarchique.

Le problème consiste à caractériser l'ensemble des solutions admissibles, c'est-à-dire des solutions qui obéissent aux contraintes. S'il est vide, on remet en cause les décisions prises aux niveaux supérieurs de la décomposition hiérarchique. Dans le cas contraire, on cherche à en extraire des sous-ensembles de solutions avec des variantes locales qui permettront aux utilisateurs de réagir aux aléas, dans les limites imposées par les containtes.

Considérons, par exemple, le cas du problème à une machine, où tout produit i est caractérisé par sa date de disponibilité r(i), la durée de fabrication p(i), et une date impérative de fin d'exécution au plus tard d(i). J. ERSCHLER, G. FONTAN, C. MERCE et F. ROUBELLAT ([1983]) ont développé des notions de dominance entre les séquences de produits. Un théorème leur permet alors de caractériser un sous-ensemble de séquences dominantes pour les vecteurs r et d fixés, quel que soit le vecteur p. Cela permet de fournir des séquences réalisables qui sont indépendantes des aléas sur les durées de fabrication.

Ce concept de dominance permet de diminuer la taille de l'ensemble des solutions réalisables, J. ERSCHLER, G. FONTAN et C. MERCE ([1985]) ont utilisé cette possibilité, dans le contexte de l'approche optimisante, pour accélérer la procédure par séparation et évaluation de K.R. BAKER et Z. SU ([1974]).

Dans le cas des ateliers de type "job shop" avec machines non reliées en parallèle (cf. 1.2.2.1. a)), F. ROUBELLAT et V. THOMAS ([1987]) définissent des "séquences de groupes d'opérations admissibles". Un groupe d'opérations associé à une machine est dit admissible s'il peut conduire à un ordonnancement admissible quel que soit la permutation de ses éléments. Une séquence de groupes d'opérations est admissible si toute solution qui en est issue par permutation est admissible. C'est la séquence de groupes admissible, calculée "off line", qui est fournie à l'atelier. La permutation des opérations à l'intérieur de chaque groupe est choisie "on line" en temps réel en fonction des perturbations locales. Un algorithme a été mis au point pour rechercher une séquence de groupes admissibles. Il repose sur les principes de scission, transfert et regroupement.

** Simulation (et études analytiques)*

Ce ne sont pas des méthodes d'ordonnancement. Néanmoins, on les trouve dans certains progiciels d'ordonnancement d'atelier pour choisir entre plusieurs règles d'ordonnancement (en temps réel).

Dès l'instant où les algorithmes utilisés sont des méthodes approchées, la simulation de leur comportement sur des exemples réels ou sur des exemples générés aléatoirement apporte une connaissance (limitée aux hypothèses des expériences faites) de leurs performances respectives ([W. GERE 1966]), [G.H. JONES 1973], [S.S. PANWALKAR / W. ISKANDER 1977], [J.H. BLACKSTONE / J.R. PHILIPS / G.L. HOGG 1981]).

Ces simulations peuvent être réalisées simplement en utilisant des langages informatiques traditionnels. Il est également possible d'utiliser des modèles et des langages plus élaborés et conçus pour la simulation (voir par exemple [G. BEL 1984], [M. AMMAR 1984]).

En particulier, les problèmes d'ordonnancement peuvent être modélisés avec des réseaux de files d'attente ([E.G. GELENBE / G. PUJOLLE 1982]) ou avec des réseaux de Pétri temporisé ([J. CARLIER / P. CHRETIENNE / C. GIRAULT 1983], [P. CHRETIENNE 1984] ou encore d'autres modèles permettant de décrire le déroulement de processus.

Des progiciels ont été développés proposant des langages de description de ces modèles et des outils permettant la simulation de "scénarios" sur les systèmes décrits (par exemple, GPSS ([J.R. SULZER / P. BOUTEILLE / P. ARQUE 1970]), SLAM ([A.A.B. PRITSKER / C.D. PEGDEN 1979]), QNAP ([D. POTIER 1983])). Des études ont été réalisées pour essayer de comparer leurs performances sur des problèmes issus de la gestion de production (voir par exemple [A. ALANCHE / B. ANCELIN 1984]).

Ces outils permettent, d'une part d'intégrer des contraintes plus complexes que ne le permettent les modèles mathématiques (par exemple, nombre de places de stockage limité, description détaillée des systèmes de transport, de manutention ou d'alimentation en outils,...), d'autre part d'étudier le comportement du système en fonction des règles de pilotage ou en fonction des ressources disponibles.

Sous réserve que les hypothèses faites (carnets de commandes, taux de panne, vitesses des déplacements,...) traduisent bien le problème réel, les simulations enrichissent les connaissances du décideur sur les conséquences de ces choix.

Comme de nombreux chercheurs et industriels, nous avons utilisé cet outil, dans le cadre d'une recherche sur le pilotage des ateliers flexibles ([C. BERTHUL / M.C. PORTMANN 1985]. Pour concevoir un système de pilotage d'atelier en présence de pannes longues, nous avons utilisé une approche hiérarchique à deux niveaux pour le court terme ; au niveau supérieur, nous avons modélisé à l'aide de la programmation linéaire une affectation de charge de travail aux différentes périodes ; au niveau bas, nous avons simulé dans le détail des stratégies de pilotage pour vérifier que les décisions prises au niveau haut étaient réalisables et donnaient des valeurs raisonnables aux différents critères.

La simulation est également utilisée pour initialier la mémoire artificielle du système de pilotage que l'on présente ci-après.

Si on prend des hypothèses adéquates (arrivées poissonniennes, temps de service exponentiels,...), il est possible d'obtenir des informations sur le comportement moyen des systèmes décrits, c'est le "mean value analysis" (voir par exemple [J.B. CAVAILLE / D. DUBOIS 1982] et [Y. DALLERY 1984]). Il permet d'évaluer les performances et de dimensionner des systèmes (choix des quantités de chaque ressource) et non d'ordonnancer.

** Systèmes experts et intelligence artificielle*

Il est difficile, en quelques paragraphes, de synthétiser les grandes lignes de l'intelligence artificielle et des systèmes experts et de présenter leur utilisation en ordonnancement (voir par exemple [K.G. KEMPF 1985], [P..RAYSON 1985], [K. KAZANTZIDIS 1985], [M.S. STEFFEN 1986]).
La plupart des produits sont seulement en phase de recherche et n'ont pas été testés dans des entreprises.

ISIS ([M.S. FOX / S.F. SMITH 1984]) est utilisé dans un atelier de la firme Westinghouse. Il utilise la technique de séparation de la "base de connaissances" et de la "base de faits" propre à l'intelligence artificielle. La base de faits contient l'environnement du problème réel y compris des contraintes "floues". La base de connaissances contient des "règles de production". Le "moteur d'inférence" est remplacé par la simulation du comportement d'un "ordonnanceur" travaillant sur un diagramme de Gantt. Le système ne comporte pas de possibilité d'apprentissage lui permettant de complèter la base de faits de connaissances déduites ou d'expériences passées.

L'utilisation de la "mémoire artificielle" ([J.M. PROTH / J. QUINQUETON/ H. RALAMBONDRAINY 1983], [E. BECHLER / J.M. PROTH / K. VOYIATZIS 1984]) est une manière d'intégrer un processus d'apprentissage dans un système de pilotage d'atelier. En raison de la grande quantité d'informations liées aux états successifs de l'atelier et aux multiples choix possibles pour le pilotage, il est nécessaire d'utiliser des techniques d'analyse des données de manière à réduire le volume de ces informations.

On considère alors des classes d'états du système, des règles d'ordonnancement et des objectifs (exprimés ici sous la forme : faire croître ou décroître la valeur d'un critère).

La mémoire artificielle contient, pour toute classe d'états E et pour toute règle d'ordonnancement R, l'espérance mathématique de la valeur des critères ou de leur variation A(E,R,T) lorsque l'on applique à un état e de E la règle R pendant l'intervalle de temps T.
La mémoire artificielle est initialisée par simulation. Des informations statistiques viennent enrichir la base de connaissance au fur et à mesure du déroulement du pilotage en temps réel.

Pour prendre une décision de pilotage, on considère un état du système s. On cherche si s est suffisamment proche de l'état représentatif e d'une classe d'états E. Si oui, on suppose que s a le même comportement que e face aux différente règles R (comportement décrit par A(E,R,T)) et on choisit R en fonction des objectifs recherchés. Sinon, il faut utiliser des méthodes classiques pour choisir la règle R et il devient nécessaire de "rafraîchir" la mémoire artificielle ; cela est fait "off line" ; des simulations sur les états non rattachables à des classes permettent de les valuer et la classification automatique fournit de nouvelles classes d'états plus conformes à l'évolution actuelle du système de production et de sa charge. L'utilisation de la mémoire artificielle est illustrée par la figure 1.3.3.5.

L'analyse sous contraintes présentée au début de ce paragraphe peut devenir un élément de base de la construction d'un système expert. J. ERSCHLER et P. ESQUIROL ([1987]) ont ainsi utilisé PROLOG pour réaliser MASCOT, un module d'analyse sous contraintes utilisant la programmation logique, qui peut servir à générer des ordonnancement réalisables.

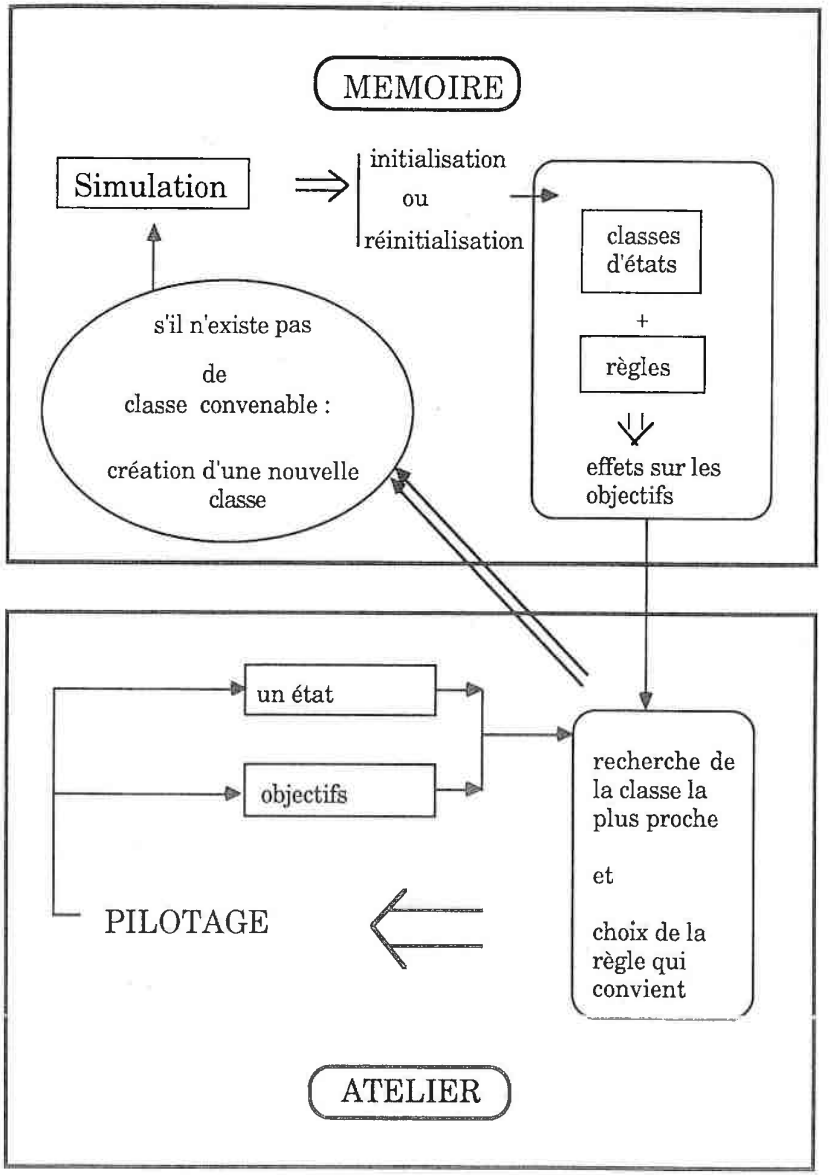

MEMOIRE
Simulation
initialisation
ou
réinitialisation
classes d'états
+
règles
effets sur les objectifs
s'il n'existe pas
de
classe convenable :
création d'une nouvelle classe
un état
objectifs
PILOTAGE
recherche de la classe la plus proche
et
choix de la règle qui convient
ATELIER

figure 1.3.3.5.

1.4. METHODES DE RESOLUTION EXISTANTES

1.4.1. Introduction

En 1.3., nous avons présenté succinctement les différentes approches possibles pour construire des algorithmes.

Nous allons maintenant détailler deux approches particulières : la programmation dynamique et les procédures par séparation et évaluation.
Ce sont des techniques générales de construction d'algorithmes très utilisées en optimisation combinatoire et tout particulièrement en ordonnancement.
Nous les avons utilisées dans le cadre de ce travail.

La littérature les concernant est très abondante. Aussi, nous avons surtout cherché à citer les articles de base, ainsi que ceux illustrant des points particuliers.

Ces techniques sont des schémas de construction d'algorithmes :
en suivant des règles générales et en les adaptant au cas particulier, on obtient un algorithme qui converge vers une solution vérifiant des caractéristiques choisies.
Les deux schémas conduisent à des méthodes d'énumération implicite :
par opposition à la méthode d'énumération explicite qui consiste à construire successivement toutes les solutions pour retenir celle(s) vérifiant les caractéristiques choisies, les méthodes d'énumération implicite retiennent ou rejettent des sous-ensembles de solutions.
Toutes les solutions seront considérées lors de la recherche, mais certaines seront rejetées sans avoir été construites explicitement car elles appartenaient à des sous-ensembles de solutions moins intéressants que d'autres.

Nous terminerons ce chapitre par l'étude de quelques cas particuliers où il est possible de concevoir des algorithmes exacts polynômiaux.

1.4.2. La programmation dynamique

Elle a été initialement développée par Richard BELLMAN ([R. BELLMAN 1954 et 1957]) pour résoudre des problèmes de chemins extrêmaux.
Le principe fondamental de l'optique de Pierre de FERMAT (1629) "la lumière suit des chemins extrêmaux" est appliqué à la théorie des graphes et devient le principe d'optimalité de BELLMAN :

> si C appartient à un chemin minimal (resp. maximal) allant de A à B, alors les sous-chemins de ce chemin allant de A à C et de C à B sont minimaux (resp. maximaux).

La démonstration est immédiate par l'absurde (elle utilise le fait que l'addition de la longueur des arcs du chemin est associative).

On utilise en fait le principe de Bellman sous la forme du corollaire suivant :

> si le chemin de A à C (resp. de C à B) est imposé, alors le plus court (ou le plus long) chemin de A à B utilisant la partie imposée AC (resp. CB) est obtenu en complétant la partie imposée par un sous-chemin optimal allant de C à B (resp. A à C).

Ce corollaire, utilisé de manière séquentielle avec des parties imposées d'un seul arc, permet d'obtenir des formules de récurrence pour la recherche de chemins extrêmaux.
Ce sont ces méthodes de calcul qui sont utilisées dans les méthodes d'ordonnancement à chemins critiques ([P.E.R.T. 1958], C.P.M. [J.E. KELLEY / M.R. WALKER 1959], méthodes de potentiels [B. ROY 1960]).
Par la suite, des méthodes analogues construites à partir de formules de récurrence ont été développées pour résoudre des problèmes :

- d'investissement ([R. FAURE 1974], [A. LAHRICHI / M. LAHRICHI 1980]),
- de gestion de stocks déterministes ou stochastiques ([M.E. VENTURA 1960], [A. KAUFMANN 1968], [A. BENSOUSSAN / J.M. PROTH 1981 et 1984],[V.I. LEOPOULOS / J.M. PROTH 1985]),
- de contrôle optimal (R. BELLMAN 1961]),
- de reconnaissance des formes et de la parole ([F. CHARPILLET / J.P. HATON / J.M. PIERREL 1984]),

- du voyageur de commerce ([N. CHRISTOFIDES 1975]),
- de coloration des graphes ([N. CHRISTOFIDES 1975],
- d'ordonnancement à une machine ([K.R. BAKER 1974], [E.G. COFFMAN 1976], ...

La modélisation de ces problèmes sous forme de cheminement dans un graphe de décision est en général possible. Ceci pourrait permettre une présentation unifiée de ces méthodes. En fait, les transformations des modèles particuliers pour obtenir le graphe de décisions ne facilitent pas la compréhension et la présentation des algorithmes spécifiques à programmer.

Nous proposons une présentation générale de la programmation dynamique non liée à l'application initiale de recherche de chemins extrémaux.

Nous considérons que la première phase de la méthode doit être la recherche d'une formulation récursive de la définition des solutions du problème.

La seconde phase doit tranformer la formulation récursive en formulation récurrente.

L'application de la formule de récurrence, étape par étape, doit supprimer tout calcul redondant et fournir, à la dernière étape, la valeur de l'objectif de la solution optimale. La remontée des calculs, à partir de repères ou marques qui ont été conservés à chaque étape, permet de retrouver, pour la ou les solutions optimales, les décisions qui ont été retenues à chaque étape. L'ensemble de ces décisions décrit les solutions optimales correspondantes.

Toute la difficulté de la programmation dynamique consiste à trouver la formulation récursive et le découpage en étapes qui conduira à la formulation récurrente. La plupart du temps, le problème posé n'a pas de formulation récursive ou récurrente et il faut le plonger dans un problème plus général qui aura cette propriété : c'est ce que l'on appelle l'extension.
L'extension consiste à définir une famille de problèmes à résoudre parmi lesquels figure le problème initial ; par exemple, en théorie des graphes, pour trouver le plus court chemin du point x_0 au point y_0, on cherche les plus courts chemins de tout point x à y_0, x_0 étant un des x. On peut alors construire une formule de récurrence sur l'ensemble des plus courts chemins de x à y_0.

La formule de récurrence porte sur la fonction objectif du problème étendu. Pour que la fonction objectif puisse se mettre sous forme récurrente, il faut qu'elle puisse se décomposer en parties élémentaires et complémentaires (par addition ou multiplication, par exemple) étape par étape.

En résumé, les démarches à effectuer pour construire un algorithme de programmation dynamique sont les suivantes:
1) rechercher une formulation récursive de la définition des solutions du problème,
2) si cela n'est pas possible, plonger le problème dans un problème plus général et revenir à 1) sinon poursuivre en 3)
3) la formulation récursive induit une recherche arborescente en profondeur d'abord avec retour arrière. Cette recherche arborescente est aveugle, c'est-à-dire qu'elle ne reconnaît pas les sous-arborescences identiques.
En organisant les calculs niveau par niveau, on peut :
- les exécuter en partant des feuilles de l'arborescence,
- les regrouper de telle sorte que les sous-arborescences identiques ne soient évaluées qu'une fois.

C'est le rôle du découpage en étapes.
4) Les calculs ne fournissent que la valeur de l'objectif des solutions optimales. Une descente rapide dans l'arborescence représentée par des repères permet de retrouver les solutions optimales.

La programmation dynamique a été fréquemment proposée pour les problèmes d'ordonnancement. Par exemple, pour les problèmes se ramenant au problème du voyageur de commerce ou à la coloration dans un graphe ([N. CHRISTOFIDES 1975]), ou encore à des problèmes moins classiques de tournées avec contraintes d'horaires ([J. DESROSIERS / P. PELLETIER / F. SOUMIS 1983]). Mais, elle a été essentiellement utilisée pour des problèmes de séquencement à une machine ([L. SCHRAGE / K.R. BAKER 1978 a) et b)], [C.N. POTTS / L.N. VAN WASSENHOVE 1982]), [E.L. LAWLER 1977], [V. SRINIVASAN 1971], [K.R. BAKER / J.B. MARTIN 1974], [I. NABESHIMA 1971]).
Plusieurs articles proposent tantôt l'extension des méthodes à des cas plus généraux (L. SCHRAGE / K.R. BAKER 1978 a) et b)], tantôt des améliorations par des méthodes de dominance ([V. SRINIVASAN 1971]), ou encore des comparaisons avec d'autres méthodes.

1.4.3. Les procédures par séparation et évaluation

1.4.3.1. Introduction

Afin de présenter un bref historique de ces méthodes, citons Pierre HANSEN ([1971]) : "...des algorithmes ont été proposés pour plusieurs classes particulières de problèmes avant que la généralité de l'approche ne soit reconnue et que les travaux théoriques ne débutent. Le premier exemple développé de P.S. est l'algorithme proposé en 1960 par A. LAND et A. DOIG pour les programmes mixtes ; en 1963, l'article de J.D.C. LITTLE, K.G. MURTY, D.W. SWEENEY et C. KAREL , utilisant une P.S. pour résoudre le problème du voyageur de commerce, eut un grand retentissement ; dès lors, les principes des P.S. étant bien dégagés, le nombre d'applications a crû rapidement. P. BERTIER et B. ROY ont donné en 1964 une première axiomatique des P.S. applicable à des problèmes de nature combinatoire... ... Récemment, B. ROY ([1970 et 1970 a)]) a modifié un des axiomes proposés..."
Dans la suite de son historique Pierre HANSEN cite encore : E. BALAS [1968], L.G. MITTEN [1970], P. HERVE [1968], N. AGIN [1966], E. LAWLER et D. WOOD [1966], M. BALINSKI [1965 et 1967] et M. BALINSKI et K. SPIELBERG [1969].

Comme nous pouvons le constater dans toutes ces références, de nombreux articles présentent les axiomatiques de ces méthodes ([P. BERTIER / B. ROY 1964], [B. ROY 1970 et 1970 a] , [P. HANSEN 1971], [E. BALAS 1968], [L.G. MITTEN 1968], [P. HERVE 1968], [N.AGIN 1966], [E.L. LAWLER / D.E. WOOD 1966]). Des travaux plus récents généralisent l'axiomatique à toutes les recherches arborescentes y compris les méthodes approchées ([H. MÜLLER-MERBACH 1974]), la programmation dynamique et les procédures par séparation et évaluation étant vues comme des cas particuliers de recherche arborescente.

Nous allons présenter au paragraphe 1.4.3.2. les procédures par séparation et évaluation de la même manière que nous avons présenté la programmation dynamique, c'est-à-dire en partant de définitions récursives des solutions du problème. Ce point de départ explique la présentation unifiée de H. MULLER MERBACH ([1974]) puisque toute définition récursive engendre une recherche arborescente.

Nous illustrerons ensuite au paragraphe 1.4.3.3. les variantes possibles en nous appuyant essentiellement sur des exemples appliqués à des problèmes d'ordonnancement.

1.4.3.2. Présentation générale

Contrairement à la programmation dynamique où la formulation récursive est souvent difficile à trouver, elle vient naturellement pour les procédures par séparation et évaluation.

Nous appelons S l'ensemble de toutes les solutions du problème posé. Nous remplaçons S par une suite $S_1, S_2, \ldots, S_p$ ($p \geq 2$) de sous-ensembles de solutions du problème qui recouvrent ou partitionnent S .
Si l'on connaît la meilleure solution de chacun de ces sous-ensembles S_i , il suffit de retenir la meilleure de toutes ces solutions pour obtenir la meilleure solution de S .

Le problème initial est ainsi remplacé par p sous-problèmes identiques et indépendants de taille inférieure.

Si, pour les résoudre, on utilise le même procédé de séparation, on obtient une formulation récursive de la solution du problème. Comme toute définition récursive, il est nécessaire de prévoir les cas où l'on remplace l'appel récursif par un calcul non récursif. Cela assure la convergence des calculs sous réserve que les appels successifs conduisent bien aux cas prévus.

Pour les procédures par séparation et évaluation, on cesse d'utiliser l'appel récursif lorsqu'on ne peut plus, ou lorsqu'on ne veut plus, séparer. La méthode la plus simple consiste à s'arrêter lorsque les sous-ensembles ne contiennent plus qu'une solution. Si une solution est unique, elle est à la fois la moins bonne et la meilleure, elle donc nécessairement optimale pour le sous-ensemble qu'elle constitue.

La méthode récursive, ainsi définie, s'arrête si S est fini. Elle fournit la solution optimale du problème posé. Cependant, elle examine toutes les solutions du problème. C'est en fait une énumération explicite par séparation de l'ensemble des solutions.

C'est pour éviter l'exploration systématique par séparation de tous les sous-ensembles de solutions que l'on introduit des valuations associées aux sous-ensembles. Ces valuations peuvent jouer différents rôles dans les algorithmes construits.

Certaines valuations, que nous qualifions de *secondaires* , fournissent des informations pertinentes sur le sous-ensemble considéré. Par exemple :

- quelles contraintes a-t-on ajoutées au problème initial pour définir ce sous-ensemble ?
- quelles contraintes supplémentaires les contraintes précédentes ont-elles induites ?
- reste-t-il encore des possibilités pour introduire de nouvelles contraintes ?
- etc...

Ces informations peuvent être représentées sous des formes simples.
Par exemple, pour le problème du voyageur de commerce, on marque dans la matrice des distances les arcs choisis par une valeur spéciale, et on rend infinies toutes les distances incompatibles avec les choix faits.
Ou encore, pour des problèmes de programmation linéaire en variables 0 et 1, on associe à chaque contrainte un majorant de sa variable d'écart.

Ces informations permettent en particulier d'éliminer des sous-ensembles qui seraient vides de solutions réalisables. Cette anomalie peut se produire si, au départ, on a pris S plus grand que l'ensemble des solutions réalisables.
On reconnaît les sous-ensembles vides de solutions réalisables lorsque, par exemple, la matrice des distances comporte une ligne ou une colonne où tous les éléments sont infinis et donc interdits, ou lorsqu'une majoration d'une variable d'écart d'une contrainte devient négative.

Elles permettent aussi de savoir quand on ne peut plus séparer un sous-ensemble. Par exemple, pour la programmation linéaire en variables 0 et 1, on ne peut plus séparer :

- soit quand un majorant de variable d'écart est négatif : sous-ensemble vide de solutions réalisables,
- soit quand un majorant de variable d'écart est nul : cela impose le choix le plus favorable pour toutes les décisions(*) restant à prendre et conduit à un ensemble vide de solution ou à une seule solution réalisable,
- soit quand toutes les décisions ont été prises, ce qui donne une seule solution réalisable.

Une valuation, que nous qualifions de *principale* , est appelée **borne** du sous-ensemble par tous les auteurs.
L'idéal pour cette valuation serait qu'elle soit toujours égale à la valeur de l'optimum du sous-ensemble considéré.
Cela est obligatoire lorsqu'un sous-ensemble, non rejeté comme intéressant, ne sera plus séparé car on a su en extraire une meilleure solution : on dit alors qu'il est exploré.

C'est le **principe de coïncidence** : pour tout sous-ensemble de solutions exploré S_i , la borne b_i doit être égale à la valeur de l'optimum.

Lorsque le sous-ensemble S_i n'est ni éliminé, ni exploré, la borne doit fournir une surestimation de la valeur de l'optimum dans le sous-ensemble, c'est-à-dire un majorant lorsque l'on cherche un maximum et un minorant lorsque l'on cherche un minimum.
Si l'on suppose que cette surestimation est de bonne qualité, c'est-à-dire qu'elle est proche de la valeur de l'optimum, alors un sous-ensemble qui a une "bonne" borne (petite pour un minorant et grande pour un majorant) a une forte probabilité de contenir des solutions strictement meilleures qu'un sous-ensemble qui a une borne plus "mauvaise" (plus grande pour un minorant et plus petite pour un majorant).

Cette idée intuitive est utilisée dans certaines procédures par séparation et évaluation dites "progressives" (P.S.E.P.), où l'on sépare en priorité le sous-ensemble de meilleure borne.

(*) ici une décision consiste à fixer une variable à la valeur 0 ou 1.

Cela conduit à deux remarques.

Tout d'abord, une procédure par séparation et évaluation est, en général, d'autant plus performante que la borne de chaque sous-ensemble est proche de la valeur de l'optimum. Cela diminue le nombre de sous-ensembles à séparer avant de trouver l'optimum, mais cela augmente le coût de chaque séparation.

D'autre part, il peut être intéressant d'intervenir pour choisir dans quel ordre seront effectués les appels de la formulation récursive.

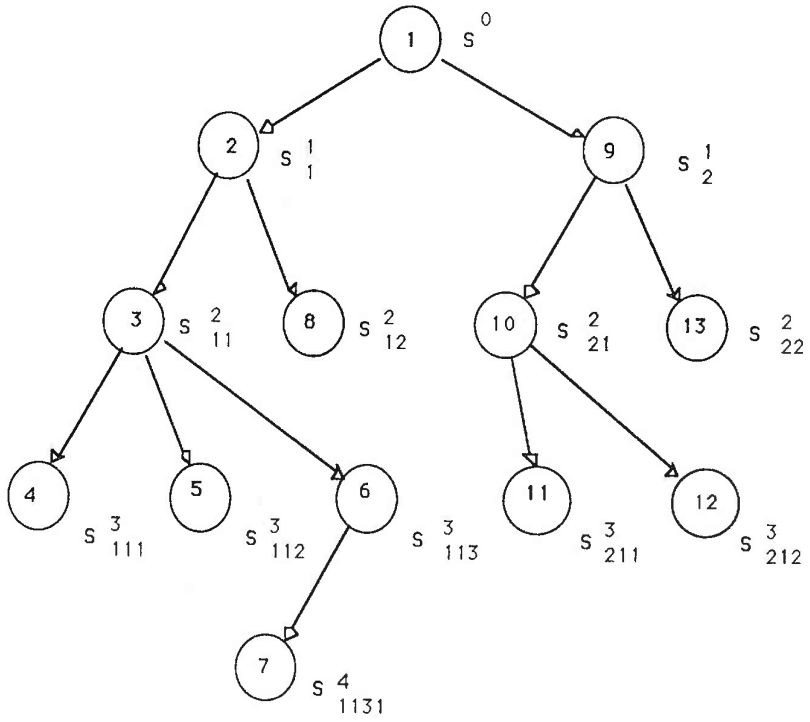
Il y aura autant de procédures par séparation et évaluation différentes pour un problème donné et pour des lois de séparation et d'évaluation retenues que d'ordres d'exécution des appels récursifs choisis.

Si le langage utilisé autorise les procédures récursives, c'est lui qui exécute systèmatiquement les appels récursifs. Il utilise alors un empilement-désempilement des appels dans l'ordre où il les rencontre : on explore l'arborescence des séparations en profondeur d'abord comme le montre le schéma de la figure 1.4.3.2. a.

On voit sur la figure 1.4.3.2. a que l'on sépare toujours le sous-ensemble que l'on vient juste de créer et que l'on construit uniquement le premier sous-ensemble de sa séparation. C'est seulement s'il n'est plus séparable que l'on construit les sous-ensembles suivants dans la même séparation.

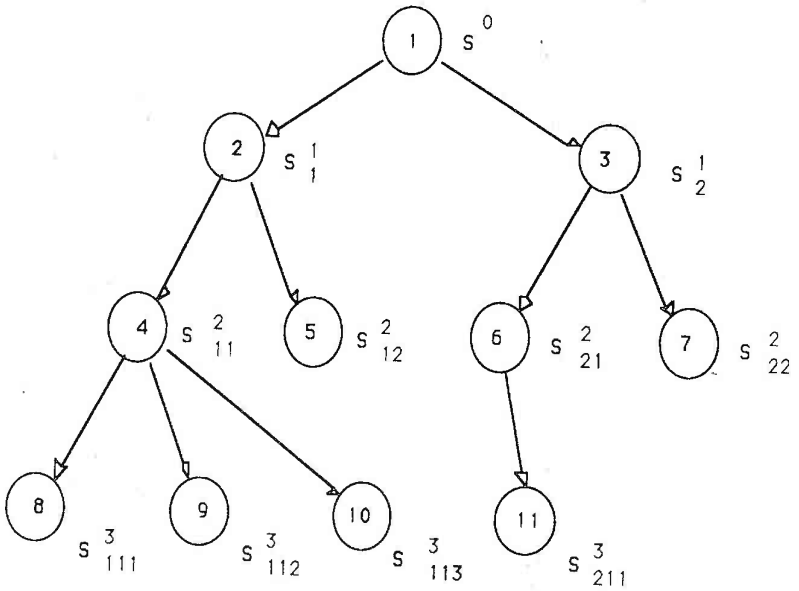
Cette exploration "en profondeur d'abord" fournit une méthode appelée "branch and bound with backtracking" qui est un cas particulier de P.S.E.S. ([B. ROY 1970], [P. HANSEN 1971]), ou procédure par séparation et évaluation SEQUENTIELLE.

Pour tout autre ordre d'exploration de l'arborescence, il est nécessaire de prévoir soi-même le passage de la formulation récursive à une formulation itérative. Ainsi, au lieu d'utiliser systématiquement "une pile en profondeur d'abord" des sous-ensembles en cours de séparation, un choix varié de structures de données pour stocker l'arborescence des séparations permet de construire de multiples procédures par séparation et évaluation différentes.



i $S^{\lambda}_{\sigma\, i}$ signifie que $S^{\lambda}_{\sigma\, j}$ a été le i-ème sous-ensemble créé, obtenu par séparation de $S^{\lambda-1}_{\sigma}$ et est le j-ème sous-ensemble de la séparation.

Figure 1.4.3.2. a P.S.E.S. avec pile "profondeur d'abord"



i $S^{\lambda}_{\sigma\ i}$ signifie que $S^{\lambda}_{\sigma\ j}$ a été le i-ème sous-ensemble créé, obtenu par séparation de $S^{\lambda-1}_{\sigma}$ et est le j-ème sous-ensemble de la séparation.

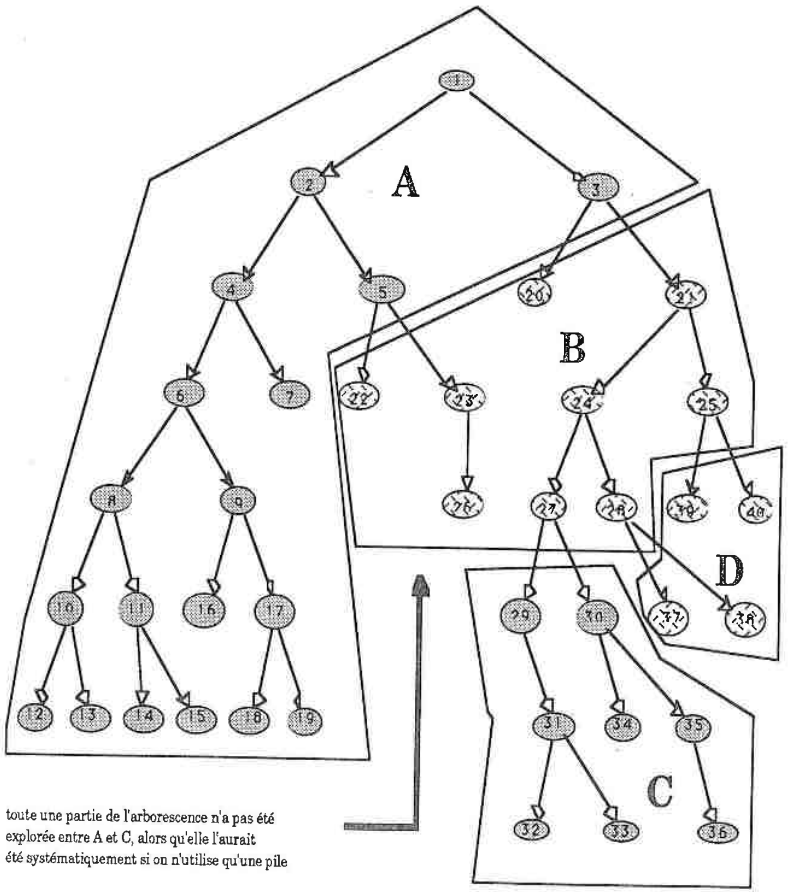
Figure 1.4.3.2. b P.S.E.S. avec file exploration niveau par niveau

Les P.S.E.S. explorent l'arborescence dans un ordre établi a priori. Cet ordre peut être donné par la structure de données utilisée pour ranger les sous-ensembles en attente de séparation.
Par exemple, si l'on choisit pour cette structure de données une file d'attente avec la règle "premier arrivé - premier servi", on obtient l'arborescence des séparations de la figure 1.4.3.2. b . On peut remarquer sur ce schéma que l'arborescence est explorée niveau par niveau. Cela nécessite plus de mémoire que lors d'une exploration en profondeur d'abord.
Il est également possible d'utiliser une pile "en largeur d'abord" : on y remplace le "dernier arrivé" par tous les sous-ensembles issus de sa séparation (alors que dans la pile en "profondeur d'abord" on en choisissait un seul et que dans la file on séparait le "premier arrivé" non encore séparé ou exclu).

Nous avons remarqué précédemment qu'il pouvait sembler judicieux d'utiliser la valeur des bornes pour choisir le sous-ensemble de solutions à séparer qui sera toujours le sous-ensemble non éliminé de plus petit minorant (ou de plus grand majorant). Pour cela on utilise une liste des sous-ensembles à séparer triée par bornes croissantes pour une minimisation (ou décroissantes pour une maximisation). On obtient une méthode "branch and bound with jumptracking" ou une P.S.E.P. : Procédure par Séparation et Evaluation Progressive ([B. ROY 1970], [P. HANSEN 1971]).

Il est à noter qu'une P.S.E.P. conduit plus vite à l'optimum lorsque les bornes sont de bonne qualité, mais que le hasard peut, sur des cas particuliers, favoriser les P.S.E.S. en leur permettant d'obtenir très vite la solution optimale et de savoir qu'effectivement elle est optimale. Par ailleurs, une P.S.E.P. est plus coûteuse en mémoire qu'une P.S.E.S. utilisant une pile en profondeur d'abord.

Aussi introduit-on ([P. HANSEN 1971]) des P.S.E.M. ou Procédures par Séparation et Evaluation Mixtes qui, pour profiter des avantages respectifs des P.S.E.S. et des P.S.E.P., les utilisent en alternance au cours de la recherche.



A "pile en largeur d'abord"

B "meilleure borne"

C "pile en largeur d'abord"

D "meilleure borne"

....

figure 1.4.3.2. c P.S.E.M. "pile largeur d'abord" en alternance avec "liste triée sur meilleure borne"

Dans cette thèse, nous avons programmé une P.S.E.M. qui régulièrement et automatiquement passe d'une liste triée à une pile en largeur d'abord et vice-versa : ceci nous permet, tantôt d'économiser la mémoire tout en explorant le plus possible les feuilles de l'arborescence, tantôt d'élaguer en partant de la racine et en séparant des sous-ensembles de bonne évaluation.
Lorsqu'on passe de la pile à la liste, puis de la liste à la pile, les feuilles que l'on atteint sont en général "éloignées" dans l'arborescence de celles que l'on atteignait auparavant comme l'illustre la figure 1.4.3.2. c ; aussi, au lieu d'avoir des solutions voisines qui ne diffèrent que par les dernières décisions prises au cours des séparations, on obtient de manière "aléatoire" des solutions très différentes.

Afin de terminer la présentation générale des procédures par séparation et évaluation, il reste à montrer comment on utilise la borne b_i associée à chaque sous-ensemble de solutions S_i pour :

- éliminer des sous-ensembles de la recherche,
- reconnaître une solution optimale du problème initial.

Pour simplifier les explications, supposons que l'on recherche un minimum et qu'après un certain nombre de séparations, on ait trouvé le minimum x_k de S_k donnant à l'objectif la valeur b_k, alors :

- tout sous-ensemble S_i tel que $b_i > b_k$ ne contient pas de solution optimale et son exploration doit être abandonnée ;

- tout sous-ensemble S_i tel que $b_i = b_k$ ne contient pas de solution strictement meilleure que x_k et son exploration peut être abandonnée si l'on ne souhaite trouver qu'une seule solution optimale ;

- si toutes les bornes b_i des sous-ensembles S_i à explorer ou explorés sont supérieures ou égales à b_k, alors x_k est optimal.

C'est ce qui permet d'arrêter l'algorithme avec la solution cherchée.

1.4.3.3. Construction de P.S.E.

Pour un même problème, on peut construire de nombreux algorithmes différents qui seront tous des procédures par séparation et évaluation. Ils seront plus ou moins performants selon les exemples numériques traités. Aucun ne sera polynômial. Nous allons les présenter en les distinguant selon :

- la nature de l'ensemble des solutions candidates,
- le principe de séparation choisi,
- la méthode de calcul de la borne b_i ,
- l'ordre dans lequel on effectue les séparations (P.S.E.S., P.S.E.P. ou P.E.S.M.). Ceci a été longuement détaillé précédemment et nous n'y reviendrons pas.

A) NATURE DE L'ENSEMBLE DES SOLUTIONS CANDIDATES

Elle dépend de la manière de modéliser le problème. Par exemple, pour le problème du voyageur de commerce, l'ensemble des solutions réalisables du problème peut être :

- soit l'ensemble des permutations de 1,2,...,n ([W.H. KOHLER / K. STEIGLITZ 1976]),
- soit l'ensemble des sous-ensembles d'arcs constituant des circuits hamiltoniens ([J.D.C. LITTLE / K.G. MURTY / D.W. SWEENEY / C. KAREL 1963]).

Mais il peut être intéressant, pour obtenir une "définition en compréhension" ([B. ROY 1970 tome 1]) plus simple de l'ensemble des solutions à séparer, de choisir comme ensemble de solutions candidates aux séparations un autre ensemble que l'ensemble des solutions réalisables.
Par exemple :

- pour le problème du sac à dos (ou du "knapsack"), on travaille sur $\{0,1\}^n$ où n est le nombre d'objets à placer dans le sac ([P.C. GILMORE /R.E. GOMORY 1966]). On n'élimine donc pas a priori les combinaisons d'objets n'entrant pas dans le sac.

Cela a pour conséquence immédiate que les séparations peuvent créer des sous-ensembles S_i vides de solutions réalisables ; ce sont les valuations secondaires qui permettront d'éliminer ces sous-ensembles ;

- pour le problème du voyageur de commerce, traité par l'algorithme de LITTLE ([1963]) l'ensemble des solutions candidates est constitué de sous-ensembles d'arcs tels que pour 2 arcs (x,y) et (x',y') appartenant à un sous-ensemble, on a $x \neq x'$ et $y \neq y'$. Il faut ultérieurement éliminer les sous-ensembles d'arcs qui contiennent des circuits de moins de n arcs.
Pour ces deux exemples, l'ensemble des solutions candidates contenait l'ensemble des solutions réalisables.

Il existe des exemples inverses : pour les problèmes d'atelier à une machine avec les hypothèses les plus simples et des critères réguliers (qui ne peuvent se détériorer en avançant une date de fin d'une tâche sans reculer celles des autres) on démontre deux résultats.
Les ordonnancements semi-actifs (toute tâche est calée à gauche) sont dominants, c'est-à-dire qu'il existe au moins une solution optimale qui est un ordonnancement semi-actif.
Les ordonnancements actifs (aucune tâche ne peut être ordonnancée plus tôt sans reculer au moins une date de fin de tâche d'une autre) sont dominants également (et sont semi-actifs).

Si on ne souhaite pas toutes les solutions optimales possibles, on choisit comme ensemble de solutions candidates l'ensemble des ordonnancements actifs qui est beaucoup plus petit que l'ensemble de tous les ordonnancements réalisables. Cela accélère la procédure, à condition que le fait de se limiter à ces seuls ordonnancements n'augmente pas trop le volume de calculs à effectuer à chaque séparation.

Dans cette thèse, nous avons utilisé comme ensemble de solutions candidates tantôt les ordonnancements actifs, tantôt les ordonnancements semi-actifs, et même des ordonnancements ε-actifs définis pour des cas particuliers.

Bien que cela soit théoriquement possible, nous n'avons pas rencontré d'exemples de P.S.E. où l'ensemble des solutions candidates ne contient pas toutes les solutions réalisables et contient des solutions non réalisables (sauf lorsque l'on décide arbitrairement d'ajouter des contraintes pour accélérer la recherche au risque de ne pas trouver l'optimum).

B) LE PRINCIPE DE SEPARATION CHOISI

Il remplace le sous-ensemble S_i par les sous-ensembles $S_{i1}, S_{i2},\ldots,S_{ip}$. Cette séparation doit vérifier le principe de conservation des solutions réalisables :

$$S_i \cap \mathcal{R} = \bigcup_{k=1}^{p} (S_{ik} \cap \mathcal{R})$$

où $\mathcal{R}$ est l'ensemble des solutions réalisables.

C'est-à-dire que toute solution réalisable de S_i se retrouve dans au moins un des S_{ik}.

Si $S_{ik1} \cap S_{ik2} = \varnothing$ pour $k_1 \neq k_2$, nous avons affaire à un partitionnement. Ce terme est utilisé abusivement ici car on autorise que S_{ik} soit vide.

Si une solution peut apparaître dans plus d'un S_{ik}, il s'agit d'un simple recouvrement.

Pour assurer le principe de finitude (dans le cas où S_i est fini) on impose la condition :

$$S_{ik} \subsetneq S_i \quad \text{si } S_i \neq \varnothing$$

Les procédés de séparation sont trés variés. Par exemple, on obtient un partitionnement :

- dans le cas du problème du voyageur de commerce ou lorsque l'on s'intéresse au séquencement d'opérations ou de travaux sur un ensemble de machines ([W.H. KOHLER / K. STEIGLITZ 1976], [K.R. BAKER 1974], [E. IGNALL / L. SCHRAGE 1965]) en procédant de la manière suivante :

si S_σ est le sous-ensemble de permutations se terminant par la sous-séquence σ, on sépare S_σ en $S_{i1\sigma}, \ldots, S_{ik\sigma}, \ldots, S_{ip\sigma}$ où $S_{ik\sigma}$ est le sous-ensemble de permutations se terminant par la sous-séquence constituée par la tâche ik suivie de la sous-séquence σ avec ik n'appartenant pas à σ.

- dans le cas du voyageur de commerce ([J.D.C. LITTLE / K.G. MURTY / D.W. SWEENEY / C. KAREL 1963]) en procédant comme suit :

si $S_{P,I}$ est le sous-ensemble de solutions où les arcs appartenant à P sont imposés et ceux appartenant à I sont interdits, on sépare $S_{P,I}$ en $S_{P\cup\{x,y\},I}$ et $S_{P,I\cup\{x,y\}}$ avec $\{x,y\} \notin P$ et $\{x,y\} \notin I$, c'est-à-dire qu'un nouvel arc est soit interdit, soit imposé (en fait, on prend $S_{P\cup\{x,y\},\ I\cup\{y,x\}}$ et $S_{P,I\cup\{x,y\}}$ car le fait d'imposer {x,y} interdit obligatoirement {y,x} qui fermerait un sous-circuit de 2 points) ;

- dans le cas des problèmes à contraintes disjonctives ([B. ROY 1966], [Y. TABOURIER 1969], [J. CARLIER 1978]) où des contraintes de précédence mutuellement exclusives traduisent le fait que deux tâches utilisant le même moyen en exemplaire unique ne doivent pas être exécutées simultanément, on sépare S_P, où P est l'ensemble des contraintes de précédence correspondant à des contraintes disjonctives pour lesquelles on a déjà décidé de l'ordre des tâches en conflit sur les moyens, en $S_{P\cup\{x,y\}}$ et $S_{P\cup\{y,x\}}$ avec $\{x,y\} \notin P$ et $\{y,x\} \notin P$;

- dans le cas du Job-shop où S_P est un ensemble d'ordonnancements partiels actifs où seul le début de l'ordonnancement est imposé, on sépare S_P en fonction de la tâche que l'on place en premier derrière P de manière à résoudre un premier conflit sur une des machines ([G.H. BROOKS / C.R. WHITE 1965]).

On obtient plus rarement un recouvrement :

- dans le cas du voyageur de commerce, on le rencontre dans deux méthodes : celle de l'arbre minimum ([M. HELD / R.M. KARP 1970]) et celle de EASTMAN ([N. CHRISTOFIDES 1975]) utilisant la méthode hongroise ([H.W. KÜHN 1955]). Dans ces deux cas, on sépare en ouvrant des sous-circuits (ou des sous-cycles) de moins de n points de manière à ne plus retrouver ces solutions non réalisables.

Si S_I est le sous-ensemble de solutions où les arcs de I sont interdits, on sépare S_I en $S_{I\cup(x1,x2)}$, $S_{I\cup(x2,x3)}$, ... , $S_{I\cup(xp,x1)}$ avec $p \leq n-1$.

Le fait d'avoir défini un procédé de séparation ne suffit pas pour séparer un sous-ensemble donné dans certains des exemples cités.
En effet, il n'y a pas forcément unicité, par exemple, de l'arc à interdire ou à imposer, ou de la contrainte disjonctive à partir de laquelle on construit deux contraintes de précédence exclusives.
L'algorithme choisi devra également prévoir la stratégie à suivre dans ces cas (par exemple, le plus grand regret dans l'algorithme de LITTLE [1963]).

C) LA METHODE DE CALCUL DE LA BORNE b_i

Il s'agit ici de disposer d'un procédé de calcul le moins coûteux possible pour trouver un minorant (resp. majorant) le plus proche possible de la borne inférieure (resp. supérieure) lorsque l'on cherche une solution minimale (resp. maximale).

Dans les cas les plus simples, et les plus fréquents, chaque décision prise pour effectuer une séparation induit une augmentation correspondante de la fonction objectif (pour un minimum). On initialise alors la fonction objectif à 0 ou à une somme de pénalités à payer a priori avant toute décision (par exemple somme du minimum des lignes puis des colonnes dans l'algorihtme de LITTLE). Puis on ajoute les pénalitées dûes à chaque décision (par exemple, nouveaux retards [K.R. BAKER 1974]).

Il est possible d'améliorer la borne en examinant plus précisément l'incidence des décisions actuelles sur les décisions ultérieures en augmentant le volume des calculs (par exemple,[E. IGNALL / L. SCHRAGE 1965]).

Une méthode plus générale d'obtention de bornes consiste à utiliser des méthodes de relaxation.La relaxation consiste à plonger le problème à résoudre, ou le sous-problème correspondant à un sous-ensemble à séparer, dans un problème plus général plus facile à résoudre. Le problème plus général est obtenu en relâchant des contraintes.
On sait résoudre certains cas particuliers d'ordonnancement d'atelier lorsque les tâches peuvent être interrompues ([K.R. BAKER 1974], [E.G. COFFMAN 1976],[R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979], [A.H.G. RINNOOY KAN 1976], [E.L. LAWLER / J. LABETOULLE 1978], [J. LABETOULLE / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOY KAN 1979]). Il est alors possible de les utiliser pour construire des procédures par séparation et évaluation, la valeur de l'objectif pour la solution optimale du problème avec tâches interruptibles sert de borne pour le problème à tâches non interruptibles.
Une autre relaxation courante consiste à modéliser le problème sous forme de programmation linéaire en nombres entiers et de négliger la contrainte d'intégrité. A.V. CABOT et S.S. ERENGUC ([1986]) proposent une extension de cette dernière technique de relaxation au cas où la fonction objectif n'est plus linéaire, mais est constituée d'une somme de fonctions concaves de chacune des variables ; ils ajoutent alors une relaxation (par minoration) de la fonction objectif en la remplaçant par son enveloppe convexe sur le domaine de définition.

Il est également possible de négliger certaines contraintes (par exemple, pour le problème du voyageur de commerce, l'interdiction des sous-circuits dans l'algorithme de EASTMAN [N. CHRISTOFIDES 1975]).

Une manière générale de relâcher simultanément certaines contraintes tout en essayant de ne pas trop les violer est d'utiliser la relaxation lagrangienne ([M.L. FISHER 1981]) :

une contrainte k de la forme $A_k X = d_k$

est ignorée et déplacée dans la fonction objectif qui devient, par exemple :

$$\min \left[CX + \sum_k w_k (A_k X - d_k) \right]$$

(si $w_k = 0$ alors la contrainte k est totalement ignorée).

La solution relâchée obtenue n'a aucune garantie ni d'optimalité, ni de faisabilité. Elle dépend du choix des coefficients de Langrange w_k. La valeur obtenue pour le nouvel objectif est un minorant de CX^*, qui est la valeur optimale.

En effet, si X° est la solution du problème relaxé, on a :

$$CX^\circ + \sum_k w_k(A_kX^\circ - d_k) \leq CX^* + \sum_k w_k(A_kX^* - d_k) = CX^*$$

L'inégalité est dûe au fait que X° est optimal pour le problème relaxé et l'égalité au fait que X^* est réalisable et vérifie donc $A_kX^* - d_k = 0$ pour tout k.

Cette manière d'obtenir des bornes pour les P.S.E. a été abondamment utilisée en ordonnancement ([M.L. FISHER 1973], [C.N. POTTS / L.N. VAN WASSENHOVE 1985], [C.N. POTTS 1985, J. GRABOWSKI / E. NOWICKI / S. SDRZACKA 1986]). Récemment, il a été montré que l'on peut améliorer ces bornes en utilisant, non pas une méthode de relaxation lagrangienne mais une méthode de décomposition lagrangienne ([M. GUIGNARD / S. KIM 1986]).

Une autre méthode de relaxation, moins fréquemment utilisée est la relaxation "surrogate" ([F. GLOVER 1968 et 1975]).

Un ensemble de contraintes k de la forme :

$A_kX = d_k$ ou $A_kX - d_k = 0$

est remplacé par leur moyenne pondérée :

$$\sum_k w_k (A_kX - d_k) = 0$$

Il est évident que cette nouvelle relaxation fournit une borne pour l'objectif du problème posé.

1.4.4. Méthodes spécifiques

1.4.4.1. Introduction

Dans ce qui précède, nous avons longuement développé les différentes approches utilisées pour résoudre les problèmes d'ordonnancement. Nous avons présenté les concepts qui sont à la base des méthodes approchées et ceux qui permettent de construire des algorithmes exacts exponentiels (programmation dynamique et procédures par séparation évaluation essentiellement). Il reste donc une dernière catégorie d'algorithmes à examiner : les méthodes exactes polynômiales utilisées pour résoudre des problèmes d'ordonnancement d'atelier.

Nous avons exclu les problèmes où les tâches sont interruptibles, car nous ne nous intéressons pas ici aux ordonnancements des tâches à l'intérieur d'un ordinateur. Par ailleurs, si les durées et les dates d'arrivées des tâches sont codées sous forme de nombres entiers, les tâches de durées unitaires sont des cas particuliers de tâches interruptibles (toutes les unités de temps) ; les résultats correspondants, connus et intéressants dans le cadre de ce travail, seront présentés au chapitre 3.
Le lecteur intéressé par les tâches interruptibles ou unitaires peut consulter J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI / J. WERGLARZ ([1986]).

Comme la plupart des problèmes d'ordonnancement d'atelier sont NP-difficiles, seuls quelques cas très particuliers ont été résolus par des algorithmes exacts polynômiaux. Nous allons les citer en considérant d'abord les problèmes à une machine, puis ceux comportant des machines en parallèle, et enfin ceux qui contiennent des machines en série.

1.4.4.2. Algorithmes polynômiaux pour les problèmes à une machine

Considérons le problème de base à une machine ([K.R. BAKER 1974]), c'est-à-dire le problème où n tâches disponibles à l'instant 0 sont à ordonnancer sur une machine toujours diponible avec des durées de réglages qui ne dépendent pas de la séquence des tâches. Pour ce problème, si l'on cherche à optimiser certains critères seulement (comme par exemple le plus grand retard), une séquence optimale est obtenue par un simple tri des tâches.

En particulier, on appelle SPT (pour "Shortest processing time") la séquence qui classe les tâches par durées croissantes, WSPT (pour "Weighted shortest processing time") la séquence qui classe les tâches dans l'ordre croissant du rapport durée divisée par le poids associé à la tâche et EDD (pour "Earliest due date") celle qui ordonne les tâches par urgence croissante. La séquence SPT minimise trois critères, le critère noté $\overline{F}$ (pour "mean Flow time") qui représente le temps moyen de présence des tâches dans l'atelier ([W.E. SMITH 1956]), le critère noté $\overline{C}$ (pour "mean Completion time") qui représente la moyenne des dates d'achèvement et le critère $\overline{J}$ (pour "mean number of Jobs") qui représente le nombre moyen de tâches simultanément présentes dans l'atelier. La séquence WSPT minimise le critère noté $\overline{F}_w$ (pour "weighted mean Flowtime") qui représente la moyenne pondérée des durées de présence des tâches dans l'atelier. La séquence EDD minimise le plus grand retard ([J.R. JACKSON 1955]).

Toujours pour ce problème de base, l'algorithme polynômial de Hogdson et Moore ([J.M. MOORE 1968]) permet, par une procédure itérative simple qui modifie EDD, de minimiser le nombre de tâches en retard.

En ajoutant des contraintes de précédence entre les tâches au modèle de base, il est encore possible de trouver quelques modèles qui ne sont pas NP-difficiles même pour des tâches non interruptibles de durées non unitaires. Ainsi E.L. LAWLER ([1973) propose un algorithme polynômial pour minimiser la durée totale (Makespan) lorsque les arrivées des tâches ne sont pas toutes simultanées et qu'il existe des contraintes de précédence entre les tâches. Plusieurs auteurs ([W.A. HORN 1972], [J.B. SIDNEY 1975], [EL. LAWLER 1978]) ont proposé des algorithmes polynômiaux pour minimiser le temps moyen de présence des tâches lorsque les contraintes de précédence forment une arborescence. Enfin, E.L. LAWLER et J.M. MOORS ([1969]) proposent un algorithme polynômial qui minimise le plus grand retard lorsqu'il existe des relations de précédence entre les tâches.

1.4.4.3. Algorithmes polynômiaux pour les problèmes d'atelier comportant des machines en parallèle

Le seul critère pour lequel des auteurs ont proposé des algorithmes polynômiaux est le "mean flow time" ou temps moyen de présence. Pour le problème de base à une machine, généralisé à m machines identiques en parallèle, R.W. CONWAY / W.L. MAXWELL / L.W. MILLER ([1967]) utilisent l'ordre SPT et une affectation progressive des tâches aux machines pour minimiser ce critère.
Des algorithmes polynomiaux ont été proposés par E. HOROWITZ et S. SAHNI ([1976)] dans le cas des machines à vitesses proportionnelles, par W.A. HORN ([1973]) et J. BRUNO / E.G. COFFMAN / S. SETHI ([1974] dans le cas de machines non reliées et par J. BLAZEWICZ / W. KUBIAK / H. RÖCK / J. SZWARCFITER ([1986]) dans le cas des machines identiques mais avec des ressources complémentaires.

1.4.4.4. Algorithmes polynomiaux pour les problèmes d'atelier comportant des machines en séries

Tous les algorithmes polynômiaux existants sont des généralisations de la célèbre "Règle de Johnson" ([1954]) valable pour minimiser la durée totale dans le cas d'un "flow-shop" à deux machines.

La seule généralisation pour les "Job shops" a été proposée par J.R. JACKSON ([1956] dans le cas de deux machines et lorsque chaque gamme ne demande que deux opérations au plus.
H. RÖCK ([1984]) intègre, toujours dans le cas de deux machines, la prise en compte de ressources limitées supplémentaires.

Pour les généralisations à plus de deux machines, elles concernent toutes les "flow shop de permutation" c'est-à-dire le cas où les produits, ne pouvant pas se dépasser, passent dans le même ordre sur toutes les machines. C.L. MONMA et A.H.G. RINNOOY KAN ([1983]) présentent un ensemble de théorèmes de dominance qui permettent d'expliquer les nombreux cas particuliers publiés auparavant, en particulier par W. SZWARC et I. NABESHIMA.

1.4.4.5. Conclusion

Si l'on exclut a priori, comme nous l'avons fait, les cas particuliers plus faciles où les tâches sont interruptibles, ou bien où les durées sont unitaires, les problèmes d'ordonnancement d'ateliers que l'on sait résoudre avec des algorithmes polynômiaux sont vraiment très rares.

L'espoir d'en résoudre d'autres et par ailleurs faible dans la mesure où beaucoup d'autres cas particuliers ont été démontrés NP-complets (voir, par exemple, [J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI /. WEGLARZ 1986]).

Il convient donc de s'intéresser aux méthodes approchées, comme celles que nous proposons dans le cadre de ce travail.

BIBLIOGRAPHIE DU CHAPITRE 1

N. AGIN
"Optimum seeking with branch and bound"
Management Science, Vol. 13, n° 4, pp 176-185, **1966**

A. ALANCHE / B. ANCELIN
"Evaluation des simulations à évènements discrets"
Le Nouvel Automatisme, pp 61-68, mars **1984**

M. AMMAR
"Conception et réalisation d'un macro-langage dédié à la simulation des systèmes de production"
Thèse de Docteur Ingénieur, ENSAM, **1984**

E.K. BAKER
"An Exact Algorithm for the Time-Constrained traveling Salesman Problem"
O.R, Vol. 31, n° 5, pp 938-945,**1983**

K.R. BAKER
Introduction to sequencing and scheduling
John Wiley, **1974**

K.R. BAKER / J.B. MARTIN
"An experimental comparison of solution algorithms for the single-machine tardiness problem"
Nav. Res. Log. Quart ., Vol. 21, pp 187-189, **1974**

K.R. BAKER / Z. SU
"Sequencing with due dates and early start times to minimize maximum tardiness"
Nav. Res. Log. Quart., vol. 21, pp 171-176, **1974**

E. BALAS
"Project scheduling with resource constraints"
Proceedings of the Nato Conference on Application of Mathematical Programming, Cambridge juin **1968**

E. BALAS
"A note on the branch and bound principle"
O.R., Vol. 16, pp 442-445 (errata p. 886), **1968**

E. BALAS
"A note on duality in disjunctive programming"
Management Science Research Report, n° 398, Carnegie-Mellon University, nov. **1976**

E. BALAS
"Disjunctive programming"
Management Science Research Report, n°415, Carnegie-Mellon University, Lecture notes for Discrete Optimization, Vancouver, août **1977**

M.L. BALINSKI
"Integer programming : methods, uses, computation"
Management Science, Vol. 12, n° 13, pp 253-313, **1965**

M.L. BALINSKI
"Some general methods in integer programming"
dans J. ABADIE "Non Linear Programming", North Holland Publishing Company, Amsterdam, **1967**

M.L. BALINSKI / K. SPIELBERG
"Methods for integer programming"
dans J.S. ARONOFSKY "Progress in Operations Research", Tome 3, pp. 195-292, John Wiley, **1969**

S.P. BANSAL
"Single machine scheduling to minimize weighted sum of completion times with secondary criterion. A branch and bound approach"
EJOR, Vol. 5, pp 177-181, **1980**

E. BECHLER / J.M. PROTH / K. VOYIATZIS
"Artificial memory in production management"
1st ORSA/TIMS, Ann Arbor, août **1984**

E. BECHLER / J.M. PROTH / K. VOYIATZIS
"Intelligence artificielle et gestion de production"
2ème symposium et exposition internationale, PARIS, novembre **1984**

M. BEGHIN-PICAVET / P. HANSEN
"Deux problèmes d'affectation non linéaires"
RAIRO Recherche Opérationnelle, Vol. 16, n° 9, pp 263-276, août **1982**

G. BEL
"Modèles et langages de simulation"
Bulletin de l'INRIA, n° 95, pp 5-9, **1984**

R. BELLMAN
"The Theory of Dynamic Programming"
Bull. Amer. Math. Soc., n° 60, pp. 503-515, **1954**

R. BELLMAN
Dynamic Programming
Princeton Univ. Press. N.J., **1957**

R. BELLMAN
Adaptative control Process, a guided Tour
Princeton Univ. Pres, **1961**

A. BENSOUSSAN / J.M. PROTH
"Gestion des stocks avec coûts concaves"
RAIRO Automatique / Systems Analysis and Control, Vol. 15, n° 3, pp 201-220, **1981**

A. BENSOUSSAN / J.M. PROTH
"Inventory planning in a deterministic environment : concave cost set-up"
Large Scale Systems, Vol. 6, pp 177-184, **1984**

J. BERNAD / M. PAKER
Les plannings
Les Editions d'Organisation, (pp 438-440), **1985**

C. BERTHUL
"Pilotage d'ateliers flexibles en présence de perturbations"
Thèse de Docteur Ingénieur, PARIS IX Dauphine, **1985**

C. BERTHUL / M.C. PORTMANN
"Control of flexible manufacturing systems with important disturbances"
PROLAMAT, Paris, pp 451-459, juin **1985**

C. BERTHUL / M.C. PORTMANN
"Flexible manufacturing system control under large disturbances"
APMS, Budapest, août **1985**

P. BERTIER / B. ROY
"Procédure de résolution pour une classe de problèmes pouvant avoir un caractère combinatoire"
Cahiers du Centre d'Etudes de Recherche Opérationnelle, Vol. 6, pp 202-208, **1964**

J.H. BLACKSTONE / J.R. PHILIPS / G.L. HOGG
"A state-of-art survey of dispatching rules for manufacturing jobshop operations"
Int. J. Prod. Res., Vol. 19, n° 2, pp 201-211, **1981**

J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI / J. WEGLARZ
Scheduling under resource constraints - Deterministic Models
J.C. BALTZER AG, **1986**

J. BLAZEWICZ / W. KUBIAK / H. RÖCK / J. SZWARCFITER
"Minimizing mean flow time under resource constraints on parallel processors"
à paraître, **1987**

J.F. BOSS
"Prise en considération des contraintes pesant sur la disponibilité des moyens dans les méthodes de chemin critique"
Revue Française de Recherche Opérationnelle, Vol. 38, pp 31-38, **1966**

G.H. BROOKS / C.R. WHITE
"An algorithm for finding optimal or near-optimal solutions to the production scheduling problem"
J. Ind. Eng., Vol. 16, n° 1, January, **1965**

J. BRUNO / E.G.COFFMAN / R. SETHI
"Scheduling independent tasks to reduce mean finishing time"
Comm. ACM, Vol. 17, n° 7, pp 382-387, **1974**

A.V. CABOT / S.S. ENRENGUC
"A branch and bound algorithm for solving a class of non linear integer programming problems"
Nav. Res. Log. Quart., Vol. 33, pp 559-567, **1986**

J. CARLIER
"Ordonnancements à contraintes disjonctives"
RAIRO, Vol. 12, n° 4, nov. **1978**

J. CARLIER
"The one-machine sequencing problem"
EJOR, Vol. 11, pp 42-47, **1982**

J. CARLIER / P. CHRETIENNE / C. GIRAULT
"Modeling scheduling problems with timed Petri Nets"
4th European Workshop on Theory and Applications of Petri Nets, Actes, pp 84-103, **1983**

G. CARPANETO / P. TOTH
"An efficient algorithm for the asymetric traveling salesman problem"
ORSA/TIMS Joint National Meeting, Atlanta, **1977**

J.B. CAVAILLE / D. DUBOIS
"Heuristic methods based on mean value analysis for flexible manufacturing systems performance evaluation"
Proc. IEEE Conference on Decision and Control, Orlando, **1982**

CESTA
Mise en oeuvre et réalités de la GPAO
CESTA, Paris, **1983**

S. CHAKRAVARTHY
"A single-machine Scheduling Problem with Random Processing Times"
Nav. Res. Log. Quart., vol. 33, pp 391-397, **1986**

A. CHARNES / W.W. COOPER
"Goal programming and multiple objective optimizations"
EJOR, Vol. 1, pp 39-54, **1977**

F. CHARPILLET / J.P. HATON / J.M. PIERREL
"Apport de la programmation dynamique en reconnaissance automatique de la parole continue"
Actes du 4ème congrès AFCET RFIA, Paris, **1984**.

P. CHRETIENNE
"Exécutions contrôlées des réseaux de Pétri temporisés"
TSI, Vol. 1, pp 23-31, **1984**

N. CHRISTOFIDES
Graph theory - An algorithmic approach
Academic Press, New York, **1974**

N. CHRISTOFIDES
"The traveling salesman problem : a survey"
Actes du Colloque de Cérisy la Salle, juin **1980**

E.G. COFFMAN
Computer and job shop scheduling theory
John Wiley & Sons, **1976**

R.W. CONWAY / W.C. MAXWELL/ L.W. MILLER
Theory of scheduling
Addison-Wesley, **1967**

M. CROUHY
La gestion informatique de la production industrielle
L'Usine Nouvelle **1983**

Y. DALLERY
"Une méthode analytique pour l'évaluation des performances d'un atelier"
Thèse d'Ingénieur, INP de Grenoble, **1986**

E. DEKEL / S. SAHNI
"Parallel Scheduling Algorithms"
O.R., Vol. 31, n° 1, **1983**

J. DESROSIERS / P. PELLETIER / F. SOUMIS
"Plus court chemin avec contraintes d'horaires"
RAIRO Recherche Opérationnelle, Vol. 17, n° 4, pp 357-377, nov **1983**

M.L. DIBON
Ordonnancement et potentiels, méthode M.P.M.
HERMANN, **1970**

G. DOUMEINGTS / D. BREUIL / L. PUN
La gestion de production assistée par ordinateur
Hermès, **1983**

L. DUPONT
"Algorithmes et Ordonnancements"
Thèse de 3ème cycle, Université de Grenoble, 24 octobre **1986**

J. ERSCHLER
"Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement"
Thèse d'Etat, Université de Toulouse, novembre **1976**

J. ERSCHLER / P. ESQUIROL
"Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de contraintes"
2ème Conférence Internationale sur les système de production, Actes, Tome 1, pp 159-173, Paris, **1987**

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"
O. R., Vol. 31, n° 1, pp 144-127, **1983**

R. FAURE
"Eléments de Programmation Dynamique, (théorie et pratique)"
Institut de Programmation, Université de Paris VI, publication n° 58, **1974**

M.L. FISHER
"Optimal solution for scheduling problems using Lagrange multipliers. Part 1"
O. R., Vol. 21, pp 1114-1127, **1973**

M.L. FISHER
"The Lagrangean relaxation method for solving integer programming problems"
Management Science, vol. 27, n° 1, janvier **1981**

M.L. FISHER / B.J. LAGEWEG / J.K. LENSTRA /A.H.G. RINNOOY KAN
"Surrogate duality relaxation for job shop scheduling"
Discr. Applied Math., n° 5, pp 65-75, **1983**

L.R. FORD / D.R. FULKERSON
"Maximal flow through a network"
Canadian Journal of Mathematics, Vol. 8, pp 399-404, **1956**

L.R. FORD / D.R. FULKERSON
"A primal-dual algorithm for the capacitated Hitchcock problem"
Nav. Res. Log. Quart., Vol. 14, n° 1, **1957**

M.S. FOX / S.F. SMITH
"ISIS : a knowledge based system for factory scheduling"
Expert Systems Journal, Vol. 1, n° 1, pp 25-45, **1984**

D.R. FULKERSON
"A network flow computation for project cost curves"
Management Science, vol. 7, n° 2, janvier **1961**

M.R. GAREY / R.L. GRAHAM / D.S. JOHNSON
"Performance guarantees for scheduling algorithms"
O.R., vol. 26, n ° 1, pp 3-21, janv-févr. **1978**

M.R. GAREY / D.S. JOHNSON
Computers and intractability : a guide to the theory of NP-completeness
FREEMAN, **1979**

M.R. GAREY / D.S. JOHNSON
"Approximation algorithms for bin packing problems : a survey"
Analysis and design of algorithms in combinatorial optimization, AUSIELLO LUCERTINI SPRINGER, pp 147-172, **1981**

R.S. GARFINKEL / G.L. NEMHAUSER
Integer Programming
J. WILEY & Sons, **1972**

L. GELDERS / P.R. KLEINDORFER
"Coordinating aggregate and detailed scheduling decisions in the one-machine job shop. Part 1 : theory"
O.R., Vol. 22, n° 1, pp 46-60, **1974**

E. GELENBE / G. PUJOLLE
Introduction aux réseaux de files d'attente
Collection technique et scientifique des télécommunications, EYROLLES, **1982**

A.M. GEOFFRION / R.E. MARTENS
"Integer programming algorithms : a framework and state of the art survey"
Management Science, vol. 18, n° 9, mai **1972**

W. GERE
"Heuristics in job shop scheduling"
Management Science, Vol. 13, n° 3, nov. **1966**

G. GIARD
Gestion de Production
ECONOMICA, **1981**

P.C. GILMORE / R.E. GOMORY
"The theory and computation of knapsack functions"
O.R., Vol. 14, pp 1045-1074, **1966**

F. GLOVER
"Surrogate constraints"
O.R., vol. 16, pp 741-749, **1968**

F. GLOVER
"Surrogate constraint duality in mathematical programming"
O.R., Vol. 23, pp 434-451, **1975**

B.L. GOLDEN / A.A. ASSAD
"A decision-theoric framework for comparing heuristics"
EJOR, Vol. 18, pp 167-171, **1984**

R.E. GOMORY
"Essentials of an algorithm for integer solutions to linear programs"
Princeton-IBM, Math. Res. Proj. Techn. Rep., n° 1, **1958**

J. GRABOWSKI / E. NOWICKI / S. SDRZACKA
"A block approach for single-machine scheduling with release dates and due dates"
EJOR, Vol. 26, pp 278-285, **1986**

R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA /A.H.G. RINNOOY KAN
"Optimization and approximation in deterministic sequencing and scheduling : a survey"
Ann. Discrete Math, vol. 5, pp 287-326, **1979**

M. GUIGNARD / S. KIM
"Lagrangean decomposition : A model yielding stronger Lagrangean bounds"
20ème anniversaire du Groupe Combinatoire de l'AFCET, décembre **1986**

P. HANSEN
"Les procédure d'optimisation par séparation : présentation générale"
Revue Belge de Statistiques, vol. 11, n° 3, pp 35-60, **1971**

M. HELD / R.M. KARP
"The traveling salesman problem and minimum spanning trees"
O.R., Vol. 18, n° 6, pp 1138-1162, **1970**

P. HERVE
"Les procédure arborescentes d'optimisation"
Revue Française d'Informatique et de Recherche Opérationnelle, Vol. 2, n° 14, pp 69-80, **1968**

J.C. HERZ
"Quelques considérations sur les problèmes d'emploi du temps"
Revue Française de Recherche Opérationnelle, vol. 38, pp 85-91, **1966**

C.A. HOLLOWAY / R.T. NELSON
"Job shop scheduling with due dates and overtime capability"
Management Science, Vol. 21, n° 1, pp 68-78, septembre **1974**

W.A. HORN
"Minimizing average flow time with parallel machines"
O.R., Vol. 1, pp 846-847, **1973**

W.A. HORN
"Single-machine job-sequencing with treelike precedence ordering and linear delay penalties"
SIAM J. on Appl. Math., Vol. 23, pp 189-202, **1972**

E. HOROWITZ / S. SAHNI
"Exact and approximate algorithms for scheduling non-identical processors"
J. ACM, Vol. 23, pp 317-327, **1976**

E. IGNALL / L. SCHRAGE
"Application of the branch and bound technique to some flow-shop scheduling problems"
O.R., Vol. 13, n° 3, May **1965**

J.R. JACKSON
"Scheduling a production line to minimize maximum tardiness"
Research Report 43, Management Sciences Research Project, UCLA, **1955**

J.R. JACKSON
"An extension of Johnson's results on job lot scheduling"
Nav. Res. Log. Quart., Vol. 3, n° 3, pp 201-203, **1956**

S.M. JOHNSON
"Optimal two-and-three-stage production schedules"
Nav. Res. Log. Quart., Vol. 1, n° 1, pp 61-68, **1954**

G.H. JONES
"An economic evaluation of job-shop dispatching rules"
Management Science, Vol. 20, n° 3, novembre **1973**

K. KAZANTZIDIS
"Les systèmes experts et la gestion de production"
Thèse de Docteur Ingénieur, Université de Paris IX, Dauphine, **1985**

A. KAUFMANN
Méthodes et modèles de la recherche opérationnelle
Tome 2, DUNOD, **1968**

J.E. KELLEY / M.R. WALKER
"Critical path planning and scheduling"
Eastern Joint Computer Conference Proceedings, pp 160-173, Boston, **1959**

K.G. KEMPF
"Manufacturing and Artificial Intelligence"
Robotics, Vol. 1, n° 1, pp 13-26, **1985**

J.P. KIEFFER / S. HAMICHI
Les progiciels de la gestion de production : principe de fonctionnement, analyse comparée, choix et mise en oeuvre
L'Usine Nouvelle, 1986

S. KIRKPATRICK / C.D. GELATT / M.P. VECCHI
"Optimization by simulated annealing"
Res. Rep. R.C. 9335, IBM Thomas J. WATSON, Center Yorktown Heights, NY, **1982**

W.H. KOHLER / K. STEIGLITZ
"Enumerative and iterative computational Approaches"
dans COFFMAN, pp 229-287, **1976**

H.W. KÜHN
"The hungarian method for the assignment problem"
Nav. Res. Log. Quart., Vol. 2, n° 1 et 2, **1955**

J. LABETOULLE / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN
"Preemptive scheduling of uniform machines subject to release dates"
Report BW 99, Mathematisch Centrum, Amsterdam, **1979**

A. LAHRICHI / M. LAHRICHI
"La notion de fonction de regret et son application à un problème d'investissement"
EDF, Bulletin de la Direction des Etudes et Recherches, Série C, n° 2, pp 25-32, **1980**

A.H. LAND / A.G. DOIG
"An automatic method for solving discrete programming problems"
Econometrica, vol. 28, pp 497-520, **1960**

E.L. LAWLER
"A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness"
Ann. Discrete Math., Vol. 1, pp 331-342, **1977**

E.L. LAWLER
"On scheduling problems with deferral costs"
Management Science, Vol. 11, pp 280-288, **1964**

E.L. LAWLER
"Optimal sequencing of a single machine subject to precedence constraints"
Management Science, Vol. 19, pp 544-546, **1973**

E.L. LAWLER
"Sequencing jobs to minimize total weighted completion time subject to precedence constraints"
Ann. Discrete Math., Vol. 2, pp 75-90, **1978**

E.L. LAWLER / J. LABETOULLE
"On preemptive scheduling of unrelated parallel processors by linear programming"
J. Assoc. Comput. Mach., Vol. 25, pp 612-619, **1978**

E.L. LAWLER / J.M. MOORS
"A functionnal equation and its application to resource allocation and scheduling problems"
Management Science, Vol. 16, n° 1, pp 77-84, **1969**

E.L. LAWLER / D.E. WOOD
"Branch and bound methods : a survey"
O.R., Vol. 14, pp 669-719, **1966**

V.I. LEOPOULOS / J.M. PROTH
"Le problème multi-produits avec coûts concaves et incitation aux lancements groupés : le cas général"
APII - 1985, vol. 19, pp 117-130

J.D.C. LITTLE / K.G. MURTY / D.W. SWEENEY / C. KAREL
"An algorithm for the travelling salesman problem"
O.R., Vol. 11, pp 972-989, **1963**

S. MARTELLO / P. TOTH
"A solution of the zero-one multiple knapsack problem"
EJOR, Vol. 4, pp 276-283, **1980**

J. MELESE
L'analyse modulaire des systèmes de gestion
Hommes et Techniques, **1972**

L.G. MITTEN
"Sequencing n jobs on two machines with arbitrary lag times"
Management Science, Vol. 15, pp 293-298, **1958**

L.G. MITTEN
"Branch and bounds methods : general formulations and properties"
O.R., Vol. 18, pp 24-34, **1970**

S. MIYAZAKI
"Combined scheduling systems for reducing job tardiness in a job shop"
Int. J. Prod. Res., Vol. 19, n° 2, pp 201-211, **1981**

C.L. MONMA / A.H.G. RINNOOY KAN
"A concise survey of efficiently solvable special cases of the permutation flow-shop problem"
RAIRO Recherche Opérationnelle, Vol. 17, n° 2, mai, pp 105-109, **1983**

J.M. MOORE
"An N job one machine sequencing algorithm for minimizing the number of late jobs"
Management Science, Vol. 15, pp 102-109, **1968**

H. MÜLLER-MERBACH
"Modelling techniques and heuristics for combinatorial problems"
Combinatorial Programming (ROY), **1974**

I. NABESHIMA
"Unified treatment on single machine scheduling problems"
Rep. Univ. Electro. Com., Vol. 22, n° 2, pp 29-38, décembre **1971**

R.T. NELSON / R.K. SARIN / R.L. DANIELS
Scheduling with multiple performance measures : the one-machine case"
Management Science, Vol. 32, n° 4, avril **1986**

T. NICHOLSON
Optimization in Industry
Vol. 1, ch. 10, Longman Press London, **1971**

J. ORLICKY
Material requirements planning
Mc Graw.Hill, **1975**

S.S. PANWALKAR / W. ISKANDER
"A survey of scheduling rules"
O. R., Vol. 25, n° 1, pp 45-61, janv-fév. **1977**

R.G. PARKER / R.H. DEANE / R.A. HOLMES
"On the use of a vehicle routing algorithm for the parallel processor problem with sequence dependent changeover costs"
AIIE Trans., Vol. 9, pp 155-160, **1977**

R.G. PARKER / R.L. RARDIN
"The traveling salesman problem : an update of research"
Nav. Res. Log. Quart., Vol. 30, pp 69-96, **1983**

T.L. PASCOE
"Allocation of resources in C.P.M."
Revue Française de Recherche Opérationnelle, n° 38, pp 31-38, **1966**

P.E.R.T. "Program Evaluation Research Task". Phase I. Summary Report.
Special Projects Office, Bureau of Ordinance, U. Department of the Navy, Washington, D.C., pp 646-699, July **1958**

J.C. PICARD / M. QUEYRANNE
"The time-dependant traveling salesman problem and its application to the tardiness problem in one machine scheduling"
O.R., Vol. 26, n° 1, janvier-février **1978**

M. PINEDO
"Stochastic scheduling with release dates and due dates"
O.R., Vol. 31, n° 3, pp 559-572, **1983**

M.C. PORTMANN
"Synthèse sur les principaux modèles et techniques d'ordonnancement"
Document pédagogique, Université de nancy I, octobre **1983**

M.C. PORTMANN
"Diversité des problèmes et survol rapide"
Commission ORDONNANCEMENT CESTA "Mises en oeuvre et réalités de la GPAO", novembre **1983**

D. POTIER
"New users ; introduction to QNAP2"
1983

C.N. POTTS
"Analysis of a heuristic of one machine sequencing with release dates and delivery times"
O.R., Vol. ??, pp 1436-1441, **198??**

C.N. POTTS
"A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time"
Management Science, Vol. 31, n° 10, octobre **1985**

C.N. POTTS / L.N. VAN WASSENHOVE
"A decomposition algorithm for the single machine total tardiness problem"
O.R. Letters, Vol. 1, n° 5, pp 177-181, novembre **1982**

C.N. POTTS / L.N. VAN WASSENHOVE
"A branch and bound algorithm for the total weighted tardiness problem"
O.R., Vol. 33, n° 12, mars -avril **1985**

A.A.B. PRITSKER / C.D. PEGDEN
Introduction to simulation and SLAM
Halted Press, New York, **1979**

J.M. PROTH / J. QUINQUETON / H. RALAMBONDRAINY / K. VOYIATZIS
"Problème d'ordonnancement : utilisation de l'intelligence artificielle"
Le nouvel automatisme, pp 59-62, nov-déc. **1983**

J.M. PROTH / J. QUINQUETON / H. RALAMBONDRAINY / K. VOYIATZIS
"Utilisation de l'intelligence artificielle dans un problème d'ordonnancement"
AFCET, CONGRES AUTOMATIQUE, pp 53-61, **1983**

P. RAYSON
"A review of expert systems principles and their roles in manufacturing"
Robotica, Vol. 3, **1985**

A.H.G. RINNOOY KAN
"Machine scheduling problems : classification, complexity and computation",
Nijhoff, The Hague **1976**

H. RÖCK
"Some new results in flow shop scheduling"
Zeitschrift für Operations Res., Vol. 28, pp 1-16, **1984**

F. ROUBELLAT / V. THOMAS
"Une méthode et un logiciel pour l'ordonnancement en temps réel d'ateliers"
2ème Conférence Internationale sur les Systèmes de Production, Tome 1, pp 87-101, Paris, **1987**

B. ROY
"Optimisation et aide à la décision"
Cahier du LAMSADE (Paris-Dauphine), n° 8, **1977**

B. ROY
"Contribution de la théorie des graphes à l'étude des problèmes d'ordonnancement"
Actes du Congrès IFORS, **1960**

B. ROY
"Prise en compte des contraintes disjonctives dans les méthodes de chemin critique"
Revue Française de Recherche Opérationnelle, n° 38, pp 69-84, **1966**

B. ROY
Algèbre moderne et théorie des graphes
Tome 2, ch. VIII, DUNOD, **1970**

B. ROY
"Procédures d'exploration par séparation et évaluation (P.S.E.P. et P.S.E.S.)"
Revue Française d'Informatique et de Recherche Opérationnelle, Vol. 1, pp 61-90, **1970 a**

B. ROY / G. CHAUVIN / M.C. PORTMANN
"La catégorie des modèles d'ordonnancement : un essai de définition"
cahier du LAMSADE, n° 15, mars **1978**

L. SCHRAGE / K.R. BAKER
"Finding an optimal sequence by dynamic programming : an extension to precedence-related tasks"
O.R., Vol. 26, n° 1, pp 111-120, janv-fév. **1975**

L. SCHRAGE / K.R. BAKER
"Dynamic programming solution of sequencing problems with precedence constraints"
O.R., Vol. 26, n° 3, pp 444-449, mai juin**1978**

J.B. SIDNEY
"Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs"
O.R., Vol. 23, pp 283-298, **1975**

E.A. SILVER / R.V.V. VIDAL / D. DE WERRA
"A tutorial on heuristic methods"
EJOR, Vol. 5, pp 153-162, **1980**

R. SLOWINSKI
"Cost minimal preemptive scheduling of independent jobs with release and due dates on open shop under resource constraints"
INFORMATION PROCESSING LETTERS, Vol. 9, n° 5, Dec. **1979**

R. SLOWINSKI
"Multiobjective network scheduling with efficient use of renewable and non renewable resources"
EJOR, Vol. 7, pp 265-273, **1981**

R. SLOWINSKI / J. WEGLARZ
"An interactive algorithm for a general class of multiobjective precedence and resource constrained scheduling problems"
Dijon, octobre **1982**

W.E. SMITH
"Various optimizers for single-state production"
Nav. res. Log. Quart., Vol. 3, pp 59-66, **1956**

V. SRINIVASAN
"A hybrid algorithm for the one machine sequencing problem to minimize total tardiness"
Nav. Res. Log. Quart., Vol. 18, pp 317-327, **1971**

R.S. STAINTON
"Production scheduling with multiple criteria objectives"
Op. Res. Quart., Pergamon Press, Vol. 28, n° 2i, pp 285-292, **1978**

M.S. STEFFEN
"A survey of artificial intelligence-based scheduling system"
Fall Industrial Conference, Boston, Dec. **1986**

J.R. SULZER / P. BOUTEILLE / P. ARQUE
La simulation, initiation pratique au GPSS
EME, **1970**

Y. TABOURIER
"Problèmes d'ordonnancement à contraintes purement disjonctives"
RIRO, 3ème année, Vol. 3, pp 51-66, **1969**

J. THEPOT
"A clustering approach of scheduling problems, application to the job-shop"
Journal ASMDA, à paraître

J. THEPOT
"Méthodes des priorités pour des problèmes d'ordonnancement à durées égales",
EIASM, W.P., n° 80, pp 32, Bruxelles, **1980**

J. THEPOT / G. LECHENAUT
"Méthode de classification pour la coloration des graphes"
Actes du Colloque de Cerisy la Salle, Regards sur la théorie des graphes, pp 313-316, juin **1980**

J. THEPOT / G. LECHENAUT
"Application de la classification hiérarchique à la coloration des graphes"
RAIRO, Vol. 15, pp 73-83, **1981**

C. THURIOT
"Lancement périodique et gestion à court terme d'un atelier flexible"
INSA de Toulouse, Thèse de Docteur Ingénieur, mars **1985**

R. TREMOLIERES / J.C. AUBERT
"Détermination d'une flotte minimale et ordonnancement des tâches dans la livraison du béton prêt à l'emploi"
Working Paper, IAE, Aix en Provence, décembre **1975**

L.N. VAN WASSENHOVE / K.R. BAKER
"A bicriterion approach to time/cost trade-offs in sequencing"
EJOR, Vol. 11, **1982**

L.N. VAN WASSENHOVE / L. GELDERS
"Four solution techniques for a general one machine scheduling problem : comparative study"
EJOR, Vol. 2, pp 281-290, **1978**

M.E. VENTURA
"Un exemple d'application de la programmation dynamique à un cas concret"
Op. Res. Quart., Vol. 2, n° 1,**1960**

D.C. WOOD
"A technique for colouring a graph applicable to large scale timetabling problems"
Computer Journal, vol. 12, **1969**

M. YAMAMOTO
"An approximate solution of machine scheduling problems by decomposition method"
Int. J. Prod. Res., Vol. 15, n° 6, pp 599-608, **1977**

CHAPITRE 2

METHODES DE DECOMPOSITION EN ORDONNANCEMENT

2. METHODES DE DECOMPOSITION EN ORDONNANCEMENT

2.1. INTRODUCTION

2.2. DECOMPOSITION TEMPORELLE

2.3. DECOMPOSITION SPATIALE

2.4. DECOMPOSITION SPATIALE ET TEMPORELLE

2.5. CONCLUSIONS

2. METHODES DE DECOMPOSITION POUR RESOUDRE DES PROBLEMES D'ORDONNANCEMENT

2.1. INTRODUCTION

2.1.1. Problème et notations

Dans le chapitre introductif sur les problèmes d'ordonnancement, nous avons mis en évidence les caractéristiques principales des problèmes que nous traitons dans le contexte général de l'ordonnancement.

Nous avons également signalé dans quelle mesure une hypothèse est obligatoire ou, au contraire, peut être supprimée ou modifiée sous réserve d'une adaptation du modèle et/ou des algorithmes.

Nous avons enfin introduit les idées générales sur les méthodes de décomposition.

Avant de détailler les méthodes de décomposition que nous proposons, nous allons revenir sur le problème considéré pour en présenter une modélisation mathématique détaillée.

Remarque importante :

Toute notation est entièrement définie lors de son introduction. Toutefois, pour éviter au lecteur de rechercher, lors de leur utilisation ultérieure, la signification des notations, nous les avons regroupées dans un glossaire en fin de document dans l'annexe F.

A) Notations utilisées

Nous essayons d'utiliser les notations les plus courantes, bien qu'elles rappellent rarement les noms français des objets désignés.

Le problème d'atelier considéré est un problème de type "job-shop" sous les hypothèses les plus classiques.

L'atelier comporte m machines utilisées en série. Une machine ne peut traiter qu'un produit à la fois.
Sur l'horizon considéré, n produits devront être fabriqués.
Chaque produit i passe sur certaines machines dans un ordre imposé qui dépend du produit et subit une opération sur chaque machine.
Le produit i est disponible à l'instant $r(i)$ qui est donc, pour i, l'instant de début de fabrication au plus tôt. La lettre r vient de la dénomination anglaise "release date".
Pour ne pas être en retard, le produit i doit être terminé à l'instant $d(i)$, qui est donc, pour i, l'instant de fin de fabrication au plus tard ("due date"). Il s'agit ici d'un délai souhaité, mais non impératif. Son non-respect induit des pénalités de retard.

La gamme linéaire du produit i comporte $q(i)$ opérations.
La k-ème opération du produit i utilise la machine dont le numéro est $\mu(i,k)$ pendant une durée $p(i,k)$ $(1 \le k \le q(i))$. La lettre p vient de l'anglais "processing time".
Les opérations ne sont pas interruptibles. Les temps de réglage sont supposés indépendants de la séquence. Les réglages effectués alors que la pièce est déjà sur la machine allongent simplement la durée de l'opération et sont inclus dans $p(i,k)$. Si une partie des réglages peut être réalisée avant que la pièce ne soit sur la machine, nous notons $s(i,k)$ la durée correspondante. La lettre s vient de l'anglais "set-up time".

Nous notons $\tau(m_1,m_2)$ la durée du déplacement de la machine m_1 vers la machine m_2. Pour étendre cette notation à la durée de déplacement nécessaire pour passer de l'entrée à la sortie de l'atelier, deux machines fictives numérotées 0 et m+1 désignent l'entrée et la sortie de l'atelier.

Enfin, $pp(i)$ sera la durée minimale de séjour du produit i dans l'atelier :

$$pp(i) = \sum_{k=1}^{q(i)-1} [\, p(i,k) + \tau(\mu(i,k), \mu(i,k+1)) \,] + p(i,q(i))$$

Les $s(i,k)$ n'interviennent pas car ils peuvent être comptabilisés avant l'arrivée du produit sur chaque machine visitée.

Toutes les informations précédentes sont supposées connues.

Un énoncé du problème d'ordonnancement $\boldsymbol{P}$ considéré est donc défini à l'aide du huit-uple : $(n, m, r, p, \mu, d, s, \tau)$

où n, m et $\mu \in \mathbb{N}^+$

r, p, d et $s \in \mathbb{R}^n$

$\tau \in \mathbb{R}^{m+2} \times \mathbb{R}^{m+2}$

et se simplifie en (n, m, r, p, μ, d) lorsque les temps de réglage avant l'arrivée du produit et les durées de transport sont négligeables.

Nous notons $t(i,k)$ les seules informations inconnues qui sont les dates de début des opérations.

B) Modélisation du problème

C'est un problème de type T (cf. § 1.2.3.). Toute solution est définie par le vecteur T :

$$T = [\, t(1,1), t(1,2), \ldots, t(1,q(1)), \; t(2,1), t(2,2), \ldots, t(2,q(2)), \; \ldots, \; t(n,1), t(n,2), \ldots, t(n,q(n)) \,]$$

Une solution T est réalisable si elle vérifie les conditions de faisabilité **C = (C1,C2,C3)** définies ci-dessous :

C 1 : condition de disponibilité du produit et de préparation de la machine

$\forall i :$ $\max [(r(i) + \tau(0,\mu(i,1))) , (t_0+(s(i,1))] \leq t(i,1)$

où t_0 est l'instant de début de l'ordonnancement.

C 2 : condition de non chevauchement entre les opérations d'un même produit

$\forall i, \forall k : 1 \leq k < q(i)$:

$t(i,k) + p(i,k) + \tau(\mu(i,k), \mu(i,k+1)) \leq t(i,k+1)$

C 3 : condition de non chevauchement des opérations sur les machines,

$\forall i_1, \forall i_2, \forall k_1 : 1 \leq k_1 \leq q(i_1), \forall k_2 : 1 \leq k_2 \leq q(i_2)$

$k_1 \neq k_2$ et $\mu(i_1,k_1) = \mu(i_2,k_2)$

====>

$t(i_1,k_1) + p(i_1,k_1) + s(i_2,k_2) \leq t(i_2,k_2)$

ou

$t(i_2,k_2) + p(i_2,k_2) + s(i_1,k_1) \leq t(i_1,k_1)$

Le produit i quitte l'atelier à l'instant c(i) . La lettre c vient de l'anglais "completion time" ou date d'achèvement.

Ici on a : $c(i) = t(i,q(i)) + p(i,q(i)) + \tau(\mu(i,q(i)), m+1)$

c(i) est une fonction de l'ordonnancement retenu T. Pour ne pas alourdir les notations, nous utilisons c(i), c'(i) ou $c_k(i)$ pour désigner les dates d'achèvement des ordonnancements T, T' ou T_k.

Les critères dits "réguliers" ([K.R. BAKER 1974] sont des fonctions des c(i) tels que pour deux solutions T et T' telles que :

$\exists i_0 : c(i_0) < c'(i_0)$ et $\forall i : c(i) \leq c'(i)$

alors critère (T) ≤ critère (T').

Les critères que nous étudierons seront tous des critères réguliers que nous chercherons à minimiser.

Notre critère principal sera la somme des retards R(T), critère équivalent à la moyenne des retards ou "mean tardiness". Nous utiliserons aussi des moyennes pondérées des retards ou "mean weighted tardiness".

$$R(T) = \sum_{i=1}^{n} \max [0 , c(i) - d(i)]$$

ou

$$R_W(T) = \sum_{i=1}^{n} \max [0 , (c(i) - d(i)).W_i]$$

où W_i est le poids attribué au produit i. Il représente une pénalité par unité de temps de retard pour le produit i.

Notre critère secondaire sera souvent la durée totale ou le "makespan" M :

$$M(T) = \max_i [c(i)]$$

Mais il est possible d'utiliser nos méthodes de décomposition avec d'autres critères comme par exemple le plus grand retard :

$$T_{max}(T) = \max_i [\max [0 , c(i) - d(i)]] = \max [0 , L_{max}(T)]$$

où

$$L_{max}(T) = \max_i [c(i) - d(i)]$$

Pour simplifier la présentation des méthodes de décomposition, nous utiliserons le modèle suivant, sans réglage ni durée de transport :

$$\min [R(T)]$$

avec $$R(T) = \sum_{i=1}^{n} \max [0 , t(i,q(i)) + p(i,q(i)) - d(i)]$$

sous les contraintes **C** suivantes :

C 1 : contrainte de disponibilité du produit

$$\forall i \quad r(i) \leq t(i,1)$$

C2 : condition de non chevauchement entre opérations du même produit

$$\forall i, \forall k : 1 \leq k < q(i) :$$
$$t(i,k) + p(i,k) \leq t(i,k+1)$$

C 3 : condition de non chevauchement des opérations sur les machines

$$\forall i_1, \forall i_2,$$

$$\forall k_1 : 1 \leq k_1 \leq q(i_1), \ \forall k_2 : 1 \leq k_2 \leq q(i_2), \ k_1 \neq k_2$$

$$\mu(i_1,k_1) = \mu(i_2,k_2)$$

====>

$$t(i_1,k_1) + p(i_1,k_1) \leq t(i_2,k_2)$$

ou

$$t(i_2,k_2) + p(i_2,k_2) \leq t(i_1,k_1)$$

Il s'agit de programmation linéaire avec des contraintes disjonctives. Il existe des algorithmes exponentiels pour résoudre ce type de problème ([CARLIER 1978], [RINNOOY KAN 1976]). Ils ne sont utilisables que pour des problèmes de petite taille. Nous les avons programmés pour les utiliser dans nos méthodes de décomposition. Les détails pratiques sont présentés dans l'annexe B.

2.1.2. Présentation générale

Dans la suite de ce chapitre, nous supposons que nous disposons d'un algorithme exact pour résoudre le problème modélisé en 2.1.1. en temps raisonnable. Cet algorithme pourra être remplacé par une heuristique de "bonne qualité" lorsque, malgré la décomposition, la taille des sous-problèmes à traiter reste trop importante.

Toutefois, nous parlerons toujours d'optimisation locale ou de solution localement optimale pour la résolution des sous-problèmes issus de nos méthodes de décomposition.

Nous allons successivement présenter la décomposition temporelle, puis la décomposition spatiale, et enfin la décomposition spatiale et temporelle.

2.2. DECOMPOSITION TEMPORELLE

2.2.1. Introduction

La décomposition temporelle correspond à une technique de plan glissant. Contrairement à la décomposition hiérarchique utilisée en Gestion de Production, on n'agrège pas les informations et l'on reste au niveau du très court terme, celui des informations détaillées. Le plan glissant n'est pas utilisé comme moyen pour relier les différents niveaux de décision et d'agrégation, mais comme technique pour construire des algorithmes approchés.

Nous considérons un problème d'ordonnancement comportant un nombre important de tâches, avec des produits qui arrivent progressivement dans l'atelier à des dates r(i) . Alors il n'est pas déraisonnable de penser que l'ordonnancement le meilleur des derniers produits est relativement peu sensible aux multiples variantes de placement des premières tâches des premiers produits, sous réserve que l'on ait toujours cherché à maximiser la charge des machines.

La plupart des méthodes exactes explorent de manière implicite l'ensemble des solutions en combinant toutes les décisions possibles à partir de l'instant initial et en progressant le long de l'axe des temps (soit en construisant progressivement la solution par prise de décision au **premier** instant où une machine est disponible avec des produits en attente de cette machine, soit en considérant a priori un ordonnancement complet avec l'hypothèse de machines en nombre illimité et en prenant une décision concernant le **premier** conflit sur les machines).

Pour chaque décision possible, on poursuivra le long de l'axe des temps avec de nouvelles décisions, construisant ainsi les branches d'une arborescence. Si les décisions finales (liées par exemple à de nouveaux produits) sont relativement peu sensibles aux décisions initiales, on retrouvera plusieurs fois sans utilité les même combinaisons de décisions finales. C'est à partir de cette hypothèse de relative indépendance entre des décisions locales suffisamment éloignées dans le temps qu'on peut construire une méthode par décomposition temporelle.

On remplace la résolution du problème initial par la résolution d'une suite de problèmes de même nature et de taille inférieure. Au départ, on ordonnance de manière optimale un premier sous-ensemble de produits, ceux qui arrivent avant une certaine date φ_1 .

A chaque itération k , on introduit un nouveau sous-ensemble de produits que l'on ordonnance de manière optimale à partir de φ_k . Les tâches déjà ordonnancées qui sont terminées avant φ_k sont définitivement planifiées. Les tâches déjà ordonnancées qui se terminent après φ_k sont réordonnancées avec le nouveau sous-ensemble de produits introduit.

2.2.2. Définitions et méthode générale

Pour construire un algorithme de décomposition temporelle, il faut disposer d'un procédé de regroupement des produits en sous-ensembles de produits. Ce procédé, comme nous l'avons introduit précèdemment, doit tenir compte de la date de disponibilité des produits (r(i)) .

Nous appelons **suite temporelle** φ associée à un problème d'ordonnancement $\boldsymbol{P}$ une suite finie d'instants $\varphi_0 , \varphi_1 , \dots , \varphi_{L+1}$ de L+2 termes telle que :

a) $\varphi_0 < \varphi_1 < \varphi_2 \dots < \varphi_{L+1}$: φ est strictement croissante,

b) $\varphi_0 \leq \min_i [r(i)]$ et $\varphi_{L+1} > \max_i [r(i)]$.

L'intervalle $[\varphi_0 , \varphi_{L+1}[$ contient toutes les dates d'arrivée de produits.

Nous appelons **suite de sous-ensembles de produits associée** au problème $\boldsymbol{P}$ et à la suite temporelle φ la suite finie de L+1 termes π définie par :

$$\forall k : 0 \leq k \leq L : \pi_k = \{ i / \varphi_{k-1} \leq r(i) < \varphi_k \}$$

Le terme général π_k de cette suite est l'ensemble des indices des produits du (k+1)-ème sous-ensemble.

Nous pouvons décrire maintenant l'algorithme général de décomposition temporelle.

Initialisation

0_0 := ordonnancement optimal des produits du sous-ensemble π_0

pour k de 1 à L faire

"ajouter le sous-ensemble π_k à l'ordonnancement O_{k-1} ":

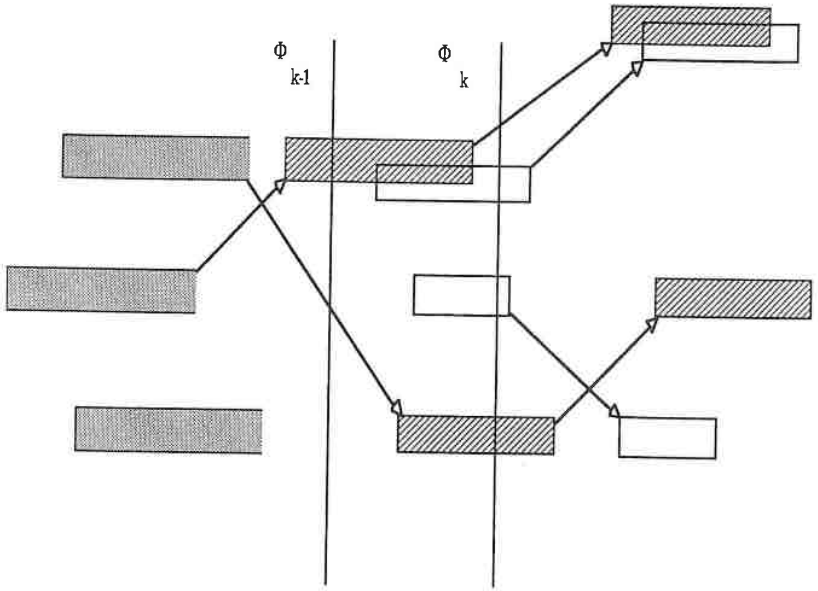
- les opérations terminées au plus tard à l'instant φ_{k-1} conservent définitivement la position qu'elles occupent dans O_{k-1} . Ces opérations sont fixées (on dit aussi figées) ;

- les opérations des produits du sous-ensemble π_k et les opérations de O_{k-1} non terminées à l'instant φ_{k-1} sont ordonnancées conjointement de manière à minimiser le critère. On réalise donc une optimisation locale pour le sous-problème correspondant, plus pauvre en opérations que le problème initial.

finpour

La figure 2.2.2. schématise une itération de la méthode de décomposition temporelle.

DECOMPOSITION TEMPORELLE



opérations fixées de O_{k-1}

opérations réordonnancées de O_{k-1}

opérations de Π_k

$O_k := O_{k-1} \oplus \Pi_k$

constitution de l'ordonnancement O_k à partir de l'ordonnancement O_{k-1} par adjonction des opérations de Π_k

figure 2.2.2.

2.2.3. Un algorithme d'ordonnancement par décomposition temporelle

Pour définir un algorithme d'ordonnancement par décomposition temporelle, il faut préciser le calcul de la suite temporelle φ, la méthode et les critères utilisés pour l'optimisation locale.

2.2.3.1. La suite temporelle φ

On peut imaginer plusieurs schémas pour construire la suite temporelle φ qui détermine les sous-ensembles de produits à introduire successivement dans l'ordonnancement .
Par exemple :
un découpage uniforme du temps :

$$\varphi_0 = \min_i [r(i)] \quad ; \quad \varphi_{L+1} = \max_i [r(i)] + 1$$

$$\Delta\varphi = \frac{\varphi_{L+1} - \varphi_0}{L+1} \quad ; \quad \varphi_k = \varphi_{k-1} + \Delta\varphi$$

un regroupement des produits par dates d'arrivées identiques :

$$\varphi_0 = \min_i [r(i)] \; ; \; \varphi_k = \min_{r(i) > \varphi_{k-1}} [r(i)] \; ; \; \varphi_{L+1} = \min(\emptyset) = +\infty$$

Le découpage uniforme du temps impose le choix d'un paramètre supplémentaire : le nombre d'itérations de la méthode. Mais il permet de faire évoluer dans le sens croissant ou décroissant la taille moyenne des sous-ensembles de produits. Ceci n'est valable bien sûr que si les dates d'arrivée sont uniformément réparties sur l'intervalle $[\varphi_0, \varphi_{L+1}-1]$.

Le regroupement des produits par dates d'arrivées identiques constitue un découpage unique. Si les sous-ensembles sont trop petits, il faut changer la suite temporelle pour faire des regroupements.

Nous avons voulu une suite temporelle unique associée à tout problème $\boldsymbol{P}$ qui soit susceptible de regrouper des produits arrivés à des dates différentes et qui ait un intérêt relativement à notre méthode d'optimisation locale.

découpage par rapport au produit non encore placé qui peut se terminer au plus tôt.

$$\varphi_0 = \min_i [r(i)] \quad ; \quad \varphi_1 = \min_i [r(i) + pp(i)]$$

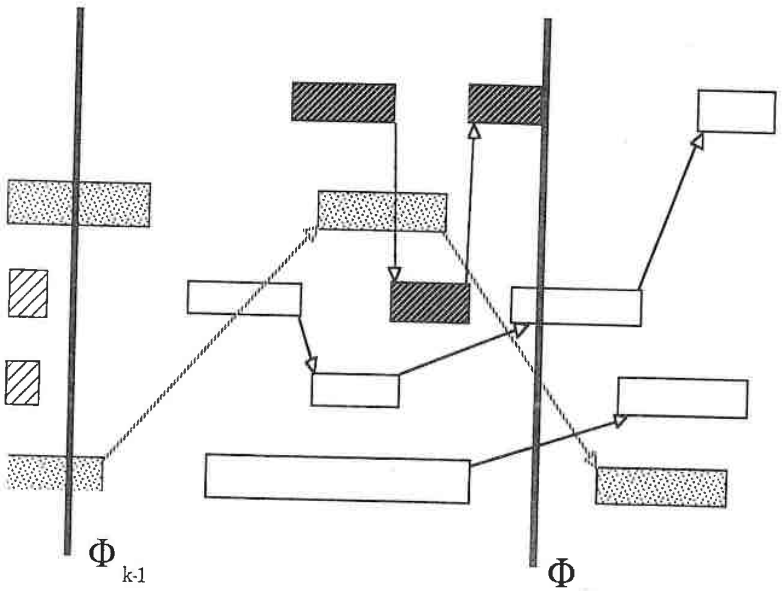
$$\varphi_k = \min_{r(i) \geq \varphi_{k-1}} [r(i) + pp(i)] \qquad \text{pour } k = 2,3,\ldots$$

Cela signifie que l'on cherche, parmi les produits en attente d'ordonnancement, celui qui, placé sur toutes les machines au plus tôt, se terminerait le premier, et que l'on introduit tous ceux qui peuvent entrer en conflit avec lui sur cette période.

La figure 2.2.3.1. met en évidence ce choix.

Ici les définitions de la suite temporelle φ et de la suite de sous-ensembles π sont liées, mais toutes deux ne dépendent que de l'énoncé du problème d'ordonnancement $\boldsymbol{P}$ et non des optimisations locales successives.

Si les sous-ensembles de produits ainsi constitués sont trop gros et ne permettent pas de réaliser les optimisations locales avec des coûts raisonnables, une variante de cette méthode consiste à remplacer $pp(i)$, durée minimale de présence du produit dans l'atelier, par $p(i,1)$, durée de sa première opération. Ces deux découpages sont équivalents lorsque chaque produit ne comporte qu'une tâche. Ce dernier découpage construit une suite temporelle analogue à celle utilisée dans les générateurs d'ordonnancements actifs ([K.R. BAKER 1974]).



date minimale de fin d'un nouveau produit

opérations définitivement fixées

opérations non terminées à l'instant Φ_{k-1} et qui correspondent à des produits déjà entrés

opérations d'un produit j non encore placé et pouvant se terminer le plus tôt possible

opérations des nouveaux produits, qui en plus du produit j , vont entrer en conflit, pour l'utilisation des machines, entres elles et avec celles qui ne sont pas terminées à l'instant Φ_{k-1}

figure 2.2.3.1.

Mais, contrairement à ces générateurs où l'on choisit successivement et individuellement chacune des tâches disponibles entre φ_{k-1} et φ_k-1 , nous recherchons un optimum local pour l'ordonnancement de l'ensemble de ces tâches.

2.2.3.2. Les critères de l'optimisation locale

Dans ce travail, nous avons considéré comme critère principal la somme des retards. Le critère de l'optimisation locale est donc la minimisation de la somme des retards des produits déjà introduits. Nous avons limité notre recherche aux seuls ordonnancements actifs qui sont dominants. Dans la mesure où le coût de la méthode le permet, à égalité sur le premier critère, nous minimisons la durée totale.

La méthode d'ordonnancement local utilisée est présentée dans l'annexe B.

2.2.4. Remarques

Dans la suite de ce travail, nous avons cherché à évaluer les méthodes de décomposition temporelle. Ce sont bien sûr des heuristiques, puisque la réunion de solutions optimales partielles ne constitue pas, en général, un optimum global, sauf dans des cas particuliers (une seule itération, L+1 sous-problèmes indépendants...).

En nous plaçant dans des cas très particuliers, nous avons réalisé des analyses dans le pire des cas : cela fera l'objet du chapitre 3.

Nous avons obtenu des analyses en moyenne par expérimentation. Elles seront présentées au chapitre 4. Comme toutes expériences faites en générant des exemples, les conclusions ne peuvent être généralisées sans risques.

2.3. DECOMPOSITION SPATIALE

2.3.1. Introduction

La décomposition spatiale cherche à constituer des sous-ateliers les plus indépendants possibles les uns des autres. Elle peut correspondre à un découpage physique réalisé dans l'atelier que l'on prend en compte pour ordonnancer. A ce niveau, on distingue en général le découpage par fonctions et le découpage par familles de produits.

Dans le découpage par fonctions, on regroupe par exemple les machines correspondant à la même phase de fabrication, et on obtient un sous-atelier de préparation, un sous-atelier d'usinage, un sous-atelier de tournage, un sous-atelier de finition,... Dans ce type de découpage, les flux de produits entre les sous-ateliers sont importants.

Dans le deuxième type de découpage, on cherche à constituer des îlots de fabrication tels qu'une famille de produits donnée est presque entièrement fabriquée dans un îlot et ne fait appel qu'exceptionnellement à des machines d'autres îlots. Ce découpage est très utilisé dans la conception des ateliers flexibles (par exemple, [A. KUSIAK / A.VANNELLI / K.R. KUMAR 1985] et [A. KUSIAK 1985]). Eventuellement, il est possible de dédoubler certaines machines pour qu'elles soient présentes dans plusieurs îlots de manière à réduire les interconnexions entre les îlots.

La méthode de décomposition spatiale, que nous proposons, relève de la deuxième catégorie de découpage. Elle permet, si on le souhaite, un découpage physique de l'atelier. Mais ce n'est pas dans cet objectif que nous l'avons développée. Nous utilisons la décomposition comme un outil de travail au sein de l'algorithme de résolution : il s'agit d'un découpage **virtuel** et non réel, propre à un carnet de commandes à ordonnancer, et qui évolue donc avec les carnets de commandes qui se présentent au cours du temps.

Nous allons tout d'abord présenter la décomposition de l'atelier en îlots de fabrication par des méthodes de classification non hiérarchique, puis nous verrons comment, à partir de ces îlots, décomposer le problème d'ordonnancement initial en sous-problèmes et, enfin, comment construire un algorithme qui gère les liens résiduels entre les îlots.

2.3.2. Méthodes de classification non hiérarchique

L'objectif de la décomposition spatiale d'un atelier est de regrouper les machines en îlots de fabrication et les produits en familles de telle sorte que :

- il y ait autant d'îlots de fabrication que de familles de produits ;
- les îlots et les familles de produits soient mis en correspondance biunivoque ;
- tout produit d'une famille soit traité essentiellement sur les machines de l'îlot associé et rarement sur celles d'autres îlots.

C'est un problème de classification croisée que nous allons traiter par une méthode de sériation par blocs. Avant de la décrire, nous allons rappeler quelques approches différentes qui conduisent également à des décompositions en blocs diagonaux.

2.3.2.1. Autres méthodes

Toutes les méthodes utilisent une matrice A de n lignes et m colonnes représentant l' importance des liens entre les éléments des deux ensembles que l'on cherche à partitionner en k classes.

La plupart des méthodes permutent les lignes et les colonnes afin de regrouper les éléments les plus grands. J.L. BURBIDGE dans les années 70 propose une méthode manuelle avec visualisation de la matrice. Par la suite, des méthodes automatiques se développent. Elles n'ont pas toujours pour objectif de trouver des blocs diagonaux non empiétants, mais parfois simplement des blocs homogènes, les blocs diagonaux étant obtenus, par exemple, s'ils sont parfaits ; c'est le cas des algorithmes proposés par W.T. Mc CORMICK, P.J. SCHWEITZER et T.W. WHITE ([1972]), J.R. SLAGLE, C.L. CHANG et S.R. HELLER ([1975]), M.V. BHAT et A. HAUPT [1976] ainsi que G. GOVAERT ([1983]).

Les autres auteurs cités dans la suite de ce paragraphe recherchent des blocs diagonaux non empiétants. Certains s'intéressent essentiellement à la constitution de familles de pièces, d'autres à la constitution de familles de machines et quelques uns seulement à une classification croisée des deux types de familles ; leurs objectifs différents les conduisent, néanmoins, à une modélisation semblable.

J. DEKLEVA et D. MENART ([1986] fabriquent de manière indépendante les familles de pièces et les familles de machines, puis testent leur cohérence.
J. Mc AULEY ([1972]) et W.T. Mc CORMICK and al. ([1972]) utilisent des mesures de ressemblance entre les rangées (lignes ou colonnes). J.R. KING ([1980]) utilise un tri des rangées à partir d'un codage en base 2.
J.R. SLAGLE and al. ([1975]) utilise des arbres de recouvrement minimaux, M.V. BHAT et A. HAUPT ([1976]) des couplages, A. KUSIAK ([1985]) et A. KUSIAK and al. ([1985]) la décomposition d'un graphe biparti en k-sous-graphes ou la programmation linéaire en variable 0-1 (formulation p-médiane). Ces derniers auteurs ont été amenés à développer des méthodes approchées pour résoudre les nouveaux modèles obtenus.

Certains auteurs travaillent sur des matrices entières ou réelles, d'autres sur des matrices en variables bivalentes ; nous n'avons pas jugé utile d'insister sur cette différence.

Il est totalement impossible de comparer ces méthodes entre elles car leur qualité dépend du critère choisi pour apprécier les bonnes solutions de la classification croisée.

2.3.2.2. Méthode utilisée

La méthode que nous utilisons a été développée par le projet SAGEP de l'INRIA ([J.M. PROTH 1985], [H. GARCIA / J.M. PROTH 1985], [H. GARCIA / B. MUTEL / J.M. PROTH 1985 et 1986]). Elle était destinée à résoudre des problèmes de technologie de groupe.

Par la suite, dans cette même équipe, nous avons étendu son champ d'application à la résolution des problèmes d'ordonnancement ([N. DRIDI / M.C. PORTMANN / J.M. PROTH 1986], [N. DRIDI / J.M. PROTH 1987], [M.C. PORTMANN 1987]).

F. MARCOTORCHINO [1986] a proposé quelques améliorations à la méthode dans le cas des matrices en variables 0-1. La légère amélioration obtenue pour la valeur du critère ne semble pas significative pour les applications en ordonnancement.

2.3.2.3. Définitions et notations

Nous désignons par Z^0 l'ensemble des indices des n produits et par Y^0 l'ensemble des indices des m machines.

Le terme général de la matrice A de n lignes et m colonnes est défini par :

$$a(i_0,j_0) = \sum_{\substack{k=1 \\ \mu(i_0,k)=j_0}}^{q(i_0)} p(i_0,k)$$

$a(i_0,j_0)$ est donc le temps total de passage du produit i_0 sur la machine j_0.

Les figures 2.3.2.3. a et b illustrent comment on passe des gammes des produits à la matrice A.

Si deux lignes de la matrice A sont identiques, cela signifie que deux produits fournissent le même volume de travail aux différentes machines et il n'est pas possible de les différencier dans la méthode de classification croisée.

Nous remplaçons tout sous-ensemble de produits indifférentiables par un produit type représentant le sous-ensemble. Les produits différents de tous les autres constituent à eux seuls un produit type.

Soit Z l'ensemble des indices des produits types et soit A' la matrice de N lignes ($N \leq n$) et m colonnes définie par :

$a'(i,j_0) = a(i_0,j_0)$ où i_0 est l'indice initial de l'un des produits correspondant au produit type i

A la matrice A' est associé un vecteur α de N éléments :

$\alpha(i)$ = poids de la ligne i dans A'
= nombre de produits correspondant au produit type i
= nombre de lignes identiques correspondant au produit type i dans A.

Les figures 2.3.2.3 b et c illustrent comment on passe de la matrice A à la matrice A'.

Si deux colonnes de la matrice A' sont identiques, cela signifie que tous les produits types demandent les mêmes durées opératoires sur les deux machines. Ces deux machines seront placées dans la même classe par la méthode de classification. Pour diminuer la taille de la matrice, il est possible d'effectuer sur les colonnes le même remplacement que sur les lignes (ce remplacement doit probablement être rare pour les problèmes réels, mais nous le proposons pour raison de symétrie du modèle).
Tout sous-ensemble de machines indifférentiables est remplacé par une machine type représentant le sous-ensemble. Les machines différentes constituent à elles seules une machine type.

Soit Y l'ensemble des indices des machines types et soit A" la matrice de N lignes ($N \leq n$) et M colonnes ($M \leq m$) définie par :

$a''(i,j) = a'(i,j_0)$ où j_0 est l'indice initial de l'une des machines correspondant à la machine type j .

A la matrice A" sont associés les vecteurs α de N éléments et β de M éléments où :

$\beta(j)$ = poids de la ligne j dans A"
= nombre de machines correspondant à la machine type j
= nombre de colonnes identiques correspondant à la machine type j dans A ou A' .

Les figures 2.3.2.3 c et d illustrent comment on passe de la matrice A' à la matrice A".

opérations

produits	durée	mach	durée	mach	durée	mach	durée	mach
1	15	1	15	2	10	1	25	3
2	10	1	10	3	5	2		
3	15	2	25	1	25	3		
4	25	2	5	1	5	3	15	4
5	5	3	8	4	5	3		
6	25	1	10	3	15	2	15	3
7	10	4						
8	15	3	25	2	15	1		
9	10	4						

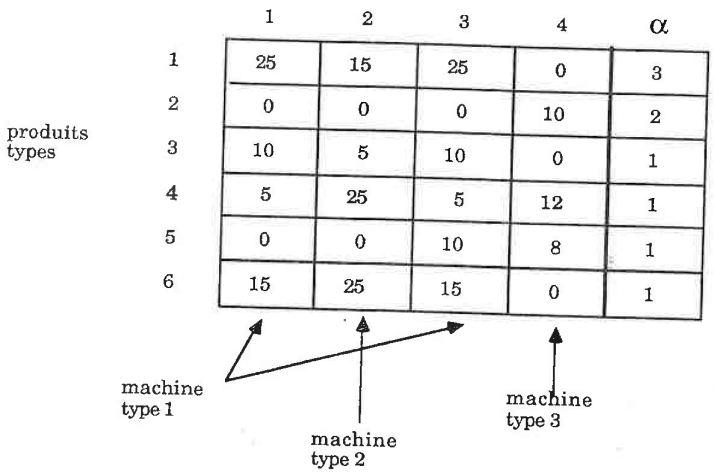
gammes des produits
pour chaque opération : durée et machine
figure 2.3.2.3. a

machines

	1	2	3	4	
1	25	15	25	0	
2	10	5	10	0	produit type 3
3	25	15	25	0	
4	5	25	5	12	produit type 4
5	0	0	10	8	produit type 5
6	25	15	25	0	
7	0	0	0	10	
8	15	25	15	0	produit type 6
9	0	0	0	10	

produit type 1 : lignes 1, 3, 6
produit type 2 : lignes 7, 9

matrice A
figure 2.3.2.3. b



	1	2	3	4	α
1	25	15	25	0	3
2	0	0	0	10	2
3	10	5	10	0	1
4	5	25	5	12	1
5	0	0	10	8	1
6	15	25	15	0	1

matrice A'

figure 2.3.2.3. c

machines types

produits types

	1	2	3	α
1	25	15	0	3
2	0	0	10	2
3	10	5	0	1
4	5	25	12	1
5	0	0	8	1
6	15	25	0	1
β	2	1	1	

matrice A"

figure 2.3.2.3. d

machines types

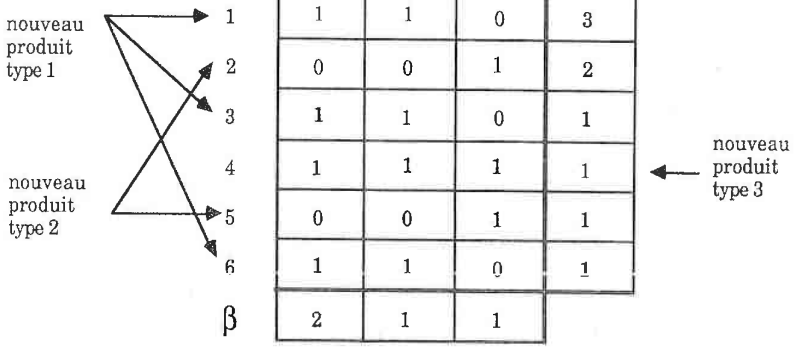
produits types

	1	2	3	α
1	1	0.6	0	3
2	0	0	0.4	2
3	0.4	0.2	0	1
4	0.2	1	0,48	1
5	0	0	0.32	1
6	0.6	1	0	1
β	2	1	1	

matrice V

figure 2.3.2.3. e

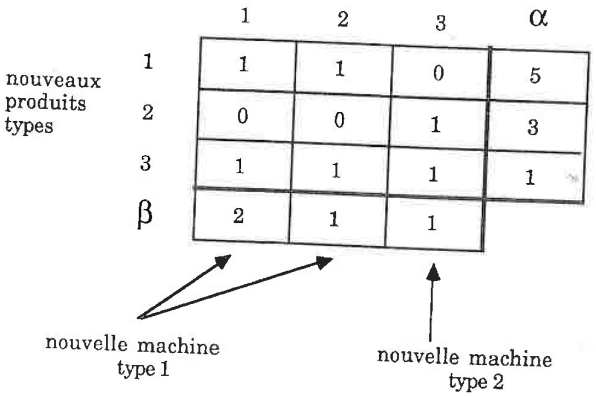
machines types



	1	2	3	α
1	1	1	0	3
2	0	0	1	2
3	1	1	0	1
4	1	1	1	1
5	0	0	1	1
6	1	1	0	1
β	2	1	1	

matrice U

figure 2.3.2.3. f



matrice U'

figure 2.3.2.3. g

nouvelles machines types

nouveaux produits types	1	2	α
1	1	0	5
2	0	1	3
3	1	1	1
β	3	1	

matrice U''

figure 2.3.2.3. h

Nous allons enfin normaliser la matrice A'' pour ramener ces éléments entre 0 et 1 de deux façons : normalisation bivalente et normalisation réelle.

Normalisation en variables 0-1.

Le terme général de la matrice U de N lignes et M colonnes est défini par :

$u(i,j) = 1$ si $a''(i,j) > 0$

0 sinon

(voir la figure 2.3.2.3. f)

Normalisation en variables réelles

Soit E défini par :

$$E = \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} [\, a''(i,j) \,]$$

Le terme général de la matrice V de N lignes et M colonnes est défini par :

$v(i,j) = a''(i,j) / E$

(voir la figure 2.3.2.3. e)

Remarque : Si de nouvelles lignes et/ou colonnes de la matrice U sont identiques, il est possible de la réduire à nouveau en corrigeant en conséquence N, M, les ensembles d'indices Z et Y ainsi que les vecteurs α et β .

Les figures 2.3.2.3. f, g , et h illustrent ces réductions toujours sur le même exemple.

Dans la suite, nous allons présenter la méthode avec la normalisation en variables réelles qui est plus générale. Nous signalerons les simplifications qui apparaissent lorsque l'on remplace la matrice V par la matrice U.

2.3.2.4. Enoncé du problème de décomposition

On se donne le nombre maximal de classes (familles de produits et groupes correspondants de machines) que l'on accepte de constituer, soit n_c ($\leq \min [N,M]$).

L'objectif du problème est de constituer :

- une partition de Z en n_c classes : $P_Z = \{Z_1, \ldots, Z_{nc}\}$
- une partition de Y en n_c classes : $P_Y = \{Y_1, \ldots, Y_{nc}\}$

qui

a) minimise

$$\sum_{\substack{i,j \\ (i,j) \notin \bigcup_{k=1}^{n} (Z_k \times Y_k)}} \alpha(i) \cdot \beta(j) \cdot v(j)$$

b) assure que $v(i,j)$ soit petit lorsque (i,j) est situé hors des blocs. (partie non hachurée de la figure 2.3.2.4. a)

La solution :

$Z_1 = Z$, $Z_2 = Z_3 = \ldots = Z_{nc} = \emptyset$

$Y_1 = Y$, $Y_2 = Y_3 = \ldots = Y_{nc} = \emptyset$

qui satisfait parfaitement a) et b) est exclue par la définition mathématique d'une partition d'un ensemble, à savoir pour Z :

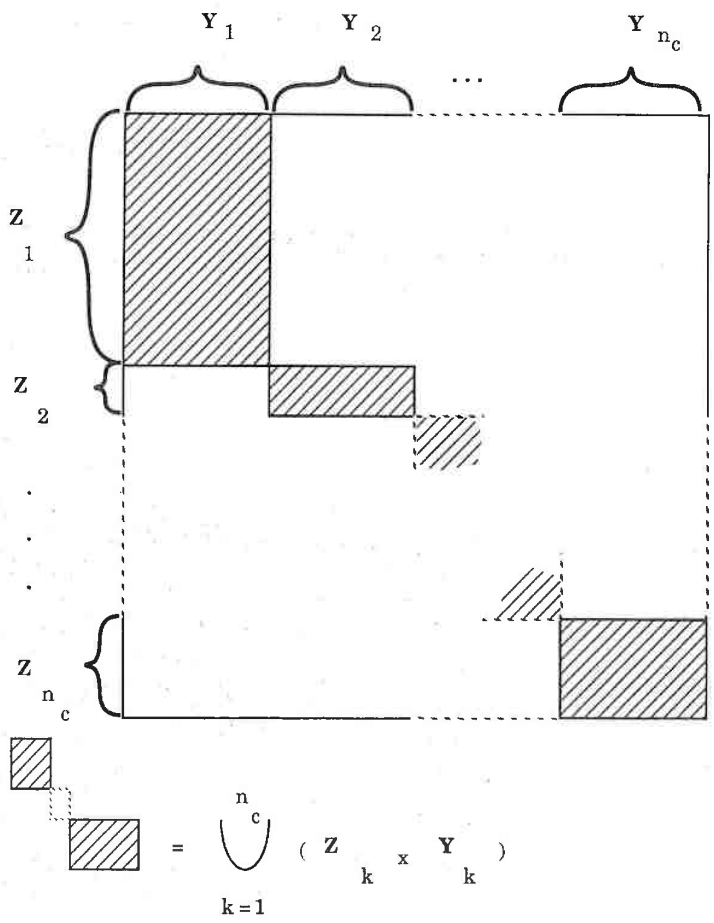
$$P_Z \text{ partition de } Z \Leftrightarrow Z = \bigcup_{k=1}^{n_c} Z_k$$

et

$$\forall k_1 \text{ et } k_2,\ k_1 \neq k_2 : \ Z_{k1} \cap Z_{k2} \neq \emptyset$$

et

$$\forall k \qquad Z_k \neq \emptyset$$



blocs diagonaux

figure 2.3.2.4.. a

Pour satisfaire a) et b), nous proposons de maximiser le critère suivant :

$$\Delta^{\lambda}(P_Z,P_Y) = h \cdot \sum_{i,j} \alpha(i) \cdot \beta(j) \cdot (v(i,j))^{\lambda} \quad (i,j) \in \bigcup_{k=1}^{n_c} (Z_k x Y_k)$$

$$+ (1-h) \cdot \sum_{i,j} \alpha(i) \cdot \beta(j) \cdot (1-v(i,j))^{1/\lambda} \quad (i,j) \notin \bigcup_{k=1}^{n_c} (Z_k x Y_k)$$

avec $0 \le h \le 1$, $0 < \lambda \le 1$ et $0 \le v(i,j) \le 1$

que nous allons commenter en utilisant la figure 2.3.2.4. a .

Si $v(i,j)$ appartient à l'un des blocs diagonaux, zône hachurée, sa participation à la mesure est :

$h \cdot \alpha(i) \cdot \beta(j) \cdot (v(i,j))^{\lambda}$

si $v(i,j) = 0$ alors $v(i,j)^{\lambda} = 0 \quad \forall \lambda > 0$,

si $v(i,j) \neq 0$ alors $v(i,j)^{\lambda}$ tend vers 1 quand λ tend vers 0 .

On a donc :

$$\lim_{\lambda \to 0} (v(i,j))^{\lambda} = u(i,j)$$

Quand $\lambda = 1$, la participation d'un point (i,j) appartenant aux blocs diagonaux est proportionnelle à sa valeur $v(i,j)$ et aux "poids" h, $\alpha(i)$ et $\beta(j)$.

Plus λ est petit et moins l'on accorde d'importance à la valeur exacte de $v(i,j)$, seul le fait qu'il soit ou non différent de 0 reste significatif.

Si (i,j) n'appartient pas à l'un des blocs diagonaux, sa participation à la mesure est :

$(1-h) \cdot \alpha(i) \cdot \beta(j) \cdot (1-v(i,j))^{1/\lambda}$

si $v(i,j) = 0$ alors $(1-v(i,j))^{1/\lambda} = 1 \quad \forall \lambda > 0$

si $v(i,j) \neq 0$ alors $(1-v(i,j))^{1/\lambda}$ tend vers 0 quand λ vers 0 .

On a donc :

$$\lim_{\lambda \to 0} (1 - v(i,j))^{1/\lambda} = 1 - u(i,j)$$

Quand $\lambda = 1$, la participation d'un point (i,j) n'appartenant pas aux blocs diagonaux est proportionnelle à $(1-v(i,j))$ et aux "poids" $1-h$, $\alpha(i)$ et $\beta(j)$.

Plus λ est petit et moins l'on accorde d'importance, dans le critère que l'on cherche à maximiser, aux grandes valeurs de $v(i,j)$. Seules les valeurs de $v(i,j)$ proches de 0 comptent.

En résumé, quand λ tend vers 0 :

si $v(i,j) = 0$ alors
il n'apporte rien en position diagonale,
il apporte le maximum en position hors blocs

si $v(i,j)$ est strictement compris entre 0 et 1 alors
il apporte d'autant plus en position diagonale que $v(i,j)$ est grand et que λ est petit.

si $v(i,j) = 1$ alors
il apporte le maximum en position diagonale,
il n'apporte rien en position hors blocs.

Ceci conduit à n'accepter hors des blocs diagonaux que des petites valeurs de $v(i,j)$.

Si l'on remplace la matrice V par la matrice U, le paramètre λ devient inutile car

$$1^{\lambda}=1 \text{ et } 0^{\lambda}=0 \quad \forall \lambda>0.$$

Soit $\Delta u(P_Z,P_Y)$ la mesure correspondante. Les remarques précédentes montrent que :

$$\lim_{\lambda \to 0} \Delta^{\lambda}(P_Z,P_Y) = \Delta u(P_Z,P_Y)$$

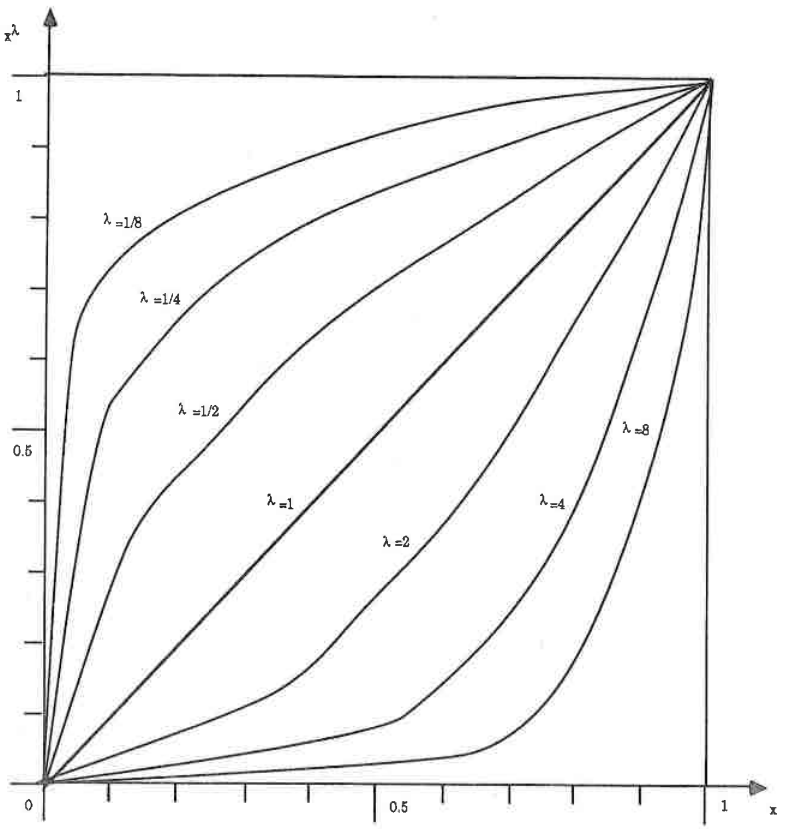
La mesure avec les variables bivalentes est donc la limite asymptotique de la mesure avec les variables réelles lorsque le paramètre λ tend vers 0.

Pour choisir la valeur de λ, on peut étudier les abaques de la figure 2.3.2.4. b. Par exemple avec $\lambda = 1/8$ $(1/\lambda = 8)$

* pour les points appartenant aux blocs diagonaux
$(v(i,j))^{1/8} = 0$ si $v(i,j) = 0$
$0.5 \leq (v(i,j))^{1/8} \leq 1$ si $v(i,j) \geq 0.01$
donc seules les valeurs presque nulles et nulles n'apportent pas de contribution à cette partie du critère.

* pour les points n'appartenant pas aux blocs diagonaux
$(1-v(i,j))^8 \leq 0.01$ si $v(i,j) \geq 0.5$
$(1-v(i,j))^8 \geq 0.4$ si $v(i,j) \leq 0.1$

On cherchera donc à mettre des valeurs ≥ 0.01 à l'intérieur des blocs diagonaux et des $v(i,j)$ les plus petits possibles et de préférence ≤ 0.5 à l'extérieur.



abaque des courbes x^{λ}

pour $0 \leq x \leq 1$ et λ = 1/8, 1/4, 1/2, 1, 2, 4 et 8

figure 2.3.2.4. b

L'interprétation du paramètre h est beaucoup plus facile. Il donne plus ou moins de poids à la partie du critère concernant les blocs diagonaux par rapport à l'extérieur de ces blocs.

En résumé, h et λ étant choisis par l'utilisateur, l'énoncé du problème de décomposition est le suivant :

trouver les partitions P_Z de Z et P_Y de Y qui maximisent $\Delta^{\lambda}(P_Z,P_Y)$.

Dans le paragraphe suivant, nous présentons l'heuristique, développée dans l'équipe, qui conduit à de bonnes solutions de ce problème.

2.3.2.5. Algorithme de classification croisée

A) INITIALISATION

Pour débuter l'algorithme, il est nécessaire d'avoir une partition initiale P_Z de l'ensemble des produits types.

Pour l'obtenir, on utilise la méthode des centre mobiles de Forgey ([FORGEY 1965]) avec par exemple, la distance euclidienne d dans $\mathbb{R}^M$ ([H. GARCIA / J.M. PROTH 1985]).

Soient z_i , $i = 1, 2, \ldots, N$, le produit type i et $\alpha(i)$ son poids. Il s'agit de partitionner Z en $P_Z = \{ Z_1, \ldots , Z_{n_c} \}$ de telle sorte que deux éléments appartenant au même Z_k soient proches l'un de l'autre au sens de la distance d.

On cherche donc à minimiser l'inertie totale interclasse :

$$\sum_{k=1}^{n_c} \sum_{i \in Z_k} \alpha(i) \cdot d^2(z_i,\zeta^k)$$

où ζ^k est le centre d'inertie de Z_k .

(ζ^k est le point de $\mathbb{R}^M$ qui minimise $I_k(u)$ avec

$$I_k(u) = \sum_{i \in Z_k} \alpha(i) \cdot d^2(z_i,u)).$$

L'heuristique débute par la recherche de n_c points ζ^0_k $(1 \le k \le n_c)$ situés dans des zones de forte densité et convenablement répartis dans l'ensemble des points ([PROTH 1985])

Phase préliminaire de la phase d'initialisation :

On se donne un rayon ρ et une distance minimale entre les points initiaux δ avec $\delta \ge 2\rho$. δ et ρ sont fournis par l'utilisateur.

On appelle densité de ζ pour le rayon ρ , le nombre de points Z situés dans la sphère de centre ζ et de rayon ρ :

$$D_\rho(\zeta) = \text{card}(\{ z_i \ / \ d^2(z_i, \zeta) \le \rho^2 \})$$

On commence par :

- $E_0 = Z =$ ensemble des points à répartir en n_c nuages de rayons ρ autour de n_c centres,
- choix arbitraire de x_1 dans E_0.

A la k-ème itération :

- $B_k = \{ z_i \ / \ z_i \in E_{k-1}$ et $d^2(z_i,x_{k-1}) \le \rho^2 \}$
- $E_k = E_{k-1} - B_k$
- $H_k = \{ z_i \ / \ z_i \in E_k$ et $d^2(z_i,x_{k-1}) \ge \delta^2 \}$
- choix arbitraire de x_{k+1} dans H_k

On arrête :

- quand E_k est vide

Ensuite :

- si $k-1 \ge n_c$ alors on trie les x_i par densités décroissantes :

$$D_\rho(x_{i1}) \ge D_\rho(x_{i2}) \ge \ldots \ge D_p(x_{ik-1})$$ et pour l de 1 à n_c : $\zeta^0_l = x_{il}$

sinon il faut recommencer avec une valeur plus faible de δ.

On obtient ainsi des points initiaux.
Puis on itère sur la méthode des centres mobiles :

A l'itération l :

1. on dispose de n_c points $\zeta^{l-1}{}_k$ $(1 \le k \le n_c)$
2. on initialise Z_k à $\emptyset$ $(1 \le k \le n_c)$
3. pour i de 1 à N faire
   3.1. recherche du plus petit indice k qui minimise $d^2(z_i, \zeta^{l-1}{}_k)$
   3.2. $Z_k = Z_k \cup \{i\}$
   finpour
4. pour k de 1 à n_c faire
   4.1. $\zeta^{l}{}_k$ = centre d'inertie de Z_k
   finpour

Si à la fin de l'itération l on a :

$$\forall k : Z^{l}{}_k = Z^{l-1}{}_k$$

alors la phase d'initialisation est terminée, sinon on poursuit les itérations.

Une itération de la méthode des centres mobiles est simple à comprendre : on part des centres d'inertie de l'itération précédente et tout point est attiré par le centre d'inertie le plus proche de lui. On obtient ainsi une nouvelle partition dont on calcule les centres d'inertie, et ainsi de suite.
Il se peut qu'un sous-ensemble Z_k ne reçoive aucun point, dans ce cas :

- ou bien on accepte de diminuer n_c (donc d'éliminer le sous-ensemble vide),
- ou bien on reprend l'algorithme avec de nouveaux paramètres pour l'obtention des $\zeta^{o}{}_k$.

A la fin de l'application de l'algorithme, on a obtenu une **partition initiale.**

B) PARTIE CENTRALE DE LA METHODE DE CLASSIFICATION CROISEE

Soit $P^{lo}{}_Z = \{Z^{lo}{}_1, \ldots, Z^{lo}{}_{nc}\}$ la partition des produits types obtenue à la fin de la phase d'initialisation.

La structure générale de l'algorithme itératif est la suivante :

$l := l_o$

répéter
  $l := l+1$
  calcul de $P^{l}{}_Y$ à partir de $P^{l-1}{}_Z$
  calcul de $P^{l}{}_Z$ à partir de $P^{l}{}_Y$
jusqu'à $(P^{l}{}_Y, P^{l}{}_Z) = (P^{l1}{}_Y, P^{l1}{}_Z)$ avec $l_1 < l$

Où, sauf pour le premier calcul de $P^{lo+1}{}_Y$ qui donne sa première valeur à $\Delta^{\lambda}(P^{lo}{}_Z, P^{lo+1}{}_Y)$, on a :

$$\Delta^{\lambda}(P^{l-1}{}_Z, P^{l}{}_Y) \ge \Delta^{\lambda}(P^{l-1}{}_Z, P^{l-1}{}_Y)$$

et $$\Delta^{\lambda}(P^{l}{}_Z, P^{l}{}_Y) \ge \Delta^{\lambda}(P^{l-1}{}_Z, P^{l}{}_Y)$$

d'où $$\Delta^{\lambda}(P^{l}{}_Z, P^{l}{}_Y) \ge \Delta^{\lambda}(P^{l-1}{}_Z, P^{l-1}{}_Y)$$

Cette propriété est évidente en raison du calcul choisi pour les partitions.

Nous allons détailler les deux modules de calcul des nouvelles partitions $P^{l}{}_Y$ et $P^{l}{}_Z$. Ils sont parfaitement identiques en inversant les notions de lignes et de colonnes et les notations qui s'y rattachent.

Calcul de P^l_Y à partir de P^{l-1}_Z :

$Y^l_1 = Y^l_2 = \ldots = Y^l_{nc} = \emptyset$

pour j de 1 à M faire

pour s de 1 à n_c faire

$$a_s(j) = h \cdot \sum_{i \in Z^{l-1}_s} \alpha(j)\,(v(i,j))^{\lambda} + (1-h) \cdot \sum_{i \notin Z^{l-1}_s} \alpha(i)\,(1-v(i,j))^{1/\lambda}$$

finpour

recherche du plus petit indice s qui maximise $a_s(j)$

$Y^l_s = Y^l_s \cup \{j\}$

finpour

Calcul de P^l_Z à partir de P^l_Y :

$Z^l_1 = Z^l_2 = \ldots = Z^l_{nc} = \emptyset$

pour i de 1 à N faire

pour s de 1 à n_c faire

$$b_s(i) = h \cdot \sum_{i \in Y^l_s} \beta(j) \cdot (v(i,j))^{\lambda} + (1-h) \cdot \sum_{i \notin Y^l_s} \beta(i)\,(1-v(i,j))^{1/\lambda}$$

finpour

recherche du plus petit indice s qui maximise $b_s(i)$

$Z^l_s = Z^l_s \cup \{i\}$

finpour

Cet algorithme converge car les ensembles de partitions $\{P_Z\}$ et $\{P_Y\}$ sont finis et l'on s'arrête lorsqu'on obtient un couple (P_Z, P_Y) déjà obtenu au préalable.

C) REMARQUES ET CONCLUSIONS

On obtient ainsi un algorithme qui ne dégrade jamais la mesure et qui converge vers un optimum local.

Il comporte un point délicat qui est la diminution éventuelle de n_c. Si l'on accepte cette diminution, le critère se dégrade chaque fois qu'une classe disparaît ([N. DRIDI / J.M. PROTH 1986]). C'est sur ce dernier point que la méthode très voisine de F. MARCOTORCHINO apporte des améliorations.

2.3.3. Définitions et méthode générale

2.3.3.1. Décomposition du problème en sous-problèmes

a) Constitution des îlots et cas particuliers

La méthode de classification croisée décrite précédemment fournit les informations suivantes :

- n_c : nombre de classes, c'est-à-dire nombre d'îlots de fabrication ou de groupes de machines types,
  mais aussi nombre de familles de produits types,
  n_c est supposé > 1 ;
- $P_Z = \{ Z_1, Z_2, \ldots, Z_{nc} \}$: partition des produits types en familles ;
- $P_Y = \{ Y_1, Y_2, \ldots, Y_{nc} \}$: partition des machines types en îlots de fabrication.

Pour traiter le problème d'ordonnancement, il faut revenir aux informations détaillées sur les produits et les machines.
A partir de (n_c, P_Z, P_Y) , de l'appartenance des produits et des machines aux produits types et aux machines types, et de l'énoncé du problème initial (n,m,r,p,μ,d) , on obtient les informations détaillées suivantes :

$\forall$ î, $1 \le$ î $\le n_c$:

$m^{\circ\circ}$(î) = nombre de machines dans l'îlot î , correspondant à la partition $Y_{\hat{i}}$

$\forall$ ô, $1 \le$ ô $\le m$:

îam(ô) = numéro de la partition de P_Y contenant la machine ô
= îlot d'appartenance de la machine ô

$\forall$ û, $1 \le$ û $\le m$:

îap (û) = numéro de la partition de P_Z contenant le produit type correspondant au produit û
= son îlot mère ou son îlot principal.

Un produit û sort de son îlot mère dans le cas suivant (condition $\mathbb{SP}$).

$\mathbb{SP}$) $\exists k,\ 1 \le k \le q(\hat{u}) :\quad \hat{i}am(\mu(\hat{u},k)) \neq \hat{i}ap(\hat{u})$.

Nous dirons que la <u>décomposition</u> est <u>parfaite</u> lorsque tout produit est traité uniquement à l'intérieur de son îlot d'appartenance, la condition $\mathbb{DP}$ est alors vérifiée :

$\mathbb{DP}$) $\forall\ \hat{u}\ ,\ 1 \le \hat{u} \le n$: $\hat{i}am(\mu(\hat{u},k)) = \hat{i}ap(\hat{u})$

$\forall\ k,\ 1 \le k \le q(\hat{u})$

Dans le cas de décomposition parfaite, nous sommes ramenés à n_c sous-problèmes indépendants identiques au problème initial et comportant chacun moins de machines et de produits. Il est alors possible de les résoudre, par des méthodes exactes s'ils sont suffisamment petits, ou par des méthodes approchées sinon, par exemple en leur appliquant à nouveau des méthodes de décompositions spatiales et/ou temporelles.

Même si la décomposition n'est pas parfaite, elle peut contenir des <u>îlots indépendants</u>, c'est-à-dire des îlots où tout produit qui le visite est dans son îlot mère et ne visite que ce seul îlot. C'est la condition d'indépendance $\mathbb{INDP}$ pour un îlot donné $\hat{i}_0$:

$\mathbb{INDP}$) $\forall\ \hat{u} :\quad \hat{i}ap(\hat{u}) = \hat{i}_0 \quad ====>$

$\forall\ k : 1 \le k \le q(\hat{u}) :\quad \hat{i}am(\mu(\hat{u},k)) = \hat{i}_0\ (=\hat{i}ap(\hat{u}))$

et

$\forall\ \hat{u} :\quad \hat{i}ap(\hat{u}) \neq \hat{i}_0 \quad ====>$

$\forall\ k : 1 \le k \le q(\hat{u}) :\quad \hat{i}am(\mu(\hat{u},k)) \neq \hat{i}_0$

Tout îlot indépendant peut être ordonnancé isolément, de la même manière que ceux issus d'une décomposition parfaite.

En cas de décomposition imparfaite, une phase préliminaire permet de supprimer éventuellement les îlots indépendants pour les traiter séparément.

Une seconde phase préliminaire permet encore de séparer le problème restant en sous-problèmes indépendants s'il existe des groupes d'îlots indépendants les uns des autres.

La condition d'indépendance d'un groupe d'îlots s'écrit

$\mathbb{INPG}$) $\forall\ \hat{u} :\ \hat{i}ap(\hat{u}) = \hat{i}$ avec $\hat{i} \in G \quad ====>$

$\forall\ k :\ 1 \le k \le q(\hat{u}) :\quad \hat{i}am(\mu(\hat{u},k)) = \hat{i}_k$ avec $\hat{i}_k \in G$

et

$\forall\ \hat{u} :\ \hat{i}ap(\hat{u}) = \hat{i}$ avec $\hat{i} \notin G \quad ====>$

$\forall\ k :\ 1 \le k \le q(\hat{u}) :\quad \hat{i}am(\mu(\hat{u},k)) = \hat{i}_k$ avec $\hat{i}_k \notin G$

La figure 2.3.3.1. a illustre ces deux phases préliminaires.

Dans la suite, on considère un seul groupe de n_c îlots dépendants entre eux à traiter par la méthode d'ordonnancement.

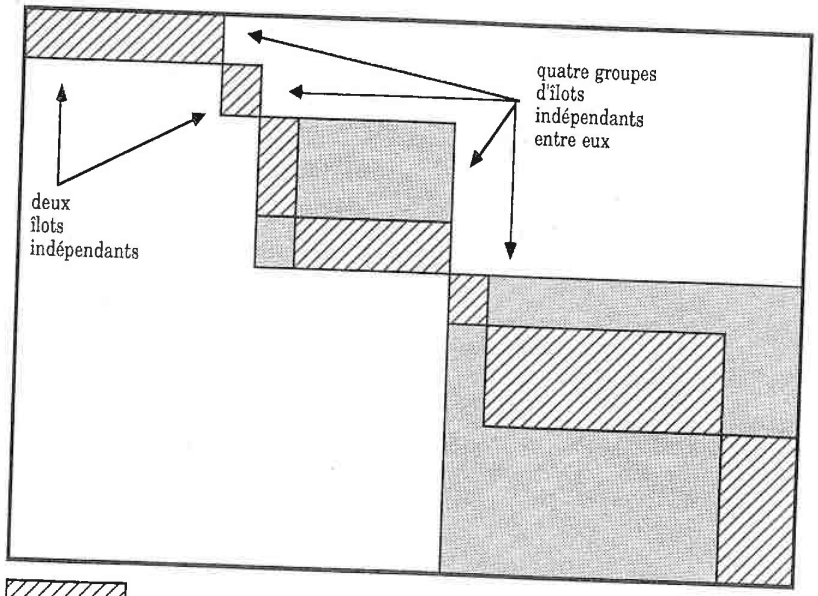


figure 2.3.3.1. a

b) Décomposition des produits

Un produit û qui ne vérifie pas la condition SP visite plusieurs îlots. Il doit être divisé en autant de sous-produits que d'îlots successivement visités. Plusieurs sous-produits du même produit peuvent être traités dans le même îlot si entre temps le produit est passé par un (ou plusieurs) autre(s) îlot(s).

Les informations nouvelles concernant les produits sont les suivantes :

sp (û) : nombre de sous-produits du produit û ;

ra(û,j) : instant de début de fabrication au plus tôt du j-ème sous-produit du produit û ;

da(û,j) : instant (souhaité) de fin de fabrication au plus tard du j-ème sous-produit du produit û ;

$pp^{\circ\circ}(\hat{u},j)$: durée totale minimale de présence du j-ème sous-produit du produit û. (durées opératoires si les transports et les réglages sont négligeables) ;

îasp(û,j) : numéro de l'îlot où s'exécute le j-ème sous-produit du produit û ;

$q^{\circ\circ}(\hat{u},j)$: nombre d'opérations dans la gamme du j-ème sous-produit du produit û ;

$p^{\circ\circ}(\hat{u},j,k)$: durée de la k-ème opération du j-ème sous-produit du produit û ;

$\mu^{\circ\circ}(\hat{u},j,k)$: machine nécessaire à la k-ème opération du j-ème sous-produit du produit û ;

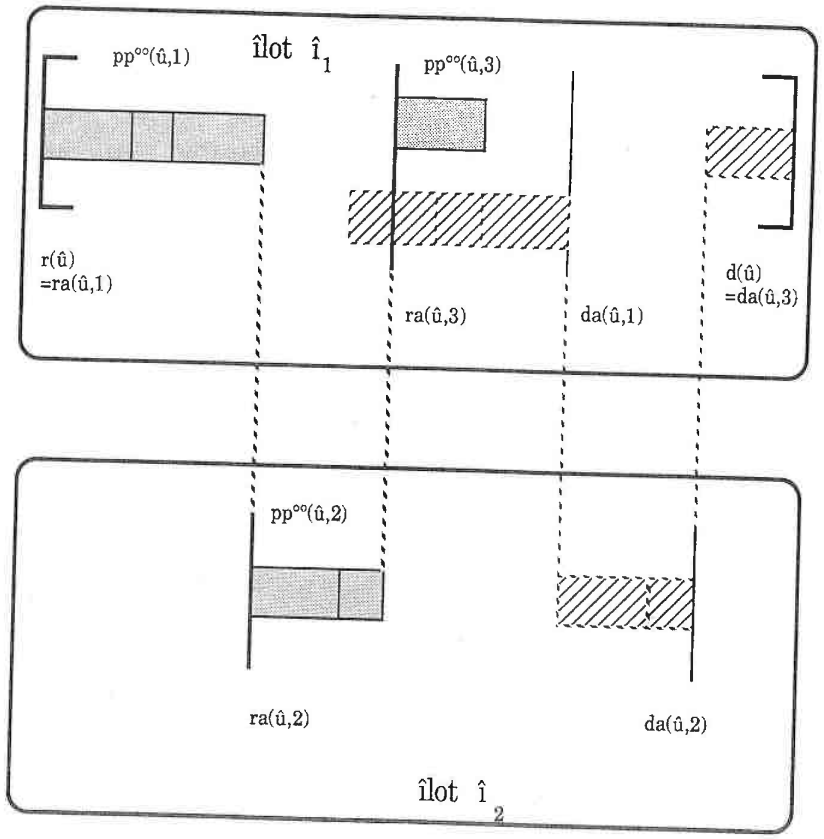
ainsi que

$n^{\circ\circ}(\hat{i})$: nombre de sous-produits dans l'îlot î .

Les figures 2.3.3.1 b) et c) illustrent la correspondance entre les informations.

Et l'algorithme suivant permet leur calcul :

$\hat{i}_0$:= îam($\mu(\hat{u},1)$) ; k:= 0 ; ks:= 0 ; rs:= r(û) ; ds:= d(û) - pp(û) ; j:= 1 ; ra(û,1):= rs ;
répéter
k := k+1 ; ks := ks+1 ; $\hat{i}_1$:= îam($\mu(\hat{u},k)$) ;
si $\hat{i}_0 \neq \hat{i}_1$ alors
da(û,j) := ds ; j:= j+1 ; ra(û,j) := rs ; ks := 1
fsi
$p^{\circ\circ}$ (û,j,ks) := p(û,k) ; $\mu^{\circ\circ}(\hat{u},j,ks) := \mu(\hat{u},k)$; rs := rs+p(û,k) ;
ds := ds+p(û,k) ; $\hat{i}_0 := \hat{i}_1$
jusqua k = q(û) ;
da(û,j) := ds



instants de début au plus tôt et de fin au plus tard des sous-produits
figure 2.3.3.1. b

exemple avec un produit û de 6 opérations (q(û)=6)

numéro des opérations	k	1	2	3	4	5	6
numéro des machines	m_k	1	2	1	3	5	2
numéro des îlots	$\hat{i}_k$		**1**			**2**	**1**
numéro des sous-produits	j		1			2	3
numéro des opérations des sous-produits	ks_j	1	2	3	1	2	1

$p^{\circ\circ}(\hat{u},j,ks_j) = p(\hat{u},k)$ et $\mu^{\circ\circ}(\hat{u},j,ks_j) = \mu(\hat{u},k)$

correspondance entre la gamme du produit û et celle de ses sous-produits
figure 2.3.3.1. c

c) Etablissement de l'indépendance des sous-produits

Les ra(û,j) et les da(û,j) sont des dates au plus tôt et au plus tard des sous-produits au sens du PERT. Ce sont des limites extrêmes à ne pas dépasser pour un sous-produit si on ne veut pas dépasser celles du produit. Nous les appelons des DELAIS EXTERNES.

Observons leur position sur l'axe des temps, par exemple sur la figure 2.3.3.1. b .
Nous constatons que l'intervalle [ra(û,1),da(û,1)] , qui représente la plage de temps sur laquelle il est possible, sans retard pour le produit û , de planifier son premier sous-produit, a une intersection non vide avec l'intervalle [ra(û,2),da(û,2)] . Cela signifie que l'on ne peut pas planifier les deux sous-produits indépendamment l'un de l'autre.

Pour acquérir cette indépendance, nous allons attribuer aux différents sous-produits d'un même produit des intervalles de temps dont les intersections sont vides. Les extrémités de ces intervalles seront appelées les DELAIS INTERNES et seront désignées par $r^{\circ\circ}(\hat{u},j)$ pour le début de l'intervalle et $d^{\circ\circ}(\hat{u},j)$ pour la fin. Nous les ferons évoluer au sein des algorithmes d'ordonnancement par décomposition.

La figure 2.2.3.1. d montre un exemple d'un tel partage.

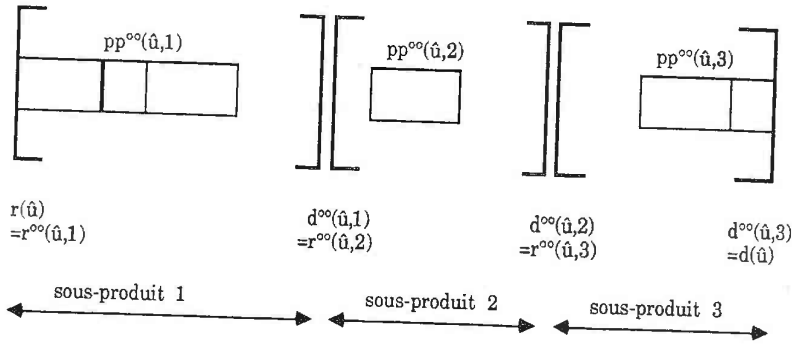


figure 2.3.3.1. d

Les délais internes doivent vérifier les conditions DI :

DI) $\forall\, \hat{u} : 1 \le \hat{u} \le n,\ \forall\, j : 1 \le j \le sp(\hat{u})$:

$ra(\hat{u},j) \le r^{\circ\circ}(\hat{u},j)$
$r^{\circ\circ}(\hat{u},j) + pp^{\circ\circ}(\hat{u},j) \le d^{\circ\circ}(\hat{u},j)$
$d^{\circ\circ}(\hat{u},j) \le r^{\circ\circ}(\hat{u},j+1)$ $\quad (j<sp(\hat{u}))$

Si le produit est "critique" au sens du PERT, c'est-à-dire si $r(\hat{u}) + pp(\hat{u}) \ge d(\hat{u})$, la seule manière raisonnable de choisir les délais internes initiaux des sous-produits est de les prendre égaux aux délais externes des sous-produits.
Dans tous les autres cas, il y a de multiples façons de choisir des délais internes vérifiant DI.

Nous aurons autant d'algorithmes différents d'ordonnancement par décomposition spatiale que de manières de choisir les délais internes initiaux. Nous devons donc nous donner une règle **R** de construction des délais internes initiaux.

Celles que nous avons utilisées utilisent la marge totale des produits :

$$marge(\hat{u}) = d(\hat{u}) - r(\hat{u}) - pp(\hat{u})$$

La règle **R1** partage la marge totale à égalité entre les sous-produits :

$r^{\circ\circ}(\hat{u},j) := d^{\circ\circ}(\hat{u},j-1)$ et $d^{\circ\circ}(\hat{u},j) := r^{\circ\circ}(\hat{u},j) + pp^{\circ\circ}(\hat{u},j) + marge(\hat{u}) / sp(\hat{u})$.

La règle **R2** partage la marge totale proportionnellement à la durée de fabrication des sous- produits :

$$r^{\circ\circ}(\hat{u},j) := d^{\circ\circ}(\hat{u},j-1) \quad \text{et} \quad d^{\circ\circ}(\hat{u},j) := r^{\circ\circ}(\hat{u},j) + pp^{\circ\circ}(\hat{u},j) + marge(\hat{u}) \times \frac{pp^{\circ\circ}(\hat{u},j)}{pp(\hat{u})}$$

On peut envisager d'autres règles, faisant intervenir le nombre d'opérations par sous-produits ou la charge des îlots concernés.

Il est très important de remarquer que ces délais internes sont arbitraires. Ils ne font pas partie des contraintes initiales du problème posé. Ce ne sont que des objets intermédiaires utiles à la méthode, mais leur choix influe sur les optimaux locaux qui seront cherchés indépendamment pour chaque îlot.

Aussi, lorsque l'on construit la solution globale à partir des solutions localement optimales, on n'obtient pas en général l'optimum global.
On ignore si la solution trouvée est optimale, sauf dans le cas où le critère minimisant la somme des retards est nul.

L'arbitraire du choix des délais internes transforme donc la méthode de décomposition spatiale en heuristique. Pour rester dans le cadre des algorithmes exacts, il faudrait choisir toutes les valeurs possibles pour les délais internes et appliquer pour chacun la méthode d'ordonnancement par décomposition spatiale, mais cela serait très coûteux, car on retrouve alors toute la combinatoire du problème général.

d) Modélisation finale des sous-problèmes

Rassemblons à présent toutes les informations obtenues.

Pour un îlot donné î, nous avons :

$n^{\circ\circ}(\hat{i})$: nombre de sous-produits de l'îlot,
$m^{\circ\circ}(\hat{i})$: nombre de machines de l'îlot
$r^{\circ\circ}(\hat{u},j)$: instant de début au plus tôt du sous-produit $(\hat{u},j)$
$d^{\circ\circ}(\hat{u},j)$: instant de fin au plus tard du sous-produit $(\hat{u},j)$
$p^{\circ\circ}(\hat{u},j,k)$: durée de la k-ème opération du sous-produit $(\hat{u},j)$
$\mu^{\circ\circ}(\hat{u},j,k)$: machine pour cette opération

En remplaçant îlot par atelier et sous-produit par produit, ce problème est exactement le problème initial, modélisé en 2.1.1. On utilisera aussi le critère "minimiser la somme des retards".

e) Construction de la solution globale

Chaque îlot est ordonnancé indépendamment des autres, de manière à minimiser localement la somme des retards. En rassemblant les îlots, on se trouve dans l'une des situations suivantes :

α) La somme totale de tous les retards est nulle

Cela signifie qu'aucun sous-produit n'a dépassé les délais internes qui lui étaient attribués et qu'aucun produit n'est en retard. La solution globale est réalisable et optimale.

β) La somme totale de tous les retards n'est pas nulle, mais on a :

NR) $\forall\ \hat{u} : c(\hat{u}) \leq d(\hat{u})$

NC) $\forall\ \hat{u}, \forall\ j < sp(\hat{u}) : t^{\circ\circ}(\hat{u},j,q^{\circ\circ}(\hat{u},j)) + p^{\circ\circ}(\hat{u},j,q^{\circ\circ}(\hat{u},j)) \leq t^{\circ\circ}(\hat{u},j+1,1)$

NR est la condition de non-retard des produits,
NC est la condition de non-chevauchement des sous-produits.

Dans ce cas β), des sous-produits sont en retard par rapport aux délais internes, jamais par rapport aux délais externes et lorsque des sous-produits sont en retard, les sous-produits qui suivent ne commencent pas immédiatement comme le montre la figure 2.3.3.1. e .
Ici, la solution globale est encore réalisable et optimale. Le retard vrai est nul.

γ) La somme totale de tous les retards n'est pas nulle :
on a la condition NC mais pas la condition NR

Dans ce cas, la solution globale est réalisable puisqu'il n'y a pas de chevauchement entre les sous-produits d'un même produit.
On ignore si elle est optimale, puisque la somme des retards vrais dépend du choix arbitraire des délais internes.

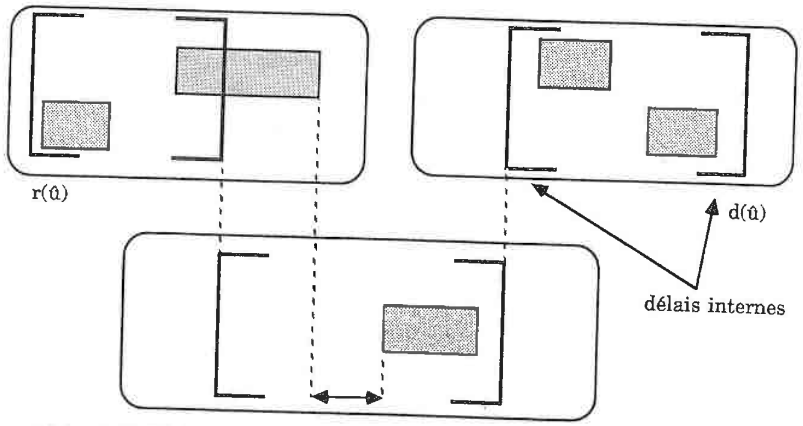
δ) La condition NC n'est pas vérifiée

Cela signifie que deux sous-produits d'un même produit se chevauchent comme le montre la figure 2.3.3.1. f . Les conditions de faisabilité **C** ne sont pas vérifiées : la solution globale n'est pas réalisable.

Dans les situations α), β) et γ), on arrête la méthode d'ordonnancement par décomposition spatiale, sans garantie d'optimalité dans le cas γ).

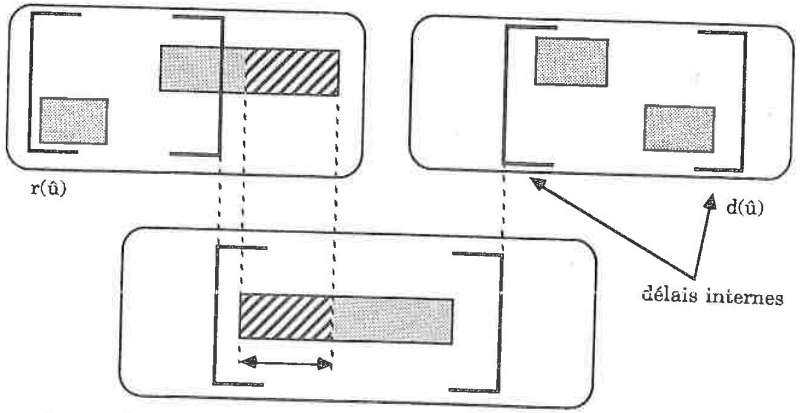
Dans la situation δ), il faut faire disparaître les chevauchements. La méthode consiste à itérer sur les ordonnancements d'îlots jusqu'à leur disparition totale.

Une telle méthode a été programmée et testée dans l'équipe SAGEP par Najoua DRIDI ([N. DRIDI / M.C. PORTMANN / J.M. PROTH 1986] et [N. DRIDI / J.M. PROTH 1987]). Nous la présentons en détail au paragraphe suivant.



existence de retards par rapport aux délais internes sans chevauchement

figure 2.3.3.1. e



existence de retards par rapport aux délais internes avec chevauchement

figure 2.3.3.1. f

2.3.4. Un algorithme d'ordonnancement par décomposition spatiale

Il est composé de deux parties : une phase préliminaire où l'on classe les îlots par ordre de priorités décroissantes, suivie d'une phase itérative comprenant l'ordonnancement de tous les îlots par priorités décroissantes, les tests de chevauchement entre les îlots et la modification en conséquence des délais internes.

L'algorithme s'arrête lorsqu'il n'y a plus de chevauchement.

2.3.4.1. Priorité des îlots

Il s'agit de classer les îlots de telle sorte qu'au cours d'une itération sur tous les îlots (nous parlerons alors de "macro-itération"), les îlots les plus contraints (qui risquent de créer le plus de retards) soient ordonnancés en priorité.

Il faut donc trouver un ordre $\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_{nc}$ des îlots à partir d'une règle de tri $\mathbf{T}$. Il y aura autant de méthodes de décomposition spatiale que de choix pour cette règle de tri.

Par exemple, on peut choisir les règles $\mathbf{T_1}$ ou $\mathbf{T_2}$.

a) Règle T_1 : charge moyenne par machine décroissante

$$cmm(\hat{i}) = \frac{1}{m^{oo}(\hat{i})} \cdot \sum_{\substack{(\hat{u},j) : \\ \hat{i}asp(\hat{u},j)=\hat{i}}} pp^{oo}(\hat{u},j)$$

$cmm(\hat{i})$ est égal à la somme de toutes les durées des opérations à faire dans l'îlot divisée par le nombre de machines de l'îlot.

Pour la règle T_1, on a :

$$cmm(\hat{i}_1) \geq cmm(\hat{i}_2) \geq \ldots \geq cmm(\hat{i}_{nc})$$

b) Règle T_2 : marge moyenne par sous-produits croissante

$$mmsp(\hat{i}) = \frac{1}{n^{oo}(\hat{i})} \cdot \sum_{\substack{(\hat{u},j) : \\ \hat{i}asp(\hat{u},j)=\hat{i}}} (d^{oo}(\hat{u},j) - r^{oo}(\hat{u},j) - pp^{oo}(\hat{u},j))$$

$mmsp(\hat{i})$ est égal à la somme des marges des sous-produits dans l'îlot.

Pour la règle T_2, on a :

$$mmsp(i_1) \leq mmsp(\hat{i}_2) \leq \ldots \leq mmsp(\hat{i}_{nc})$$

La règle T_1 est statique : elle ne dépend pas des délais internes. Il suffit de l'appliquer une fois pour toutes a priori.

La règle T_2 est dynamique : elle varie avec les délais internes. On peut l'appliquer une seule fois a priori avec les valeurs initiales des délais internes ou au contraire l'appliquer à nouveau au début de chaque macro-itération.

2.3.4.2. Macro-itération sur tous les îlots

La seule information importante transmise d'une macro-itération à la suivante est la valeur des délais internes. Initialement, ils ont été calculés grâce au choix de la règle **R** de partage des marges, lors du découpage en sous-produits.

On ordonnance les n_c îlots dans l'ordre $\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_{nc}$ obtenu en appliquant la règle **T**.

A la fin de l'ordonnancement de $\hat{i}_k$, pour tout sous-produit de cet îlot $(\hat{u},j)$, on effectue les traitements d'ajustage des délais internes suivants :

Si $j > 1$ et si $\hat{i}asp(\hat{u},j-1) = \hat{i}_l$ avec $l > k$ alors

(le sous-produit qui précède est dans un îlot non encore ordonnancé au cours de la macro-itération)

on attribue à $(\hat{u},j-1)$ toute la marge aval disponible :

$d^{oo}(\hat{u},j-1) = r^{oo}(\hat{u},j) = t^{oo}(\hat{u},j,1)$

où $t^{oo}(\hat{u},j,1)$ est la date de début de $(\hat{u},j,1)$ dans l'ordonnancement de $\hat{i}_k$.

Si $j < sp(\hat{u})$ et si $\hat{i}asp(\hat{u},j+1) = \hat{i}_l$ avec $l > k$ alors

(le sous-produit qui suit est dans un îlot non encore ordonnancé au cours de la macro-itération)

on attribue à $(\hat{u},j+1)$ toute la marge amont disponible :

$d^{oo}(\hat{u},j) = r^{oo}(\hat{u},j+1) = t^{oo}(\hat{u},j,q^{oo}(\hat{u},j)) + p^{oo}(\hat{u},j,q^{oo}(\hat{u},j))$.

A la fin de l'ordonnancement des n_c îlots, c'est-à-dire en fin de macro-itération, on recherche les chevauchements entre sous-produits d'un même produit.

S'il n'y en a pas, la méthode spatiale est terminée.

Sinon, pour tout chevauchement entre les sous-produits $(\hat{u},j)$ et $(\hat{u},j+1)$, on corrige à nouveau les délais internes (exclusivement en les augmentant) :

(on reporte le délai de début au plus tôt du deuxième sous-produit à la fin effective du premier)

$d^{oo}(\hat{u},j) = r^{oo}(\hat{u},j+1) = t^{oo}(\hat{u},j,q^{oo}(\hat{u},j)) + p^{oo}(\hat{u},j,q^{oo}(\hat{u},j))$.

Il est possible d'apporter à la méthodes des améliorations techniques.

Dans une macro-itération, il n'est pas utile de réordonnancer un îlot si depuis son dernier ordonnancement, on n'a pas modifié les délais internes.
On peut essayer de réduire les chevauchements en donnant dans le critère un poids plus grand aux retards des sous-produits dont les suivants dans l'ordre de la gamme ont été planifiés dans les îlots qui précédaient ; il faut alors choisir un poids intermédiaire pour les derniers sous-produits de chaque produit dont le retard est toujours compatible avec les conditions de faisabilité, mais traduit un retard vrai pour le produit.

On évite également les chevauchements en exigeant qu'entre deux sous-produits d'un même produit figurant dans le même îlot, il y ait l'intervalle de temps minimum nécessaire à l'exécution des opérations intermédiaires à exécuter à l'extérieur de l'îlot. Cette condition est vérifiée à l'initialisation lors du partage des marges, mais peut être perdue par la modification des délais internes. On la rétablit lorsqu'on ordonnance chaque îlot.

2.3.5. Remarques

L'algorithme défini au paragraphe 2.3.4. converge puisqu'à chaque macro-itération la date de début de premier chevauchement croît strictement et qu'il existe au moins une solution réalisable sans chevauchement que l'on peut obtenir à partir de n'importe quel ordonnancement en repoussant globalement tout ce qui suit les chevauchements.

On peut imaginer des variantes moins draconiennes pour la suppression progressive des chevauchements. Par exemple, à chaque macro-itération, on ne change les délais que pour le ou les premiers chevauchements (dont l'instant de début de conflit entre opérations est minimal).

Dans ce cas, on augmente probablement le nombre de macro-itérations et donc le coût total de l'algorithme, mais on recule le moment où les retards vrais apparaissent. Le critère "somme des retards" devrait, en moyenne, être meilleur.

2.4. DECOMPOSITION SPATIALE ET TEMPORELLE

2.4.1. Introduction

L'idée générale consiste à croiser décomposition spatiale et décomposition temporelle.

Nous avons déjà suggéré une première possibilité de croisement. Dans la méthode de décomposition spatiale, si un îlot $\hat{i}_k$ contient trop d'opérations à ordonnancer, il faut remplacer la méthode exacte d'ordonnancement par une méthode approchée, par exemple une méthode de décomposition temporelle.

Le croisement peut également être fait dans l'autre sens. Au cours d'une méthode de décomposition temporelle, remplacer la méthode exacte d'optimisation locale par une méthode approchée, par exemple une méthode de décomposition spatiale.

C'est plutôt cette deuxième approche qui est à l'originie de la méthode proposée, puisque nous utilisons une méthode de décomposition temporelle où à chaque itération un sous-ensemble de sous-produits est ajouté dans un seul îlot. Pour croiser plus généralement les deux méthodes selon la deuxième approche, il faudrait faire une décomposition en îlots différente à chaque itération de la méthode de décomposition temporelle, ce que nous évitons : la décomposition en îlots est faite a priori comme dans la méthode de décomposition spatiale.

Comme le critère essentiel est la somme des retards vrais, c'est-à-dire les retards des produits et non pas la somme des retards des sous-produits par rapport aux délais internes, on tiendra compte en permanence de l'incidence des retards des sous-produits sur le retard des produits.

Au paragraphe suivant, nous allons présenter les lignes générales des méthodes de décomposition spatiales et temporelles, puis nous présenterons les particularités de notre algorithme.

2.4.2. Définitions et méthode générale

a) Décomposition spatiale

Comme pour la méthode de décomposition spatiale :

- on applique la méthode de classification croisée pour regrouper les machines en îlots de fabrication ;

- on remplace le problème initial par un ensemble de sous-problèmes identiques associés à des groupes d'îlots indépendants et dorénavant on s'intéresse à l'ordonnancement d'un seul groupe d'îlots indépendant des autres groupes ;

- si le groupe d'îlots est réduit à un seul îlot, on lui applique simplement la méthode de décomposition temporelle ;

- si le groupe d'îlots contient plus d'un îlot, on décompose les produits en sous-produits et on leur affecte des délais internes par répartition de la marge des produits.

b) Décomposition temporelle

Il faut disposer d'un procédé de regroupement des sous-produits en sous-ensembles de sous-produits.

Nous utilisons deux critères pour constituer ces sous-ensembles : il regroupent des sous-produits d'un même îlot et ils sont liés dans le temps par une suite î-temporelle φ.

Cette suite î-temporelle φ a des propriétés légèrement différentes de celle utilisée dans la méthode de décomposition temporelle.

Nous appelons suite î-temporelle φ associée à un problème d'ordonnancement décomposé en îlots, une suite finie d'instants $\varphi_0, \varphi_1, \ldots, \varphi_{L+1}$ de L+2 termes telle que :

α) $\varphi_0 \leq \varphi_1 \leq \varphi_2 \ldots \leq \varphi_{L+1}$: φ est croissante

β) $\varphi_0 \leq \min_i [r(i)]$ et $\varphi_{L+1} = +\infty$

l'intervalle $[\varphi_0, \varphi_{L+1}[$ contient toutes les dates d'arrivée de sous-produits.

Un sous-produit $(\hat{u},j)$ est accessible lorsqu'il est le premier sous-produit de la gamme du produit $\hat{u}$ $(j=1)$ ou lorsque les sous-produits $(\hat{u},ja)$ qui le précèdent dans la gamme $(ja<j)$ ont déjà été introduits dans les ordonnancements partiels de la méthode de décomposition spatiale et temporelle.

A l'itération k $(1 \leq k \leq L+1)$, nous choisissons un îlot $\hat{\imath}_k$ et nous ajoutons à son ordonnancement partiel le sous-ensemble de sous-produits défini par :

$\pi_k = \{ (\hat{u},j) \ / \ (\hat{u},j)$ accessible et $\hat{\imath}asp(\hat{u},j) = \hat{\imath}_k$ et $r^{\circ\circ}(\hat{u},j) \leq \varphi_k$ et $(\hat{u},j)$ non encore introduit dans l'ordonnancement$\}$

Cette suite π de sous-ensembles de sous-produits ne peut pas être construite a priori, comme dans le cas de la décomposition temporelle, car nous allons faire évoluer les $r^{\circ\circ}(\hat{u},j)$ au cours des itérations.

c) Ordonnancement local

A chaque itération, comme dans la méthode de décomposition temporelle :

- on ne remet pas en cause les tâches planifiées qui sont terminées avant l'arrivée du premier sous-produit à ajouter,

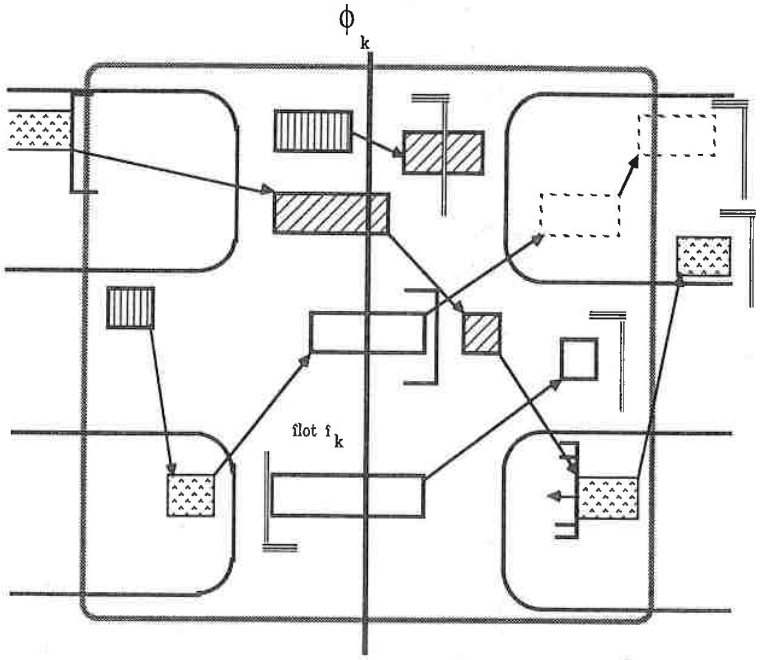
- on ordonnance conjointement les tâches non terminées avec celles ajoutées en minimisant une somme pondérée de retards internes et de retards externes :

* des poids forts pénalisent les retards vrais par rapport aux délais externes $da(\hat{u}_j)$,

* des poids infinis interdisent qu'un sous-produit se mette en parallèle avec son successeur déjà planifié dans un autre îlot ; dans ce cas, pour éviter des chevauchements, les délais internes $d^{\circ\circ}(\hat{u}_j)$, ajustés au début du sous-produit suivant $t^{\circ\circ}(\hat{u}_{j+1})$, deviennent des délais impératifs ou "deadlines",

* des poids faibles pénalisent les simples dépassements par rapport aux délais internes $d^{\circ\circ}(\hat{u}_j)$ lorsque les sous-produits suivants ne sont pas planifiés et que le produit conserve encore une marge totale positive ou nulle.

La figure 2.4.2. illustre les particularités de l'optimisation locale.



tâches planifiées dans les autres îlots, leur position est fixée pendant tout l'ordonnancement local de l'îlot f_k

tâches planifiées de l'îlot f_k qui sont définitivement fixées

tâches déjà ordonnancées de l'îlot f_k qui sont réordonnancées

tâches introduites dans l'îlot f_k à cette itération

tâches suivantes qui seront introduites plus tard dans d'autres îlots

délais externes

délais internes

délais internes impératifs

optimisation locale dans l'îlot f_k

figure 2.4.2.

2.4.3. Un algorithme d'ordonnancement par décomposition spatiale et temporelle

Pour finir de préciser la méthode, il faut donner la méthode de calcul de la suite î-temporelle φ et la manière de choisir l'îlot de la k-ème itération. Il faut également préciser l'évolution des délais internes.

2.2.3.1. La suite î-temporelle φ et le choix de l'îlot $\hat{i}_k$

On peut imaginer plusieurs schémas pour construire la suite î-temporelle φ et la suite d'îlots associés.

Par exemple :
un découpage uniforme du temps et une boucle sur tous les îlots pour une valeur donnée de φ

$$\varphi_{(o,nc)} = \min_i [r_i]$$

$$\varphi_{max} = \max [\max_i [r(i) + pp(i)] , \max_i [d(i)]] . KD$$

où KD est un coefficient de dilatation supérieur ou égal à 1.

φ_{max} est une valeur approchée empirique de la future durée totale de l'ordonnancement (car les délais internes $r^{oo}(\hat{u},j)$ qui varient dynamiquement peuvent devenir beaucoup plus grands que $\max_i [r(i)]$).

$$\Delta\varphi = \frac{\varphi_{max} - \varphi_{(o,nc)}}{L+1} ;$$

$$\forall k_1 \ 1 \le k_1 \le L : \quad \varphi_{k1,1} = \varphi_{k1-1,nc} + \Delta\varphi$$

$$\forall k_2 \ 1 \le k_2 \le n_c : \quad \varphi_{k1,k2} = \varphi_{k1,1}$$

$\varphi_{(o,nc)} < \varphi_{(1,1)} = \varphi_{(1,2)} = \dots = \varphi_{(1,nc)} < \varphi_{(2,1)} = \varphi_{(2,2)} \dots$ est la suite î-temporelle ainsi définie.

Les îlots sont ordonnancés dans l'ordre :

$$1, 2, \dots, n_c, 1, 2, \dots, n_c, \dots, 1, 2, \dots, n_c ;$$

ou encore dans l'ordre :

$$\hat{i}_1, \hat{i}_2, \dots, \hat{i}_{nc}, \hat{i}_1, \hat{i}_2, \dots, \hat{i}_{nc}, \dots, \hat{i}_1, \hat{i}_2, \dots, \hat{i}_{nc}$$

si on cherche à choisir un ordre sur les îlots comme dans le cas de la décomposition spatiale.

Comme dans le cas de la décomposition temporelle, nous avons retenu un découpage du temps en utilisant le sous-produit non encore placé qui se terminerait le plus tôt possible si on lui accordait en priorité les machines nécessaires, et l'on choisit un des îlots qui contient un tel sous-produit :

$\varphi_o := \min_i [r(i)]$; $\pi_o = \emptyset$;

$SPA_o :=$ { $(\hat{u},1)$ } = ensemble des sous-produits accessibles ;

$k := 0$;

tantque $SPA_k \neq \emptyset$ faire

$k := k + 1$;

$\varphi_k := \min_{(\hat{u},j) \in SPA_{k-1}} [r^{oo}(\hat{u},j) + pp^{oo}(\hat{u},j)]$

$(\hat{u}^*,j^*)$:= l'un des $(\hat{u},j)$: $\varphi_k = r^{oo}(\hat{u}^*,j^*) + pp^{oo}(\hat{u}^*,j^*)$

$\hat{i}_k := \hat{i}asp(\hat{u}^*,j^*)$

$\pi_k := \{ (\hat{u},j) / (\hat{u},j) \in SPA_{k-1}$ et $\hat{i}asp(\hat{u},j) = \hat{i}_k$ et $r^{oo}(\hat{u},j) < \varphi_k \}$

Ordonnancement local des opérations de π_k dans l'îlot $\hat{i}_k$.

Ajustement des délais internes et modification des poids.

$SPA_{k+1} = SPA_k - \pi_k + \{ (\hat{u},j+1) / (\hat{u},j) \in \pi_k$ et $j < sp(\hat{u}) \}$

fintantque

2.2.3.2. L'évolution des délais internes et la modification des poids

La participation de tout sous-produit au critère "retards pondérés" est constituée de deux parties :

- le retard vrai par rapport au délai externe du sous-produit (poids 100 par exemple) ;

- le retard virtuel par rapport au délai interne du sous-produit (poids 1 par exemple) ou
  le retard interdit par rapport au délai interne ajusté au début planifié du sous-produit suivant (poids 10^6 par exemple).

Appelons $WE(\hat{u},j)$ le poids par rapport au délai externe et $WI(\hat{u},j)$ le poids par rapport au délai interne pour tout sous-produit $(\hat{u},j)$.

A l'initialisation :

On calcule les $ra(\hat{u},j)$, les $da(\hat{u},j)$ et les $WE(\hat{u},j)$, en utilisant les définitions pour les premiers.
On initialise les $WI(\hat{u},j)$ à 1 et les $r^{oo}(\hat{u},j)$ et $d^{oo}(\hat{u},j)$ en utilisant la règle de partage de la marge totale des $\hat{u}$.
On initialise le critère somme des retards vrais SRV à 0 .

A la fin de l'ordonnancement local ajoutant les tâches de π_k dans l'îlot $\hat{i}_k$:

$\forall(\hat{u},j) \in \pi_k$: soit $f^{oo}(\hat{u},j) = t^{oo}(\hat{u},j,q^{oo}(\hat{u},j)) + p^{oo}(\hat{u},j,q^{oo}(\hat{u},j))$

si $j > 1$ alors
    ajustement des délais en amont et interdiction de chevauchement pour le sous-produit amont :
    $WI(\hat{u},j-1) := 10^6$; $d^{oo}(\hat{u},j-1) := t^{oo}(\hat{u},j,1)$
fsi

si $j < sp(\hat{u})$ alors
    ajustement des délais en aval :
    $r^{oo}(\hat{u},j+1) := f^{oo}(\hat{u},j)$
fsi

si $f^{oo}(\hat{u},j) > da(\hat{u},j)$ alors
    on comptabilise ce retard vrai :
    SRV := SRV + $(f^{oo}(\hat{u},j) - da(\hat{u},j))$ x $WE(\hat{u},j)$
    on décale les délais externes de manière à ne plus reconsidérer ce retard vrai (il ne faudrait pas le compter pour tous les sous-produits suivants) :
    décal := $f^{oo}(\hat{u},j) - da(\hat{u},j)$
    pour jj de j à $sp(\hat{u})-1$ faire
    $da(\hat{u},jj) := da(\hat{u},jj) + décal$
    $ra(\hat{u},jj+1) := ra(\hat{u},jj+1) + décal$
    finpour
fsi

2.4.4. Remarques

La méthode de décomposition spatiale et temporelle ne crée jamais de chevauchements puisqu'on les interdit par l'utilisation de poids infinis.

Il existe toujours une solution pour le problème d'ordonnancement local puisqu'il y a toujours leur planification précédente pour les sous-produits déjà planifiés qui ont des délais impératifs et que l'on peut reculer aussi loin que nécessaire les sous-produits que l'on ajoute.

L'ordonnancement partiel en cours de l'ensemble des îlots est donc toujours réalisable.

Les délais internes liés à des poids de 1 ont une importance très réduite dans l'optimisation locale. Ils servent essentiellement à faire un choix entre des solutions équivalentes pour les retards vrais en utilisant la connaissance de la marge totale des produits.

La méthode converge puisqu'à chaque itération on introduit au moins une opération de plus dans l'ordonnancement partiel.

Dans le cas général, la qualité de la méthode ne peut être étudiée que par expérimentation. Ceci sera vu au chapitre 4

2.5. CONCLUSIONS

Les méthodes d'ordonnancement par décomposition, proposées dans ce chapitre, permettent de construire de multiples variantes d'algorithmes approchés d'ordonnancement.

Nous les avons développées avec le critère "minimisation de la somme des retards". Le choix de ce critère est dû à son intérêt pratique (un carnet de commandes doit être effectué dans un atelier en respectant au mieux les délais), et à son intérêt théorique (c'est un des critères les plus difficiles à minimiser, difficile au sens de la complexité des problèmes).

Toutefois, il est possible d'appliquer ces méthodes avec d'autres critères, dès l'instant où l'on dispose d'un algorithme pour résoudre les sous-problèmes avec le critère retenu.

Les méthodes par décomposition temporelle, et spatiale et temporelle, peuvent ne pas utiliser de délais externes si le critère choisi n'en utilise pas, mais des délais internes devront être construits dans le dernier cas pour interdire les chevauchements.

BIBLIOGRAPHIE DU CHAPITRE 2

K.R. BAKER
Introduction to sequencing and scheduling
John Wiley & Sons, **1974**

M.V. BHAT / A. HAUPT
"An efficient clustering algorithm"
IEEE Trans. Syst. Man. and Cyber., Vol. SMC-6, pp 61-64, **1976**

J.L. BURBIDGE
"Production flow analysis"
Prod. Engr., Vol. 42, pp 742, **1963**

J. DEKLEVA / D. MENART
"Advantages and disadvantages of different production cell identification methods"
Advance in Production Management Systems 85, IFIP, pp 73-82, **1986**

N. DRIDI / M.C. PORTMANN / J.M. PROTH
"Une méthode de décomposition pour les problèmes d'ordonnancement en production"
Lecture Notes in Control and Information Sciences, Proceedings of the Seventh International Conference on Analysis and Optimization of Systems, pp 247-254, Antibes, juin **1986**

N. DRIDI / J.M. PROTH
"Ordonnancement des tâches : une méthode basée sur la technologie de groupe"
2nd International Conference on Production Systems INRIA, tome 1, pp 61-74, avril **1987**

J. CARLIER
"Ordonnancements à contraintes disjonctives"
RAIRO, Vol. 12, pp 333-351, **1978**

E.W. FORGEY
"Cluster analysis of multivariate data"
AAAS Biometric Society (WNAR), Reverside, California, USA, **1965**

H. GARCIA / B. MUTEL / J.M. PROTH
"Familles de produits et îlots de fabrication : le cas de machines multiples"
INRIA, Rapport de Recherche n° 469, décembre **1985**
et
Lecture Notes in Control and Information Sciences, Proceedings of the Seventh International Conference on Analysis and Optimization of Systems, pp 255-270, Antibes, juin **1986**

H. GARCIA / J.M. PROTH
"Group technology in production management : the short horizon planning level"
Journal Applied Stochastic Model and Data Analysis, Vol. 1, pp 25-34, **1985**

H. GARCIA / J.M. PROTH
"A new cross-decomposition algorithm : the GPM. Comparison with the bond energy method"
Control and Cybernetics, Vol. 15, n°2, pp 155-165, **1986**

G. GOVAERT
"Classification croisée"
Thèse de Doctorat d'Etat, Université de Paris VI, 20 juin **1983**

J.R. KING
"Machine-component grouping in production flow analysis : approach using a rank order clustering algorithm"
OMEGA (The Int. Journal of Management Science), Vol. 8, n° 2, pp 193-199, **1980**

K.R. KUMAR / A. KUSIAK / A. VANNELLI
"Grouping of parts and components in flexible manufacturing systems"
EJOR, Vol. 24, pp 387-397, **1986**

A. KUSIAK / A. VANELLI / K.R. KUMAR
"Grouping problem in scheduling flexible manufacturing systems"
Robotica, Vol. 3, pp 245-252, **1985**

A. KUSIAK
"The part families problem in flexible manufacturing systems"
Annals of Operation Research, Vol. 3, pp 279-300, **1985**

A. KUSIAK
"Generalized group technology concept"
Working paper n° 03/86, Université de Manitoba, janvier **1986**

A. KUSIAK
"Knowledge engineering and optimization in automated manufacturing systems"
2nd International Conference on Production Systems INRIA, tome 1 pp 15-28, Paris, avril **1987**

F. MARCOTORCHINO
"Block seriation problems and relational analysis methods"
non publié, **1986**

F. MARCOTORCHINO
"Une approche unifiée des problèmes de décomposition en production"
2nd International Conference on Production Systems INRIA, cours tutoriaux, pp 49-76, Paris, avril **1987**

J. Mc AULEY
"Machine grouping for efficient production"
The Production Engineer, pp 53-57, février **1972**

W.T. Mc CORMICK / P.J. SCHWZEITZER/ T. W. WHITE
"Problem decomposition and data reorganization by a clustering technique"
O.R., Vol. 20, pp 993-1009, **1972**

M.C. PORTMANN
"Méthodes de décomposition spatiales et temporelles en ordonnancement de la production"
2nd International Conference on Production Systems INRIA, tome 1, pp 103-113, Paris, avril **1987**

J.M..PROTH
"Regroupement en familles de produits en fonction des outils disponibles"
Rapport de Recherche INRIA n ° 370, mars **1985**

A.H.G. RINNOOY KAN
"Machine scheduling problems : classification, complexity and computation"
Nijhoof, The Hague, **1976**

J.R. SLAGLE / C.L. CHANG/ S.R. HELLER
"A clustering and data reorganization algorithm"
IEEE Trans. Syst. Man. and Cyber. , Vol. SMC-5, pp 125-128, **1975**

M. YAMAMOTO
"An approximate solution of machine scheduling problems by decomposition method"
Int. J. Prod. Res. Vol. 15, n° 6, pp 599-608, **1977**

CHAPITRE 3

EVALUATION DES METHODES DE DECOMPOSITION TEMPORELLE POUR LES PROBLEMES A UNE MACHINE

3. EVALUATION DES METHODES DE DECOMPOSITION TEMPORELLE POUR LES PROBLEMES A UNE MACHINE

3.1. INTRODUCTION GENERALE

3.2. CAS PARTICULIER : DUREES OPERATOIRES UNITAIRES

3.3. CAS PARTICULIER : DUREES OPERATOIRES EGALES ET NON UNITAIRES

3.4. AUTRES CAS PARTICULIERS

3.5. CAS GENERAL ET CONCLUSIONS

3. EVALUATION DES METHODES DE DECOMPOSITION TEMPORELLE POUR LES PROBLEMES A UNE MACHINE

3.1. INTRODUCTION GENERALE

Les méthodes de décomposition présentées au chapitre 2 sont des méthodes approchées. Pour les évaluer il faut, comme nous l'avons présenté au paragraphe 1.3.2. du chapitre 1, s'intéresser à la complexité de l'algorithme et à la valeur du ou des critères. Il est très difficile d'obtenir des résultats théoriques dans le cas général, c'est pourquoi nous avons considéré, pour ces évaluations, le cas du problème d'ordonnancement à une machine et plus particulièrement, pour certains résultats, le cas où les durées opératoires sont toutes identiques.

Dans une première partie, nous allons rappeler le problème et les principaux résultats connus, puis nous intéresser aux variantes possibles des méthodes de décomposition temporelle et à leurs propriétés dans le cas des ateliers à une machine.

Dans la suite du chapitre, nous étudierons les cas plus particuliers

3.1.1. Rappel du problème et des résultats antérieurs

A. Problème posé et notations

Pour le cas où l'atelier se réduit à une seule machine, les notations deviennent plus simples et les méthodes peuvent être améliorées en tenant compte des hypothèses particulières.

Lorsqu'il n'y a qu'une machine, la gamme d'un produit se réduit à une opération et il y a correspondance biunivoque entre produits et opérations. Plutôt que de retenir systématiquement l'un de ces deux termes ou de les utiliser indifféremment, dans la suite de ce chapitre, nous avons choisi d'utiliser le mot tâche qui désigne l'unique opération à réaliser sur le produit.

Les caractéristiques du problème d'ordonnancement à une machine considéré sont les suivantes :

- la machine est toujours en état de fonctionner et ne peut exécuter qu'une tâche à la fois ;
- une tâche commencée ne peut être interrompue ;
- les réglages sont effectués alors que le produit est déjà sur la machine ; leur durée ne dépend pas de l'ordre de passage des tâches et est incluse dans la durée de la tâche ;
- les durées de transport sont négligeables ;
- les tâches sont indépendantes : il n'y a pas de contraintes de précédence entre les tâches ;
- la tâche i est exécutable à partir de l'instant r_i appelé date d'arrivée ou de disponibilité de la tâche i ;
- la tâche i a une durée p_i ;
- la tâche i a un délai d_i (date de fin au plus tard souhaitée) ; si le délai d_i n'est pas respecté, cela engendre un coût de retard ρ_i lié à la tâche i :

$$\rho_i = c_i - d_i$$

où c_i est la date d'achèvement de la tâche i alors que t_i est la date de début d'exécution de la tâche i :

$$t_i = c_i - p_i$$

Ces notations sont conformes à celles utilisées dans le chapitre 2 et regroupées dans le glossaire de l'annexe F. Simplement, l'indice correspondant aux opérations a disparu.

Nous supposons que le codage retenu pour toutes les informations est choisi de telle sorte que r_i, p_i et d_i soient des données entières, t_i et c_i seront alors entiers ; cette hypothèse n'est pas restrictive pour les problèmes réels, il suffit de choisir convenablement l'unité de temps (dixième ou centième d'heure, de minute ou de seconde).

B. Résultats antérieurs

Les revues scientifiques sont très riches en articles sur les problèmes d'ordonnancement. Pour s'en convaincre, il suffit de consulter la revue "International Abstract of Operation Research" ; le premier chapitre de cette thèse illustre également cette richesse.

Près de la moitié des articles publiés s'intéressent aux problèmes d'ordonnancement à une machine.

Mais lorsque l'on a comme objectif de minimiser la somme des retards par rapport aux délais ou lorsque l'on suppose que toutes les tâches ne sont pas disponibles à l'instant "0" (les r_i ne sont pas tous égaux), alors la littérature propose beaucoup moins de résultats.
Une première série d'articles propose des méthodes exactes de résolution. Comme nous l'avons vu au paragraphe 1.4., il s'agit essentiellement de procédures par séparation et évaluation ("branch and bound") ou de programmation dynamique.

Lorsque l'objectif est de minimiser la somme des retards (ou un critère plus compliqué dont la somme des retards est un cas particulier), mais que toutes les tâches sont disponibles dès l'instant "0", on peut citer en particulier : H. EMMONS ([1969], V. SRINIVASAN ([1971]), I. NABESHIMA ([1971]), A.H.G. RINNOOY KAN / B.J. LAGEWEG et J.K. LENSTRA ([1974]), L. GELDERS et P.R. KLEINDORFER ([1973]), M.L. FISHER ([1976]), L. VAN WASSENHOVE et L. GELDERS ([1978]), J. SHWIMER ([1972]), E.L. LAWLER ([1977]), L. SCHRAGE et K.R. BAKER ([1978]), C.N. POTTS et L.N. WASSENHOVE ([1982]) et T.T. SEN / L.M. AUSTIN / P. GHANDFOROUSH ([1983]).

Lorsque les tâches sont disponibles à des instants différents, mais que l'objectif n'est pas de minimiser la somme totale des retards, on peut citer en particulier : J.R. JACKSON ([1955]), P. BRATLEY / M. FLORIAN et P. ROBILLARD ([1973]), K.R. BAKER et Z.S. SU ([1974]), G.B. Mc MAHON et M. FLORIAN ([1975]), B.J. LAGEWEG / J.K. LENSTRA et A.H.G. RINNOY KAN ([1975]), M.I. DESSOUKY et J.S. DEOGUN ([1981]), J. CARLIER ([1982]), A.M.A. HARIRI et C.N. POTTS ([1983]), M.E. POSNER ([1986]) et J. GRABOWSKI / E. NOWICKI / S. SDRZALKA ([1986]). Dans ce cas, une étude stochastique sur deux critères a été réalisée par M. PINEDO ([1983]).

Dans le cas général du job shop, une procédure par séparation et évaluation a été proposée par A.H.G. RINNOOY KAN ([1976]).

Lorsque les tâches sont disponibles à des instants différents, quelques auteurs proposent, analysent et comparent des méthodes approchées. On peut citer en particulier : H. KISE / T. IBARAKI / H. MINE ([1978]), C.N. POTTS ([1978]) et N.G. HALL et W.S.R. RHEE ([1986]).

Une deuxième série d'articles s'intéresse à la complexité des problèmes. Les principaux résultats sont regroupés dans le répertoire du livre de M.R. GAREY et D.S. JOHNSON ([1979]. Le lecteur, non habitué à la théorie de la complexité des problèmes, trouvera les définitions dans l'annexe A.

Dans le cas général, le problème est NP-difficile au sens fort si l'on choisit comme critère la somme pondérée des retards. Il le reste si l'on choisit des poids égaux à un, c'est-à-dire si l'on prend comme critère la somme ou la moyenne des retards. Par contre, dans le cas particulier où toutes les tâches ont des durées unitaires, le problème est polynômial ([R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979]).

Il faut donc s'attendre à trouver des algorithmes non polynômiaux pour résoudre le problème dans le cas général. Tout algorithme polynômial sera une méthode approchée.

3.1.2. Présentation de plusieurs variantes de méthodes de décomposition temporelle

Nous avons signalé au chapitre 2 que les méthodes de décomposition temporelle pouvaient être construites pour n'importe quel critère et que, dans certains cas particuliers, nous n'utiliserons pas un critère, mais une hiérarchie de critères lors des optimisations locales. C'est ce que nous allons détailler dans ce paragraphe en mettant en évidence l'intérêt d'une telle hiérarchie de critères.

Notre critère principal sera toujours la somme des retards que nous minimisons localement.

Lorsque plusieurs solutions sont équivalentes au sens du premier critère, nous utilisons un ou plusieurs critères secondaires, dans un ordre choisi, pour retenir la solution localement optimale.

La durée totale de l'ordonnancement partiel obtenu (ou "makespan") peut être un tel critère secondaire. Il a l'avantage de minimiser la somme des durées improductives où la machine est inutilisée entre deux tâches et, ainsi, de libérer plus tôt la machine pour les futurs produits. Il a l'inconvénient d'ignorer l'urgence relative des tâches et peut pénaliser des tâches urgentes.

Un autre critère secondaire utilisé est nouveau. Nous l'appellerons U (pour "urgence"). Il compte le nombre de fois où tout couple de tâches (i,j) vérifie la propriété u définie ci-dessous. Nous cherchons à le maximiser.

Définition

On dit qu'un couple de tâche (i,j) appartenant à la séquence σ (qui définit un ordonnancement sur une machine) a la propriété u si :

i est avant j dans la séquence σ ===> $(d_i \leq d_j)$ ou $(c_i - p_i < r_j)$

et on définit $U(\sigma)$ par :

$U(\sigma) = \text{card}(\{ (i,j) / i \in \sigma , j \in \sigma , (i,j) \text{ vérifie } u \})$

Vérifier la propriété u consiste à placer en priorité la tâche la plus urgente sauf si elle n'est pas encore accessible au moment où l'on peut commencer une autre tâche moins urgente, auquel cas on peut placer cette dernière pour éviter de laisser la machine inutilisée.

Il est possible d'inverser i et j dans une séquence σ et de vérifier la propriété u pour (i,j) ou (j,i) dans les deux séquences comme le montre la figure 3.1.2.

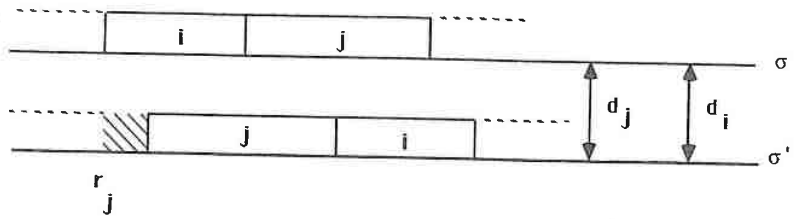


figure 3.1.2. : propriété u vérifiée dans $\sigma(i,j)$ et $\sigma'(j,i)$

Il est évident que :

- U est maximal pour la séquence EDD ("Earliest Due Date"), c'est-à-dire la séquence $\sigma = (i_1,i_2,\ldots,i_n)$ avec $d_{i1} \leq d_{i2} \leq \ldots \leq d_{in}$;

- U est maximal pour la séquence J construite à partir de la règle de Jackson ([J.R. JACKSON 1955]) : à partir de l'instant initial, chaque fois que la machine est disponible, placer la tâche la plus urgente parmi les tâches exécutables ;

- $U(EDD) = U(J) = \frac{n(n-1)}{2}$ puisque tout couple (i,j) avec i placé avant j vérifie la propriété u.

L'intérêt de U comme critère secondaire, utilisé pour choisir parmi les séquences qui minimisent localement la somme des retards, est de trier dans l'ordre EDD les tâches non encore en retard (mais qui risquent de le devenir aux itérations suivantes), avec une possibilité de choix utilisée lorsque le fait de respecter EDD peut créer des temps improductifs sur la machine.

Pour désigner les différentes variantes des méthodes de décomposition temporelle, nous utilisons les notations suivantes :

DTR désigne la méthode de décomposition temporelle qui localement minimise la somme des retards et retient n'importe quelle solution en cas d'optima locaux multiples ;

DTRM désigne la méthode de décomposition temporelle qui localement minimise la somme des retards et à égalité sur ce premier critère retient l'une des solutions qui minimise le "makespan".

DTRU désigne la méthode de décomposition temporelle qui localement minimise la somme des retards et à égalité sur ce premier critère retient l'une des solutions qui maximise le nombre de fois où localement on vérifie la propriété u ;

DTRMU et DTRUM désignent de manière analogue des méthodes de décomposition temporelle où les hiérarchies de critères utilisées pour les optimisations locales sont respectivement : somme des retards, "makespan" et "urgence" et somme des retards, "urgence" et "makespan".

Nous disposons ainsi d'une famille de méthodes de décomposition temporelle que nous pourrions enrichir avec d'autres variantes.

3.1.3. Propriété des méthodes de décomposition temporelle

Au paragraphe 1.4.3.3. A), dans le cadre de la présentation des procédures par séparation et évaluation, nous avons été amenés à définir rapidement les notions d'ordonnancements actifs et semi-actifs ([K.R. BAKER 1974). Nous allons rappeller ces définitions, ainsi que celle des ordonnancements sans délai qui sera utile pour des comparaisons ultérieures. Nous introduisons également une nouvelle notion, celle d'"ordonnancement u-actif".

Nous montrerons ensuite que si les optimisations locales construisent des solutions partielles vérifiant une propriété donnée, (actif, semi-actif,...), alors la solution globale vérifie cette propriété.

A) Définitions

Un ordonnancement est semi-actif si toute tâche i commence au plus tôt compte tenu de sa date d'arrivée r_i et de la date d'achèvement c_j de la tâche j qui la précède immédiatement : $t_i = \max [r_i, c_j]$; on dit encore que l'ordonnancement est calé à gauche.

Un ordonnancement est actif si, pour avancer une tâche quelconque, on est obligé de reporter le début d'une autre ; c'est-à-dire s'il n'existe pas d'intervalle où la machine est disponible et où l'on pourrait placer une tâche déjà exécutable (planifiée plus tard) sans modifier le reste de la solution.

Un ordonnancement est sans délai si la machine n'est jamais laissée inoccupée alors qu'il y a des tâches exécutables.

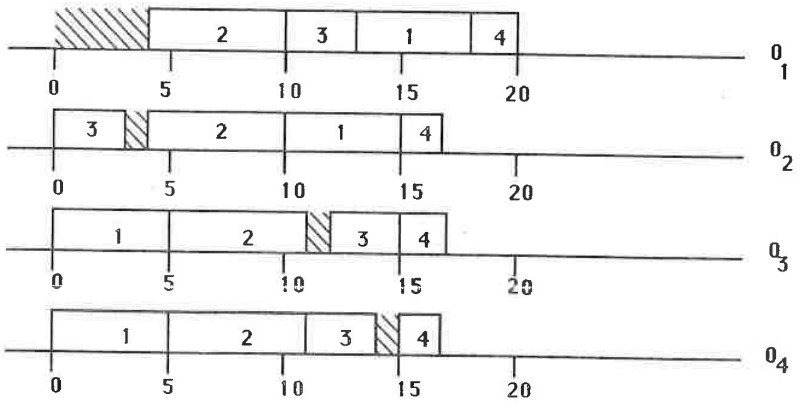
Un ordonnancement est u-actif s'il est actif et si tout couple de tâches (i,j) vérifie la propriété u.

Illustrons ces différentes notions à l'aide de la figure 3.1.3.
La figure 3.1.3. a fournit l'énoncé du problème.
La figure 3.1.3. b présente le diagramme de Gantt de plusieurs ordonnancements.
La figure 3.1.3. c résume les propriétés de ces ordonnancements. On peut remarquer ici que l'ordonnancement O_1 est obtenu à partir de la séquence EDD. Bien que cet ordonnancement maximise le critère U, il n'est pas u-actif.

i	r_i	p_i	d_i
1	0	5	12
2	4	6	8
3	0	3	10
4	15	2	17

énoncé d'un problème à une machine

figure 3.1.3. a



ordonnancements

figure 3.1.3. b

	semi-actif	actif	u-actif	sans délai
0_1	OUI	NON	NON	NON
0_2	OUI	OUI	OUI	NON
0_3	NON	NON	NON	NON
0_4	OUI	OUI	NON	OUI

propriétés des ordonnancements

figure 3.1.3. c

Il est connu ([K.R. BAKER 1974]) que :

- un ordonnancement sans délai est toujours actif,

- un ordonnancement actif est toujours semi-actif,

- si le critère est régulier, il existe toujours une solution optimale qui est un ordonnancement actif (et donc semi-actif), par contre, il se peut que parmi les solutions optimales ne figure aucun ordonnancement sans délai.
(Rappelons (cf paragraphe 2.1.1. B)) qu'un critère est dit régulier si, lorsqu'on avance une tâche sans retarder les autres, la valeur du critère ne peut que s'améliorer ou rester invariante).

B) La méthode

Rappelons tout d'abord la méthode de décomposition temporelle dans le cas particulier d'une seule machine. Nous choisissons pour la suite temporelle φ un découpage par rapport à la tâche non encore placée qui peut se terminer au plus tôt (cf. 2.2.3.1.).

L'algorithme général est le suivant :

W := {ensemble des indices des tâches}
$\varphi_o := \min_{i \in W} [r_i]$ et $\varphi_1 := \min_{i \in W} [r_i + p_i]$

$k := 0$ et $O := \emptyset$ et $P_0 := \emptyset$

tantque $P_k \neq W$ faire

$k := k + 1$

$\pi_k := \{i / i \in W$ et $\varphi_{k-1} \leq r_i < \varphi_k\}$

$P_k := P_{k-1} \cup \pi_k$

$O := O \oplus \pi_k$

$\varphi_{k+1} := \min_{i \in W} [r_i + p_i]$

fintantque

où $O := O \oplus \pi_k$ se traduit par :

- ne pas modifier les tâches de O qui sont terminées au plus tard à l'instant φ_{k-1} ;

- considérer les autres tâches de O et les tâches de π_k et les ordonnancer à partir de l'instant q_k ($q_k \leq \varphi_{k-1}$) où la machine est disponible.
Cet ordonnancement local, qui ne remet pas en cause le début de l'ordonnancement, est optimal pour la hiérarchie choisie (cf. § 3.1.2.).

C) Propriétés de la méthode de décomposition temporelle

La méthode de décomposition temporelle "conserve" les notions définies au paragraphe A) comme l'exprime le théorème suivant que nous allons démontrer.

Théorème 3.1.3.

Si les optimisations locales fournissent des ordonnancements semi-actifs (ou actifs, ou u-actifs, ou sans délai), alors la solution globale obtenue est un ordonnancement semi-actif (ou actif, ou u-actif, ou sans délai).

Démonstration (*)

Elle se fait par récurrence sur le nombre d'itérations.

a) *Le théorème est évident s'il n'y a qu'une itération, puisque la solution locale a la propriété souhaitée et est également solution globale.*

b) *Supposons que la propriété souhaitée soit vraie jusqu'à l'itération* K *et plaçons nous à l'itération* K+1.

Soit M_K *la date d'achèvement de la dernière tâche à la fin de l'itération* K *(ou "makespan" à la fin de* K *).*

Deux cas sont à envisager :

α) $M_K \leq \min_{i \in \pi_{k+1}} [r_i]$

Dans ce cas, l'ordonnancement local de l'itération K+1 *est totalement indépendant de l'ordonnancement partiel précédemment obtenu ; chacun d'entre eux étant semi-actif (ou actif, ou sans délai), leur réunion est semi-active (ou active ou sans délai.*

Pour le cas u-actif, la propriété u *est vérifiée pour tout couple interne aux deux ordonnancements indépendants et elle est vérifiée entre toute tâche* i *du premier ordonnancement et toute tâche* j *du deuxième ordonnancement car l'on a* $c_i - p_i < M_K \leq r_j$.

(*) Dans cette thèse, toutes les démonstrations sont en italiques pour permettre au lecteur de les éviter lors d'une lecture rapide.

β) $M_K > \min_{i \in \pi_{k+1}} [r_i]$

Dans ce cas, jusqu'à q_{K+1} *, date de fin de la dernière tâche figée, l'ordonnancement partiel correspondant est semi-actif (ou actif, ou sans délai), selon l'hypothèse de récurrence restreinte au début de l'ordonnancement. Puis l'optimisation locale à partir de* q_{K+1} *est à nouveau totalement indépendante de cet ordonnancement partiel. On en déduit comme précédemment que la réunion des deux ordonnancements partiels est semi-active (ou active, sans délai).*
Pour le cas u-actif, la propriété u *est vérifiée pour tout couple interne aux deux ordonnancements indépendants, mais il faut examiner deux cas possibles lorsque la tâche* i *est dans le premier ordonnancement et* j *dans le second :*

i) $j \in \pi_{K+1}$ *alors* u *est vérifiée car on a :*

$$c_i - p_i < q_{K+1} \leq \varphi_K \leq r_j$$

ii) $j \notin \pi_{K+1}$ *alors* u *est vérifiée car à l'itération précédente on avait* i *avant* j *dans un ordonnancement partiel u-actif et donc :*
$d_i \leq d_j$ ou $c_i - p_i < r_j$,
condition qui reste vraie puisque la tâche i *n'a pas été déplacée.*

A l'exclusion des ordonnancements u-actifs qui nécessiteraient une définition plus précise liée aux opérations dans le cas de plusieurs machines, il est à noter que le théorème 3.1.3. reste valable pour les méthodes de décomposition spatiale et temporelle appliquées dans un atelier à plusieurs machines ; la même démonstration s'applique en remplaçant q_{K+1} et M_K par des vecteurs ayant la même signification pour chaque machine.

Le théorème 3.1.3. s'applique donc à un domaine plus large que celui qui est défini par les hypothèses de ce chapitre.

3.1.4. Complexité des méthodes de décomposition temporelle

Nous avons défini au paragraphe 1.3.2. les notions d'algorithmes polynômiaux et exponentiels.

Un algorithme est polynômial si l'on peut majorer le nombre d'opérations qu'il exécute par la valeur d'un polynôme dont la variable est la taille du problème.

Ici, tout énoncé contient 3n informations (pour chaque tâche : date d'arrivée, durée, délai). Il faut donc chercher un polynôme en n.

On peut remarquer qu'à chaque itération d'un algorithme de décomposition temporelle on introduit au moins une nouvelle tâche, et que l'algorithme s'arrête lorsqu'il n'y a plus de nouvelles tâches à introduire. Le nombre d'itérations d'un algorithme de décomposition temporelle est donc toujours majoré par n , le nombre de tâches.

Aussi, s'il est possible d'utiliser un algorithme polynômial pour les optimisations locales, la méthode de décomposition temporelle utilise au plus n fois des algorithmes polynômiaux et est elle-même, dans ce cas, un algorithme polynômial. Nous rencontrerons cette possibilité dans des cas particuliers ; ce sera également le cas si l'on remplace l'optimisation locale par un algorithme approché polynômial.

Au contraire, si le nombre d'opérations effectuées par les algorithmes locaux ne peut pas être borné par un polynôme, il en est de même pour la méthode de décomposition temporelle (dont le nombre d'opérations sera minoré par une fonction exponentielle du nombre n de tâches). C'est le cas général puisque le problème considéré est NP-difficile.

Pour les optimisations locales, dans le cas général du "job-shop", nous avons construit deux procédures par séparation et évaluation (voir paragraphe 1.4.3.). L'une a été mise au point par Najoua DRIDI, à partir de l'algorithme d'énumération de tous les ordonnancements actifs proposé par A.H.G. RINNOOY KAN ([1976]) et l'autre se base sur la notion de contraintes disjonctives (voir l'annexe B ainsi que : [B. ROY / B. SUSSMANN 1964], [E. BALAS 1968], [J. CARLIER 1978]).

Ce sont ces algorithmes non polynômiaux que nous utilisons, dans les cas les plus généraux, lors de nos optimisations locales au cours de la méthode de décomposition temporelle.

Comme la réunion de solutions localement optimales ne conduit pas en général à une solution globalement optimale, nous nous intéresserons à l'erreur commise sur le critère lorsque l'on applique les méthodes de décomposition temporelle.

3.1.5. Présentation du chapitre

Dans le cas particulier, où toutes les durées des tâches sont unitaires, le problème est polynômial ([R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979]. Il peut donc être résolu par un algorithme polynômial. Nous montrerons, dans ce cas, qu'une méthode de décomposition temporelle se simplifie en un algorithme polynômial connu qui fournit une solution optimale. C'est l'objectif du paragraphe 3.2.

Un autre cas particulier est celui, où toutes les durées des tâches sont égales à une constante D , mais ne sont pas unitaires ; c'est-à-dire que D ne divise pas le plus grand commun diviseur des dates d'arrivées r_i , ou encore qu'un changement d'unité de temps ne peut pas rendre les tâches unitaires tout en conservant un codage entier pour les r_i. Dans le livre de M.R. GAREY et D.S. JOHNSON ([1979]), ce cas est rattaché au précédent, mais la référence citée reste la même que dans le paragraphe précédent ([R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINNOOY KAN 1979] et ne concerne que les durées unitaires! A notre connaissance, le problème reste donc ouvert. On ne sait donc pas s'il peut être résolu par un algorithme polynômial. Le paragraphe 3.3. a pour objectif de montrer que, dans ce cas, une méthode de décomposition temporelle conduit à un algorithme polynômial qui ne fournit pas toujours une solution optimale. Une étude dans le pire des cas fournit l'ordre de grandeur de l'erreur commise sur la somme des retards en fonction du nombre de tâche n et de la durée D .

Le paragraphe 3.4. correspond à d'autres cas particuliers dont certains ont déjà été étudiés. Le paragraphe 3.5. fait le point des conclusions que l'on peut tirer dans le cas général à partir des cas particuliers précédents.

3.2. CAS PARTICULIER : DUREES OPERATOIRES UNITAIRES

3.2.1. Durées unitaires ou égales et arrivées simultanées

Nous avons précisé au paragraphe 3.1.5. que si D divise le P.G.C.D. des dates d'arrivées r_i, alors l'hypothèse de durées égales est équivalente à celle de durées unitaires. C'est en particulier le cas lorsque toutes les dates d'arrivées des tâches sont identiques ($r_i = 0 \quad \forall i$).

Dans le cas où les tâches sont disponibles dès l'instant 0, il est connu ([H. EMMONS 1969]) que si les séquences SPT ("shortest processing time") et EDD ("earliest due date") sont identiques, alors elles minimisent la somme des retards.

Lorsque les durées sont égales, les $n!$ séquences sont des séquences SPT et parmi celles-ci figurent nécessairement une séquence EDD qui, coïncidant avec SPT, est optimale.

Dans le cas où les durées sont égales et où les tâches sont exécutables dès l'instant 0, on a :

- tout algorithme de tri qui classe les tâches dans l'ordre EDD est un algorithme polynômial exact (de complexité $n \log n$) ;

- la méthode de décomposition temporelle ne comporte qu'une seule itération ($\varphi_0 = 0$, $\varphi_1 = D$ et $\forall i : \varphi_0 = 0 = r_i < D = \varphi_1$). Si celle-ci est réalisée par l'algorithme de tri précédent, alors la méthode de décomposition temporelle est polynômiale et optimale.

3.2.2. Simplification des méthodes

Nous considérons la méthode de décompositon temporelle DTRU définie au paragraphe 3.1.2.

L'algorithme général est le suivant :

$W :=$ (ensemble des indices des tâches)

$\varphi_0 := \min_{i \in W} [r_i]$ et $\varphi_1 := \min_{i \in W} [r_i + p_i]$

$k := 0$ et $O := \emptyset$ et $P_0 := \emptyset$

tantque $P_k \neq W$ faire

$k := k + 1$

$\pi_k := \{ i \ / \ i \in W \text{ et } \varphi_{k-1} \leq r_i < \varphi_k \}$

$P_k := P_{k-1} \cup \pi_k$

$O := O \oplus \pi_k$

$\varphi_{k+1} := \min_{i \in W} [r_i + p_i]$

fintantque

où $O := O \oplus \pi_k$ se traduit par :

- ne pas modifier les tâches de O qui sont terminées au plus tard à l'instant φ_{k-1} ;

- considérer les autres tâches de O et les tâches de π_k et les ordonnancer à partir de l'instant q_k ($q_k \leq \varphi_{k-1}$) où la machine est disponible.

- cet ordonnancement local minimise la somme des retards et, à égalité, le critère U (méthode DTRU).

Pour simplifier l'algorithme, étudions les "invariants de boucle" de cet algorithme itératif, c'est-à-dire la définition des objets calculés par récurrence et montrons que ces définitions sont plus simples dans le cas particulier.(*)

(*) cette partie technique est également mise en italique.

En notant R_k *la plus petite date d'arrivée d'un produit* π_k *on a :*

$$R_k = \min_{j \in W\text{-}P_k} [r_j] = \min_{j \in W\text{-}P_k} [r_{j+1}] - 1 = \varphi_{k-1} \quad \text{quand} \quad p_i = 1$$

On en déduit :

$$\pi_k = \{ i / i \in W \text{ et } \varphi_{k-1} \leq r_i < \varphi_k \}$$

$$= \{ i / i \in W \text{ et } R_{k-1} + 1 \leq r_i < r_k + 1 \}$$

$$= \{ i / i \in W \text{ et } R_{k-1} < r_i \leq R_k = \min_{j \in \pi_k} [r_j] \}$$

$$= \{ i / i \in W \text{ et } r_i = R_k \}$$

(ces identités sont valables pour $k = 1$ *si l'on pose* $R_0 = \varphi_0 - 1$ *).*

La partie principale de l'algorithme devient alors :

W := {indices des tâches}

k := 0 et O := ø et P_0 := ø

tantque $P_k \neq W$ faire

k := k + 1

$R_k := \min_{i \in W\text{-}P_k} [r_j]$

$\pi_k := \{ i / i \in W \text{ et } r_i = R_k \}$

$P_k := P_{k-1} \cup \pi_k$

$O := O \oplus \pi_k$

fintantque

A chaque itération, l'algorithme ajoute à O *toutes les tâches de plus petite date d'arrivée non encore incluses dans* O .

Examinons maintenant le "module" $O := O \oplus \pi_k$ *lorsque les durées sont unitaires et que la méthode de décomposition choisie est* DTRU :

- on ne modifie pas les tâches O *qui sont terminées au plus tard à l'instant* φ_{k-1} *(qui vaut ici* $R_{k-1} + 1$ *) ;*

- on considère les autres tâches de O *et on les ordonnance conjointement avec les nouvelles tâches de* π_k *qui arrivent à l'instant* R_k .

Reprenons les mêmes notations qu'au paragraphe 3.1.3. : soit M_{k-1} *la date d'achèvement de la dernière tâche en fin d'itération* k-1.

Deux cas sont à envisager :

α) $M_{k-1} \leq R_k$

Dans ce cas, l'ordonnancement local de l'itération k *est totalement indépendant de l'ordonnancement partiel précédemment obtenu.*
Il s'agit d'ordonnancer les tâches de π_k *qui sont toutes disponibles à l'instant* R_k *et ont des durées égales (unitaires), de manière à minimiser la somme des retards et à égalité sur les retards de maximiser le critère* U.
Or, on a déjà rappelé en 3.2.1. *que la séquence* EDD *minimise dans ce cas la somme des retards, et constaté en* 3.1.2. *que la séquence* EDD *maximise le critère* U.
En outre, toute séquence qui ne respecte pas l'ordre croissant des délais ne maximise pas U *car* $c_i - 1 < r_j = r_i = R_k$ *est impossible dans notre cas particulier et la propriété* u *ne peut être vérifiée que par la condition* $d_i \leq d_j$.
L'ordonnancement local de la méthode DTRU *retient donc obligatoirement une séquence* EDD, *placée à partir de* R_k.

β) $M_{k-1} > R_k$

Comme les durées sont unitaires, il n'y a pas de tâche qui commence avant R_k *et se termine après* R_k *, la dernière tâche figée se termine donc en* R_k ($q_k = R_k$).

Toutes les tâches réordonnancées ainsi que les nouvelles tâches de π_k *sont de durée unitaire et sont exécutables à partir de* R_k *, date de disponibilité de la machine après les tâches figées.*

On retrouve, pour l'ordonnancement local, les mêmes hypothèses qu'en α) et on retient donc la séquence EDD *pour les optimisations locales.*

En examinant le comportement de DTRU *dans ce cas particulier, on remarque que :*

- *seules les tâches placées à l'itération* k *entre les instants* R_k R_{k+1} *restent définitivement à la place qui leur est attribuée ;*

- *l'ordonnancement des autres est ignoré par la suite, et donc inutile ;*

- les tâches placées entre R_k et R_{k+1} sont les plus urgentes parmi *celles disponibles au plus tard en* R_k *et non encore placées ; à chaque instant* R_k+t *, "on place la plus urgente parmi les tâches disponibles".*

Comme cette dernière propriété est vraie quel que soit l'instant et quelle que soit l'itération, on peut simplifier une dernière fois l'algorithme DTRU *qui devient :*

$k := 0$ et $t := \min_i [r_i]$

tanque $k<n$ faire

si il existe une tâche j non placée telle que $r_j \leq t$

alors :

- choisir une de ces tâches j de délai minimal d_j
- la placer en t
- $k := k+1$; $t := t+1$

sinon $t := \min_{j \text{ tâche non placée}} [r_j]$

fsi

fintanque

3.2.3. Optimalité et complexité

A) Optimalité

L'algorithme que nous venons d'obtenir est connu sous l'appellation "règle de Jackson" ([J.R. JACKSON 1955]). On sait qu'il minimise le plus grand retard. Dans notre cas particulier, il minimise également la somme des retards.

Nous n'avons pas trouvé ce résultat dans la littérature publiée, c'est pourquoi nous en proposons une démonstration.

Théorème 3.2.3.

Dans le cas du problème d'ordonnancement à une machine, lorsque les durées sont unitaires, l' "algorithme de Jackson" minimise la somme des retards des tâches.

La démonstration se fait en deux parties.

Première partie :

On montre que , pour un ordonnancement considéré, l'échange de deux tâches qui ne vérifie pas la propriété u *ne dégrade pas le critère " somme des retards ".*

Soit σ *une séquence de tâches qui définit un ordonnancement semi-actif.*

Soient i *et* j *deux tâches quelconques placées dans cet ordre dans* σ .

Supposons que (i,j) ne vérifie pas la propriété u .

On a :

$$d_i > d_j \quad \text{et} \quad c_i - 1 \geq r_j$$

Il est donc possible d'inverser i *et* j *sans toucher aux autres tâches comme le montre la figure* 3.2.3 .

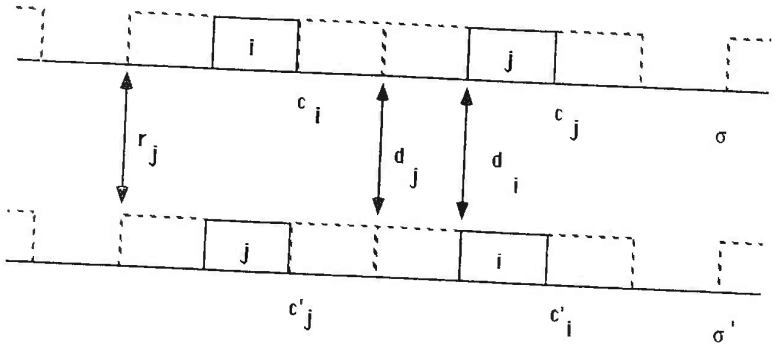


figure 3.2.3.

Soient T_σ *et* $T_{\sigma'}$ *la somme des retards correspondant aux séquences* σ *et* σ'. *En utilisant les notations introduites en* 3.1.1., *on a :*

$$T_\sigma - T_{\sigma'} = \rho_i + \rho_j - \rho'_i - \rho'_j$$

avec $\rho_i = \max\ (0,\ c_i - d_i)$

$\rho_j = \max\ (0,\ c_j - d_j)$

$\rho'_i = \max\ (0,\ c_j - d_i)$

$\rho'_j = \max\ (0,\ c_i - d_j)$

et $d_i > d_j$

et $c_j > c_i$

d'où $\rho_i \le \rho'_i$; $\rho_i \le \rho'_j$; $\rho_j \ge \rho'_i$; $\rho_j \ge \rho'_j$;

Examinons les différents cas possibles :

a) $d_i \ge c_j$

alors $\rho_i = \rho'_i = 0$ *et comme* $\rho_j \ge \rho'_j$ *on a* $T_\sigma \ge T_{\sigma'}$

b) $d_i < c_j$

α) $d_j \le c_i$

alors $\rho'_j = c_j - d_i$; $\rho_j = c_j - d_j$; $\rho'_j = c_i - d_j$

et $T_\sigma - T_{\sigma'} = \max\ (0,\ c_i - d_i) + (c_j - d_j) - (c_i - d_j) - (c_j - d_i)$

$= \max\ (0,\ c_i - d_i) - (c_i - d_i)$

et donc $T_\sigma \ge T_{\sigma'}$

β) $d_j > c_i$

alors $\rho'_j = 0 \ge \rho_i \ge 0$ *d'où* $\rho_i = 0$;

$T_\sigma - T_{\sigma'} = \rho_j - \rho_i$ *et comme* $\rho_j \ge \rho'_i$

on a $T_\sigma \ge T_{\sigma'}$

Dans tous les cas, on a trouvé $T_\sigma \ge T_{\sigma'}$

Donc l'échange de deux tâches quelconques ne vérifiant pas la propriété u *ne dégrade pas le critère " somme des retards". Ceci achève la première partie de la démonstration.*

Ce résultat est insuffisant pour prouver que DTRU *est optimal car* (j,i) *est bien un couple supplémentaire vérifiant la propriété* u, *mais l'échange peut avoir créé plusieurs couple* (j,v) *qui ne respectent plus cette propriété alors qu'ils la vérifiaient précédemment. Donc à partir de* σ, *il pourrait se créer un phénomène périodique d'échanges de tâches qui ne vérifient pas la propriété* u, *sans jamais maximiser* U *et donc sans conduire à* DTRU *pour prouver son optimalité. Ceci justifie la nécessité de la deuxième partie de la démonstration.*

Deuxième partie

Soit σ *une séquence active optimale. Nous allons montrer maintenant que l'on peut construire une suite de séquences (par permutation de couples de tâches ne vérifiant pas la propriété* u) *qui converge vers* DTRU *où tous les couples de tâches vérifient la propriété u.*

Nous notons k_u *et* k'_u *les rangs respectifs de toute tâche* u *dans deux séquences quelconques* σ et σ' .

Nous définissons un ordre sur les couples de rang des tâches en utilisant l'ordre lexicographique strict dont nous rappelons la définition :

$(a,b) <\bullet (c,d)$

$\Leftrightarrow \quad a < c$

ou $\quad a = c$ et $b < d$

Supposons maintenant que (i,j) *soit le premier couple qui ne vérifie pas la propriété* u *dans la séquence* σ, *c'est-à-dire :*

(i,j) *ne vérifie pas* u

et $\forall u, \forall v$ *tels que* (u,v) *ne vérifie pas* u :

$(k_i,k_j) <\bullet (k_u,k_v)$.

Nous construisons maintenant la séquence σ' *à partir de* σ *en inversant* i *et* j *sans toucher aux autres tâches.*

Soit (i',j'), *le premier couple qui dans* σ' *ne vérifie pas* u.

Nous allons montrer que $(k_i,k_j) <\bullet (k'_{i'},k'_{j'})$.

a) Montrons tout d'abord que l'on a $k'_{i'} \geq k_i$

En effet, on a $k_u < k_i \Leftrightarrow k'_u < k'_j$ *car le passage de* σ *à* σ' *ne modifie pas le début de la séquence* σ.

Soit u *une tâche quelconque telle que* $k_u < k_i$ ($\Leftrightarrow k'_u < k'_j$), *et soit* v *une tâche quelconque telle que* $k_u < k_v$ ($\Leftrightarrow k'_u < k'_v$ *toujours parce que le début de la séquence* σ *qui contient* u *est invariant ,* v *étant n'importe où après* u).

(i,j), *premier couple ne vérifiant pas la propriété* u *dans* σ

$\Rightarrow$ (u,v) *vérifie la propriété* u *dans* σ

$\Rightarrow (d_u \leq d_v)$ ou $(c_u - D < r_v)$

$\Rightarrow$ *car* $c_u = c''_u$: $(d_u \leq d_v)$ ou $(c''_u - D < r_v)$

$\Rightarrow$ (u,v) *vérifie la propriété* u dans σ' $\quad \forall$ v : $k_v > k_u$

$(\Rightarrow u \neq i')$

donc $\forall u : k_u < k_i = k'_j$, $\forall v : k_u < k_v$, (u,v) *vérifie la propriété* u *dans* σ'

$\Rightarrow k_u = k'_u < k'_{i'}$

On a donc :

$\forall u : k'_u < k'_j : k'_u < k'_{i'}$

d'où $k'_j - 1 < k'_{i'}$ ($\Leftrightarrow k'_j \leq k'_{i'}$)

et comme $k'_j = k_i$

on a bien $k_i \leq k'_{i'}$

b) *Montrons que dans le cas où* $k'_{i'} = k_i$ *alors on a obligatoirement* $k'_{j'} > k_j$ *en effet :*

(i,j), *premier couple ne vérifiant pas la propriété* u *dans* σ

$\Rightarrow \forall v : k_i < k_v < k_j$: $(d_i \leq d_v)$ ou $(c_i - 1 < r_v)$

avec $k_i = k'_j$

$k_j = k'_i$

$c'_j = c_i$

$d_j < d_i$

et $k_v = k'_v \Rightarrow$

$\forall v : k'_j < k'_v < k'_i$: $(d_j < d_v)$ ou $(c'_j - 1 < r_v)$

et donc $\forall v : k'_j < k'_v < k'_i$: (j,v) *vérifie la propriété* u *dans la séquence* σ'

en outre comme (j,i) *vérifie aussi la propriété* u *car* $d_i > d_j$

on a $k'_{j'} > k'_i = k_i$

Ce qui termine la démonstration : $(k_i,k_j) <\bullet (k'_{i'},k'_{j'})$

Définissons à présent la suite de séquences :

$\sigma_o = \sigma$ *(séquence optimale pour le critère "somme des retards")*

et σ_p *est la séquence construite à partir de* σ_{p-1} *en inversant le premier couple de tâches qui ne vérifie pas la propriété* u.

σ_p *est une suite de séquences qui converge, car l'ensemble des séquences est fini et le fait que les rangs des premiers couples ne vérifiant pas la propriété* u *sont strictement croissants implique qu'on ne peut pas retrouver deux fois la même séquence.*

Tant qu'il existe au moins un couple ne vérifiant pas la propriété u, *on échange les tâches du premier d'entre eux, donc à la dernière itération on effectue le dernier échange possible et pour la dernière séquence possible, il n'existe plus de couple de tâches ne vérifiant pas la propriété u.*

Cette séquence limite σ* *est optimale car les échanges successifs n'ont pas dégradé le critère somme des retards. Tout couple de tâches vérifie la propriété* u *: elle maximise le critère* U *. Elle est identique à* DTRU *, au numérotage des tâches identiques près, puisque respecter* u *pour tout couple de tâches interdit de placer une tâche moins urgente avant une tâche plus urgente déjà arrivée et oblige donc à respecter la règle de Jackson.*
DTRU *est donc optimal pour le critère somme des retards.*

B. Complexité de l'algorithme

L'algorithme de Jackson est polynômial. Il peut être programmé en O(nlog(n)) c'est-à-dire de telle sorte que son nombre d'opérations soit majoré par K.n.Logn, où K est une constante ([J. CARLIER 1982]).

Il est évident que, sans optimisation particulière, le nombre d'opérations de DTRU est majoré par un polynôme de degré 2 : il y a au plus 2n itérations puisque l'on place une tâche au moins une itération sur 2 ; et la recherche de la tâche disponible de plus petit délai ou de la tâche non placée de plus petite date d'arrivée demande au plus, à chaque itération, d'examiner les tâches restantes en nombre inférieur à n.

C. Remarques

1) Les méthodes de décomposition temporelle DTRMU ou DTRUM sont identiques à DTRU car dans le cas des durées unitaires, DTRU minimise aussi le "makespan".

2) Les méthodes de décomposition temporelle DTR et DTRM, qui ne cherchent pas à maximiser le critère U, ne minimisent pas forcément le critère somme des retards.
En effet, considérons ce qui se passe à l'itération k et les conséquences possibles sur la solution globale. A l'itération k, on minimise la somme des retards pour les tâches à ordonnancer à partir de R_k ; si une sous-séquence de tâches n'induit pas de retard quel que soit l'ordre local choisi pour l'exécution des tâches, alors DTR ou DTRM retient un ordre quelconque. Dans ce cas, une tâche moins urgente peut se trouver entre R_k et R_{k+1} , alors qu'une tâche plus urgente située après R_{k+1} va se trouver en conflit avec des tâches au moins aussi urgentes apparaissant en R_{k+1}. Cela crée des retards qui sont évitables et évités par les méthodes de décomposition temporelle utilisant le critère U comme critère secondaire.

3.3. CAS PARTICULIER : DUREES OPERATOIRES EGALES ET NON UNITAIRES

3.3.1. Introduction

Nous avons déjà vu au paragraphe 3.2.1., que si la durée D , commune à toutes les tâches, divise le PGCD des dates d'arrivée r_i , alors, par changement de l'unité de temps, on se ramène à des durées unitaires et la méthode de décomposition temporelle DTRU (identique dans ce cas à DTRMU et DTRUM) est polynômiale et optimale. Nous nous intéressons dans ce chapitre aux durées égales non unitaires.

Dans ce cas, le problème reste ouvert : on ne sait pas s'il est polynômial ou NP-difficile.

Une partie de la réponse est connue dans un cas particulier : lorsque les durées sont identiques, quelles que soient les dates d'arrivée et les délais, le problème de décision "Existe-t-il un ordonnancement de retard nul ?" est polynômial ([J. CARLIER 1982]). Donc, s'il existe une solution telle que la somme des retards est nulle, alors cette solution peut être trouvée par un algorithme polynômial. Dans le cas contraire, le problème reste ouvert.

Dans ce chapitre, nous allons successivement, dans le cas particulier des durées égales et pour le critère somme des retards, montrer que les ordonnancements u-actifs constituent un sous-ensemble "dominant" (§ 3.3.2.), étudier la simplification des méthodes de décomposition temporelle (§3.3.3.), prouver que ces méthodes sont polynômiales mais qu'elles ne fournissent pas toujours la solution optimale (§ 3.3.4.), étudier les erreurs commises dans le pire des cas, et les comparer avec les erreurs commises par d'autres méthodes (§3.3.5.).

3.3.2. Propriété de dominance

Nous avons défini au paragraphe 3.1.3. les ordonnancements u-actifs comme des ordonnancements actifs où tout couple de tâches vérifie la propriété u (rappel : (i,j) vérifie la propriété u dans l'ordonnancement correspondant à la séquence σ si i est placé avant j et si $d_i \leq d_j$ ou $c_i - p_i < r_j$).

Nous allons montrer que, dans le cas des durées égales, le sous-ensemble des ordonnancements u-actifs est "dominant" pour le critère somme des retards, c'est-à-dire qu'il contient au moins une solution optimale. Cela permet de n'explorer que les ordonnancements u-actifs pour chercher une solution optimale.

Théorème 3.3.2.

Dans le cas du problème à une machine, lorsque toutes les durées des tâches sont égales, le sous-ensemble des ordonnancements u-actifs est dominant pour le critère "somme des retards".

Démonstration

Elle se fait selon le même schéma que la démonstration d'optimalité du paragraphe 3.2.3 dans le cas des durées unitaires.

Soit $\sigma_0 = \sigma^*$ un ordonnancement actif optimal. Il en existe au moins un puisque l'ensemble des ordonnancements actifs est dominant pour tout critère régulier.
Soit $(i_0 j_0)$ le premier (dans l'ordre lexicographique) couple de tâches qui ne vérifie pas la propriété u.

Soit σ_1 l'ordonnancement obtenu en inversant i_0 et j_0 dans la séquence σ_0.
On montre dans un premier temps que l'échange de i_0 et j_0 ne dégrade pas le critère somme des retards, puis dans un second temps que la position du premier couple qui ne vérifie pas la propriété u croît strictement dans l'ordre lexicographique.

a) L'échange de i_0 et j_0 ne dégrade pas le critère "somme des retards" noté T.

En effet, comme (i_0, j_0) ne vérifient pas la propriété u, on a :

$$d_{i_0} > d_{j_0} \quad \text{et} \quad c_{i_0} - D \geq r_{j_0}$$

Il est donc possible d'inverser i et j sans toucher aux autres tâches et on obtient l'ordonnancement σ'_1 des exemples de la figure 3.3.2.

La démonstration que $T\sigma'_1$ est inférieur ou égal à $T\sigma_0$ est identique à celle faite au paragraphe 3.2.3. car le fait que les durées étaient unitaires n'intervenait pas dans la démonstration on a donc $T\sigma'_1 \leq T\sigma_0$.

Comme le montre la figure 3.3.2., l'ordonnancement σ'_1 n'est pas forcément semi-actif : soit σ''_1 l'ordonnancement semi-actif obtenu en calant σ'_1 à gauche.
Le passage de σ'_1 à σ''_1 consiste à avancer des tâches, comme le critère est régulier, on a $T\sigma''_1 \leq T\sigma'_1$.

Enfin σ''_1 peut ne pas être actif si un intervalle de temps où la machine est inutilisée a une longueur suffisante pour accueillir une tâche déjà arrivée et planifiée plus tard, on peut avancer cette tâche sans toucher aux autres et sans dégrader le critère.
Il peut y avoir plusieurs tâches candidates à l'utilisation de l'intervalle de temps disponible : on choisira toujours celle de plus petit rang dans la séquence et à égalité de rang la plus urgente. Comme cette opération a laissé un nouvel intervalle de longueur au moins égale à la durée D de la tâche avancée, on cherche s'il existe des tâches déjà arrivées planifiées plus tard et on place la plus urgente, on en cale d'autres à gauche, et ainsi de suite.

On passe ainsi de σ''_1 à σ_1 comme le montrent les différents exemples de la figure 3.3.2.
On n'a jamais retardé de tâches. Donc, comme le critère T est régulier, on a $T\sigma_1 \leq T\sigma''_1$.

Finalement, σ_1 *est actif, puisqu'il n'y existe plus d'intervalle où on peut avancer une tâche sans en retarder d'autres, et* $T\sigma_1 \leq T\sigma_0$.

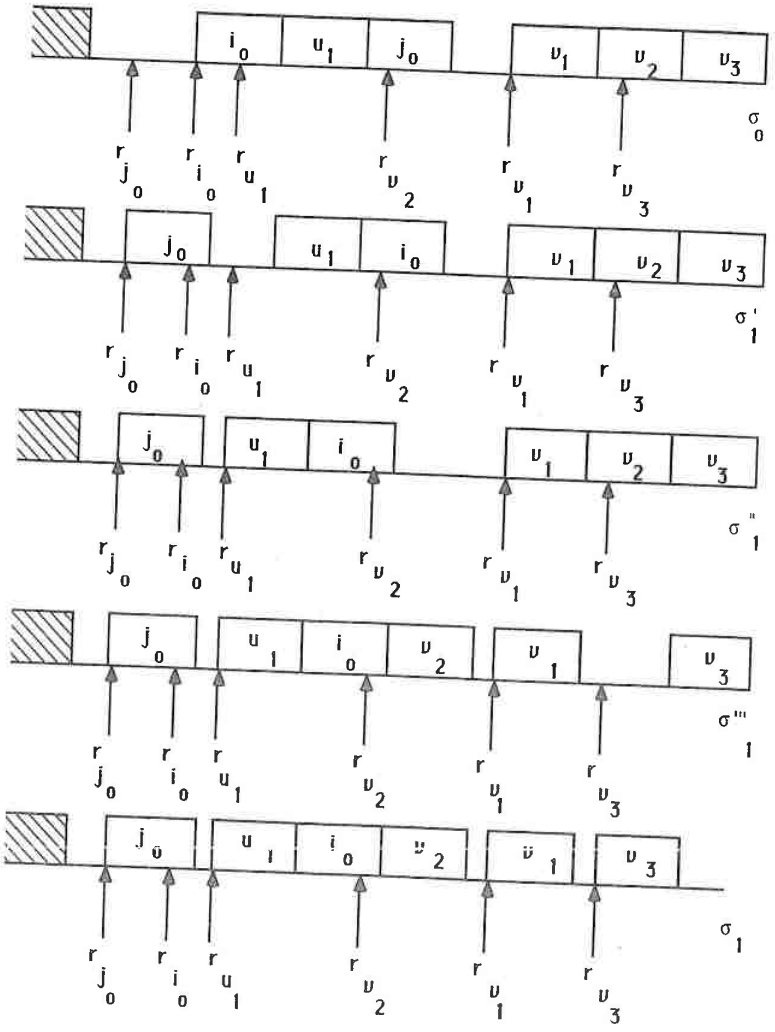


figure 3.3.2. a

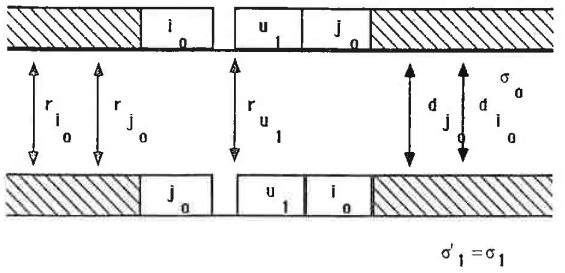


figure 3.3.2. b

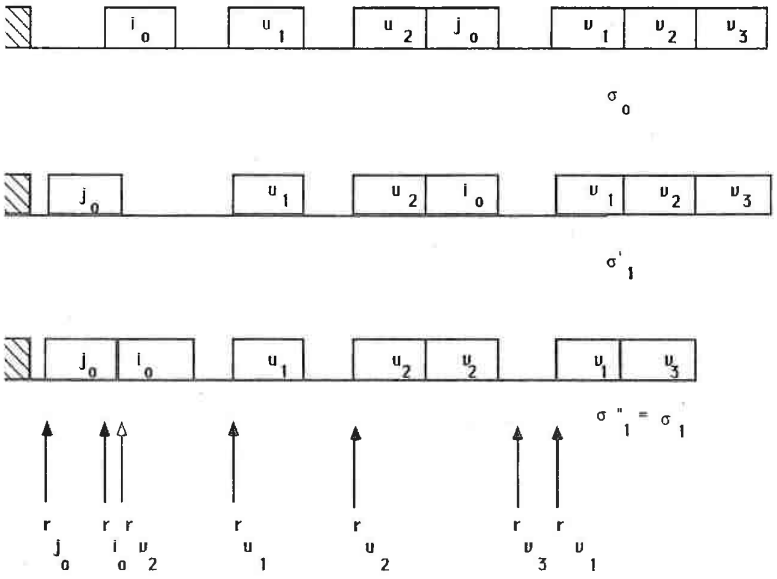


figure 3.3.2. c

Exemples de passage de σ_0 à σ_1 par inversion des tâches du premier couple de tâches ne vérifiant pas la propriété u et avancées de tâches pour rendre l'ordonnancement actif.

figures 3.3.2.

b) *Soit* (i_1,j_1) *le premier (dans l'ordre lexicographique) couple de tâches qui ne vérifie pas la propriété* u *dans l'ordonnancement* σ_1.

Nous choisissons des notations analogues à celles du paragraphe 3.2.3. :

Soit k_j *le rang de la tâche* j *dans* σ_0,

k'_j *son rang dans* σ'_1,

et k''_j *son rang dans* σ_1.

Soit $<\bullet$ *l'ordre lexicographique sur les rangs des couples de tâches.*

Nous allons montrer que $(k_{i0},k_{j0}) <\bullet (k''_{i1},k''_{j1})$

α) *On a* $k''_{i1} \geq k_{i0}$

En effet, on a $k_u < k_{i0} \Leftrightarrow k''_u < k''_{j0}$ *car le passage de* σ_0 *à* σ_1 *ne modifie en aucun cas le début de la séquence* σ_0.

Soit u *une tâche quelconque telle que* $k_u < k_{i0}$ $(\Leftrightarrow k''_u < k''_{j0})$, *et soit* v *une tâche quelconque telle que* $k_v > k_u$ $(\Leftrightarrow k''_v > k''_u$ *toujours parce que le début de* σ_0 *ne change pas*).

(i_0,j_0), *premier couple ne vérifiant pas la propriété* u *dans* σ_0

$\Rightarrow$ (u,v) *vérifie la propriété* u *dans* σ_0

$\Rightarrow$ $(d_u \leq d_v$ ou $c_u - D < r_v)$

$\Rightarrow$ $(d_u \leq d_v$ ou $c''_u - D < r_v)$

$\Rightarrow$ (u,v) *vérifie la propriété* u *dans* σ_1

$\Rightarrow u \neq i1$

$\Rightarrow k''_u < k''_{i1}$

On a donc :

$\forall u : k''_u < k''_{j0}$: $k''_u < k''_{i1}$ $\Rightarrow$ $k''_{j0} - 1 < k''_{i1}$ $(\Leftrightarrow k''_{j0} \leq k''_{i1})$

et comme $k''_{j0} = k''_{i0}$

on a bien $k''_{i1} \geq k_{i0}$

β) *Plaçons nous dans le cas où* $k''_{i1} = k_{i0}$ *et montrons qu'alors* $k''_{j1} > k_{j0}$.

Soit v *une tâche quelconque, différente de* j_0, *telle que* $k_v \geq k_{i0}$ $(\Leftrightarrow k''_v > k''_{j0})$, *montrons que si* (j_0,v) *ne vérifie pas propriété* u *dans* σ_1 *alors* $k''_v > k_{j0}$.

Trois cas sont à considérer :

i) $k_v > k_{j0}$ et $k''_v > k_{j0}$

(*la tâche* v *était après* (i_0,j_0) *dans la troisième partie de l'ordonnancement et a gardé un rang la situant toujours dans cette partie*)

On a bien $k''_v > k_{j0}$

ii) $k_v > k_{j0}$ et $k''_v < k_{j0}$

(*la tâche* v *était après* (i_0,j_0) *dans la troisième partie de l'ordonnancement et se trouve maintenant dans la deuxième partie*).

Ce cas est impossible, car lorsqu'on considère σ''_1, *il existe au plus un intervalle entre* j_0 et i_0 *où placer une tâche et on y place une tâche de rang inférieur ou égal à* k''_{i0} (i_0 *convenant toujours s'il n'y a pas d'autres tâches possibles*). *Ensuite, si la tâche avancée n'était pas* i_0, *on répète le même processus et les tâches de la deuxième partie de l'ordonnancement sont toujours les mêmes. Après avoir avancé* i_0, *le rang de la tâche avancée pour prendre la place de* i_0 *reste un rang de la troisième partie.*

Donc une tâche telle que $k_v > k_{j0}$ *vérifie toujours* $k''_v > k_{j0}$.

iii) $k_{io} \leq k_v < k_{jo}$

(i_o,v) *vérifie la propriété* u *dans* σ_o

$\Rightarrow (d_{io} \leq d_v$ ou $c_{io} - D < r_v)$

$\Rightarrow$ *avec* $d_{jo} < d_{io}$ et $c''_{jo} \leq c_{io}$: $(d_{jo} \leq d_v)$ ou $(c''_{jo} - D < r_v)$

$\Rightarrow (j_o,v)$ *vérifie la propriété* u *dans* σ_1 *(pour tous les* v *de la deuxième partie)*

Ce qui montre que lorsque $i_1 = j_o$ $(\Leftrightarrow k''_{i1} = k_{io} = k''_{jo})$ *alors* $k''_{j1} > k_{jo}$

On a donc dans tous les cas :

$(k_{io},k_{jo}) <\bullet (k''_{i1},k''_{j1})$

En répétant les modifications qui ont fait passer de la séquence σ_o *à la séquence* σ_1, *on obtient une suite de séquence :*

$\sigma_o, \sigma_1, \ldots$ *telle que le critère "somme des retards" reste optimal et la suite d'éléments* (k_i,k_j) *de couples de rang du premier couple de tâches ne vérifiant pas la propriété* u *est strictement croissante.*

Comme dans le cas des durées unitaires, cette suite de séquences converge vers une séquence optimale dont tous les couples de tâches vérifient la propriété u . *En outre, cette séquence est un ordonnancement actif puisque tous les éléments de la suite le sont.*

Ceci achève la démonstration du théorème 3.3.2. Le sous-ensemble des ordonnancements u-actifs est dominant dans le cas des durées égales (unitaires ou non).

Remarques

Le théorème 3.3.2. ne signifie pas que tout ordonnancement u-actif est optimal.

Le théorème 3.3.2. permet de prouver que dans le cas des durées égales, lorsque toutes les tâches arrivent à l'instant 0, EDD minimise la somme des retards (voir le cas des durées unitaires). En effet, en partant d'une séquence optimale, comme toutes les tâches sont inversibles, on peut obtenir ou conserver la propriété u en plaçant toute paire de tâches dans l'ordre d'urgence croissante et la suite σ tend alors vers EDD.

Cela ne prouve rien pour le cas de tâches de durées différentes, où on sait que EDD ne minimise pas toujours la somme des retards, même si toutes les tâches sont disponibles dès l'instant 0.

3.3.3. Simplification des méthodes pour les durées égales

Revenons à l'algorithme général de décomposition temporelle dans le cas comportant une seule machine présenté au paragraphe 3.1.2., pour le simplifier dans le cas des durées égales non unitaires.

L'algorithme général est le suivant : (cf 3.1.3. B) et 3.2.2.)

W := (ensemble des indices des tâches)

$\varphi_0 := \min_{i \in W} [r_i]$ et $\varphi_1 := \min_{i \in W} [r_i + p_i]$

$k := 0$ et $O := \emptyset$ et $P_0 := \emptyset$

tantque $P_k \neq W$ faire

$k := k + 1$

$\pi_k := \{i \ / \ i \in W$ et $\varphi_{k-1} \leq r_i < \varphi_k\}$

$P_k := P_{k-1} \cup \pi_k$

$O := O \oplus \pi_k$

$\varphi_{k+1} := \min_{i \in W} [r_i + p_i]$

fintantque

où $O := O \oplus \pi_k$ se traduit par :

- ne pas modifier les tâches de O qui sont terminées au plus tard à l'instant φ_{k-1} ;

- considérer les autres tâches de O et les tâches de π_k et les ordonnancer à partir de l'instant q_k $(q_k \leq \varphi_{k-1})$ où la machine est disponible.

Nous examinons les invariants de boucle de cet algorithme itératif dans le cas particulier des durées égales non unitaires.

On a :

$$\forall j \in \pi_k : \min_{i \in \pi_k} [r_i] \leq r_j \leq \min_{i \in \pi_k} [r_i] + D - 1 = \varphi_k - 1$$

ou encore :

$$\forall j_1 \text{ et } \forall j_2 \in \pi k : \quad | r_{j_1} - r_{j_2} | < D$$

Cela est immédiat à partir du calcul de φ_k *et parce que* $W - P_{k-1} \supseteq \pi_k$:

$$\varphi_k = \min_{i \in W-P_{k-1}} [r_i+p_i] = \min_{i \in W-P_{k-1}} [r_i] + D \leq \min_{i \in \pi_k} [r_i] + D$$

et $\forall j \in \pi_k : \min_{i \in \pi_k} [r_i] \leq r_j$

avec $r_j \leq \varphi_k \Rightarrow r_j \leq \min_{i \in \pi_k} [r_i] + D - 1$

Considérons maintenant le module $O := O \oplus \pi_k$ *et reprenons les notations des paragraphe* 3.1.3. *et* 3.2.2.
Soit M_K *la date d'achèvement de la dernière tâche à la fin de l'itération* K *(par convention, on prendra* $M_0 = \min_{i \in W} [r_i] = \varphi_0 = R_0$*).*

Deux cas sont à envisager :

a) $M_{k-1} \leq \min_{i \in \pi_k} [r_i] = R_k$

Dans ce cas, l'ordonnancement local de l'itération K *est totalement indépendant de l'ordonnancement partiel précédemment obtenu* O.

Il suffit donc d'ordonnancer de manière optimale (pour le critère T *) les tâches du sous-ensemble* π_k *à partir de* R_k.

α) étude de la forme des séquences localement optimales

Nous notons n_k *le nombre de tâches dans* π_k .

Soit j *la tâche placée en premier, et* v *une tâche quelconque , on a :*

$$c_j = r_j + D \geq \min_{i \in \pi_k} [r_i] + D > \min_{i \in \pi_k} [r_i + D - 1] \geq r_v$$

Toutes les tâches de π_k *sont donc exécutables à la fin de l'exécution de la première, tout ordonnancement actif (ou semi-actif) les place donc de manière contiguë, le seul intervalle où la machine peut être laissée indisponible se trouvant devant* j.

En outre, si l'on impose la première tâche j, *on connaît une séquence qui minimise la somme des retards pour des tâches de durées égales toutes disponibles à partir du même instant* c_j *: c'est la séquence* EDD *pour le sous-ensemble* π_k - {j}.

Pour trouver une solution optimale de cette optimisation locale, il suffit donc de construire l'ordre EDD *correspondant au sous-ensemble* π_k, *puis de calculer la somme des retards pour les* n_k *séquences obtenues en plaçant une tâche* j *en tête et en complétant par* EDD - {j} *comme le montre la figure* 3.3.3. a . *La séquence optimale est celle qui correspond à la meilleure somme des retards.*

β) propriétés de dominance

Pour limiter les calculs, on peut, comme on l'a montré au paragraphe 3.3.2., *ne considérer que les ordonnancements u-actifs. On obtient alors une méthode de décomposition temporelle* DTRU.

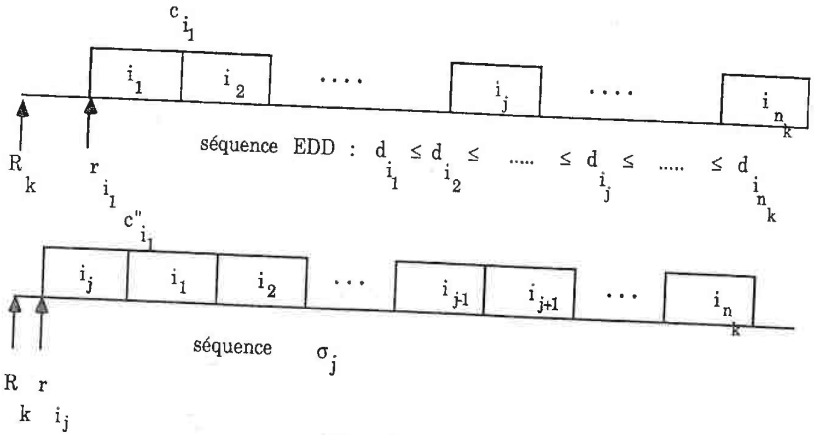


figure 3.3.3. a

Considérons, dans ce but, la séquence EDD/FIFO, *c'est-à-dire celle où les tâches sont classées par délais croissants, et où, à égalité sur les délais, on place en premier la tâche disponible la première.*

Soit EDD/FIFO = $(i_1, i_2, \dots, i_{nk})$.

Considérons une séquence σ_j *(construite à partir de* EDD/FIFO *en plaçant* i_j *en tête).*

Soit i_v *l'indice d'une tâche quelconque placée avant* j *dans* EDD/FIFO :

$i_1 \le i_v < i_j$ et $d_{ij} \ge d_{iv}$.

Deux cas permettent des éliminations de séquences :

cas i)

Si $d_{ij} > d_{iv}$ et $r_{ij} \ge r_{iv}$ *alors* (i_j,i_v) *ne vérifie pas la propriété* u *dans* σ_j *car* $(d_{ij} > d_{iv})$ et $(c''_{ij} - D = r_{ij} \ge r_{iv})$

La séquence σ_j *n'est donc pas un ordonnancement* u-*actif, elle peut être éliminée de la recherche de l'optimum local dans une méthode* DTRU.

cas ii)

Si $d_{ij} = d_{iv}$ et $r_{ij} \ge r_{iv}$ *alors la propriété* u *est vérifiée par* (i_j,i_v) *dans* σ_j, *mais la séquence* σ_j *est dominée par la séquence* σ_v. *En effet, la suite des délais des tâches correspondant à* σ_v *est égale à la suite des délais des tâches correspondant à* σ_j ; *mais, comme* σ_v *commence plus tôt (ou au moins aussi tôt) que* σ_j, *la suite des dates de fin de tâches de* σ_v *est inférieure ou égale à celle de* σ_j *et l'on a donc* $T\sigma_v \le T\sigma_j$.

La séquence σ_j *peut donc, dans ce cas, être également éliminée de la recherche.*

En utilisant ces éliminations dûes à des propriétés de dominance, on démontre le théorème suivant :

Théorème 3.3.2.

Si la séquence EDD fournit un ordonnancement semi-actif qui commence à la plus petite date de disponibilité r_i et qui ne laisse jamais la machine inoccupée entre deux tâches, alors cet ordonnancement est optimal pour le critère "somme des retards" dans le cas où les durées opératoires sont égales.

γ) variantes des méthodes de décomposition temporelle

Ces éliminations ne suffisent pas pour passer de la méthode de décompositon temporelle DTRU *à la méthode de décomposition temporelle* DTRUM.

En effet, si l'on considère le cas particulier :

$r_{i1} > r_{i2} > \ldots > r_{ink}$

et $d_{i1} > d_{i2} > \ldots > d_{ink}$

et $r_{i1} + n_k \cdot D < d_{i1}$

alors on a :

- *toutes les séquences sont optimales avec un retard nul ;*
- *aucune séquence* σ_j *n'est éliminée par les propriétés de dominance utilisées ci-dessus ;*
- *toutes les séquences* σ_j *sont des ordonnancements u-actifs.*

Il faut ajouter la condition de minimisation du "makespan" parmi les solutions minimisant T *pour obtenir la méthode de décomposition temporelle* DTRUM.

δ) algorithme de l'optimisation locale

Avec les propriétés de dominance présentées ci-dessus, l'algorithme d'optimisation locale dans le cas a) *s'écrit :*

1. Construire la séquence EDD/FIFO notée σ_1
2. j* := première tâche de la meilleure solution trouvée = indice 1

   MT := meilleure somme des retards = $T\sigma_1$

   r_{min} := plus petite date de début essayée = r_{i1}

3. pour j de 2 à n_k faire

   si $r_{ij} \geq r_{min}$ alors

   ($\exists v$: $r_{ij} \geq r_{iv}$ et $j > v$ d'où $d_{ij} \geq d_{iv}$)

   la séquence σ_j n'est pas examinée

   sinon

   $r_{min} := r_{ij}$

   calcul de $T\sigma_j$

   si $T\sigma_j < MT$ alors

   $j^* := j$; $MT := T\sigma_j$

   fsi

   fsi

4. La solution localement optimale est σ_{j^*}.

b) $M_{k-1} > \min_{i \in \pi_k} [r_i] = R_k$

Soit q_k *la date de fin de la dernière tâche figée.*

On a $R_k - D < q_k \leq R_k$

Soit ω_k *l'ensemble des tâches de* 0 *telles que :*

$\forall j \in \omega_k$: $c_j - D \geq q_k$

ω_k *est l'ensemble des tâches à réordonnancer avec* π_k.

Puisque la méthode de décomposition temporelle ne modifie plus ce qui précède l'instant q_k, *considérons* q_k *comme date de début de l'ordonnancement et modifions en conséquence les dates d'arrivées :*

$\forall j \in \omega_k \cup \pi_k$: $r'_j = \max [r_j, q_k]$;

Le problème consiste à ordonnancer de manière optimale (pour le critère T) les tâches du sous-ensemble $\omega_k \cup \pi_k$ *à partir de* q_k.

α) étude de la forme des séquences localement optimales

Nous notons n_k *le nombre de tâche de* $\omega_k \cup \pi_k$.

Soit X_k *la plus grande date d'arrivée des tâches de* π_k :

$$X_k = \max_{i \in \pi_k} [r_i] = \max_{i \in \omega_k \cup \pi_k} [r_i] = \max_{i \in \omega_k \cup \pi_k} [r'_i]$$

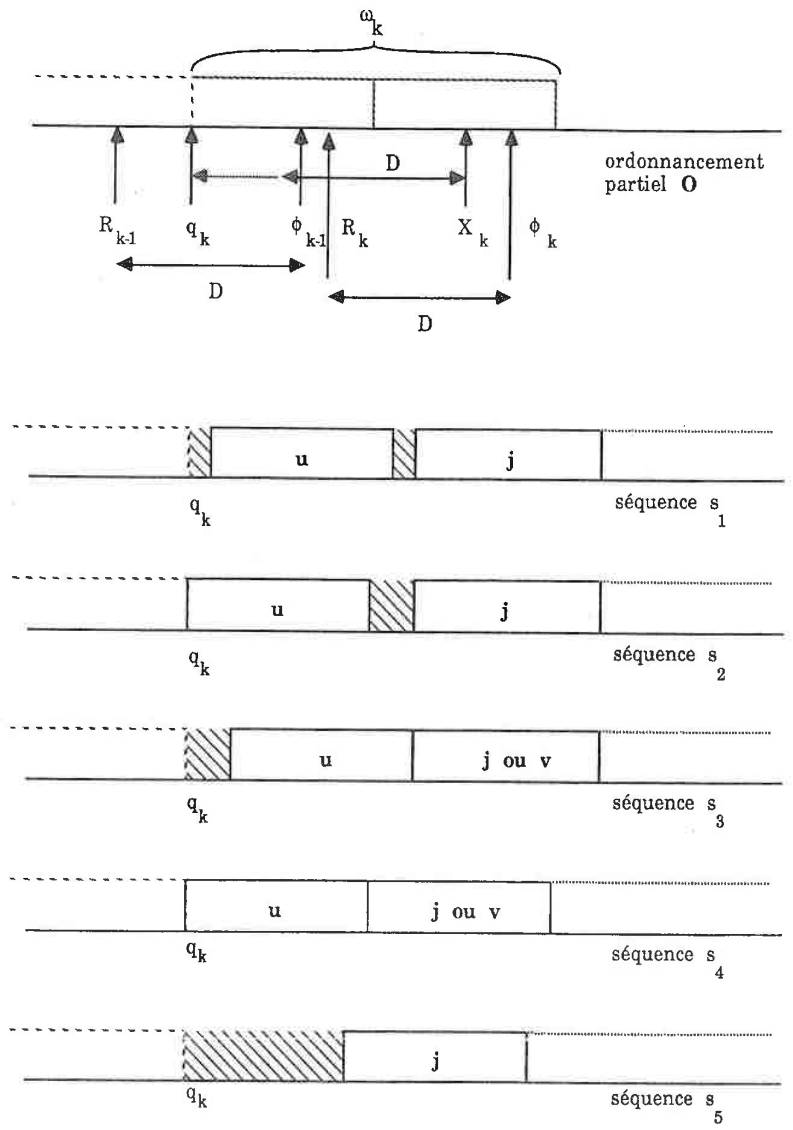
Deux cas sont à envisager.

i) si $q_k \geq X_k - D$ *alors, comme au* a) , *à la fin de l'exécution de la première tâche choisie* j , *toutes les autres tâches sont exécutables et l'on peut utiliser l'algorithme présenté à la fin du* cas a) *au paragraphe* δ) *en remplaçant simplement* r_{ij} *par* r'_{ij}.

ii) si $q_k < X_k - D$ *alors il y a plusieurs façons différentes de laisser dans l'ordonnancement local des intervalles de temps où la machine est inutilisée comme le montre la figure* 3.3.3. b .

Dans le cas de la séquence s_5 , *où l'on place en premier une tâche* j *de* π_k, *toutes les autres tâches de* $\omega_k \cup \pi_k$ *sont exécutables à la fin de la tâche* j *et on complète la fin de séquence de manière optimale par* EDD/FIFO *comme dans le cas* a).

Dans tous les autres cas où l'on commence par placer une tâche u *de* ω_k, *il faut encore choisir une tâche* j *de* π_k *ou* v *de* ω_k *avant d'être sûr de pouvoir compléter de manière optimale par* EDD/FIFO.



durées égales non unitaires : différents cas possibles pour l'optimisation locale dans le cas où $q_k < X_k - D$

figure 3.3.3. b

β) propriétés de dominance

Les conditions de dominance utilisées en a) *pour éliminer des séquences* σ_j *avec* $j \in \pi_k$ *restent valables pour éliminer des séquences* σ_v *avec* $v \in \pi_k \cup \omega_k$ *et* r_j *remplacé par* r'_j.

Pour u *fixé appartenant à* ω_k *placé en tête, on peut également, en notant* q'_u *la fin de* u, *corriger temporairement les* r'_v *des autres tâches en* $r''_v = \max [q'_u, r'_v]$ *et appliquer les conditions de dominance à la fin de la séquence qui suit* u. *Pour ne pas traiter à part le cas où aucune tâche* u *de* ω_k *n'est en tête nous noterons par vide, soit le symbole* ^, *le fait que l'optimisation locale commence par une tâche de* π_k $(u = ^)$.

γ) algorithme de l'optimisation locale

Notons optloc (τ,T,q,s) *la partie d'algorithme qui, à partir d'un sous-ensemble de tâches* τ *dont les dates de disponibilité* r_i *sont comprises entre* q *et* q + D - 1, *cherche une solution optimale ayant une somme des retards inférieure ou égale à la valeur* T *en utilisant l'algorithme vu au paragraphe* δ *du cas* a). *La solution trouvée est* s. *Dans le cas où aucune solution ne peut être au moins aussi bonne que* T, s *prend la valeur* ø.

Examinons maintenant l'algorithme général de l'optimisation locale dans le cas b) :

1. $R_k := \min_{i \in \pi_k} [r_i]$;
2. $\omega_k := \{ i / r_i < R_k$ et $c_i > R_k \}$;
3. $q_k := \min_{i \in \omega_k} [c_i] - D$

4. σ_k := EDD/FIFO $(\pi_k \cup \omega_k) = (i_1, i_2, \ldots)$

   calcul de $T\sigma_k$; $\sigma^* := \sigma_k$;

   MT := $T\sigma_k$; $r_{min} := r_{i1}$;

5. pour tout u de $\omega_k \cup \{^\wedge\}$ dans l'ordre σ_k faire

   5.1. on place u en tête
   d'où
   si u = ^ alors q := q_k ; $r_u := R_k$
   sinon q := date de fin de u
   fsi

   5.2. si $r_u < r_{min}$ alors

   $\tau := \omega_k \cup \pi_k - (u)$

   $T := MT - \rho_u$

   optloc (τ, T, q, s)
   si s ≠ ø alors

   $\sigma^* := u\ s$

   $MT := T_s + \rho_u$

   $r_{min} := r_u$

   fsi

   fsi

La solution localement optimale est σ^*.

Cet algorithme se modifie simplement pour obtenir les différentes variantes DTRU *et* DTRUM.

Pour DTRU, *où on ne cherche pas, à égalité sur les retards, à minimiser la durée totale, on demande au module* "optloc" *de fournir une solution strictement meilleure que* T, *ce qui accélère les calculs.*

Pour DTRUM, *dans et après l'appel du module* "optloc", *si* $(T_s + \rho_u)$ *est égal à* MT, *on retient la solution de meilleure durée totale.*

3.3.4. Non optimalité et complexité

L'algorithme simplifié que nous venons d'obtenir au paragraphe précédent est polynômial.

En effet, si n est le nombre de tâches, il y a au plus n optimisations locales. Pour chacune, on obtient la séquence initiale EDD/FIFO avec une complexité en $O(n \log_2 n)$ (tri). On construit ensuite au plus n^2 séquences pour lesquelles on calcule la somme des retards en balayant les n tâches.

L'algorithme est donc polynômial en $O(n^4)$.

Par contre, il ne fournit pas toujours la solution optimale comme le montre l'exemple de la figure 3.3.4 .

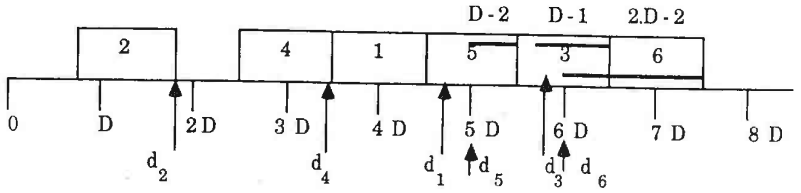
L'énoncé est fourni par la figure 3.3.4. a .
Nous lui appliquons une méthode de décomposition temporelle avec minimisation locale de la somme des retards notée DTR (le détail du déroulement de l'algorithme est fourni dans l'annexe E) et nous obtenons la solution illustrée par la figure 3.3.4. b qui a la valeur 4.D - 5 pour le critère "somme des retards".
La solution optimale est représentée sur la figure 3.3.4. c , et a pour somme des retards la valeur 3 .
L'erreur commise ici par l'algorithme de décomposition temporelle s'élève à 4.D - 2 où D est la durée des tâches.

numéro des tâches i	1	2	3	4	5	6
r_i	0	D - 1	2 D	3D - 2	4 D	5 D - 1
d	5D - 1	2D - 1	6D - 1	4D - 2	5 D	6 D

D = durée égale des tâches

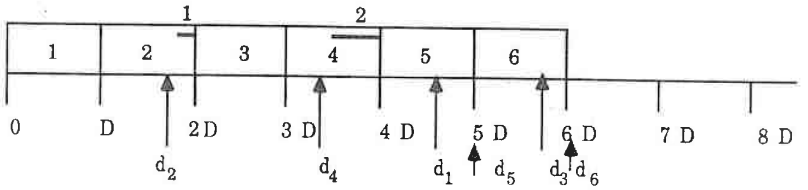
énoncé d'un problème de 6 tâches

figure 3.3.4. a



solution trouvée (somme des retards = 4 D - 5)

figure 3.3.4. b



solution optimale (somme des retards=3)

figure 3.3.4. c

C'est une méthode polynômiale approchée.
Il est donc intéressant de chercher un majorant de l'erreur commise dans le pire des cas. C'est l'objet du paragraphe suivant.

3.3.5. Calculs d'erreurs dans le pire des cas

Les ordonnancement u-actifs étant dominants pour le critère "somme des retards" dans le cas des durées égales, nous limiterons notre étude aux méthodes de décomposition temporelle DTRU et DTRUM définies au paragraphe 3.1.2. et simplifiées dans le cas des durées égales au paragraphe 3.3.3.

Comme les démonstrations directes sont difficiles et afin de permettre les comparaisons, nous allons procéder par étapes (la figure 3.3.5. met en évidence les inclusions entre les différents ensembles d'ordonnancements considérés) :

- au paragraphe 3.3.5.1., nous majorons la plus grande erreur qui puisse être commise sur la somme des retards dans l'ensemble des ordonnancements semi-actifs (indépendamment de toute méthode), et nous montrons que ce majorant peut être atteint ;

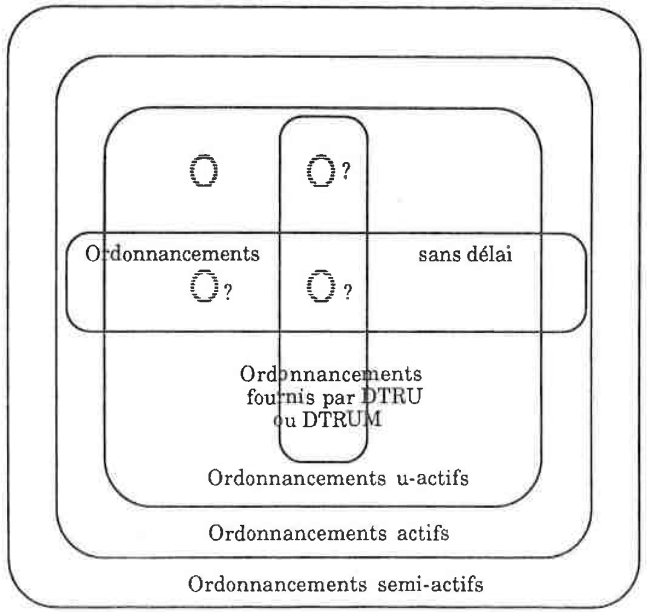
- au paragraphe 3.3.5.2., nous majorons la plus grande erreur qui puisse être commise sur la somme des retards dans l'ensemble des ordonnancements actifs (contenu dans l'ensemble des ordonnancements semi-actifs et contenant au moins une solution optimale) , et nous montrons que ce majorant peut être atteint ;

- au paragraphe 3.3.5.3, nous majorons la plus grande erreur qui puisse être commise sur la somme des retards dans l'ensemble des ordonnancements u-actifs (contenu dans l'ensemble des ordonnancements actifs et contenant au moins une solution optimale lorsque les durées des tâches sont identiques), et nous montrons que ce majorant peut être atteint ;

- au paragraphe 3.3.5.4., nous majorons la plus grande erreur qui puisse être commise sur la somme des retards dans l'ensemble des ordonnancements u-actifs sans délai (contenu dans l'ensemble des ordonnancements u-actifs et dans l'ensemble des ordonnancements sans délai, ce dernier ne contenant pas toujours une solution optimale), et nous montrons que ce majorant peut être atteint ;

- au paragraphe 3.3.5.5., nous majorons la plus grande erreur sur la somme des retards que puissent commettre les méthodes de décomposition temporelle DTRU et DTRUM (dont les solutions sont contenues dans l'ensemble des ordonnancements u-actifs) et nous montrons que les majorants sont des bornes supérieures de l'erreur ;

- au paragraphe 3.3.5.6., nous comparons les résultats obtenus.



Existence d'ordonnancements optimaux démontrée (dans le cas où les durées opératoires sont égales)

? Les sous-ensembles ne contiennent pas toujours des ordonnancements optimaux

figure 3.3.5.

3.3.5.1. Erreur maximale dans le sous-ensemble des ordonnancements semi-actifs

Théorème 3.3.5.1.

Pour le problème à une machine, dans le cas où les durées sont égales à D, si σ' et σ'' sont deux séquences quelconques de n tâches, définissant deux ordonnancements semi-actifs O' et O'' ayant respectivement comme somme des retards $T_{\sigma'}$ et $T_{\sigma''}$, on a :

$$\left| T_{\sigma'} - T_{\sigma''} \right| \leq \max [\, n.\Delta r \,,\, n.(n-1).D \,]$$

où $\Delta r = \max_i [r_i] - \min_i [r_i] = r_{max} - r_{min}$

et cette valeur majorante peut être atteinte.

Démonstration

Elle comporte trois parties.

Dans la première partie, on démontre que $|T_{\sigma'} - T_{\sigma''}| \leq |T_M - T_m|$ *où* T_M *et* T_m *représentent la somme des retards de deux ordonnancements spécifiques "extrêmes" que l'on définira.*

Dans la deuxième partie, on montre que $|T_M - T_m|$ *est majoré par* $\max [\, n.\Delta r \,,\, n.(n-1).D \,]$.

Dans la troisième partie, on met en évidence deux ordonnancements semi-actifs σ' *et* σ'' *pour lesquels cette valeur majorante est atteinte.*

Première partie

Soient σ' *et* σ'' *deux séquences quelconques définissant des ordonnancements semi-actifs dont les sommes des retards par rapport aux délais sont respectivement* $T_{\sigma'}$ *et* $T_{\sigma''}$.

Nous supposons que $T_{\sigma'}$ *est supérieur ou égal à* $T_{\sigma''}$ *(dans le cas contraire, on échange les notations* σ' *et* σ'' *).*

On a donc $|T_{\sigma'} - T_{\sigma''}| = T_{\sigma'} - T_{\sigma''}$.

$\sigma' = (i'_1, i'_2, ..., i'_n)$ *définit l'ordonnancement semi-actif* $0'$.

Soit $R_{max} = r_{min} + \max [\, \Delta r \,,\, (n-1).D \,] = \max [\, r_{max} \,,\, r_{min} + (n-1).D \,]$

A partir de σ', *on construit l'ordonnancement* 0^1 *en posant* $c^1_{i'1} = R_{max}$ *et en calant à gauche derrière* i'_1 *les autres tâches* $i'_2, \ldots, i'_n$.

On a :

$$c^1_{i'1} = R_{max} = \max [\, r_{max} \,,\, r_{min} + (n-1).D \,] \geq r_{max} \geq r_{i'1} \geq c'_{i'1}$$

En retardant le début de la séquence $(i'_1, ..., i'_n)$, *on ne peut que retarder ou maintenir au même instant le début de toutes les tâches.*

Donc $\forall k : c^1_{i'k} \geq c'_{i'k}$

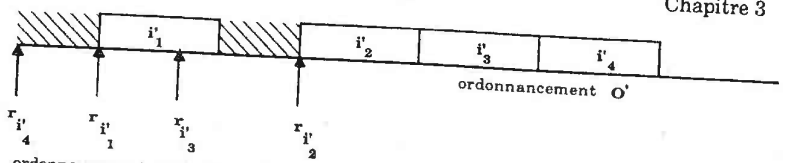
et comme le critère T *est régulier, on a* $T_1 \geq T_{\sigma'}$ *où* T_1 *est la somme des retards de l'ordonnancement* 0^1.

Les figures 3.3.5.1. a et b illustrent de tels ordonnancements $0'$ et 0^1.

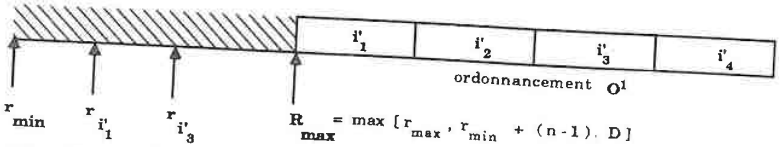
Considérons l'ordonnancement 0^1. *Toutes les tâches sont disponibles à l'instant* R_{max} *et ont des durées égales. Dans ce cas, la séquence* EDD *minimise le critère somme des retards* (cf. 3.2.1.) *et la séquence* LDD, *plaçant les tâches dans l'ordre décroissant des délais le maximise.*

Si 0^M *est l'ordonnancement obtenu en plaçant les tâches dans l'ordre* LDD *à partir de l'instant* R_{max} (voir figure 3.3.5.1. c) *et si* T_M *est la somme des retards correspondants, d'après ce qui précède, on a :* $T_1 \leq T_M$ *et donc*

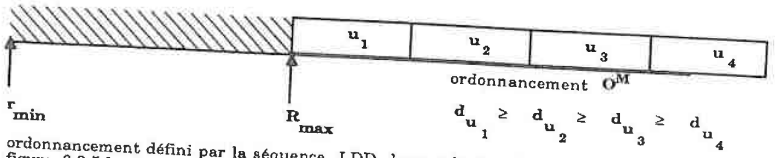
$T_{\sigma'} \leq T_1 \leq T_M$.



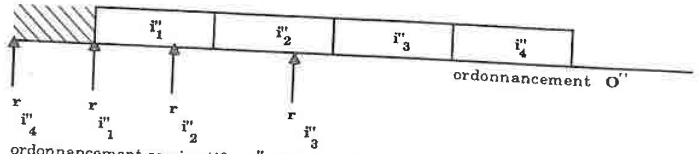
ordonnancement semi-actif O' défini par la séquence σ'
figure 3.3.5.1. a



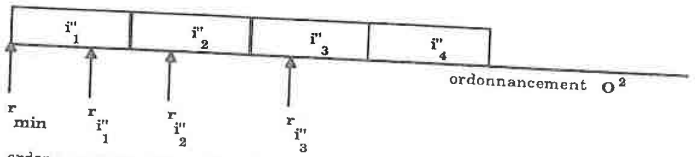
ordonnancement défini par la séquence σ' lorsque le début imposé est R_{max}
figure 3.3.5.1. b



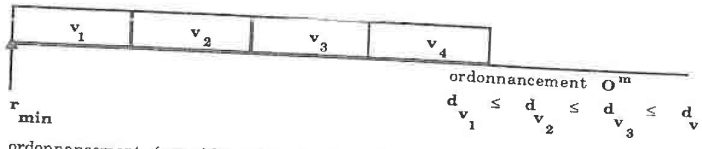
ordonnancement défini par la séquence LDD lorsque le début imposé est R_{max}
figure 3.3.5.1. c



ordonnancement semi-actif O'' défini par la séquence σ''
figure 3.3.5.1. d



ordonnancement (non réalisable) où la séquence σ'' est entièrement calée à gauche sur r_{min}
figure 3.3.5.1. e



ordonnancement (pas obligatoirement réalisable) où la séquence EDD est entièrement calée à gauche sur r_{min}

figure 3.3.5.1. f

Par ailleurs, $\sigma'' = (i''_1, i''_2, ..., i''_n)$ *définissant l'ordonnancement semi-actif* $0''$, *à partir de* σ'', *on construit l'ordonnancement* 0^2 *en posant* $c^2_{i''1} = r_{min} + D$ *et en calant à gauche derrière* i''_1 *les autres tâches* $i''_2, ..., i''_n$ *sans tenir compte des dates de disponibilité* $r_{i''1}, r_{i''2}, ..., r_{i''n}$.

0^2 *n'est pas toujours réalisable pour les dates de disponibilité du problème, mais on a :*

$$\forall k : \quad c^2_{i''k} \leq c''_{i''k}$$

et comme le critère T *est régulier :* $T_2 \leq T_{\sigma''}$ *où* T_2 *est la somme des retards de l'ordonnancement* 0^2.

Les figures 3.3.5.1. d et e illustrent de tels ordonnancements $0''$ et 0^2.

A partir de l'ordonnancement 0^2, *on construit l'ordonnancement* 0^m *en utilisant la séquence* EDD *et en la plaçant à partir de* r_{min} *sans tenir compte des dates de disponibilité des tâches* (voir figure 3.3.5.1. f). EDD *placé ainsi au plus tôt sans interruption minimise la somme des retards* (cf. 3.2.1.).

On a ainsi $T_m \leq T_2$ *où* T_m *est la somme des retards de l'ordonnancement* 0^m.

En regroupant toutes les inégalités on obtient : $T_m \leq T_2 \leq T_{\sigma''} \leq T_1 \leq T_M$

d'où $T_{\sigma'} - T_{\sigma''} \leq T_M - T_m$

ou plus généralemeni $|T_{\sigma'} - T_{\sigma''}| \leq |T_M - T_m|$

Deuxième partie

Nous allons montrer que $|T_M - T_m| \leq \max [n.\Delta r , n.(n-1).D]$.

On a :

$$\forall u : \quad c^m_u \le r_{min} + n.D \le R_{max} + D \le c^M_u$$

par définition de R_{max} *et construction des ordonnancements* 0^M *et* 0^m.

Donc toutes les tâches de 0^m *se terminent au moins aussi tôt que celles de* 0^M.

On a :

$$\forall u : \quad \Delta\rho_u = \rho^M_u - \rho^m_u = \max\ [0, c^M_u - d_u] - \max\ [0, c^m_u - d_u]$$

d'où on déduit en examinant les différents cas possibles :

$$\Delta\rho_u \le c^M_u - c^m_u$$

(l'écart sur les retards d'une tâche entre deux ordonnancements ne peut pas dépasser l'écart positif entre les dates d'achèvement des tâches).

D'où

$$T_M - T_m = \sum_u \Delta\rho_u \le \sum_u c^M_u - \sum_u c^m_u =$$

$$\sum_{k=1}^{n} (R_{max} + k.D) - \sum_{k=1}^{n} (r_{min} + k.D) = n.(R_{max} - r_{min}) =$$

$$n.(r_{min} + \max\ [\Delta r, (n-1).D] - r_{min}]) = \max\ [n.\Delta r, n.(n-1).D]$$

Donc $|T_M - T_m| \le \max\ [n.\Delta r, n.(n-1).D]$.

Troisième partie

On atteint cette valeur maximale pour l'exemple de la figure 3.3.5.1. *g*.

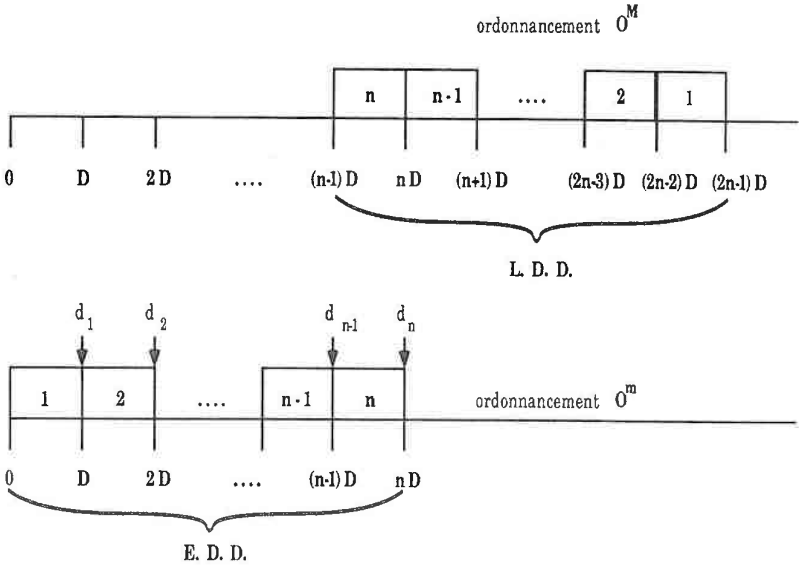
Pour cet exemple , on a :

$$T_M - T_m = \sum_{k=1}^{n-1} [c^M_k - c^m_k] = \sum_{k=1}^{n-1} c^M_k - \sum_{k=1}^{n-1} c^m_k =$$

$$\sum_{k=1}^{n-1} [n.D + k.D] - \sum_{k=1}^{n-1} kD = n.(n-1).D = n.\Delta r = \max\ [n.\Delta r, n.(n-1).D]$$

i	1	2	3	....	n-1	n
r_i	0	D	2.D	....	(n-2).D	(n-1).D
d	D	2.D	3.D	....	(n-1).D	n.D

énoncé d'un problème d'ordonnancement



ordonnancements semi-actifs "extrémaux" pour le critère somme des retards

figure 3.3.5.1. g

3.3.5.2. Erreur maximale dans le sous-ensemble des ordonnancements actifs

Théorème 3.3.5.2.

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D, si O est un ordonnancement actif quelconque dont la somme des retards par rapport aux délais est égale à T, et si T^* désigne la somme des retards de la solution minimale, on a :

$$\Delta\rho = T - T^* \leq \frac{n^2}{4} . (2.D-1)$$

et cette valeur est une borne supérieure.

Démonstration

Elle comporte trois parties.

Dans la première partie, on démontre la formule dans le cas particulier où l'ordonnancement O *ne laisse jamais la machine inoccupée sur des intervalles de temps supérieurs ou égaux à* D.

Dans la deuxième partie, on montre que si l'ordonnancement O *laisse la machine inoccupée sur des intervalles de temps supérieurs ou égaux à* D, *alors le problème de majoration de l'erreur se découpe en plusieurs sous-problèmes pour lesquels on peut appliquer la formule démontrée dans la première partie, afin d'obtenir la formule dans le cas général.*

Dans la troisième partie, on met en évidence deux ordonnancements actifs O *et* O^* *pour lesquels cette valeur majorante est atteinte.*

Première partie

On s'intéresse au cas où, dans l'ordonnancement O, *les intervalles où la machine est inutilisée sont de longueurs inférieures ou égales à* $D-1$. *(*$\Leftrightarrow < D$ *puisque toutes les informations sont supposées être des nombres entiers).*

Comme dans la démonstration du théorème 3.3.5.1., *nous allons construire des ordonnancements, non réalisables, notés* O^M *et* O^m, *et les utiliser pour calculer un majorant de* $T - T^*$. *Pour obtenir ces ordonnancements respectivement moins bon et meilleur que* O, *nous accepterons que la machine exécute deux tâches en parallèles et que l'ordonnancement ne respecte pas les dates de disponibilité des tâches.*

a) construction de O^M et O^m

Cette construction est illustrée par les figures 3.3.5.2. a, b et c.
Soit W *l'ensemble des indices des tâches de* O.

On pose :

$J_1 = \{j/j \in W, c_j > d_j \text{ et } c_j - D = r_j\}$

$J_2 = \{j/j \in W, c_j > d_j \text{ et } c_j - D > r_j\}$

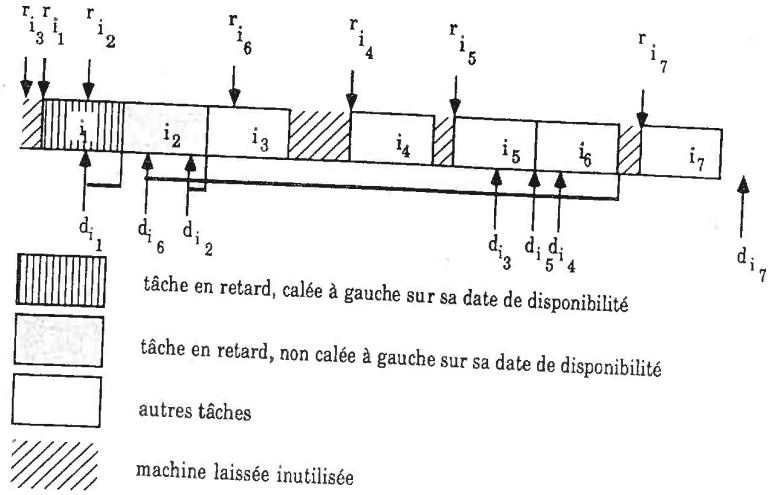
$J_3 = \{j/j \in W, c_j \leq d_j\}$

J_1 *est l'ensemble des tâches en retard qui ne peuvent pas être avancées compte tenu de leur date de disponibilité. Leur retard est inévitable et est minimal dans l'ordonnancement* O.

J_2 *est l'ensemble des tâches en retard qui peuvent être planifiées plus tôt. Si l'ordonnancement* O *n'est pas optimal, c'est en avançant au moins une tâche de* J_2 *que l'on peut l'améliorer.*

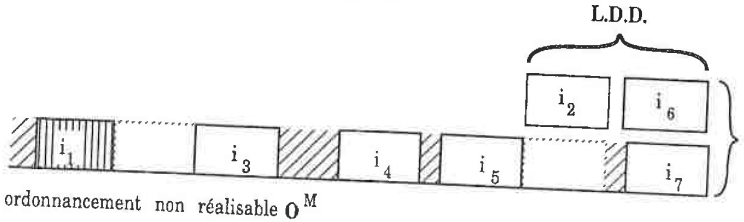
J_3 *est l'ensemble des tâches qui ne sont pas en retard. Leur participation au critère "somme des retards" est nulle. Les avancer ne permet pas d'améliorer le critère.*

Soient i *un élément quelconque de l'ensemble* $\{1, 2, 3\}$ *et* μ *un élément quelconque de l'ensemble de notations* $\{M,m,*\}$ *désignant respectivement les éléments associés aux ordonnancements* O^M, O^m *et* O^*.



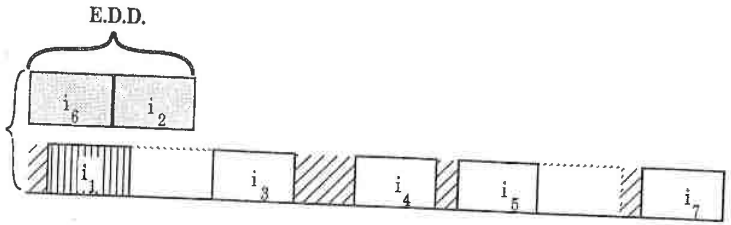
ordonnancement actif O

figure 3.3.5.2. a



ordonnancement non réalisable O^M

figure 3.3.5.2. b



ordonnancement non réalisable O^m

figure 3.3.5.2. c

Nous noterons :

k_i *la cardinalité de l'ensemble d'indices* J_i ,

j_i *l'indice d'une tâche quelconque de* J_i ,

ρ^{μ}_{ji} *le retard de la tâche* j_i *dans l'ordonnancement* O^{μ},

ρ^{μ}_{Ji} *la somme des retards des tâches de* J_i *dans l'ordonnancement* O^{μ}.

L'ordonnancement O^M *est construit de la manière suivante à partir de l'ordonnancement* O :

- les tâches des ensembles J_1 *et* J_3 *sont maintenues à leurs emplacements respectifs ;*

- les tâches de l'ensemble J_2 *sont déplacées vers la droite, à l'emplacement des* k_2 *dernières tâches de* O *(éventuellement en parallèle avec les autres tâches). On les maintient dans leur ordre initial. Aucune n'est avancée : la somme des retards se dégrade ou reste constante. On permute ensuite ces tâches entre elles pour les mettre dans l'ordre* LDD.(*)

On a :

$$T = \rho_{J1} + \rho_{J2} + \rho_{J3} = \rho^M_{J1} + \rho_{J2} + \rho^M_{J3} \leq \rho^M_{J1} + \rho^M_{J2} + \rho^M_{J3} = T_M$$

et donc $T \leq T_M$

(*) En fait, cette dernière modification est facultative car si toutes les tâches J_2 sont en retard dans O^M :

$$\rho^M_{J2} = \sum_{j_2 \in J_2} c_{j2} - \sum_{j_2 \in J_2} d_{j2}$$

et le fait de passer à LDD ne change pas la valeur du critère.

L'ordonnancement 0^m *est construit de la manière suivante à partir de l'ordonnancement* 0 :

- *les tâches des ensembles* J_1 *et* J_3 *sont maintenues à leurs emplacements respectifs ;*

- *les tâches de l'ensemble* J_2 *sont déplacées vers la gauche, placées à partir du début, de manière contiguë (éventuellement en parallèle avec les autres tâches), en les maintenant d'abord dans leur ordre initial (toutes avancent ou restent à leur place initiale), puis en les triant dans l'ordre* EDD.

On a :

$$\rho^m_{J2} \leq \rho_{J2}$$

d'où

$$T_m = \rho^m_{J1} + \rho^m_{J2} + \rho^m_{J3} = \rho_{J1} + \rho^m_{J2} + \rho_{J3} \leq \rho_{J1} + \rho_{J2} + \rho_{J3} = T$$

et donc $T_m \leq T$

Par ailleurs,

$\rho^m_{J2} \leq \rho^*_{J2}$ *car on ne peut faire mieux pour* J_2 *dans* 0^m,

$\rho^m_{J1} \leq \rho^*_{J1}$ *car on ne peut pas avancer plus les tâches de* J_1,

$\rho^m_{J3} = \rho_{J3} = 0 \leq \rho^*_{J3}$ *car on ne peut faire mieux que* 0 *comme somme des retards de* J_3,

d'où

$$T_m = \rho^m_{J1} + \rho^m_{J2} + \rho^m_{J3} \leq \rho^*_{J1} + \rho^*_{J2} + \rho^*_{J3} = T^*$$

Enfin, par définition de T^* : $T^* \leq T$

On a donc :

$$T_m \leq T^* \leq T \leq T_M$$

d'où

$$T - T^* \leq T_M - T_m = \rho^M_{J2} - \rho^m_{J2}$$

Nous allons maintenant majorer $\rho^M_{J2} - \rho^m_{J2}$.

$$\rho^M_{J2} - \rho^m_{J2} = \sum_{j_2 \in J_2} [\max [0, c^M_{j2} - d_{j2}] - \max [0, c^m_{j2} - d_{j2}]]$$

Soit $J_4 = \{j / j \in J_2$ et $c^M_{j2} > c^m_{j2}\}$

on a :

$$\rho^M_{J2} - \rho^m_{J2} \leq \sum_{j \in J_4} c^M_{j4} - \sum_{j \in J_4} c^m_{j4}$$

on obtient ce résultat en montrant que :

$$A \geq B \Rightarrow \max [0, A] - \max [0, B] \leq A - B$$

et que :

$$A < B \Rightarrow \max [0, A] - \max [0, B] \leq 0$$

Si k_4 *est la cardinalité de* J_4, *on majore* $\sum_{j \in J_4} c^M_{j4}$ *en prenant les dates des* k_4 *dernières tâches de* 0 *et on majore* $-\sum_{j \in J_4} c^m_{j4}$ en prenant $-\sum_{j=1}^{k_4} j \,.\, D$, *c'est-à-dire les dates des* k_4 *premières tâches de* 0^m.

Nous examinons les dates de fin possibles pour les dernières tâches de 0 *dans le pire des cas .*

Pour la dernière : $n.D + r.(D - 1)$

c'est-à-dire la somme des durées de toutes les tâches plus la somme des durées des r *intervalles où la machine est laissée inutilisée. Ces intervalles sont, par hypothèse de la première partie, de durée inférieure ou égale à* $D - 1$

puis $(n - 1).D + r.(D - 1)$ *pour l'avant dernière*

etc...

jusqu'à $(n - k4 + 1).D + r.(D - 1)$.

D'où

$$T - T^* \leq \rho^M_{J2} - \rho^m_{J2} \leq \sum_{k=1}^{k_4} [(n + 1 - k).D + r.(D - 1)] - \sum_{k=1}^{k_4} k.D =$$

$$k_4.(n + 1).D + k_4.r.(D - 1) - D.k_4.(k_4 + 1) = k_4.(n - k_4).D + k_4.r.(D - 1)$$

Par ailleurs, on a :

$k_1 + k_2 + k_3 = n, \quad k_4 \le k_2$

et $r \le k_1$ *car dans les ordonnancements actifs, la machine n'est laissée inoccupée que devant des tâches qui sont planifiées au plus tôt à leur date de disponibilité.*

On en déduit :

$r \le k_1 = n - k_2 - k_3 \le n - k_2 \le n - k_4$

D'où

$T - T^* \le k_4.(n - k_4).D + k_4.(n - k_4).(D - 1) = k_4.(n - k_4).(2.D - 1)$

avec $0 \le k_4 \le n$

La fonction $f(x) = x.(n - x)$ *est maximale pour* $x = n/2$.

Et donc $T - T^* \le \frac{n^2}{4} (2.D - 1)$

Ceci termine la première partie de la démonstration.

Deuxième partie

Considérons un ordonnancement 0 *où la machine est laissée inoccupée pendant* p *intervalles de temps de longueurs supérieures ou égales à* D. *Soient* $0_1, 0_2, \dots, 0_k, \dots, 0_{p+1}$ *les ordonnancements partiels de* $n_1, n_2, \dots, n_{p+1}$ *tâches séparés par ces intervalles, chacun d'entre eux ayant une somme des retards de ses tâches notée* T_k.

Considérons le $k^{ème}$ *ordonnancement partiel* 0_k. *S'il est précédé d'un intervalle de durée supérieure ou égale à* D, *c'est qu'aucune de ces tâches n'a une date de disponibilité suffisamment petite pour venir utiliser cet intervalle. L'ordonnancement de* 0_{k-1} *n'a donc, tel qu'il a été fait dans* 0, *aucune influence sur l'ordonnancement de* 0_k.

$T_k - T^*_k$ *ne dépend que des caractéristiques des tâches de* 0_k.

Alors on peut appliquer à chaque 0_k *(qui ne contient pas d'intervalles sur lesquels la machine est laissée inutilisée, de longueur supérieure ou égale à* D *) le résultat de la première partie :*

$$T - T^* = \sum_{k=1}^{p+1} (T_k - T^*_k) \le \sum_{k=1}^{p+1} \frac{n_k^2}{4} (2.D - 1) = \frac{2.D - 1}{4} \sum_{k=1}^{p+1} n_k^2 \le$$

$$\frac{2.D - 1}{4} \left(\sum_{k=1}^{p+1} n_k \right)^2 = \frac{n^2}{4} (2.D - 1)$$

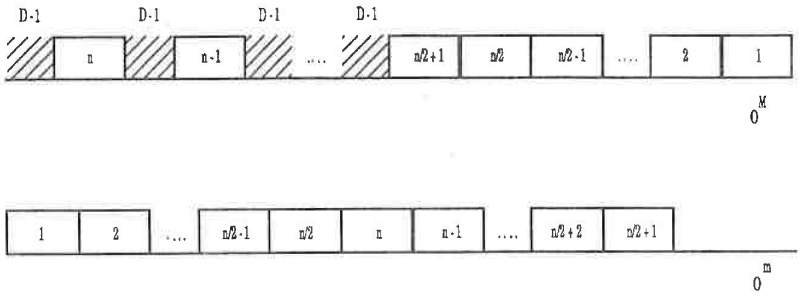
Ce qui termine la deuxième partie de la démonstration.

Troisième partie

On atteint cette valeur maximale pour l'exemple de la figure 3.3.5.2. d.

i	1	2	...	n/2	n/2+1	n/2+2	...	n-1	n
r_i	0	D	...	(n/2-1).D	(n-1).D - n/2	(n-3).D - n/2 - 1	...	3.D - 2	D - 1
d_i	D	2.D	...	n/2.D	n.D	n.D	...	n.D	n.D

énoncé d'un problème d'ordonnancement



ordonnancements actifs extrêmaux réalisables pour le critère somme des retards (n pair)

figure 3.3.5.2 d

où:

$$T_M - T_m = \sum_{k=1}^{n/2} c^M_k - \sum_{k=1}^{n/2} c^m_k = \sum_{k=1}^{n/2} [\, n/2.(2.D - 1) + k.D \,] - \sum_{k=1}^{n/2} k.D =$$

$$\frac{n^2}{4} (2.D - 1)$$

3.3.5.3. Erreur maximale dans le sous-ensemble des ordonnancements u-actifs

Théorème 3.3.5.3.

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D, si $\mathbf{0}$ est un ordonnancement u-actif dont la somme des retards par rapport aux délais est égale à T et si T^* désigne la somme des retards de la solution minimale, on a :

$$\Delta\rho = T - T^* \leq \frac{n}{2} \cdot \frac{D.(D-1)}{2.D-1} \cdot \left[\frac{n.(3.D-2)}{2.D-1} + 1 \right]$$

et cette valeur est une borne supérieure.

Démonstration

Elle suit le même schéma que celui du théorème 3.3.5.2. dans le cas des ordonnancements actifs.

Première partie

On s'intéresse au cas où, dans l'ordonnancement $\mathbf{0}$, *les intervalles où la machine est inutilisée sont de longueurs inférieures ou égales à* $D - 1$.

On construit deux ordonnnancements $\mathbf{0}^M$ *et* $\mathbf{0}^m$ *tels que* $T - T^* \leq T_M - T_m$, *puis on majore* $T_M - T_m$.

Les constructions des ordonnancements extrêmaux et le calcul des majorations dépendent de l'importance de la durée totale d'inutilisation de la machine entre les tâches de l'ordonnancement $\mathbf{0}$.

Soit L *la somme des intervalles où la machine est laissée inutilisée dans* $\mathbf{0}$ *et soit* $\lambda_0.(D-1) \leq L < (\lambda_0 + 1).(D-1)$.

On remplace l'ordonnancement $\mathbf{0}$ *par l'ordonnancement* $\mathbf{0}'$ *où la dernière tâche est reportée de* $(L - \lambda_0 (D-1))$ (figure 3.3.5.3.a).

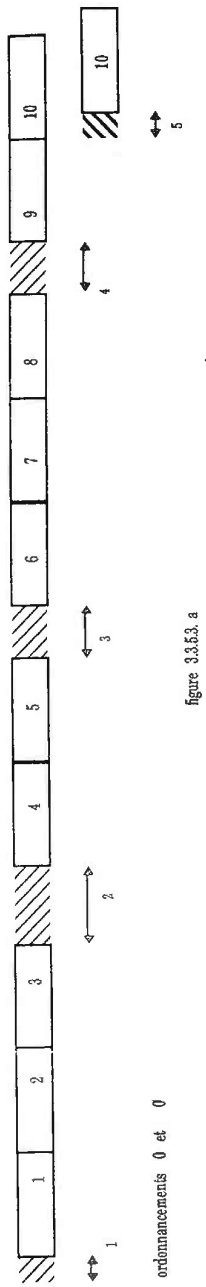


figure 3.3.5.3. a

ordonnancements 0 et 0

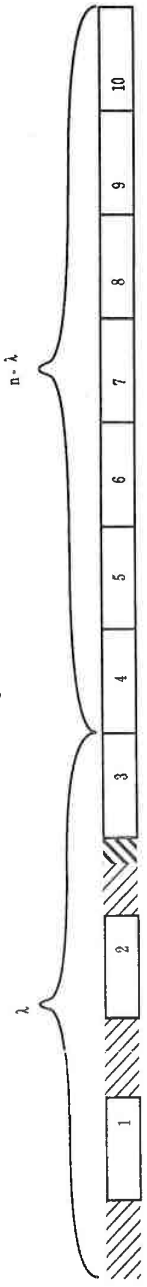
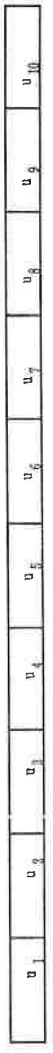


figure 3.3.5.3. b

ordonnancement 0^M

les intervalles où la machine est laissée inutilisée sont avancés le plus tôt possible



$d_{u_1} \leq d_{u_2} \leq \ldots \leq d_{u_{10}}$

ordonnancement (non réalisable) 0^m = séquence E.D.D. calée à gauche

figure 3.3.5.3. c

Si L' *est la somme des intervalles inutilisés de* $0'$, *on a :* $0 \le T' - T \le D - 2$ *et* $L' = \lambda.(D - 1)$.

Nous allons distinguer deux cas selon que $L' \le (n - \lambda) . D$ *et* $L' \ge (n - \lambda) . D$ *Ces deux cas correspondent aux situations suivantes : il existe peut-être des tâches qui peuvent être avancées pour combler tous les intervalles où la machine est inutilisée, ou il n'existe sûrement pas assez de tâches non bloquées sur leur instant de début pour combler complètement tous les intervalles de* $0'$.

A) cas où $L' \le (n - \lambda) . D$

a) construction de 0^M et 0^m

La construction de 0^M *est illustrée par les figures* 3.3.5.3. a *et* b .
Les tâches de 0 *sont maintenues dans le même ordre. Les intervalles où la machine est laissée inutilisée sont placés le plus tôt possible.*

Les λ *intervalles sont dans* 0^M *devant les* λ *premières tâches.*
Cette modification de la durée et de l'emplacement des intervalles ne peut en aucun cas avancer des tâches :

$\forall i,\ 1 \le i \le n :\ c_i \le c^M_i$, *et comme le critère "somme des retards" est régulier :* $T' \le T_M$, *d'où* $T \le T' \le T_M$.

La construction de 0^m *est illustrée par les figures* 3.3.5.3. a *et* c .
Comme pour le théorème 3.3.5.1. , *la séquence* EDD *est calée à gauche même si elle n'est pas réalisable.*

On a donc $T_m \le T^*$.

D'où $T_m \le T^* \le T \le T_M$ *et* $T - T^* \le T_M - T_m$

b) majoration de $T_M - T_m$

On a : $T_M - T_m \le \sum_{i=1}^{n} c^M_i - \sum_{i=1}^{n} c^m_i$

Pour obtenir plus rapidement une formule simple, considérons les écarts sur les λ *premières tâches (dont la première avance de* $(D - 1)$, *la seconde de* $2.(D - 1)$, *...) et sur les* $(n - \lambda)$ *dernières tâches (écarts de* $L' = \lambda . (D - 1)$ *).*

$$T_M - T_m \le \frac{\lambda (\lambda + 1)}{2} . (D - 1) + (n - \lambda) . \lambda . (D - 1)$$

$$T_M - T_m \le (D - 1) . \frac{\lambda}{2} . [\, 2.n + 1 - \lambda \,]$$

La fonction $f(\lambda) = (D - 1) . \frac{\lambda}{2} [\, 2.n + 1 - \lambda \,]$ *croît, lorsque* λ *croît, de* $f(0)$ *jusqu'à une valeur maximale obtenue pour* $\lambda = n + 1/2$, *puis décroît.*

Mais nous avons ici une contrainte sur λ :

$\lambda . (D - 1) = L' \le (n - \lambda) . D \quad \Rightarrow \quad \lambda . (2 . D - 1) \le n . D$

Et en donnant à λ *la valeur (éventuellement non entière)* $n . \frac{D}{2 . D - 1}$, *on obtient comme nouvelle majoration de* $T_M - T_m$:

$$T_M - T_m \le \frac{(D - 1)}{2} . \frac{n . D}{2 . D - 1} . [\, 2.n + 1 - \frac{n . D}{2 . D - 1} \,] =$$

$$\frac{n}{2} . \frac{D . (D - 1)}{2 . D - 1} [\, \frac{n . (3 . D - 2)}{2 . D - 1} + 1 \,]$$

B) Cas où $L' \ge (n - \lambda) . D$

a) construction de 0^M et 0^m

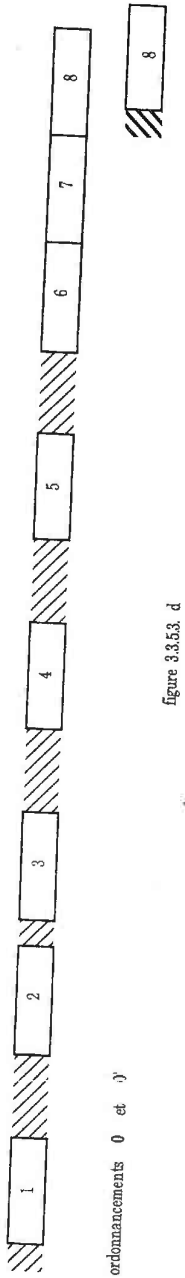
La construction de 0^M *est analogue à celle du cas* A). *Elle est illustrée dans ce cas par les figures* 3.3.5.3. d *et* e .
On a de la même façon $T \le T' \le T_M$.

La construction de 0^m *est illustrée par la figure* 3.3.5.3. f . *On avance les tâches au maximum en tenant compte du fait que :*

$L' \ge (n - \lambda) . D$ et $L' = \lambda . (D - 1) \quad \Rightarrow \quad \lambda \ge \frac{n . D}{2 D - 1}$

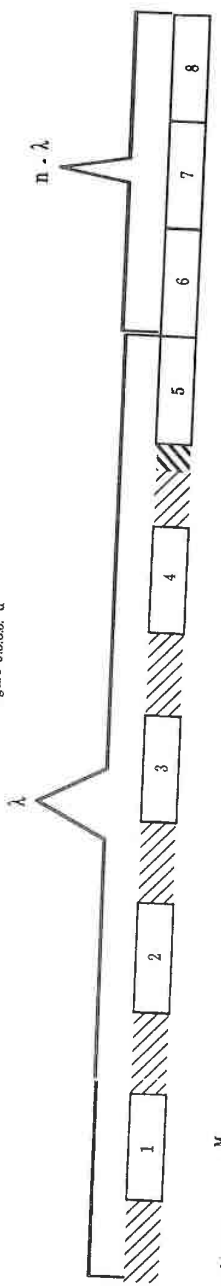
et $r_{max} = (\lambda - 1) . D + L' = \lambda . (2 . D - 1) - D \ \ge \ (n - 1) . D$

Donc si $\lambda > \frac{nD}{2D - 1}$ *alors* $r_{max} > (n - 1) . D$



ordonnancements $\mathbf{0}$ et $\mathbf{0}'$

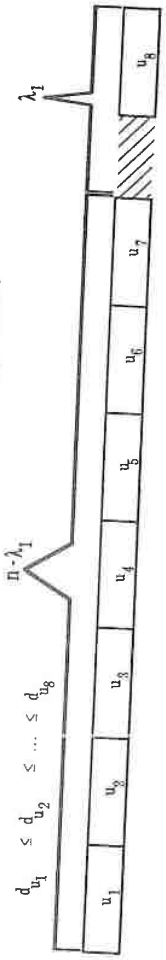
figure 3.3.5.3. d



ordonnancement $\mathbf{0}^M$

les intervalles où la machine est laissée inutilisée sont avancés le plus tôt possible

figure 3.3.5.3. e



ordonnancement (non réalisable ?) $\mathbf{0}^m$

séquence E.D.D calée à gauche en tenant compte des r_i minimaux des derniers emplacements possibles

figure 3.3.5.3. f

La dernière tâche (dans l'ordre des arrivées) est placée dans $\mathbf{0}^m$ *à l'instant* $\max [r_{max} , (n-1) . D] = \lambda . (2 . D - 1) - D \geq (n-1) . D$.

L'avant dernière tâche (dans l'ordre des arrivées est placée à l'instant $\max [r_{max} - (2 . D - 1) , (n-2) . D]$ *et, en remontant ainsi l'ordre des arrivées, la tâche qui arrive au rang* $(n - \lambda + 1)$ *est placée à l'instant* $\max [r_{max} - (\lambda - 1) . (2 . D - 1) , (n - \lambda) . D]$.

Les λ *dernières tâches ne peuvent pas être placées plus tôt compte tenu des* λ *dernières dates d'arrivées dans* $\mathbf{0}'$ *et des* $(n - \lambda)$ *tâches placées devant. On cale à gauche les* $(n - \lambda)$ *premières tâches.*

Si, dans les emplacements ainsi définis, on place les tâches dans le même ordre que $\mathbf{0}'$ *, alors toute tâche est placée au moins aussi tôt dans* $\mathbf{0}^m$ *que dans* $\mathbf{0}'$*, et le critère "somme des retards" ne peut pas se dégrader ; il en est de même si, en conservant les mêmes emplacements, on utilise la séquence* EDD : $T_m \leq T'$.

On a $T_m \leq T^*$ *puisque toute tâche est placée le plus tôt possible tout en respectant* EDD *(éventuellement on avance la dernière tâche d'au plus* $D - 2$ *pour revenir à la dernière position dans* $\mathbf{0}$ *légèrement meilleure que celle de* $\mathbf{0}'$*).*

D'où $T_m \leq T^* \leq T \leq T' \leq T_M$ *et* $T - T^* \leq T_M - T_m$

b) majoration de $T_M - T_m$

Soit λ_1 *le nombre de tâches précédées d'un intervalle où la machine est inutilisée dans* $\mathbf{0}^m$.

Ces λ_1 *tâches n'interviendront pas dans le calcul de* $T_M - T_m$ *puisqu'elles occupent la même position dans les deux ordonnancements.*

$$T_M - T_m \leq \sum_1 c^{M}_i - \sum_1 c^{m}_i$$

$$T_M - T_m \le \frac{(\lambda - \lambda_1).(\lambda - \lambda_1 + 1)}{2} . (2 . D - 1)$$

$$+ [\ \lambda . (n - \lambda) . (2 . D - 1) + \frac{(n - \lambda).(n - \lambda + 1)}{2} . D\ ]$$

$$- \frac{(n - \lambda_1).(n - \lambda_1 + 1)}{2} . D = h(\lambda, \lambda_1)$$

Supposons d'abord λ *fixé et cherchons la valeur maximale en faisant varier* λ_1 avec $0 \le \lambda_1 \le \lambda$:

$$g(\lambda_1) = h(\lambda, \lambda_1) =$$

$$\frac{(\lambda_1)^2}{2} \cdot (D - 1) + \lambda_1 . [\ -\lambda . (2 . D - 1) - \frac{1}{2} \cdot (2 . D - 1) + n . D + \frac{1}{2} . D\ ]$$

$$+ [\ \frac{\lambda . (\lambda + 1)}{2} (2 . D - 1) + \lambda . (n - \lambda) . (2D - 1)$$

$$+ \frac{(n - \lambda).(n - \lambda + 1)}{2} . D - \frac{n.(n + 1)}{2} D\ ]$$

$$g(\lambda_1) = \frac{(\lambda_1)^2}{2} \cdot (D - 1) + \lambda_1 . [(n + 1/2) . D - (\lambda + 1/2) . (2D - 1)]$$

$$+ [\ (\frac{\lambda.(\lambda + 1)}{2} + \lambda.(n - \lambda)).(2D - 1) + (\frac{(n - \lambda).(n - \lambda + 1)}{2} - \frac{n.(n + 1)}{2}).D\ ]$$

$$g'(\lambda_1) = \lambda_1 . (D - 1) + (n + 1/2) . D - (\lambda + 1/2) . (2D - 1)$$

$g'(\lambda_1)$ *s'annule pour*

$$\lambda'_1 = \frac{(\lambda + 1/2) . (2 D - 1) - (n + 1/2) . D}{D - 1}$$

$$\lambda . (2 . D - 1) \ge n . D \Rightarrow \lambda'_1 \ge \frac{1/2 .(2 .D - 1) - 1/2 .D}{D - 1} = 1/2$$

$$\lambda \le n \Rightarrow \lambda'_1 \le n + 1/2$$

La fonction $g(\lambda_1)$ *décroît quand* λ_1 *croît de* 0 *à* λ'_1 *et croît quand* λ_1 *croît de* λ'_1 *à* n *(si* $\lambda \le n - 1$, *ce que nous supposons, car si* $\lambda = n$ *alors toutes les tâches de* **0** *sont calées à gauche sur leurs dates d'arrivée et* **0** *est alors optimal :* $T - T^* = 0$*).*

$g(\lambda_1)$ *est donc maximal aux extrémités de l'intervalle* [0,n] .

En outre $\lambda_1 \le \lambda$

et $\lambda'_1 \le \lambda$ ($\Leftrightarrow (\lambda + 1/2).(2D - 1) - (n + 1/2)\, D \le \lambda . (D - 1)$

$\Leftrightarrow \lambda . D \le (n + 1/2).D - D + 1/2 = (n - 1/2).D + 1/2$)

$g(\lambda_1)$ *est donc maximal en* 0 *ou en* λ .

Avec $\lambda_1 = 0$, *on retrouve l'exemple de la figure* 3.3.5.3. c ; *on couvre les intervalles où la machine était inutilisée d'où* $\lambda . (D - 1) \le (n - \lambda)\, D$, *avec* $\lambda . (D - 1) \ge (n - \lambda)\, D$, *cela conduit à* $\lambda = \frac{n . D}{D - 1}$, *et on retrouve la majoration du cas* A).

$\lambda_1 = \lambda$ *entraîne* $(n - \lambda) = 0$: *on n'a pas de tâches contigües au début de* 0^m *et* $g(\lambda) = 0$.

On retrouve donc un seul cas extrême, celui du cas A).

Deuxième partie

Pour un ordonnancement **0** *tel que les intervalles où la machine est inutilisée sont de durée inférieure ou égale à* D - 1 , *on a trouvé dans la première partie :*

$T - T^* \le A.n^2 + B.n$ avec $A > 0$ et $B > 0$

Les ordonnancements u-actifs étant des ordonnancements actifs, on a indépendance entre les sous-séquences séparées d'un intervalle de durée supérieure ou égale à D - 1 :

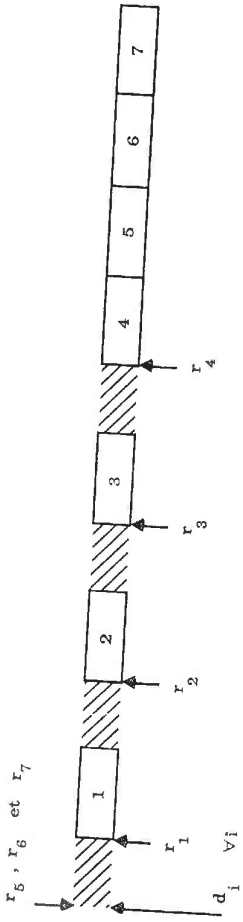
$$T - T^* = \sum_k (T_k - T^*_k) \le \sum_k (A.n^2_k + B.n_k) \le A . \sum_k n^2_k + B . \sum_k n_k \le A.n^2 + B.n$$

Troisième partie

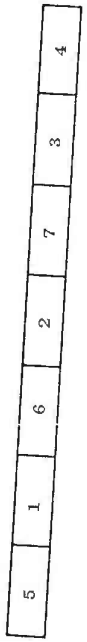
Cette valeur majorante est atteinte pour l'exemple de la figure 3.3.5.3. g .

i	1	2	...	D	...	k.D	...	k.D+1	...	n
r_i	(2.D - 1) - D	2.(2.D - 1) - D	...	D.(2.D - 1) - D		k.D.(2.D - 1) - D		0	...	0
d_i	0	0	...	0	...	0	...	0	...	0

énoncé d'un exemple général



ordonnancement 0 (n = 7 , D = 4 , λ = 4 , n - λ = 3 , k = 1)



ordonnancement 0*

figure 3.3.5.3. g

Toutes les tâches ont un délai de 0 *et sont donc en retard quel que soit l'ordre. La somme des retards est égale à la somme des dates de fin.*

$D . (n - k . D) = k . D . (D - 1)$ $(\Rightarrow n = (2 . D - 1) . k)$ *et les* $(n - k . D)$ *dernières tâches disponibles à l'instant* 0 *comblent exactement les* $k . D$ *intervalles de longueur* $D - 1$ *laissés dans* 0^M *devant les premières tâches.*

$T_M - T_m = \sum_i c^M_i - \sum_i c^m_i$ *est exactement égale à la formule majorante calculée précédemment lors de la démonstration de la première partie.*

Ainsi pour l'exemple numérique de la figure 3.3.5.3. g *on a :*

$$T_M - T_m = (2 . 4 - 1) . \frac{4 . 5}{2} + (2 . 4 - 1) . 4 . 3 + \frac{3 . 4}{2} . 4 - \frac{7 . 8}{2} . 4$$

$$= 70 + 84 + 24 - 112 = 66$$

et $$\frac{n}{2} . \frac{D . (D - 1)}{2 . D - 1} . \left[\frac{n . (3 . D - 2)}{2 . D - 1} + 1 \right] =$$

$$\frac{7}{2} . \frac{4 . 3}{7} \left[\frac{7 . (3 . 4 - 2)}{7} + 1 \right] = 66$$

3.3.5.4. Erreur maximale dans le sous-ensemble des ordonnancements u-actifs sans délai

Un ordonnancement est u-actif si :

$\forall i, \forall j$ tels que $c_i < c_j$, on a : $(d_i \leq d_j)$ ou $(c_i - D < r_j)$

Un ordonnancement est sans délai si la machine n'est jamais laissée inutilisée alors qu'il existe une tâche exécutable.

Un ordonnancement est u-actif et sans délai si on ne laisse jamais la machine inoccupée alors qu'il existe une tâche exécutable et si parmi les tâches exécutables on choisit la plus urgente de manière à respecter la propriété u. On retrouve la règle de Jackson déjà rencontrée au paragraphe 3.2.2. dans le cas plus particulier des tâches unitaires où il était optimal.

L'erreur maximale que l'on peut commettre en appliquant la règle de Jackson dans le cas des durées égales est formulée par le théorème 3.3.5.4.

Théorème 3.3.5.4.

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D, si **O** est un ordonnancement u-actif sans délai quelconque dont la somme des retards par rapport aux délais est égale à T, et si T* désigne la somme des retards de la solution minimale, on a :

$\Delta\rho = T - T^* \leq (n-1).(D-1)$

et cette valeur est une borne supérieure.

Démonstration

Comme celle d'autres théorèmes démontrés précédemment, elle procède par "encadrement" : elle utilise la construction d'ordonnancements, éventuellement non réalisables, qui sont meilleurs que l'ordonnancement initial et dont la somme des retards est au moins aussi bonne que la valeur optimale, puis on majore l'écart du critère entre les deux ordonnancements.

Dans une première partie, on démontre la formule dans le cas particulier où l'ordonnancement **0** *ne laisse jamais la machine inoccupée.*

Dans une deuxième partie, on montre que si l'ordonnancement **0** *laisse la machine inoccupée entre deux tâches, alors le problème de majoration de l'erreur se découpe en plusieurs sous-problèmes indépendants pour lesquels on peut appliquer la formule démontrée dans la première partie, afin d'obtenir la formule dans le cas général.*

Dans la troisième partie, on met en évidence deux ordonnancements **0** *(u-actif sans délai) et* **0*** *(u-actif avec délai) pour lesquels cette valeur majorante est atteinte.*

Première partie

On s'intéresse au cas où l'ordonnancement u-actif sans délai **0** *ne laisse jamais la machine inoccupée (toutes les tâches sont contiguës).*

Pour simplifier les notations, on suppose que les tâches sont numérotées dans l'ordre de leur position dans **0** : $\sigma = (1, 2, ..., n)$ *est la séquence des tâches de l'ordonnancement* **0** .

Comme l'ordonnancement est u-actif, on a :

$\forall i, \forall j : i < j \Rightarrow (d_i \leq d_j)$ ou $(c_i - D < r_j)$

Si $d_i \leq d_j$ *, alors on ne gagnerait rien à transposer* i *et* j *dans la séquence quel que soit* r_j *, puisqu'ici la séquence est contiguë.*

Si $d_i > d_j$ *, alors, on a* $c_i - D < r_j$ *, mais si* $c_i \leq r_j$ *, alors on ne gagne rien à transposer* i *et* j *dans la séquence puisque cela retarderait au moins* i *sans avancer* j .

Le seul moyen d'améliorer l'ordonnancement **0** *est donc d'échanger des tâches telles que :*

$d_i > d_j$ et $c_i - D < r_j < c_i$

Nous allons donc nous intéresser plus particulièrement aux tâches qui ne respectent pas EDD *et à leur voisinage.*

A partir de l'ordonnancement **0** *, nous définissons la suite* v *(unique) de la manière suivante :* v *est la suite finie croissante de* k *indices de tâches comprenant tous les indices tels que la tâche correspondante possède un délai strictement supérieur à celui de la tâche suivante :*

$1 \le v_1 < v_2 < \ldots < v_k < n$

$\forall_i,\ 1 \le i \le k :\ d_{vi} > d_{vi+1}$

Par convention, nous posons $v_0 = 0$ *et* $v_{k+1} = n$.

Par définition de v *, on a :*

$\forall i,\ 0 \le i \le k :\quad d_{v_i+1} \le d_{v_i+2} \le \ldots \le d_{v_{i+1}}$

c'est-à-dire qu'entre v_i *(exclu) et* v_{i+1} *(inclus), les tâches sont dans l'ordre* EDD *(sinon la suite* v *ne contiendrait pas tous les indices tels que* $(d_{v_i} > d_{v_i+1})$.

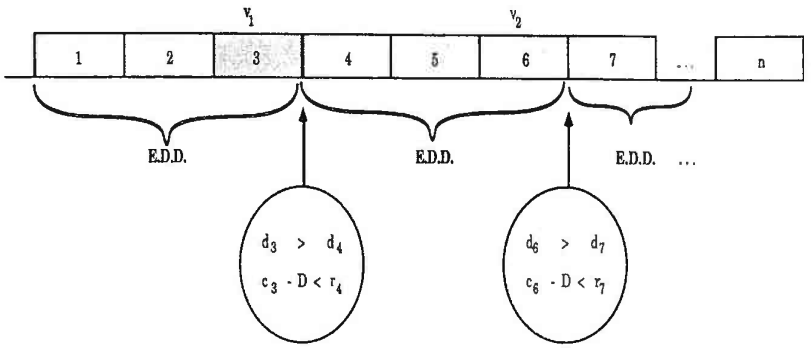
Ces propriétés de la suite v *sont illustrées sur la figure* 3.3.5.4. a .

On construit l'ordonnancement 0^m *(non réalisable) de la manière suivante (voir figure* 3.3.5.4.b *) :*

- *on maintient les tâches* $1, 2, \ldots, v_1$ *à leur emplacement initial.*
- *on avance les tâches* $v_1+1, \ldots, n$ *de* D - 1.

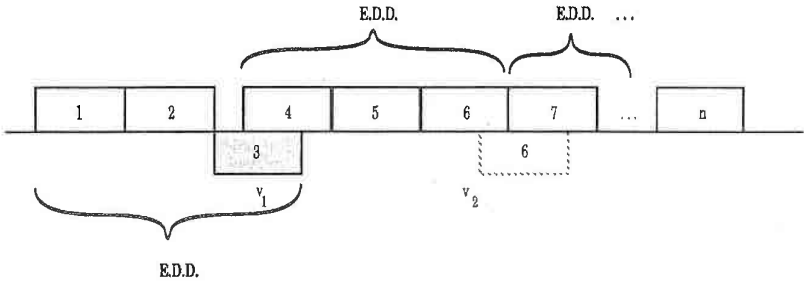
La somme des retards de l'ordonnancement 0^m *, notée* T_m *, est inférieure ou égale à la somme des retards de toute solution optimale, comme nous allons le montrer.*

En effet, pour tout i *compris entre* 0 *et* k *, considérons la sous-séquence* $\sigma_i = (v_i + 1,\ v_i + 2, \ldots, v_{i+1})$, *la séquence complète* σ *étant constituée de* $\sigma_0, \sigma_1, \ldots, \sigma_k$.



ordonnancement **0**

figure 3.3.5.4. a



ordonnancement 0^m (la fin de l'ordonnancement avance de D - 1)

figure 3.3.5.4. b

Nous avons déjà vu que σ_i *est dans l'ordre* EDD *et* σ_i *ne laisse jamais la machine inutilisée puisque cette suite de tâches était contigüe (hypothèse de la première partie) et a été avancée globalement de la même quantité* (0 *pour* i = 0 *et* D - 1 *pour* $i \neq 0$).

La seule manière permettant d'améliorer σ_i *serait d'avancer au moins une tâche* j *strictement avant le début de cette sous-séquence, c'est-à-dire strictement avant* $c_{vi} - D + 1$.

Examinons les différents cas possibles :

a) $r_j > c_{vi} - D$ *est sant intérêt puisqu'on ne pourrait pas avancer* j *strictement avant* $c_{vi} - D + 1$.

b) $r_j \leq c_{vi} - D$ et v_i avant j *entraîne* $d_{vi} \leq d_j$ *car sinon* j *aurait pris la place de* v_i *dans l'ordonnancement* **0** *u-actif. On a alors* $d_j \geq d_u \quad \forall u \in \sigma_{i-1}$ *et on ne gagnerait pas par échange avec les tâches de* σ_{i-1}.
Plus généralement :

$r_j \leq c_{vi'} - D$ avec $i' \leq i$ *entraîne :* $\forall u \in \sigma_{i'-1} : \; d_u \leq d_{vi'} \leq d_j$ *et on ne gagne pas par échange avec les tâches de* $\sigma_{i'-1}$.

On a donc $T^* \leq T^m \leq T$ *ou* $T - T^* \leq T - T^m$

Majorons donc $T - T^m$

$$T - T^m \leq \sum_{u\,:\,c_u > c^m_u} (c_u - c^m_u)$$

or

$\forall u \in \sigma_0 : \quad c_u = c^m_u$

$\forall u \notin \sigma_0 : \quad c_u = c^m_u + (D-1)$

$$T - Tm \leq \sum_{u \notin \sigma_0} (D-1) \leq (n-1)(D-1)$$

car σ_0 *contient au moins la tâche* v_i *si* $k \geq 1$ *et la séquence est optimale avec* $T = T^*$ *si* $k = 0$.

Deuxième partie

Supposons que la règle de Jackson fournisse un ordonnancement **0** *qui laisse* r *fois la machine inutilisée entre deux tâches. Cela signifie, puisque l'ordonnancement* **0** *est sans délai, qu'aucune tâche n'était en attente. Les* $r+1$ *sous-séquences de tâches contiguës constituent donc des ordonnancements indépendants pour lesquels on peut appliquer la formule de la première partie :*

$\forall k, \; 0 \leq k \leq r : \; T_{\sigma_k} - T^*_{\sigma_k} \leq (\text{card}(\sigma_k) - 1).(D-1)$

où $\text{card}(\sigma_k)$ *est le nombre de tâches contenues dans* σ_k *d'où :*

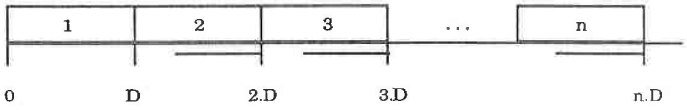
$$T - T^* = \sum_{k=0}^{r} (T_{\sigma_k} - T^*_{\sigma_k}) \leq (D-1).\left[\sum_{k=0}^{r} \text{card}(\sigma_k) - (r+1)\right] \leq (n-1).(D-1)$$

Troisième partie

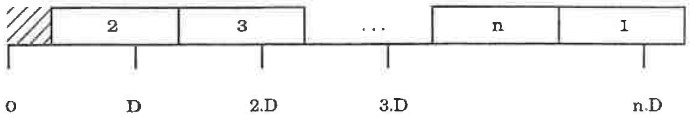
Ce majorant est atteint pour l'exemple de la figure 3.3.5.4. c .

i	1	2	3	...	n
r_i	0	1	2	...	n-1
d	n.D+1	D+1	2.D+1	...	(n-1).D+1

énoncé d'un problème d'ordonnancement



ordonnancement u-actif sans délai de retard (n-1).(D-1)



ordonnancement optimal de retard nul

figure 3.3.5.4. c

3.3.5.5. Erreur maximale dans le sous-ensemble des ordonnancements fournis par les méthodes de décomposition temporelle DTRU, DTRUM et DTRMU

Théorème 3.3.5.5.

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D, si 0 (resp. $0'$) est un ordonnancement u-actif fourni par une méthode de décomposition temporelle DTRU (resp. DTRUM) dont la somme des retards par rapport aux délais est égale à T (resp. T') et si T* désigne la somme des retards de la solution minimale, on a :

$$\Delta\rho = T - T^* \leq \frac{n^2}{4} \cdot \frac{D.(D-1)}{2.D-1}$$

et

$$\Delta\rho' = T' - T^* \leq \frac{(n-1/2)^2}{4} \cdot \frac{D.(D-1)}{2.D-1}$$

La première de ces deux valeurs peut être atteinte exactement et la deuxième asymptotiquement (car elle n'est jamais entière).

Démonstration

Nous allons d'abord obtenir les majorants, puis montrer qu'ils peuvent être atteints exactement ou asymptotiquement dans une deuxième partie.

Première partie

Nous utilisons certains des résultats déjà obtenus au cours des démonstrations des théorèmes précédents.

Tout d'abord, si nous démontrons que la formule est majorante pour un ordonnancement qui ne laisse jamais la machine inutilisée sur des intervalles de longueurs supérieures ou égales à D, alors la formule est vraie dans le cas général. En effet, les ordonnancements partiels entre deux intervalles sont actifs et leurs performances ou contre-performances sont indépendantes. On applique la formule à chaque ordonnancement partiel et on en déduit la formule dans le cas général car :

$(n_1)^2 + (n_2)^2 + \ldots + (n_k)^2 \leq (n_1 + n_2 + \ldots + n_k)^2$ *et*

$(n_1 - 1/2)^2 + \ldots + (n_k - 1/2)^2 = (n_1)^2 - n_1 + 1/4 + (n_2)^2 - n_2 + 1/4 + \ldots + (n_k)^2 - n_k + 1/4$

$\leq (n_1 + n_2 + \ldots + n_k - 1/2)^2 = (n_1 + n_2 + \ldots + n_k)2 - (n_1 + n_2 + \ldots + n_k) + 1/4)$

Ensuite, si l'erreur par rapport à la séquence optimale consiste à placer une tâche non urgente trop tôt, alors nous avons vu au cours de la démonstration du théorème 3.3.5.4. *que cela ne provoquait qu'une erreur proportionnelle à* n *et donc nettement inférieure à celles que l'on peut obtenir en laissant un intervalle inutilisé pour attendre l'arrivée d'une tâche urgente. Nous n'utilisons donc que ce deuxième type d'erreurs pour obtenir les "pires cas".*

Par ailleurs, plus ce deuxième type d'erreurs se produit tôt, plus les conséquences sont importantes sur les erreurs commises sur l'ensemble des autres tâches. Les ordonnancements 0^M *et* 0^m, *comme ceux des figures* 3.3.5.3. b *et* c, *qui comportent en tête* λ *tâches précédées d'intervalles de durée* D - 1 *où la machine est inutilisée, suivies de* $(n - \lambda)$ *tâches contigües, donnent la forme des "pires cas" possibles pour tous les ordonnnancements u-actifs avec délais. Le choix de la valeur* λ *qui maximise l'erreur commise dépend de la famille d'ordonnancements u-actifs avec délais considérée.*

Comme au paragraphe 3.3.5.3., *on a toujours :*

$T - T^* \leq T_M - T_m \leq \sum_i c^M_i - \sum_i c^m_i$ *mais ce majorant est trop grand,*

car avancer une tâche strictement avant son délai n'améliore plus le critère "somme des retards" et cette situation va se produire dans ce paragraphe.

Nous allons examiner les conditions nécessaires sur les délais des tâches pour qu'une méthode de décomposition temporelle fournissant des ordonnancements u-actifs puissent proposer la solution 0^M *(alors que l'optimum est la solution* 0^m*) tout en maximisant l'écart sur le critère "somme des retards".*

Soient

J_1 *l'ensemble des indices des* k_1 *premières tâches* $(k_1 = \lambda)$ *qui créent dans* 0^M *des intervalles où la machine est inutilisée.*

J_2 *l'ensemble des indices des* k_2 *tâches qui sont repoussées, au cours des premières itérations de la méthode de décomposition temporelle, par les tâches de* J_1 *. Elles arrivent au plus tard lors de l'itération qui place la* k_1 ème *tâche de* J_1 $(r_{j_2} < k_1 . (2 . D - 1))$.

J_3 *l'ensemble des indices des* k_3 *autres tâches* $(r_{j_3} \geq k_1 . (2 . D - 1))$.

Considérons l'ordonnancement partiel u-actif (conduisant à 0^M *) juste avant l'itération qui introduit la première tâche de* J_3 *. Les* k_1 *tâches de* J_1 *sont suivies des* k_2 *tâches de* J_2 *calées à gauche et classées dans l'ordre* EDD. *Pour que* 0^m *puisse être réalisable, on suppose qu'il existe des tâches de* J_2 *susceptibles de venir combler complètement les intervalles où la machine est inutilisée dans* 0^M.

Supposons qu'une tâche de J_2 *(qui est susceptible de combler un intervalle) soit en retard dans l'ordonnancement partiel, alors l'avancée de cette tâche dans l'intervalle fait gagner au minimum* $(2 . D - 1)$ *sur le critère "somme des retards" et fait perdre au plus* 1 *pour la tâche de* J_1 *retardée. Donc toute tâche de* J_2 *qui est susceptible de venir combler des intervalles ne doit pas être en retard dans l'ordonnancement partiel.*

Et au plus une tâche de J_2 *telle que* $r_{j_2} \geq k_1 (2 . D - 1) - D - 1$ *peut être en retard de au plus une unité de temps, pour qu'on puisse obtenir l'ordonnancement partiel et cela à condition que la tâche de* J_1 *reportée par l'avancée d'une tâche de* J_2 *prenne du retard.*

Donc, si on numérote les tâches dans l'ordre de l'ordonnancement partiel, on a :

$\forall i,\ k_1 < i \leq k_1 + k_2 :\quad k_1 . (D - 1) + i . D \leq d_i$

(avec éventuellement une exception où on remplace dans le deuxième membre de la formule d_i *par* $d_i + 1$ *mais alors* $\exists j_1 :\ d_{j1} \leq c^M_{j1}$)

Pour que les tâches de J_2 *participent au maximum à la valeur du critère "somme des retards", lors de l'introduction ultérieure des tâches de* J_3 *, nous leur choisissons le délai minimal autorisé pour atteindre l'ordonnancement partiel précédent :* $d_i = k_1 . (D - 1) + i . D$.

Considérons maintenant les tâches de J_3 *: elles viennent s'interclasser avec les tâches de* J_2 *lors des itérations ultérieures. Elles participent au maximum à l'écart sur la somme des retards entre* 0^M *et* 0^m *si elles ne sont en avance ni dans* 0^M *, ni dans* 0^m *. On leur attribue par exemple un délai de* 0 *. Pour pouvoir les avancer au maximum dans* 0^m *, on leur attribue la plus petite date d'arrivée possible* $k_1 . (2 . D - 1)$.

Alors 0^M *est constitué de* J_1 *suivi de* J_3 *suivi de* J_2.

La seule information que nous n'avons pas encore fixée concerne les délais de J_1 *. Leur choix pour obtenir* 0^M *et pour maximiser l'erreur dépend de la méthode de décomposition temporelle choisie, et nous allons distinguer deux cas.*

A) Méthode de décomposition temporelle DTRU

Elle n'examine pas le "makespan" comme critère secondaire.

Alors, même s'il existe d'autres ordonnancements partiels de même somme des retards et de "makespan" inférieur, on peut trouver celui qui nous intéresse ici (J_1 *suivi de* J_2) *en prenant :*

$\forall j_1,\ d_{j_1} = k_1 . (2 . D - 1) + D$

(on a bien $\forall j_1, \forall j_2 :\quad d_{j1} \leq d_{j2}$
et la somme des retards de l'ordonnancement partiel est nulle et donc optimale).

Ce choix des d_{j_1} *permet de ne pas perdre sur l'écart du critère "somme des retards" pour les tâches* j_1 *qui sont reportées dans* 0^m *par rapport à* 0^M.

Pour calculer l'écart maximum correspondant, nous remplaçons 0^M (J_1 *suivi de* J_3 *suivi de* J_2) par 0^1 (J_1 *suivi de* J_2 *suivi de* J_3) *qui a même somme des retards (aucune tâche de* J_2 *et* J_3 *n'est strictement en avance, ni dans* 0^M, *ni dans* 0^1 *).*

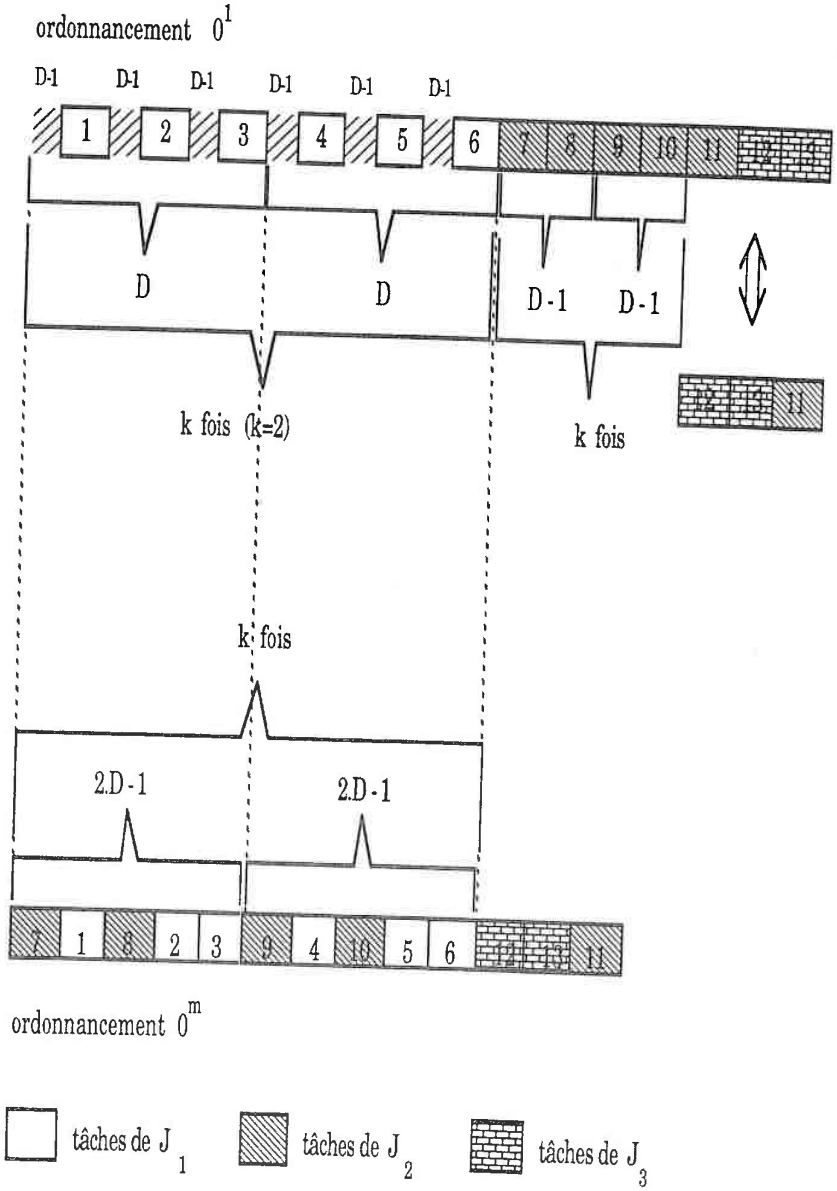


figure 3.3.5.5.

On obtient ainsi les ordonnancements 0^1 *et* 0^m *de la figure* 3.3.5.5.

Comme l'illustre la figure, mettre dans J_2 *une tâche qui n'est pas utilisée pour combler les intervalles est équivalent pour la somme des retards à mettre une tâche de plus dans* J_3. *Nous choisirons donc de combler exactement les intervalles :* $k_1 . (D - 1) = k_2 . D = k . D . (D - 1)$

Les tâches $J_1 \cup J_2$ *ne sont en retard ni dans* 0^1, *ni dans* 0^m *alors que celles de* J_3 *sont toujours en retard :*

$$T_M - T_m = T_1 - T_m = \sum_{i \in J_3} c^1{}_i - \sum_{i \in J_3} c^m{}_i$$

$$= \text{(durée totale des intervalles multipliée par } \text{card}(J_3)\text{)}$$

$$= k . D . (D - 1) . (n - k . (2 . D - 1))$$

$f(x) = x . D . (D - 1) . (n - x . (2 . D - 1))$ *est maximal pour* $x = \frac{n}{2 . (2 . D - 1)}$

et en donnant à k *cette valeur, on obtient :*

$$T_M - T_m = \frac{n^2}{4} \cdot \frac{D . (D - 1)}{2 . D - 1}$$

Cette valeur est bien atteinte si elle est entière, c'est-à-dire en choisissant :

$$n = k . 2 . (2 . D - 1)$$

d'où

$$k_1 = k . D = \frac{n . D}{2 . (2D - 1)}$$

$$k_2 = k . (D - 1) = \frac{n . (D - 1)}{2 . (2D - 1)}$$

$$k_3 = n - k (2D - 1) = \frac{n}{2}$$

B) Méthodes de décomposition temporelle DTRUM et DTRMU

Elles examinent le "makespan" comme critère secondaire.

Pour obtenir l'ordonnancement partiel J_1 *suivi de* J_2, *il faut que tous les autres ordonnancements qui ont un "makespan" strictement inférieur aient une somme des retards strictement supérieure : aucune tâche de* J_1 *ne peut être strictement en avance.*

Nous prenons par exemple : $\forall j_1$, $d_{j_1} = 0$.

Le calcul de l'écart maximal est le même que dans le cas précédent à une exception près : il faut retirer de cet écart le retard pris par les tâches de J_1. *Ce retard est minimal dans le cas de la figure* 3.3.5.5. *où, pour chacun des* k *groupes de tâches de* J_1 *et* J_2 *intercalés dans* 0^m :

on a *la première tâche de* J_1 *reportée de* 1 ,
la seconde *reportée de* 2 ,
...
la $(D-1)^{ème}$
et *la* $D^{ème}$ *reportée de* $(D-1)$,
qui n'est pas déplacée.

D'où :

$$T_M - T_m = T_1 - T_m = k \,.\, D \,.\, (D-1) \,.\, (n - k \,.\, (2\,.\,D-1)) - k \,.\, \frac{D\,.\,(D-1)}{2}$$

$f(x) = x \,.\, D \,.\, (D-1) \,.\, [n - x \,.\, (2\,.\,D-1) -, 1/2]$ est maximal pour $x = \frac{n - 1/2}{2\,.\,(2\,.\,D-1)}$

d'où

$$T_M - T_m \leq \frac{(n-1/2)^2}{4} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

mais cette valeur ne peut pas être atteinte car elle n'est pas entière.

En prenant $x = \frac{n-1/2}{2\,.\,(2\,.\,D-1)}$ *ou* $x = \frac{n}{2\,.\,(2\,.\,D-1)}$ *qui peuvent être des entiers, on obtient dans les deux cas :*

$$T_M - T_m = \frac{(n-1/2)^2}{4} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

qui est la valeur la plus grande que l'on puisse atteindre, elle diffère de la précédente de

$$\frac{1}{16} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

Et l'écart entre DTRUM *(ou* DTRMU *)* *et* DTRU *dans le pire des cas est*

$$\frac{n}{4} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

3.3.5.6. Conclusions et perspectives

Rappelons les formules trouvées dans le pire des cas :

pour les ordonnancements semi-actifs :

$$\max\,[\,n\,.\,\Delta r\,,\,n\,.\,(n-1)\,.\,D\,]$$

pour les ordonnancements actifs :

$$\frac{n^2}{4}\,.\,(2.D-1)$$

pour les ordonnancements u-actifs :

$$\frac{n}{2} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1} \,.\, [\,\frac{n\,.\,(3\,.\,D-2)}{2\,.\,D-1} + 1\,]$$

pour les ordonnancements u-actifs sans délai :

$$(n-1)\,.\,(D-1)$$

pour les ordonnancements fournis par DTRU :

$$\frac{n^2}{4} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

pour les ordonnancements fournis par DTRUM et DTRMU :

$$\frac{n\,(n-1)}{4} \,.\, \frac{D\,.\,(D-1)}{2\,.\,D-1}$$

Pour obtenir des ordres de grandeurs comparables, supposons que n soit grand par rapport à D , et D grand par rapport à 1 .

On obtient alors :

$(T - T^*)_{max}$ (semi-actifs) $\geq n^2 \,.\, D$

$(T - T^*)_{max}$ (actifs) $\geq \frac{n^2}{2} \,.\, D$

$(T - T^*)_{max}$ (u-actifs) $\# \frac{3}{8} \,.\, n^2 \,.\, D$

$(T - T^*)_{max}$ (DTRU...) $\# \frac{1}{8} \,.\, n^2 \,.\, D$

Ceci montre dans quelle mesure le fait de restreindre l'ensemble des ordonnancements sur lesquels on travaille restreint l'erreur maximale dans le pire des cas. Pour presque tous ces exemples, l'erreur est en n^2.

Les ordonnancements u-actifs sans délai sont, relativement à l'erreur dans le pire des cas, les plus intéressants puisque l'erreur est seulement multiple de n. Ceci s'explique par le fait suivant : placer trop tôt une tâche non urgente n'est pas un phénomène qui accumule les causes d'erreurs.

On peut considérer ce résultat comme un paradoxe.

En effet, la méthode de Jackson qui construit les ordonnancements u-actifs sans délai est une méthode où à chaque instant t, on ne remet pas en cause le passé et on prend une décision optimale en fonction des tâches arrivées au plus tard à l'instant t et non encore placées ; par contre, on ignore totalement les tâches qui vont arriver après l'instant t.

Dans les méthodes de décomposition temporelle, on prend également une décision localement optimale à l'instant t, sans remettre en cause le passé, mais en connaissant les tâches qui arrivent entre t et $t + (D - 1)$.

Une connaisance plus importante du problème peut donc conduire dans le pire des cas à des résultats moins bons.

La suite de théorèmes que nous venons de démontrer pour le critère "somme des retards", qui nous intéresse plus particulièrement dans le cadre de ce travail, pourrait être reprise pour d'autres critères.

En suivant d'autres voies pour essayer de démontrer les théorèmes prédédents, nous avons trouvé (et démontré) en particulier que :

$$\Delta M \text{ (actifs)} \leq n \cdot \frac{D \cdot (D-1)}{2 \cdot D - 1}$$

et

$$\Delta M \text{ (actifs)} \leq \max \left(E\left(\frac{n}{2}\right) \cdot (D-1), E\left(\frac{n-1}{2}\right) \cdot D \right)$$

où E désigne la partie entière d'un réel, et ces valeurs majorantes peuvent toutes les deux être atteintes.

3.4. AUTRES CAS PARTICULIERS

3.4.1. Présentation

Au paragraphe 3.2.1., nous avons rappelé un résultat connu ([H. EMMONS 1969]) : si les séquences SPT et EDD sont identiques et si toutes les tâches sont disponibles à l'instant 0 , alors elles minimisent la somme des retards.

Nous proposons une généralisation de ce résultat pour un cas particulier où les tâches ne sont pas toutes disponibles à l'instant 0 ; nous analysons alors le comportement des méthodes de décomposition temporelle

Nous avons déjà défini (cf. 3.3.3.) les séquences EDD/FIFO , nous utilisons ici les séquences FIFO ou "first in, first out" qui sont les séquences où les tâches sont ordonnancées par dates d'arrivée r_i croissantes et les séquences LIFO ou "last in, first out" qui classent les tâches par dates d'arrivée r_i décroissantes.

Nous allons montrer au paragraphe 3.4.2. que, si les séquences SPT , EDD et FIFO sont identiques, elles minimisent plusieurs critères dont le critère "somme des retards" et que la séquence inverse (LPT, LDD et LIFO) maximise alors ces critères. Des corollaires de ce théorème seront présentés.

Au paragraphe 3.4.3., nous examinerons, pour ce même cas particulier, le déroulement des méthodes de décomposition temporelle de type DTRUM (cf. 3.1.2.) et montrerons qu'elles sont optimales et polynômiales.

3.4.2. Les séquences SPT, EDD et FIFO sont identiques

A) Définitions

Nous utilisons les notations définies au paragraphe 3.1.1. . En outre, pour toute séquence σ qui définit un ordonnancement semi-actif, $0(\sigma)$, nous notons :

$\sigma = (i_1, i_2, \ldots, i_n)$

T_σ la somme des retards : $T_\sigma = \sum_{k=1}^{n} \rho_k$

M_σ le "Makespan" ou durée totale : $M_\sigma = \max_k [c_k]$

C_σ la somme des dates d'achèvement des tâches : $C_\sigma = \sum_{k=1}^{n} c_k$

F_σ la somme des durées des temps de présence des tâches (ou "flow time") :

$$F_\sigma = \sum_{k=1}^{n} f_k = \sum_{k=1}^{n} [c_k - r_k] = \sum_{k=1}^{n} c_k - \sum_{k=1}^{n} r_k = C_\sigma - R_\sigma$$

où $R_\sigma = \sum_{k=1}^{n} r_k$

MR_σ le plus grand retard : $MR_\sigma = \max_k \rho_k$

Comme R_σ est une constante pour un problème d'ordonnancement donné, minimiser (ou maximiser) F_σ est analogue à minimiser (ou maximiser) C_σ ; nous ne considérons que l'un ou l'autre de ces deux critères dans les démonstrations qui suivent.

B) Théorèmes

Théorème 3.4.2. a

Si les séquences EDD , SPT et FIFO sont identiques, elles définissent sur une machine un ordonnancement semi-actif qui minimise :

- la somme des retards,
- le plus grand retard,
- la durée totale,
- la somme des durées des temps de présence des tâches (ou la somme des dates d'achèvement).

Théorème 3.4.2. b

Si les séquences EDD , SPT et FIFO sont identiques, la séquence inverse LDD , LPT et LIFO définit sur une machine un ordonnancement semi-actif qui maximise :

- la somme des retards,
- le plus grand retard,
- la durée totale
- la somme des durées des temps de présence des tâches (ou la somme des dates d'achèvement).

Démonstration

Nous allons démontrer simultanément ces deux théorèmes complémentaires. La technique de démonstration utilisée est connue sous le nom "d'échange de paires de tâches consécutives".

Soient :

σ_{ij} *une séquence où la tâche* i *précède immédiatement la tâche* j ,

σ_{ji} *la séquence obtenue à partir de* σ_{ij} *en inversant l'ordre des deux tâches* i *et* j ,

Z *l'ensemble des indices des tâches,*

U *l'ensemble des indices des tâches qui précèdent* (i,j) *dans* σ_{ij} *et dans* σ_{ji} *dont l'ordonnancement ne change pas,*

V *l'ensemble des indices des tâches qui suivent* (i,j) *dans* σ_{ij} *et dans* σ_{ji} *dont l'ordre ne change pas,*

W *la réunion des ensembles* U *et* V ,

q *l'instant de fin de la dernière tâche de* V,

c^{ij}_z *et* c^{ji}_z *les instants de fin de toute tâche* z *dans les séquences* σ_{ij} *et* σ_{ji},

p^{ij}_z *et* p^{ji}_z *les retards respectifs de toute tâche* z *dans les séquences* σ_{ij} *et* σ_{ji}.

Pour simplifier, on écrira M_{ij} , M_{ji} , T_{ij} , T_{ji} , C_{ij} , C_{ji} , MR_{ij} *et* MR_{ji} *à la place de* $M_{\sigma_{ij}}$, $M_{\sigma_{ji}}$, $T_{\sigma_{ij}}$, $T_{\sigma_{ji}}$, $C_{\sigma_{ij}}$, $C_{\sigma_{ji}}$, $MR_{\sigma_{ij}}$, $MR_{\sigma_{ji}}$.

Par hypothèse, les séquences EDD , SPT *et* FIFO *coïncident et, toujours pour simplifier, nous supposons que les tâches ont été numérotées selon l'ordre de ces séquences identiques :*

EDD = SPT = FIFO = (1 , 2 , ... , n)

On a :

$d_1 \le d_2 \le \ldots \le d_n$ d'après EDD (délais croissants)

$p_1 \le p_2 \le \ldots \le p_n$ d'après SPT (durées opératoires croissantes)

$r_1 \le r_2 \le \ldots \le r_n$ d'après FIFO (dates de disponibilité croissantes)

Et pour deux tâches quelconques x *et* y *on a :*

$r_x \geq r_y$ et $p_x \geq p_y$ et $d_x \geq d_y$

ou bien

$r_x \leq r_y$ et $p_x \leq r_y$ et $d_x \leq d_y$

Nous supposons de manière arbitraire que $r_i \geq r_j$ et $p_i \geq p_j$ et $d_i \geq d_j$ *et nous allons montrer qu'alors on a* $M_{ij} \geq M_{ji}$, $T_{ij} \geq T_{ji}$ et $C_{ij} \geq C_{ji}$. *Nous en déduirons les deux théorèmes.*

Dans un premier temps, nous allons montrer que dans tous les cas on a :
$\sigma^{ji}_i \leq c^{ij}_j$, $\sigma^{ji}_j \leq c^{ij}_i$ et $\sigma^{ji}_j \leq c^{ij}_j$
(il est évident par ailleurs que $c^{ij}_i < c^{ij}_j$ et $\sigma^{ji}_j < \sigma^{ji}_i$*)*

Puis nous distinguerons des cas en fonction des délais pour les critères utilisant le retard des tâches.

Etude comparative des dates d'achèvement des séquences σ_{ij} et σ_{ji}

a) $q \geq r_i$ (avec $r_i \geq r_j \Rightarrow q \geq r_j$)

Ce cas est illustré par la figure 3.4.2.a .

On a : $\sigma^{ji}_i = q + p_i + p_j = c^{ij}_j$

$\sigma^{ji}_j = q + p_j \leq q + p_i = c^{ij}_i$

et $\sigma^{ji}_j = q + p_j < q + p_j + p_i = c^{ij}_j$

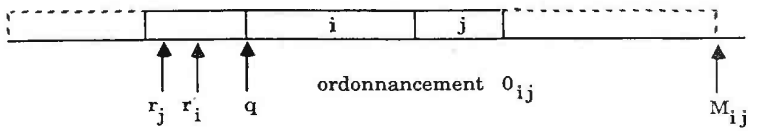
b) $q < r_i$

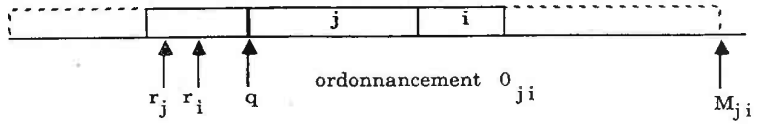
C'est le cas où la machine est laissée inutilisée avant la tâche i *comme le montrent les figures* 3.4.2. a *et* b .

On a ici :

$c^{ij}_i = r_i + p_i$, $\quad c^{ij}_j = r_i + p_i + p_j$,

$\sigma^{ji}_j = \max [q, r_j] + p_i$, $\quad \sigma^{ji}_i = \max [q + p_j, r_j + p_j, r_i] + p_i$





$q \geq r_i \geq r_j$, $p_i \geq p_j$, $d_i \geq d_j$

figure 3.4.2. a

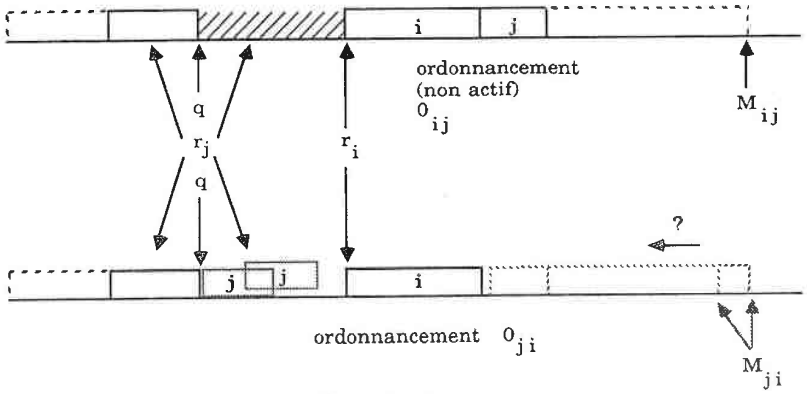


figure 3.4.2. b

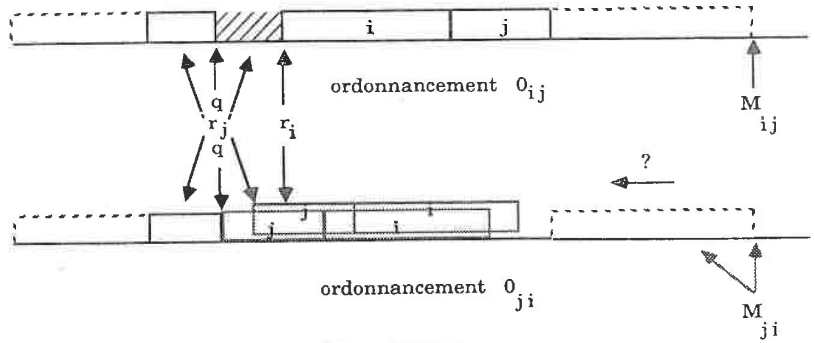


figure 3.4.2. c

$$\max [q, r_j] + p_j \le r_i + p_j < r_i + p_i + p_j \Rightarrow c^{ji}_j \le c^{ij}_j$$

$$\left.\begin{array}{l} q + p_j + p_i < r_i + p_i + p_j \\ r_j + p_j + p_i \le r_i + p_i + p_j \\ r_i + p_i < r_i + p_i + p_j \end{array}\right\} \Rightarrow c^{ji}_i \le c^{ij}_j$$

$$\left.\begin{array}{l} q + p_j < r_i + p_j \le r_i + p_i \\ r_j + p_j \le r_i + p_i \end{array}\right\} \Rightarrow c^{ji}_j \le c^{ij}_i$$

Pour les autres tâches, on a :

$\forall u \in U : \quad c^{ij}_u = \omega^{ji}_u$

et comme l'ordre des tâches de V *est le même dans* 0_{ji} *et dans* 0_{ji} *, que ces ordonnancements sont semi-actifs (c'est-à-dire calés à gauche) et que l'on a* ω^{ji}_i = fin de (i,j) dans $0_{ji} \le c^{ij}_j$ = fin de (i,j) dans 0_{ij}, *on a :*

$\forall v \in V : \quad cji_v \le c^{ij}_v$

Etude comparative des critères qui n'utilisent pas la notion de retard

$$M_{ij} = \max_{z \in Z} [c^{ij}_z] = \max [\max_{v \in V} [c^{ij}_v], c^{ij}_j] \ge \max [\max_{v \in V} [\omega^{ji}_v], \omega^{ji}_i] = \max_{z \in Z} [\omega^{ji}_z] = M_{ji}$$

$$C_{ij} = \sum_{z \in Z} c^{ij}_z = \sum_{\substack{z \ne i \\ z \ne j}} c^{ij}_z + c^{ij}_i + c^{ij}_j \ge \sum_{\substack{z \ne i \\ z \ne j}} \omega^{ji}_z + \omega^{ji}_j + \omega^{ji}_i = \sum_{z \in Z} \omega^{ji}_z = C_{ji}$$

On a donc $M_{ij} \ge M_{ji}$, $C_{ij} \ge C_{ji}$ (et $F_{ij} \ge F_{ji}$) .

Etude comparative des critères qui utilisent la notion de retard

$$T_{ij} = \sum_{\substack{k \ne i \\ k \ne j}} \rho^{ij}_k + \rho^{ij}_i + \rho^{ij}_j \quad \text{et} \quad T_{ji} = \sum_{\substack{k \ne i \\ k \ne j}} \rho^{ji}_k + \rho^{ji}_j + \rho^{ji}_i$$

$$MR_{ij} = \max [\max_{\substack{k \ne i \\ k \ne j}} [\rho^{ij}_k], \rho^{ij}_i, \rho^{ij}_j] \quad \text{et} \quad MR_{ji} = \max [\max_{\substack{k \ne i \\ k \ne j}} [\rho^{ij}_k], \rho^{ji}_j, \rho^{ji}_i]$$

$\forall k, \ k \ne i \ \text{et} \ k \ne j : \quad c^{ij}_k \ge \omega^{ji}_k \Rightarrow \rho^{ij}_k \ge \rho^{ji}_k$

d'où

$$\sum_{\substack{k \ne i \\ k \ne j}} \rho^{ij}_k \ge \sum_{\substack{k \ne i \\ k \ne j}} \rho^{ji}_k \quad \text{et} \quad \max_{\substack{k \ne i \\ k \ne j}} [\rho^{ij}_k] \ge \max_{\substack{k \ne i \\ k \ne j}} [\rho^{ji}_k]$$

Soient $\rho_1 = \rho^{ij}_i + \rho^{ij}_j$, $\rho_2 = \rho^{ji}_j + \rho^{ji}_i$, $\rho_3 = \max [\rho^{ij}_i , \rho^{ij}_j]$ *et*

$\rho_4 = \max [\rho^{ji}_j, \rho^{ji}_i]$

On aura $T_{ij} \ge T_{ji}$ si $\rho_1 \ge \rho_2$

et $MR_{ij} \ge MR_{ji}$ si $\rho_3 \ge \rho_4$

Nous allons démontrer que $\omega^{ji}_i \le c^{ij}_j$, $\omega^{ji}_j \le c^{ij}_i$ et $\omega^{ji}_j \le c^{ij}_j$ entraîne $\rho_1 \ge \rho_2$ et $\rho_3 \ge \rho_4$ *en examinant tous les cas possibles.*

$\rho^{ij}_i = \max [0, c^{ij}_i - d_i]$ $\qquad$ $\rho^{ij}_j = \max [0, c^{ij}_j - d_j]$

$\rho^{ji}_j = \max [0, c^{ji}_j - d_j]$ $\qquad$ $\rho^{ji}_i = \max [0, c^{ji}_i - d_i]$

$c^{ij}_j \ge \omega^{ji}_j \quad \Rightarrow \quad \rho^{ij}_j \ge \rho^{ji}_j$

α) $d_i \ge c^{ij}_j$ $\quad (\Rightarrow d_i > c^{ij}_i \ \text{et} \ d_i \ge \omega^{ji}_i)$

$\rho^{ij}_i = \rho^{ji}_i = 0$

$\rho_3 = \rho_1 = \rho^{ij}_j \ge \rho^{ji}_j = \rho_2 = \rho_4$

β) $d_i < c^{ij}_j$

i) $d_j \le \omega^{ji}_j$ $\quad (\Rightarrow d_i \le c^{ij}_j)$

$$\begin{aligned} \rho_1 - \rho_2 &= \max [0, c^{ij}_i - d_i] + (c^{ij}_j - d_j) - (\omega^{ji}_j - d_j) - \max [0, \omega^{ji}_i - d_i] \\ &= \max [-\omega^{ji}_j, c^{ij}_i - \omega^{ji}_j - d_i] - \max [-c^{ij}_j, \omega^{ji}_i - c^{ij}_j - d_i] \\ &\ge \max [-\omega^{ji}_j, - d_i] - \max [-c^{ij}_j, - d_i] \\ &= \min [c^{ij}_j, d_i] - \min [\omega^{ji}_j, d_i] \ \ge \ 0 \end{aligned}$$

$\rho_3 = \max [\max [0, c^{ij}_i - d_i], c^{ij}_j - d_j]$

$\rho_4 = \max [\omega^{ji}_j - d_j, \max [0, \omega^{ji}_i - d_i]]$

$c^{ij}_j - d_j \geq o^{ji}_j - d_j$ et $c^{ij}_j - d_j \geq 0$ et $c^{ij}_j - d_j \geq o^{ji}_i - d_i$ $\Rightarrow$ $\rho_3 \geq \rho_4$

ii) $d_j > o^{ji}_j$ (avec $d_i \geq d_j$ $\Rightarrow c^{ji}_j < d_j \leq d_i < c^{ij}_j$)

$\rho_1 - \rho_2 = \max [0, c^{ij}_i - d_i] + (c^{ij}_j - d_j) - 0 - (o^{ji}_i - d_i)$

$= \max [0, c^{ij}_i - d_i] + (c^{ij}_j - o^{ji}_i) + (d_i - d_j) \geq 0$

$\rho_3 = \max [\max [0, c^{ij}_i - d_i], c^{ij}_j - d_j]$

$\rho_4 = o^{ji}_i - d_i$

et $c^{ij}_j - d_j \geq o^{ji}_i - d_i$ $\Rightarrow$ $\rho_3 \geq \rho_4$

Ceci termine la démonstration que $T_{ij} \geq T_{ji}$ *et* $MR_{ij} \geq MR_{ji}$.

Nous avons donc :

$(r_i \geq r_j$, $p_i \geq p_j$ et $d_i \geq d_j)$ $\Rightarrow$

$(M_{ij} \geq M_{ji}$, $C_{ij} \geq C_{ji}$, $F_{ij} \geq F_{ji}$, $T_{ij} \geq T_{ji}$ et $MR_{ij} \geq MR_{ji})$

Quand les séquences EDD , SPT *et* FIFO *coïncident, on ne peut pas dégrader l'un quelconque des critères en inversant dans une séquence deux tâches qui ne vérifient pas l'ordre commun à ces trois séquences.*

Donc si σ_{min} *(respectivement* σ_{max} *) est une séquence qui minimise (resp. maximise) l'un des critères donnés, et si* $\sigma^1_{min}, \sigma^2_{min} \ldots$ *(resp.* $\sigma^1_{max}, \sigma^2_{max} \ldots$*) sont les suites de séquences obtenues en inversant à chaque fois les tâches, d'une paire de tâches consécutives qui ne respectent pas l'ordre commun (respectivement l'ordre inverse), ces suites convergent vers l'ordre commun (resp. inverse) qui est donc minimal (resp. maximal).*

Ceci termine la démonstration des théorèmes 3.4.2.a *et* 3.4.2.b .

C) Corollaires et théorème auxiliaire

En corollaires du théorème 3.4.2.a, on peut retrouver des résultats connus :

Celui de H. EMMONS ([1969]) en prenant les dates de disponibilité toutes identiques :

> Si $\forall i$, $r_i = 0$ et si les séquences EDD et SPT coïncident, alors elles minimisent la somme des retards.

On obtient un autre cas particulier en prenant pour les délais des valeurs égales :

> Si $\forall i$, $d_i = d$ et si les séquences FIFO et EDD coïncident, alors elles minimisent la somme des retards.

En prenant en outre les dates de disponibilités toutes identiques, on obtient un résultat cité dans K. R. BAKER ([1974]) :

> Si $\forall i$, $r_i = 0$ et $d_i = d$ alors la séquence SPT minimise la somme des retards.

En prenant les durées égales, on obtient un résultat déjà obtenu au paragraphe 3.3. :

> Si $\forall i$, $p_i = D$ et si les séquences FIFO et EDD coïncident, alors elles minimisent la somme des retards.

On peut également obtenir un autre résultat cité dans K. R. BAKER ([1974]) :

> Si $\forall i$, $r_i = 0$ et si quelle que soit la séquence, toutes les tâches sont en retard, alors la séquence SPT minimise la somme des retards.

On obtient ce résultat en construisant un nouveau problème par modification des délais : $\forall i$, $r'i = r_i = 0$, $p'_i = p_i$ et $d'_i = 0$ $(= d_i - d_i)$

Pour ce nouveau problème, les séquences EDD, FIFO et SPT *coïncident, et en particulier* SPT *minimise la somme des retards, on a, si toutes les séquences sont en retard et les* $d_i \geq 0$:

$T = T' + \sum_i d_i$ *où* $\sum_i d_i$ *est une constante.*

Donc T *et* T' *sont minimuax simultanément pour la séquence* SPT.

Nous présentons un autre corollaire du théorème 3.4.2. et un théorème auxiliaire.

Corollaire 3.4.2.

Si les séquences EDD et SPT sont identiques et si elles définissent sur une machine un ordonnancement semi-actif qui ne laisse jamais la machine inutilisée entre les tâches, alors cet ordonnancement minimise :
- la somme des retards,
- le plus grand retard,
- la durée totale,

et la somme des durées des temps de présence (ou la somme des dates d'achèvement).

Démonstration

Soit $\pi = (\{r_i\}, \{p_i\}, \{d_i\})$ *l'énoncé du problème* π *considéré et soit* $\pi' = (\{r'_i\}, \{p'_i\}, \{d'_i\})$ *l'énoncé d'un nouveau problème* π' *obtenu en posant :*

$\forall_i \in W, \quad r'_i = 0, \quad p'_i = p_i, \quad d'_i = d_i$

On a : $\forall_i \in W$, $r'_i \leq r_i$ *et donc, toute solution réalisable pour* π *l'est également pour* π' *: l'ensemble des solutions de* π' *contient l'ensemble des solutions de* π *et la meilleure solution de* π' *est au moins aussi bonne que la meilleure solution de* π *pour un critère quelconque à minimiser noté* CR : *si* CR* *(resp.* CR'* *)* *désigne la meilleure valeur du critère quelconque* CR *pour le problème* π *(resp.* π' *), on a :* $CR'^* \leq CR^*$.

Par ailleurs, pour le problème π' *, les séquences* EDD, SPT *et* FIFO *coïncident et, en notant* CR *la valeur d'un quelconque des critères considérés pour cette séquence commune on a, d'après le théorème* 3.4.2.a : $CR = CR'^*$

D'où $CR = CR'^* \leq CR^*$, *ce qui montre l'optimalité des séquences* EDD *et* SPT *qui coïncident pour le problème* π *sans que* FIFO *ne crée d'intervalles inutilisés.*

Théorème 3.4.2.c

Si les séquences EDD , SPT et FIFO sont identiques, toute séquence σ qui définit un ordonnancement semi-actif **0** qui minimise simultanément
- la somme des retards,
- la durée totale,
- la somme des durées des temps de présence,

a le même ensemble de dates de début et de fin de tâches que la séquence vérifiant EDD , SPT et FIFO.

Démonstration

Il s'agit ici d'une démonstration par l'absurde.

Nous notons σ^* *la séquence identique respectant simultanément les ordres* EDD , SPT et FIFO *et nous marquons d'une étoile toutes les entités relatives à cette séquence.*

Nous supposons que la séquence σ *optimise les critères considérés et a au moins une date de début ou de fin de tâche strictement différente de celles de* σ^* .

$\sigma = (i_1, i_2, ..., i_n)$ et $\sigma^* = (i^*_1, i^*_2, ... i^*_n)$

Soit k *le plus petit entier tel que* i_k *et* i^*_k *n'ont pas des dates de début et de fin identiques. On a :*

$\forall j < k$: $c_{i_j} = c^*_{i^*_j}$ et $c_{i_j} - p_{i_j} = c^*_{i^*_j} - p^*_{i^*_j}$

et $c_{i_k} \neq c^*_{i^*_k}$ ou $c_{i_k} - p_{i_k} \neq c^*_{i^*_k} - p^*_{i^*_k}$

Nous allons distinguer les quatre cas possibles et montrer qu'ils sont incompatibles avec nos hypothèses.

a) $c_{i_k} - p_{i_k} < c^*_{i^*_k} - p^*_{i^*_k}$

Comme les dates de début et de fin sont identiques, cela signifie que, dans la séquence σ^* *, la machine est laissée inutilisée entre la fin de la* $(k-1)^{ème}$ *tâche (instant* $c^*_{i^*_{k-1}} = c_{i_{k-1}} \leq c_{i_k} - p_{i_k}$*) et le début de la* $k^{ème}$ *(instant* $c^*_{i^*_k} - p^*_{i^*_k}$*) alors que d'après la séquence* σ *, on sait qu'une* $k^{ème}$ *tâche est disponible à l'instant* $c_{i_k} - p_{i_k}$ *strictement antérieur à* $c^*_{i^*_k} - p^*_{i^*_k}$ *. Cela signifierait que la séquence* σ^* *ne respecterait pas la séquence* FIFO.

b) $c_{i_k} - p_{i_k} > c^*_{i^*_k} - p^*_{i^*_k}$

Dans ce cas, la machine est laissée inutilisée dans la séquence σ *entre les instants* $c_{i_{k-1}}$ $(= c^*_{i^*_{k-1}})$ *et* r_{i_k} $(= c_{i_k} - p_{i_k})$

Nous construisons un nouvel ordonnancement $\mathbf{0}'$ *en conservant le début de l'ordonnancement* $\mathbf{0}$, *correspondant à la séquence* σ *jusqu'à la tâche* i_k-1, *et en réordonnançant dans l'ordre commun à* EDD, FIFO et SPT *les tâches restantes à partie de l'instant* r_{ik}. *D'après le théorème* 3.4.2.a *appliqué à partir de l'instant* r_{ik}, *la fin de l'ordonnancement* $\mathbf{0}'$ *est au moins aussi bonne pour les critères considérés que la fin de* $\mathbf{0}$ *et cette propriété est vraie pour les ordonnancements complets puisque le début reste inchangé (critères de type "additif" ou concernant la dernière tâche).*

$\sigma' = (i_1, i_2, \ldots, i_{k-1}, i'_k, i'_{k+1}, \ldots, i'_n)$

Soit r *le plus petit entier* $(\geq k)$ *tel que* i'_r *et* i^*_r *n'ont pas des dates de début et de fin identiques.*

On a : $r_{i_r} < r_{i_k}$ *(en effet* i_r *est une des tâches de date de disponibilité minimale de la fin de séquence, qui contient au moins une tâche* j *telle que* $r_j \leq r_{i^*_k}$ *d'après la comparaison entre* σ *et* σ^*).

En avançant la tâche i_r *de* r_{i_k} *à* r_{i_r} *on ne dégrade aucun critère et on améliore strictement le critère somme des dates de fin de tâches.*
Ceci est en contradiction avec l'optimalité de la séquence σ.

c) $c_{i_k} - p_{i_k} = c^*_{i^*_k} - p^*_{i^*_k}$

α) $c_{i_k} < c^*_{i^*_k}$

Cela implique $p^*_{i^*_k} > p_{i_k}$

Ce cas est impossible, car les durées des (k-1) *premières tâches étant identiques, lors du placement de la* kème, σ^* *placerait une tâche de durée* $p^*_{i^*_k}$ *alors qu'une tâche de durée strictement inférieure* p_{i_k} *est disponible ; cela contredit l'ordre* SPT.

β) $c_{i_k} > c^*_{i^*_k}$

Comme au b), *on fabrique* $\mathbf{0}'$ *à partir de* $\mathbf{0}$ *en réordonnançant les* (n-k+1) *dernières tâches dans l'ordre commun à* FIFO, SPT et EDD. *Aucun critère n'est dégradé et le critère "somme des dates de fin" est amélioré strictement.*
Pour le démontrer, on considère deux tâches consécutives qui ne respectent pas SPT :

$p_r > p_{r+1}$ ($\Rightarrow r_r \geq r_{r+1}$ *selon nos hypothèses*)

$c_{r+1} = c_r + p_{r+1} \Rightarrow c_r + c_{r+1} = 2.c_{r+1} - p_{r+1}$

en les échangeant simplement sur l'intervalle $[c_r - p_r, c_{r+1}]$ *on obtient de nouvelles dates de fin :* c''_r *et* c''_{r+1} *avec*

$c''_r = c_{r+1}$ et $c''_{r+1} = c_{r+1} - p_r \Rightarrow c''_r + c''_{r+1} = 2.c_{r+1} - p_r,$ et $p_r > p_{r+1}$

$\Rightarrow c''_r + c''_{r+1} < c_r + c_{r+1}$

Un seul échange suffit donc pour améliorer ce critère, or il y en a au minimum un pour passer de $\mathbf{0}$ *à* $\mathbf{0}'$.

D) Remarques

La condition "les séquences EDD, FIFO et SPT coïncident " est une condition très forte. On peut se demander ce que devient le théorème si seulement deux de ces séquences coïncident.

Nous allons rapidement présenter quelques résultats concernant le maintien ou non des propriétés des séquences.

α) <u>FIFO minimise toujours le makespan M</u>

La démonstration se fait par échange de paires de tâches consécutives :

σ_{ij}, σ_{ji}, M_{ij}, M_{ji} *et* q *sont définies comme pour la démonstration des théorèmes* 3.4.2. a et b.

On suppose ici $r_j < r_i$. *Et on note* m_{ij} *et* m_{ji} *la fin de la paire* {i,j} *respectivement dans* σ_{ij} *et* σ_{ji}. *Si* $m_{ji} \leq m_{ij}$, *alors on aura* $M_{ji} \leq M_{ij}$ *(même sous-séquence semi-active terminant la solution et pouvant commencer au moins aussi tôt). Montrons donc que* $m_{ji} \leq m_{ij}$.

a) $q \geq r_i \Rightarrow m_{ji} = q + p_j + p_i = m_{ij}$

b) $q < r_i \Rightarrow m_{ji} = \max [\, q + p_j + p_i , r_j + p_j + p_i , r_i + p_i \,] \leq$

$\max [\, r_i + p_j + p_i , r_i + p_j + p_i , r_i + p_i \,] = r_i + p_i + p_j = m_{ij}$

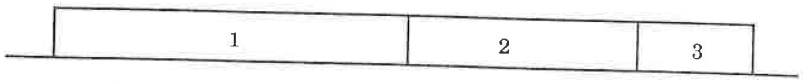
Ensuite, à partir d'une séquence minimisant M, des échanges de paires consécutives conduisent à FIFO sans dégrader M.

β) FIFO et EDD identiques, mais différents de SPT peuvent ne minimiser ni T, ni C, ni F

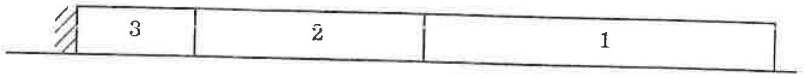
Montrons le par un contre-exemple (figure 3.4.2.d).

i	1	2	3
r_i	0	1	2
p_i	15	10	5
d_i	0	1	2

énoncé d'un problème d'ordonnancement



T = (15 - 0) + (25 - 1) + (30 - 2) = 67
C = 15 + 25 + 30 = 70
ordonnancement selon les ordres FIFO et EDD.



T = (7 - 2) + (17 - 1) + (30 - 2) = 53
C = 7 + 17 + 32 = 56
ordonnancement strictement meilleur que les ordres FIFO et EDD pour les critères "somme des retards" et "somme des dates d'achèvement".

figure 3.4.2. d

γ) FIFO et SPT identiques minimisent C et F

Ce résultat est un corollaire du théorème 3.4.2. a puisque ces critères n'utilisent pas les délais.

δ) FIFO et SPT identiques mais différents de EDD peuvent ne pas minimiser T

Montrons le par un contre-exemple (figure 3.4.2. e)

i	1	2
r_i	0	1
p_i	10	11
d_i	22	12

énoncé d'un problème d'ordonnancement

T = max [0 , 10 - 22] + max [0 , 21 - 12] = 9
séquence selon FIFO et SPT

T = max [0 , 12 - 12] + max [0 , 22 - 22] = 0
ordonnancement optimal

figure 3.4.2. e

On peut noter que si, quel que soit l'ordre retenu, toutes les tâches sont en retard, la séquence commune à FIFO et SPT qui minimise C minimise aussi T.

ε) EDD et SPT identiques, mais différents de FIFO peut ne minimiser ni T, ni M, ni C

Montrons le par un contre-exemple (figure 3.4.2.f)

i	1	2
r_i	9	0
p_i	10	15
d_i	10	15

énoncé

R = (19 - 10) + (34 - 15) = 28 ; C = 19 + 34 = 53 et M = 34
séquence commune à EDD et SPT

R = (15 - 15) + (25 - 10) ; C = 15 + 25 = 40 et M = 25
séquence strictement meilleure pour les 3 critères

figure 3.4.2.f

3.4.3. Conséquences pour les méthodes de décomposition temporelle

Théorème 3.4.3.a

On considère le problème d'ordonnancement à une machine.
Si les séquences EDD, SPT et FIFO coïncident, alors les méthodes de décomposition temporelle DTRUM sont polynômiales et optimales.

Démonstration

On considère d'abord le problème d'ordonnancement où, à partir d'un instant q, *on doit ordonnancer un ensemble de tâches telles que les séquences* EDD, SPT *et* FIFO *coïncident, de manière à minimiser la somme des retards* T, *puis à égalité sur ce premier critère, maximiser le critère d'urgence* U *et, enfin, à égalité sur ce second critère, minimiser la durée totale* M.

On peut remarquer que :
EDD *minimise la somme des retards* (théorème 3.4.2.a)
toute séquence qui respecte pas EDD *ne maximise pas* U *(avec les hypothèses présentes).*

En effet, $U(EDD) = \frac{n(n-1)}{2}$ (cf 3.1.2.) *est le maximum maximorum du critère* U *obtenu quand tous les couples vérifient la propriété* U.
Or, ici, si la tâche i *est placée avant* j *avec* $d_i > d_j$, *cela entraîne* $r_i \geq r_j$ $(\Rightarrow c_i - p_i \geq r_j)$ *et la propriété* U *n'est pas vérifiée pour ce couple.*

Donc, parmi les séquences minimisant la somme des retards, le deuxième critère ne retient que les séquences vérifiant EDD.
Parmi ces séquences, à égalité sur la valeur du délai d_i, *il peut exister des sous-séquences locales qui ne vérifient pas* FIFO ou SPT, *et dans ce cas, la séquence complète, qui respecte* EDD, *peut minimiser ou non la somme des retards.*

Le théorème 3.4.2. a *nous indique que la séquence commune aux ordres* EDD, SPT et FIFO, *qui peut être obtenue par un tri (complexité polynômiale en* $0(n.\log n)$), *satisfait à toutes les exigences demandées : minimisation de* T, *maximisation de* U *et minimisation de* M *(tout en apportant des qualités supplémentaires non exigées par* DTRUM *).*

Nous utilisons donc cette solution au sein de la méthode DTRUM *pour résoudre le sous-problème posé au cours des optimisations locales.*

Démontrons maintenant le théorème 3.4.3 *par récurrence sur le nombre* K *d'itérations.*

a) *Si* $K = 1$ *alors la construction de la séquence commune à* EDD, FIFO et SPT *est polynômiale et fournit une solution optimale.*

b) *Supposons qu'en fin d'itération* K, *la méthode de décomposition temporelle* DTRUM *ait fourni pour l'ensemble des* n_k *tâches planifiées l'ordre commun, pour ces tâches, à* EDD, FIFO et SPT *et considérons l'itération* $K+1$.
Soit M_k *la valeur du "makespan" à la fin de l'itération* K.

$$\alpha)\quad M_k \leq r_{min} = \min_{i \in W - N_k} [r_i]$$

où N_k *est l'ensemble des tâches déjà planifiées.*
Alors l'itération $K+1$ *est indépendante des itérations précédentes, la solution locale est fournie par la sous-séquence commune aux ordres* EDD, FIFO et SPT *pour les tâches ajoutées et comme :*

$$\max_{i \in N_K} [r_i] < M_K \leq \min_{i \in \pi_k} [r_i]$$

La concaténation des deux sous-séquences vérifiant EDD, FIFO et SPT *constitue une séquence vérifiant* EDD, FIFO et SPT.

$$\beta)\quad M_K > r_{min} = \min_{i \in W - N_K} [r_i]$$

Si q *est le début de la tâche planifiée* j *vérifiant* $q = c_j - p_j \leq r_{min} < c_j$, *alors l'optimisation locale choisit, à partir de* q, *l'ordre* EDD/FIFO/SPT *pour les tâches à réordonnancer et pour les tâches ajoutées.*

On a :

$$\max_{i \in F} [r_i] \leq r_j \leq c_j - p_j \leq r_{min}$$

où F *est l'ensemble des tâches fixées,*
et

$$r_j = \min_{i \in N_K - F} [r_i]$$

d'après la formule de récurrence à la fin de l'itération K.
A nouveau, les sous-séquences correspondant à F *et à l'ordonnancement local sont indépendantes, respectent* EDD/FIFO/SPT *et leur juxtaposition respecte* EDD/FIFO/SPT.
Donc si à la fin de l'itération K, *la solution partielle respecte les ordres* EDD, FIFO et SPT, *alors ces ordres sont à nouveau respectés à la fin de l'itération* K+1 *et le coût de chaque itération est de complexité polynômiale (tri).*

Il y a au plus n *itérations et à la fin de chacune, la solution partielle est optimale pour le sous-ensemble de tâches considéré. La solution finale est donc obtenue avec une complexité en* $0(n^2 \log n)$ *et est optimale.*

Cela termine la démonstration du théorème 3.4.3.

Si l'on ne recherche pas la polynômialité de la méthode, on peut démontrer par récurrence sur le nombre d'itérations en distinguant les cas α) et β) comme ci-dessus, un théorème plus général sur le méthodes de décomposition temporelle utilisant le "makespan" M comme critère secondaire :

Théorème 3.4.3.b

On considère le problème d'ordonnancement à une machine.
Si les séquences EDD, SPT et FIFO coïncident, alors les méthodes de décomposition temporelle DTRM, DTRMU et DTRUM sont optimales.

3.5. CAS GENERAL ET CONCLUSIONS

3.5.1. Introduction

Nous nous intéressons dans cette dernière partie du chapitre 3 au problème général à une machine lorsque les durées sont quelconques, sans liens particuliers avec les délais et les dates de disponibilité, comme c'était le cas au paragraphe 3.4.

Nous allons nous intéresser successivement aux ordonnancements u-actifs (paragraphe 3.5.2.) et aux erreurs commises par les méthodes de décomposition temporelle (paragraphe 3.5.3.).

3.5.2. Ordonnancements u-actifs

Théorème 3.5.2.

Lorsque les durées sont quelconques, le sous-ensemble des ordonnancements u-actifs ne constitue plus un sous-ensemble dominant pour le critère "somme des retards".

Démonstration

Montrons le par un contre-exemple où un ordonnancement non u-actif est strictement meilleur que tout ordonnancement u-actif (figure 3.5.2.).

i	1	2	3
r_i	0	0	0
p_i	15	5	2
d_i	0	1	2

énoncé d'un problème d'ordonnancement

1	2	3

T = (15 - 0) + (20 - 1) + (22 - 2) = 54
seule séquence u-active : EDD (quand $r_i = 0$)

3	2	1

T = (2 - 2) + (7 - 1) + (22 - 0) = 28
séquence non u-active meilleure que EDD

figure 3.5.2.

Ce contre-exemple a été construit en utilisant des résultats connus ([K. BAKER 1974]) *et rappelés au paragraphe* 3.4.2. *: si* $\forall i,\ r_i = 0$ *et si quelle que soit la séquence toutes les tâches sont en retard, alors* SPT *est optimal.*
Comme, par ailleurs, vérifier u *quand* $r_i = 0$, *oblige à respecter l'ordre des délais, il suffit de choisir des délais suffisamment petits pour que toutes les tâches soient en retard, et de faire coïncider* EDD *et* LPT *pour obtenir un contre-exemple.*

En prenant dans l'exemple $r_2 = 1$ *ou* 2, *on obtient un contre-exemple où les* r_i *ne sont pas tous nuls.*

En conclusion, se limiter, dans le cas général aux ordonnancements u-actifs comme dans les méthodes de décomposition temporelle DTRU, DTRUM et DTRMU, c'est prendre le risque d'éliminer a priori les solutions optimales.

3.5.3. Majoration d'erreurs

Les problèmes où toutes les durées des tâches sont égales sont des cas particuliers du cas plus général qui nous intéresse ici. Cela signifie que l'on peut faire au moins aussi mal que dans le cas où toutes les durées sont égales.

En particulier, on a :

$$\Delta T_{s_a} \geq \max [n \,.\, \Delta r,\ n \,.\, (n-1) \,.\, p_{min}]$$

$$\Delta T_a \geq \frac{n^2}{4} \cdot (2 \,.\, p_{min} - 1)$$

$$\Delta T_{ua} \geq \frac{n}{2} \cdot \frac{p_{min} \cdot (p_{min} - 1)}{2 \,.\, p_{max} - 1} \,.\, \left[\frac{n \,.\, (3 \,.\, p_{min} - 2)}{2 \,.\, p_{max} - 1} + 1 \right]$$

$$\Delta T_{DTRUM} \geq \frac{n\,(n-1)}{2} \cdot \frac{p_{min} \cdot (p_{min} - 1)}{2 \,.\, p_{max} - 1}$$

$$\Delta T_{usd} \geq (n-1) \,.\, (p_{min} - 1)$$

où $p_{min} = \min_i [\, p_i \,]$ et $p_{max} = \max_i [\, p_i \,]$

et ΔT_μ pour $\mu \in$ (sa, a ua, DTRUM, usd) est l'écart maximal sur la somme des retards que l'on peut obtenir entre deux ordonnancements quelconques d'un même problème dans l'ensemble des ordonnancements semi-actifs, actifs, u-actifs, fournis par DTRUM et u-actifs sans délais.

On peut se demander si on obtient des majorants en remplaçant p_{min} par p_{max} et p_{max} par p_{min}. La réponse est négative, car, ici, non seulement on augmente l'écart en reportant le plus tard possible l'instant de début des tâches et en ne respectant pas EDD, mais aussi en ne respectant pas SPT.

Nous nous contenterons de fournir un exemple simple pour montrer que $\max [n \,.\, \Delta r,\ n \,.\, (n-1) \,.\, p_{max}]$ n'est pas majorant dans l'ensemble des ordonnancements semi-actifs. *L'énoncé et les ordonnancements extrêmaux sont fournis par la figure* 3.5.3. (n impair).

Pour construire cet exemple, nous avons utilisé les résultats du paragraphe 3.4. *en choisissant les valeurs des paramètres de telle sorte que* FIFO, SPT et EDD *soient trois séquences identiques. Il est alors évident que* 0^M *maximise la somme des retards et que* 0^m *la minimise. Par ailleurs* $T_M = T_1$ *car si des tâches de durées égales et toutes disponibles au même instant sont toutes en retard quel que soit leur ordre, alors la somme des retards ne dépend pas de leur ordre.*
On a :

$$T_M - T_m = T_M - T_1 = \frac{n-1}{2} \,.\, [\, n \,.\, D + \frac{n-1}{2} \,.\, D \,] + \frac{n-1}{2} [\, n \,.\, D - \frac{n-1}{2} \,]$$

$$= D \,.\, \frac{n-1}{2} \,[\, 2 \,.\, n + \frac{n-1}{2} \,] - \frac{(n-1)^2}{4} = D \,.\, (n-1) \,.\, n + \frac{(n-1)}{4} \,.\, (D-1)$$

donc $\Delta T_{sa} > n \,.\, (n-1) \,.\, p_{max}$ (car $p_{max} = D > 1$)

et comme ici $\Delta r = (n-1) \,.\, p_{max}$

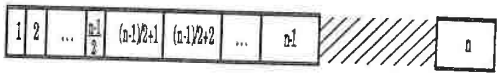
on a bien

$$\Delta T_{sa} > \max [\, n \,.\, \Delta r,\ n \,.\, (n-1) \,.\, p_{max} \,]$$

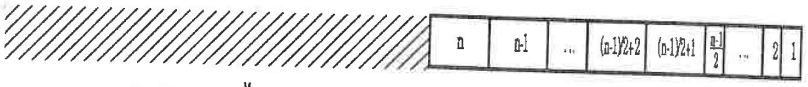
Et il est possible de trouver des écarts encore plus grands en ne se limitant pas à 2 valeurs possibles pour la durée des tâches.

i	1	2	...	(n-1)/2	(n-1)/2+1	(n-1)/2+2	...	n-1	n
r_i	0	1	...	(n-1)/2-1	(n-1)/2	(n-1)/2+D	...	(n-1)/2+((n-1)/2).D	(n-1).D
p_i	1	1	...	1	D	D	...	D	D
d_i	1	2	...	(n-1)/2	(n-1)/2+D	(n-1)/2+2.D	...	(n-1)/2.(D+1)	n D

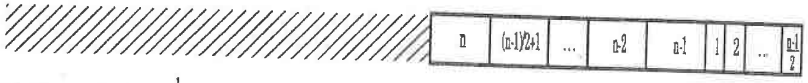
énoncé d'un problème d'ordonnancement



ordonnancement minimal (FIFO, SPT et EDD): 0^m



ordonnancement maximal (LIFO, LPT et LDD): 0^M



autre ordonnancement maximal : 0^1

figure 3.5.3.

3.5.4. Conclusion

Dans le cas des durées quelconques, trop de paramètres interviennent pour que l'on puisse trouver des bornes supérieures de forme simple comme dans le cas des durées égales. On ne pourra avoir une idée des erreurs commises que par simulation.

De même, les simulations sont nécessaires, même dans le cas des durées égales, pour obtenir des calculs d'erreurs en moyenne. C'est l'objet d'une partie du chapitre 4.

BIBLIOGRAPHIE DU CHAPITRE 3

K.R. BAKER
Introduction to sequencing and scheduling
John Wiley, **1974**

K.R. BAKER / Z. SU
"Sequencing with due dates and early start times to minimize maximum tardiness"
Nav. Res. Log. Quart., vol. 21, pp 171-176, **1974**

E. BALAS
"Machine sequencing via disjunctive graphs : an implicit enumeration algorithm"
Carnegie Mellon University, Revised Dec. **1968**

P. BRATLEY / M. FLORIAN / P. ROBILLARD
"On sequencing with earliest starts and due dates with application to computing bounds for the (n/m/G/Fmax) problem"
Nav. Res. Log. Quart., Vol. 20, n° 1, pp 57-67, **1973**

J. CARLIER
"Ordonnancements à contraintes disjonctives"
RAIRO, Vol. 12, n° 14, novembre **1978**

J. CARLIER
"The one-machine sequencing problem"
EJOR, Vol. 11, pp 42-47, **1982**

M.I. DESSOUKY / J.S. DEOGUN
"Sequencing jobs with unequal ready times to minimize mean flow time"
Soc. Industrial Appl. Math. J. Comput., Vol. 10, pp 192-202, **1981**

H. EMMONS
"One machine scheduling to minimize certain functions of job tardiness"
O.R., Vol. 17, n° 4, pp 701-705, **1969**

M.L. FISHER
"A dual algorithm for the one-machine scheduling problem"
"Mathematical programming, Vol. 11, n° 3, 458-481, **1976**

M.R. GAREY / D.S. JOHNSON
Computers and intractability : a guide to the theory of NP-completeness
FREEMAN, **1979**

L. GELDERS /.R. KLEINDORFER
"Coordination aggregate and detailed scheduling decisions in the one-machine job-shop. Part 1 : theory"
O.R., Vol. 22, n° 1,, pp 46-60, **1974**

J. GRABOWSKI / E. NOWICKI / S. SDRZALKA
"A block approach for single-machine scheduling with release dates and due dates"
EJOR, Vol. 26, pp 278-285, **1986**

R.L. GRAHAM / E.L. LAWLER / J.K. LENSTRA / A.H.G. RINOOY KAN
"Optimization and approximation in deterministic sequencing and scheduling : a survey"
Ann. Discrete Math., Vol. 5, pp 287-326, **1979**

A.M.A. HARIRI / C.N. POTTS
"An algorithm for single machine sequencing with release dates to minimize total weighted completion time"
Discrete Applied Mathematics, Vol. 5, pp 99-109, **1983**

J.R. JACKSON
"Scheduling a production line to minimize maximum tardiness"
Research Report 43, Management Science,
Research Project, University of California, Los Angeles, **1955**

H. KISE / T. IBARAKI / H. MINE
"Approximate algorithms for the one-machine maximum lateness scheduling problem with ready times"
Technical report, Kyoto, **1978**

B.J. LAGEWEG / J.K. LENSTRA / A.H.G. RINNOOY KAN
"Minimizing maximum lateness on one machine : computational experience on some applications"
Mathematical Centrum, Amsterdam, août **1975**
et
Statistica neerlandica, Vol. 30, pp 25-41, **1976**

E.L. LAWLER
"A 'pseudopolynômial' algorithm for sequencing jobs to minimize total tardiness"
Ann. Discrete Math., Vol. 1, pp 331-342, **1977**

G.B. Mc MAHON / M. FLORIAN
"On scheduling with ready times and due dates to minimize maximum lateness"
O.R., Vol. 23, pp 475-482, **1975**

I. NABESHIMA
"Unified treatment on single machine scheduling problems"
Rep. Univ. Electro-Comm, Vol. 22, n° 2, pp 29-38, déc. **1971**

M. PINEDO
"Stochastic scheduling with release dates and due dates"
O.R., Vol. 31, n° 3, pp 559-572, **1983**

M.E. POSNER
"A sequencing problem with release dates and clustered jobs"
Management Science, Vol. 32, n° 6, pp 731-738, juin **1986**

C.N. POTTS
"Analysis of heuristics for sequencing jobs on one machine with release dates and delivery dates"
Preprints BW 95/78, Amsterdam, **1978**

C.N. POTTS / L.N. VAN WASSENHOVE
"A decomposition algorithm for the single machine total tardiness problem"
O.R. Letters, Vol. 1, n° 5, pp 177-181, novembre **1982**

A.H.G. RINNOOY KAN
"Machine scheduling problems : classification, complexity and computation"
Nijhoff, The Hague, **1976**

A.H.G. RINNOOY KAN / B.J. LAGEWEG / J.K. LENSTRA
"Minimizing total costs in one-machine scheduling"
Combinatorial programming : methods and application, Versailles, sept. **1974**
ou
O.R. , Vol. 23, pp 908-927, **1975**

B. ROY / B. SUSSMANN
"Les problèmes d'ordonnancement avec contraintes disjonctives"
SEMA, Direction Scientifique, Rapport de Recherche n° 9, octobre **1964**

L. SCHRAGE / K.R. BAKER
Dynamic programming solution of sequencing problems with precedence constraints"
O.R., Vol. 26, n° 3, pp 444-449, **1978**

T.T. SEN / L.M. AUSTIN / P. GRANDFOROUSH
"An algorithm for the single-machine sequencing problem to minimize total tardiness"
IIE trans., Technical Note, Vol. 15, n° 4, pp 363-366, déc. **1983**

J. SHWIMER
"On the n-job, one-machine sequence-independent scheduling problem with tardiness penalties : a branch and bound solution"
Management Science, Vol. 18, pp 301-313, **1972**

V. SRINIVASAN
"A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness"
Nav. Res. Log. Quart., Vol. 18, n° 3, pp 317-327, **1971**

L. VAN WASSENHOVE / L. GELDERS
"Four solution techniques for a general one machine scheduling problem : a comparative study"
EJOR, Vol. 2, pp 281-290, **1978**

CHAPITRE 4

EVALUATION EXPERIMENTALE DES METHODES DE DECOMPOSITION SPATIALE ET/OU TEMPORELLE

4. EVALUATION EXPERIMENTALE DES METHODES DE DECOMPOSITION SPATIALE ET/OU TEMPORELLE

4. EVALUATION EXPERIMENTALE DES METHODES DE DECOMPOSITION SPATIALE ET/OU TEMPORELLE

4.1. INTRODUCTION

Nous avons vu, au paragraphe 1.3.2., que, lorsque l'on utilisait des méthodes approchées, se posait le problème de leur évaluation, et que cette évaluation pouvait se faire de deux façons : en moyenne ou dans le pire des cas. Le chapitre 3 s'est intéressé, pour des cas particuliers, à des évaluations dans le pire des cas. Ce chapitre va s'intéresser aux évaluations en moyenne.

Les évaluations en moyenne peuvent être obtenues de deux façons :

- on associe aux différentes caractéristiques du problème posé des variables aléatoires dont on précise la loi de probabilité, et on cherche l'espérance mathématique de l'écart par rapport à l'optimum des solutions fournies par les méthodes approchées ;

- on génère un large échantillon de problèmes, on applique des méthodes approchées ou exactes, et on effectue des statistiques sur les résultats fournis par les différentes méthodes.

La première façon nous semble peu réaliste par rapport aux exemples industriels car on ne sait obtenir des résultats que sur des cas très simples : une seule loi de probabilité par caractéristique avec une moyenne et un écart type. Or, si l'on considère les délais, par exemple, une loi plus réaliste serait probablement de la forme : x % des articles ont des délais raisonnables conformes en moyenne aux décisions prises au niveau de la gestion de production à moyen terme, et (100-x) % des articles ont des délais très courts, presque impossibles à satisfaire. Il n'y a donc pas, en général, des lois pour les caractéristiques de l'ensemble des produits, mais des lois propres à des familles de produits ou à des familles de machines.

C'est pourquoi nous n'avons considéré que la deuxième approche pour évaluer en moyenne, qui ne comporte aucune restriction sur la complexité des lois utilisées dans les générateurs.

Nous allons, dans le paragraphe 4.2., présenter les différentes formes de générateurs que nous avons conçus, puis présenter les différents résultats obtenus en les utilisant. Les résultats sont donnés sous forme de tableaux synthétiques, accompagnées de remarques. Lorsque nos méthodes sont comparées avec d'autres, celles-ci sont brièvement rappelés. Au paragraphe 4.3., nous ne nous intéressons qu'aux différentes variantes de nos méthodes de décomposition ; au paragraphe 4.4., nous les comparons avec d'autres méthodes, essentiellement avec des méthodes utilisant des règles empiriques dans le cas général, et avec des méthodes plus spécifiques dans le cas où l'atelier ne comporte qu'une machine. Nous terminons par quelques remarques générales sur les résultats obtenus.

4.2. GENERATEURS DE PROBLEMES

4.2.1. Remarques générales

Les conclusions que l'on peut tirer d'un plan d'expériences dépendent du plan d'expériences choisi. Cela signifie que des résultats expérimentaux doivent toujours être accompagnés de la description du générateur qui a fourni les problèmes.

La plupart des générateurs utilisés dans les résultats publiés proposent une loi de probabilité par donnée caractéristique de chaque type d'information (dates d'arrivée, durées opératoires, délais,...). Les résultats que l'on obtient ne sont valables que pour des ateliers où les caractéristiques des différents objets sont uniformes.

4.2.2. Générateur pour les méthodes de décomposition spatiale

Dans les méthodes de décomposition purement spatiale, on suppose implicitement l'existence d'îlots de fabrication imparfaits dans l'atelier. En d'autres termes, on suppose que certains groupes de produits passent essentiellement sur certains groupes de machines avec seulement de rares incursions dans les autres groupes de machines. Il est évident que les méthodes de décomposition spatiale sont d'autant plus performantes qu'il y a moins de sorties des produits hors de leur îlot mère (voir paragraphe 2.3.3.) ; le cas idéal est le découpage en îlots totalement indépendants, ou au moins en groupe d'îlots indépendants entre eux.

Le générateur destiné à tester ces méthodes doit travailler dans ce cadre. On lui fournit en paramètres : le nombre d'îlots, le nombre de machines par îlots, le nombre de produits, le nombre moyen d'opérations par produit, les lois des durées opératoires, et surtout le "coefficient de pollution" que l'on accepte hors des îlots mères, c'est-à-dire le nombre de fois où des produits quittent leur îlot d'origine pour subir des opérations sur un autre îlot.

4.2.3. Générateurs de solutions

Pour évaluer l'erreur commise sur la valeur du critère par des méthodes approchées, il faut connaître la valeur du critère à l'optimum ou, à défaut, en avoir une bonne estimation.

Si, pour obtenir la valeur de la solution optimale, on utilise un algorithme exact, le problème posé demande un volume important de calculs. On ne peut donc les utiliser que pour des problèmes de petite taille, et ceci ne fournit pas des résultats intéressants pour les problèmes concrets.

Pour des problèmes de taille plus importante, on peut, pour évaluer les méthodes approchées :

- les comparer à des méthodes exactes en limitant, pour chacune, le nombre total de calculs. On compare alors les meilleures solutions trouvées par chacune des méthodes,

- les comparer aux meilleurs résultats obtenus par d'autres méthodes approchées,

- les expérimenter sur des exemples dont on connaît, a priori, la valeur de la solution optimale ; c'est ce que nous obtenons avec ce que nous appelons des générateurs "de solutions".

Le principe de ces générateurs de solutions est simple. Au lieu de générer des problèmes d'ordonnancement, nous générons des solutions de ces problèmes dont nous tirons l'énoncé correspondant.

Notre générateur fonctionne de la manière suivante :

- dans une première phase, on génère de manière aléatoire un diagramme de Gantt comportant des tâches indépendantes séparées par des intervalles où les machines sont inutilisées,

- dans une deuxième phase, on rassemble plusieurs tâches qui se suivent dans le temps pour constituer des gammes de produits et, pour chacun d'eux, on génère, en amont de la première tâche, une date d'arrivée et un délai par rapport à la fin de la dernière tâche.

Pour chaque produit, la suite des durées des tâches qui constituent sa gamme, ainsi que les machines associées et les délais, fournissent toutes les informations de l'énoncé.

Les lois de probabilité que nous utilisons sont liées à des types de machines. Pour chaque type de machines, il est possible de définir le pourcentage de petites, moyennes et longues tâches, leurs lois de probabilité, ainsi que la probabilité qu'une tâche soit suivie, dans le diagramme de Gantt, d'une période où la machine est inutilisée, et la loi de la durée de cette période.
Ceci permet d'obtenir des machines "goulots", des machines peu chargées, des machines chargées de nombreuses tâches courtes ou de quelques tâches longues, etc.

Comme ce générateur est utilisé pour tester des méthodes de décomposition spatiale, il comporte la possibilité, dans la première phase, de fournir un nombre d'îlots, une loi de probabilité pour le nombre de machines par îlot, une loi pour la répartition des types de machines dans les îlots. Dans la deuxième phase, un îlot mère est affecté à chaque produit, et on constitue sa gamme avec une certaine probabilité de lui attribuer des tâches longues dans l'îlot mère et courtes à l'extérieur. Malheureusement, la possibilité de respecter cette probabilité décroît au fur et à mesure que l'on affecte des tâches aux produits, le choix de tâches disponibles s'amenuisant. On n'a donc pas clairement un coefficient de "pollution" choisi par l'utilisateur comme pour le générateur présenté au paragraphe 4.2.2.

Considérons maintenant la partie "génération d'un délai par rapport à la fin de la dernière tâche d'un produit". Si pour tout produit, nous supposons qu'il n'est pas en retard dans la planification proposée par le diagramme de Gantt, c'est-à-dire que nous générons le délai après l'instant de fin de la dernière tâche, alors la solution générée est optimale pour le critère "somme des retard" avec un retard total nul, et nous connaîtrons exactement l'erreur commise sur ces exemples par les méthodes approchées.

Cette possibilité est intéressante pratiquement, mais il faut être prudent et ne pas en tirer de conclusions générales : en effet, la théorie montre que, dans certains cas particuliers, les problèmes deviennent polynômiaux pour les critères de retard lorsqu'il existe des solutions (optimales) dont le retard est nul.

En générant, sur le diagramme de Gantt du retard pour quelques produits, on connaît un majorant de la valeur optimale avec lequel on pourra comparer les résultats des méthodes de décomposition. Toutefois, on ignore s'il n'existe pas une solution de retard nul (à moins d'insérer un produit "critique au sens du PERT", c'est-à-dire ayant toutes ses tâches contiguës, en retard).

Les deux générateurs ont été utilisés pour nos expériences. Un avantage du deuxième est de fournir des familles de solutions voisines en répétant plusieurs fois la deuxième phase sur un même diagramme de Gantt.

4.3. COMPARAISONS DES METHODES DE DECOMPOSITION

La figure 4.3. présente une série de résultats. Elle comporte 26 problèmes d'ordonnancement générés par un générateur d'énoncés (cf 4.2.2.).

On peut noter qu'aucune méthode de décomposition ne domine nettement les autres. Chacune rencontre des difficultés sur des exemples différents (sauf l'exemple 50, difficilement traité par toutes les méthodes). (Les débordements de mémoire (disque) de la décomposition temporelle sont dûs à un malencontreux encombrement de la zône disque commune à plusieurs utilisateurs).

Les "contre-performances" des méthodes de décomposition sont les suivantes :

-- en ce qui concerne le coût d'obtention de la solution (≥100s cpu) ou les débordements mémoire (dûs à l'utilisation de la file) :

- exemples 16, 45, 47, 50 et 17 (soit 5 fois sur 26) pour les décompositions temporelles,
- exemples 16, 48, 45, 47, 50, 17, 2 et 11 (soit 8 fois sur 26) pour les décompositions spatiales,
- exemple 50 (soit une fois sur 26) pour les décompositions spatiales et temporelles.

numéro	tailles			sans décomposition		décomposition temporelle		décomposition spatiale		décomposition spatiale et temporelle		nombre d'îlots
	m	n	o	retards	coût	retards	coût	retards	coût	retards	coût	
16	12	20	94	56.9 *	1412	###	###	22.3	630	18.6	56	3
48	12	20	94	244.9 *	1339	21.6	38	270.9 *	988	43.2	18	3
49	12	20	67	66.9 *	1142	13.2	45 [19]	67.9	51	2.4	13	5
45	8	20	65	114.8 *	841	2.8 [###]	370 [###]	61.9	458	0	41	3
47	8	20	65	72.6 *	836	15.3 [0]	360 [59]	38.3 *	464	0	40	3
50	8	20	57	83.8 *	805	###	###	34.5 *	373	32.1	267	4
51	10	20	56	45.4 *	1039	45.4	19 [7]	45.4	5	45.4	10	4
17	7	18	50	19.8 *	710	###	###	0.6	100	6	18	3
2	8	15	50	131.9 *	686	9.1	14 [12]	13.8	112	14	12	3
11	8	15	50	131.9 *	713	9.1	14 [12]	13.8	116	14	14	3
18	6	17	40	85.7 *	537	11.2	58 [20]	11.2	28	11.2	11	3
8	7	14	38	56.9 *	582	19.3 [10.4]	16 [9]	18	5	10.4	11	3
19	6	15	37	0	6	0	4	0	3	0	9	2
10	6	14	37	0	20	0 [0.8]	11 [12]	0	4	0	7	2
7	6	14	37	0	20	0	11	0	3	0	9	3
46	7	16	35	10.3	96	10.3	5	10.6	3	10.3	8	3
15	7	14	32	0	10	0	5	0.2	30	0	5	3
9	6	14	32	94.9 *	469	80.8	22 [15]	51	5	56.6	8	3
1	5	12	29	0	12	0.4	5 [4]	0	3	1.1	12	2
3	5	12	29	0	12	0.4	4	0.8	4	1.1	7	2
4	5	12	29	0	10	0	4	7.9	5	0	7	2
5	5	12	29	0	18	0.1	7 [4]	0	3	0.1	5	2
12	5	12	29	0	12	0	4	1.4	4	0	7	2
13	5	12	29	0	12	0.4	5	0	3	0.4	7	2
14	5	12	29	0	12	0	5	0	3	0	8	2
6	6	12	26	0	3	0	4	0	2	0	7	3

m = nombre de machines
n = nombre de produits
o = nombre d'opérations

retards = valeur du critère "somme des retards"
coût = durée "cpu" sur le DPS8 (système multics) exprimée en secondes

* = lors des optimisations locales ou globales, on a conservé la meilleure solution trouvée en au plus 10 000 séparations
= méthode arrêtée pour cause de débordement mémoire

[ ] = variante de la méthode de décomposition temporelle où les sous-ensembles de produits introduits sont plus petits (on cherche l'opération et non pas le produit pouvant se terminer le plus tôt possible), les résultats correspondants ne sont indiqués que s'ils sont différents

figure 4.3.

-- en ce qui concerne la valeur du critère "somme des retards", la méthode fournit la plus mauvaise valeur obtenue pour les exemples :
- 16, 50, 5 et 13 (soit 4 fois sur 26) pour les décompositions temporelles,
- 48, 49, 46, 15, 4 et 12 (soit 6 fois sur 26) pour les décompositions spatiales,
- 1, 3, 5 et 13 (soit 4 fois sur 26) pour les décompositions spatiales et temporelles.

Chacune des méthodes fournit la meilleure des valeurs obtenues pour le critère "somme des retards" dans les exemples suivants (une étoile marque les cas où la valeur est strictement meilleure que celles fournies par toutes les autres méthodes, y compris la méthode sans décomposition interrompue à 10 000 séparations) :
- 48*, 51, 2*, 11*, 18, 8, 19, 10, 7, 46, 15, 4, 12, 14 et 6 (soit 15 fois sur 26) pour les décompositions temporelles,
- 51, 17*, 18, 19, 10, 7, 9*, 1, 5, 13, 14 et 6 (soit 12 fois sur 26) pour les décompositions spatiales,
- 16*, 49*, 45*, 47, 50*, 51, 18, 8, 19, 10, 7, 46, 15, 4, 12, 14 et 6 (soit 17 fois sur 26) pour les décompositions spatiales et temporelles.

Les moyennes des coûts d'obtention des solutions sont respectivement :
- 436.7 secondes cpu pour les résolutions sans décomposition,
- 44.8 secondes cpu (sur seulement 23 exemples) pour les décompositions temporelles,
- 130.9 secondes cpu pour les décompositions spatiales,
- 23.7 secondes cpu pour les décompositions spatiales et temporelles.

Les moyennes des valeurs du critère "somme des retards" sont respectivement :
- 46.8 pour les résolutions sans décomposition,
- 10.4 (sur 23 exemples) pour les décompositions temporelles,
- 25.8 pour les décompositions spatiales,
- 10.3 pour les décompositions spatiales et temporelles.

Les résultats montrent l'intérêt des méthodes de décomposition : temps réduits pour de faibles pertes sur le critère "somme des retards". Ils donnent un léger avantage à la méthode de décomposition spatiale et temporelle. Ces méthodes nous ont permis en outre d'aborder des exemples plus gros non accessibles aux autres méthodes et pour lesquels nous connaissions la valeur optimale (nulle) du critère "somme des retards" grâce au générateur de solutions (cf 4.2.3.).

En outre, nous avons constaté que pour obtenir des solutions correctes pour de gros exemples récalcitrants, il suffit souvent de couper arbitrairement en deux l'îlot contenant le plus de machines.

Il faut cependant se garder de tirer des conclusions générales déduites de petits échantillons (qui ont pourtant coûté relativement cher en heures machine, surtout pour les comparaisons avec les méthodes sans décomposition).

Il serait intéressant, mais coûteux, de développer un plan systématique d'expériences pour étudier en particulier :

- l'influence du découpage en îlots,

- l'influence du choix de la règle de répartition des marges entre les sous-produits,

- l'influence de la règle de découpage du temps pour constituer les sous-ensembles de produits ou de sous-produits.

4.4. COMPARAISON AVEC D'AUTRES METHODES

Dans le cas général, si l'on cherche à comparer avec des règles empiriques, par exemple, l'opération la plus urgente ou ayant le moins de marges d'abord, il suffit de descendre une seule branche de nos méthodes arborescentes dans le cas sans décomposition. Les coûts d'obtention de la solution sont alors très faibles, mais les performances médiocres par rapport aux méthodes de décomposition.

Il nous a semblé plus intéressant de réaliser les expériences correspondant aux cas étudiés au chapitre 3 (une machine et durées égales) pour obtenir des résultats moyens.

Les résultats se présentent comme le montre la figure 4.4.

Une série de tableaux de résultats figure à l'annexe C.
Ils comparent l'algorithme de Jackson avec la méthode de décomposition temporelle DTRU.

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 500

delais : valeur minimale valeur maximale
0 500

durees : 10 nombre moyen de taches : 50

types de taches	1	2
pourcentages	25	75
delais minimaux	0	0
delais maximaux	100	500

numero de serie :99 taille de l'echantillon : 10

RESULTATS

	Jackson	DTRU
cout moyen	3	22
somme des retards (moyenne)	7121	7542
pourcentage de meilleures valeurs	90	10
ecart moyen par rapport aux meilleures valeurs	1	422

figure 4.4.

On peut remarquer en particulier, que, pour les exemples générés, "Jackson" est en général meilleure que "DTRU" mais que cela peut s'inverser (n^{os} de série 2, 5 et 11) quand les tâches sont relativement moins urgentes.

Jackson a une complexité en 0 (nlogn), DTRU en $0(n^3)$, la série d'exemples 11, 12, 13, 14, 15 destinée à le vérifier est insuffisante pour en tirer des conclusions, et cela d'autant moins que pour la série 1, 2, 3, 4, 5 où tous les exemples comportent 50 tâches, DTRU est entre 3 et 19 fois plus coûteux en temps d'obtention des solutions que Jackson.

Comme nous l'avons déjà dit à plusieurs reprises, les expériences doivent être très nombreuses avec toutes sortes de possibilités pour les paramètres de la génération pour tirer des conclusions de séries d'expériences.

CONCLUSION

CONCLUSION

Dans ce travail, nous avons proposé, non pas un nouvel algorithme pour ordonnancer les tâches dans un atelier, mais de nouvelles méthodes générales qui permettent de construire des familles d'algorithmes.

Ces méthodes approchées, qui utilisent des décompositions spatiales et temporelles, répondent à certains besoins des utilisateurs en entreprise :
- nécessité de concevoir des solutions qui peuvent évoluer de manière dynamique au cours du temps en fonction de l'arrivée de nouveaux produits ou de pertubations aléatoires,
- besoin de découper l'atelier en entités plus petites de manière à mieux le piloter et à mieux l'analyser.

Ces méthodes ont été évaluées de manière théorique dans des cas simples (décomposition temporelle et une seule machine). Cela a permis de dégager de nouvelles notions de dominance pour le critère utilisé : la somme totale des retards, critère peu étudié dans la littérature.

D'autres cas "simples" pourraient faire l'objet de recherches particulières, par exemple, une machine par îlot dans le cas de la méthode de décomposition spatiale et temporelle.

La suppression progressive des chevauchements dans les méthodes de décomposition purement spatiale peut être améliorée, mais au prix d'une augmentation non négligeable du coût de la méthode.

Les expériences pratiques montrent l'intérêt de ces méthodes. Mais la grande variété de possibilités des hypothèses des problèmes traités et des stratégies différentes dans les méthodes utilisées ne permet pas de tirer de conclusions générales.

ANNEXE A

NOTIONS DE COMPLEXITE DES PROBLEMES

ANNEXE A

NOTIONS DE COMPLEXITE DES PROBLEMES

A.1) NOTIONS DE COMPLEXITE DES ALGORITHMES

La fonction complexité ou complexité d'un algorithme est l'expression de son coût, exprimé en nombre d'opérations élémentaires, en fonction de la longueur des entrées dans le pire des cas.

La longueur des entrées est le nombre de caractères lus. Elle dépend du schéma d'encodage des paramètres du problème.

Souvent, on prendra comme longueur des entrées le nombre n de données lues.

Soit g une fonction de IN dans IN .

Si f(n) est la fonction complexité d'un algorithme donné, on dit que f(n) est en O(g(n)) si il existe une constante c telle que :

$\forall n : |f(n)| \leq c. |g(n)|$

Cette définition est usuelle en mathématique.

Par extension, on dit qu'un algorithme est en O(g(n)) si sa fonction complexité est en O(g(n)) .

Un algorithme est dit polynômial si sa complexité est en O(P(n)) où P est un polynôme.

Les algorithmes pour lesquels la fonction complexité ne peut pas être bornée par un polynôme sont dits exponentiels.

Coût de l'exploration de l'ensemble des solutions

Exemple à une machine

n tâches

n! séquences possibles à comparer

(hyp. : une séquence est explorée en une nano-seconde)

n	durée de l'exploration
10	3 millisecondes
11	40 millisecondes
12	0,5 secondes
13	6 secondes
14	1,5 minutes
15	20 minutes
16	6 heures
17	4 jours
18	2,5 mois
19	4 ans
20	77 ans
21	16 siècles

figure A1

Un algorithme exponentiel n'est pas utilisable pour de grandes valeurs de la longueur des entrées. Nous illustrons ce fait par la figure A1 . On y considère un simple problème à une machine où l'on cherche à minimiser la somme des pénalités de retards par rapport à des délais. Pour n produits, il y a n! séquences possibles. En prenant l'hypothèse optimiste de pouvoir examiner une séquence en une nano-seconde, la durée des calculs en fonction du nombre de produits est donnée sur la figure A1 . En supposant qu'un algorithme amélioré n'explore, par exemple, qu'un centième des séquences possibles, on ne pourrait pas l'utiliser pour n plus grand que 18.

Si un algorithme polynômial n'explore que n^p séquences, chacune en une nano-seconde, la figure A2 montre la supériorité des algorithmes polynômiaux sur les algorithmes exponentiels pour des valeurs relativement élevées de p . Elle illustre également l'intérêt de minimiser le degré du polynôme de la fonction complexité des algorithmes polynômiaux.

n	$n^5 \times 10^{-9}$ s	$n^{10} \times 10^{-9}$ s	$n! \times 10^{-9}$ s
10	0.1 millisecondes	10 secondes	3 millisecondes
20	3.2 millisecondes	2.8 heures	77 ans
30	24.3 millisecondes	6.8 jours	84 millions de millions de siècles
40	0.1 seconde	121.3 jours	ASTRONOMIQUE !!!

figure A2

A2) NOTIONS DE COMPLEXITE DES PROBLEMES

Face à un type de problème, la question importante est : peut-on espérer trouver un algorithme polynômial pour le résoudre ou cela est-il impossible ?

Toute une théorie de la complexité des algorithmes a été développée depuis 1970, on trouve en particulier dans le livre de M.R GAREY et D.S JOHNSON ([1979]) la présentation des principaux résultats ; de nombreux résultats ont été démontrés pour les problèmes d'ordonnancement, voir par exemple A.H.G. RINNOOY KAN ([1976]) et J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI / J WEGLARZ ([1986]).

Nous essayons de présenter brièvement ces notions, même si cela nuit légèrement à leur rigueur mathématique.

La notion essentielle est la transformée polynômiale de tout énoncé d'un problème P1 en énoncé d'un autre problème P2. Le problème P2 est alors au moins aussi difficile que le problème P1 car la transformation peut ne donner que des cas particuliers de P2.

Au moyen de cette notion, si on démontre que P1 est au moins aussi difficile que P2 et que P2 est au moins aussi difficile que P1, alors on définit des classes d'équivalence dans l'ensemble des problèmes.

Parmi les problèmes "décidables" (c'est-à-dire ceux pour lesquels il existe des algorithmes susceptibles de les résoudre), on distingue alors :

-- la classe des problèmes polynômiaux que l'on peut résoudre sur un ordinateur déterministe par des algorithmes polynômiaux,

-- elle est contenue dans la classe des problèmes N-P que l'on peut résoudre sur un ordinateur non déterministe par des algorithmes polynômiaux (pour les problèmes de décision qui consiste à savoir s'il existe au moins une solution vérifiant une propriété donnée, tester si une solution vérifie bien la propriété doit pouvoir se faire en temps polynômial pour les problèmes de N-P).

-- la classe N-P contient une classe de problèmes dit NP-complets dont il a été démontré qu'ils sont au moins aussi difficiles que n'importe quel problème de N-P.

-- la classe des problèmes pseudo-polynômiaux dont la complexité des meilleurs algorithmes peut s'exprimer sous forme d'un polynôme de la taille du problème et de la taille de la plus grande information pouvant figurer dans un énoncé.

-- la classe des problèmes N-P complets au sens fort, tels que l'ensemble des problèmes particuliers, où la taille de la plus grande information est limitée par un polynôme de la taille des données, reste dans l'ensemble des problèmes N-P complets.

BIBLIOGRAPHIE

M.R. GAREY / D.S. JOHNSON
Computers and intractability : a guide to the theory of N.P.-completeness
FREEMANN, **1979**

J. BLAZEWICZ / W. CELLARY / R. SLOWINSKI / J WEGLARZ
Scheduling under resource constraints - Deterministic Models
J.C. BALTZER AG, **1986**

A.H.G. RINNOOY KAN
Machine scheduling problems : classification, complexity and computation
Nijhoff, The Hague **1976**

ANNEXE B

METHODE EXACTE D'ORDONNANCEMENT UTILISEE POUR LES OPTIMISATIONS LOCALES

ANNEXE B

METHODE EXACTE D'ORDONNANCEMENT UTILISEE POUR LES OPTIMISATIONS LOCALES

B.1) STRATEGIE

Cette méthode est une procédure par séparation et évaluation. Comme nous l'avons signalé au paragraphe 1.4.3., il s'agit d'une PSEM. En effet, nous alternons deux stratégies.

Dans un premiers temps, nous utilisons une pile en largeur d'abord, c'est-à-dire que tous les sous-ensembles issus d'une séparation remplacent l'ensemble séparé au sommet de la pile, et que l'on sépare toujours le sous-ensemble situé au sommet. Ceci permet d'atteindre très rapidement les feuilles de l'arborescences où sont situées les solutions réalisables. En poursuivant avec cette stratégie, on sépare en premier lieu les sous-ensembles les plus proches du sommet de la pile, et donc les "branches" les plus à gauche de l'arborescence qui contiennent des solutions réalisables peu différentes les unes des autres.

Après un nombre fixé de séparations, on change de stratégie d'exploration. On transforme la pile de sous-ensembles à séparer en une liste triée par valeur croissante de la borne (minorant de la somme des retards) et on passe à une stratégie d'exploration de type PSEP. Comme la valeur de la borne croît de la racine vers l'extrémité des branches, cette stratégie sépare plutôt les sous-ensembles proches de la racine. En outre, les sous-ensembles créés ayant également des bornes de faible valeur, ils sont rarement éliminés par l'existence de bonnes solutions réalisables déjà trouvées. Comme à chaque séparation on remplace un sous-ensemble par deux autres, si on élimine peu, cette stratégie a tendance à encombrer la mémoire (parfois jusqu'à dépasser les dimensions prévues dans le programme).

C'est pourquoi, au bout d'un nombre fixé de séparations selon cette stratégie, on reprend la stratégie avec la pile. Celle-ci va à nouveau nous entraîner en priorité vers les feuilles de l'arborescence, mais comme le choix du sous-ensemble se trouvant au sommet de la pile (ancienne liste gérée en pile) est tout à fait aléatoire, les nouvelles solutions réalisables que l'on atteint seront probablement très différentes de celles obtenues précédemment par la stratégie de la pile, ou par la stratégie de la file qui a pu atteindre elle-aussi des feuilles.

Cette stratégie de pile sera abandonnée au profit de celle de la liste triée au bout d'un nombre fixé de séparations et ainsi de suite.

La figure 1.4.3.2. c du chapitre 1, illustre l'alternance de ces deux stratégies et leurs effets.

B.2) PRINCIPE DE SEPARATION

Nous avons retenu une méthode utilisant des contraintes disjonctives. C'est-à-dire que si deux tâches i et j utilisent la même machine k, nous devons avoir (i avant j) ou (j avant i).

Nous avons d'abord supposé que les moyens étaient illimités et avons planifié toutes les opérations au plus tôt. Nous obtenons ainsi un échéancier des dates de début de tâches.

Sur cet échéancier, nous recherchons le premier instant où deux opérations j_1 et j_2 entrent en conflit pour l'utilisation de la même machine. Nous séparons alors l'ensemble de toutes les solutions en deux sous-ensembles, celui où l'opération j_1 passe avant l'opération j_2 et réciproquement. Chaque nouveau sous-ensemble est obtenu en ajoutant aux contraintes de précédence traduisant les gammes des produits une nouvelle contrainte de précédence entre des opérations de produits différents utilisant la même machine. Les nouveaux échéanciers sont construits en repoussant les tâches situées sur des chemins issus de la tâche retardée (à la première séparation, c'est seulement la fin de la gamme du produit concerné). Le problème initial est bien remplacé par deux sous-problèmes identiques où on recherche le premier conflit.

Une solution est bien sûr réalisable lorsqu'on n'y trouve plus de conflit machines.

B.3) EVALUATION

Elle est obtenu très simplement en calculant la somme des retards des produits sur le planning correspondant à l 'échéancier. Comme la solution n'est pas, en général, réalisable, il s'agit d'un minorant de la valeur du critère pour le sous-ensemble de solutions réalisables considéré. En effet, pour obtenir une solution réalisable, les séparations ultérieures retardent des tâches mais n'en avancent jamais.

Contrairement aux méthodes qui ajoutent une (ou plusieurs) opérations à chaque séparation et ne regardent pas toujours les conséquences des choix fait sur les futures opérations, cette évaluation a le mérite de considérer en permanence l'ensemble de tous les produits.

B.4) VARIANTES

Il est possible de transformer cette méthode en méthode approchée, soit en limitant le nombre total de séparations autorisées et en fournissant la meilleure solution trouvée, soit en ne créant à chaque séparation que le sous-ensemble plaçant en premier une opération choisie selon une règle empirique. (Ceci évite de programmer séparément les tests des méthodes empiriques et rend les comparaisons plus justes puisque les coûts fixes de préparation des données non liés à la méthode restent les mêmes).

ANNEXE C

TABLEAUX DE RESULTATS EXPERIMENTAUX

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 500

delais : valeur minimale valeur maximale
0 500

durees : 10 nombre moyen de taches : 50

numero de serie : 1 taille de l'echantillon : 30

RESULTATS

	Jackson	DTRU
cout moyen	3	22
somme des retards (moyenne)	4859	5097
pourcentage de meilleures valeurs	96	3
ecart moyen par rapport aux meilleures valeurs	9	247

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale 0 valeur maximale 500

delais : valeur minimale 0 valeur maximale 1000

durees : 10 nombre moyen de taches : 50

numero de serie : 2 taille de l"echantillon : 30

RESULTATS

	Jackson	DTRU
cout moyen	3	27
somme des retards (moyenne)	2489	2435
pourcentage de meilleures valeurs	36	63
ecart moyen par rapport aux meilleures valeurs	71	17

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale 0 valeur maximale 500

delais : valeur minimale 0 valeur maximale 250

durees : 10 nombre moyen de taches : 50

numero de serie : 3 taille de l"echantillon : 30

RESULTATS

	Jackson	DTRU
cout moyen	3	20
somme des retards (moyenne)	8962	9069
pourcentage de meilleures valeurs	83	16
ecart moyen par rapport aux meilleures valeurs	51	158

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 250

delais : valeur minimale valeur maximale
0 500

durees : 10 nombre moyen de taches : 50

numero de serie : 4 taille de l'echantillon : 30

RESULTATS

	Jackson	DTRU
cout moyen	4	75
somme des retards (moyenne)	2012	2769
pourcentage de meilleures valeurs	100	0
ecart moyen par rapport aux meilleures valeurs	0	756

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 750

delais : valeur minimale valeur maximale
0 500

durees : 10 nombre moyen de taches : 50

numero de serie : 5 taille de l'echantillon : 30

RESULTATS

	Jackson	DTRU
cout moyen	3	9
somme des retards (moyenne)	9728	8230
pourcentage de meilleures valeurs	16	86
ecart moyen par rapport aux meilleures valeurs	1501	3

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 100

delais : valeur minimale valeur maximale
0 100

durees : 10 nombre moyen de taches : 10

numero de serie :11 taille de l'echantillon : 10

RESULTATS

	Jackson	DTRU
cout moyen	0	1
somme des retards (moyenne)	246	210
pourcentage de meilleures valeurs	60	70
ecart moyen par rapport aux meilleures valeurs	45	8

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 200

delais : valeur minimale valeur maximale
0 200

durees : 10 nombre moyen de taches : 20

numero de serie :12 taille de l'echantillon : 10

RESULTATS

	Jackson	DTRU
cout moyen	1	4
somme des retards (moyenne)	910	877
pourcentage de meilleures valeurs	70	30
ecart moyen par rapport aux meilleures valeurs	61	27

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 300

delais : valeur minimale valeur maximale
0 300

durees : 10 nombre moyen de taches : 30

numero de serie :13 taille de l"echantillon : 10

RESULTATS

	Jackson	DTRU
cout moyen	2	8
somme des retards (moyenne)	2093	2140
pourcentage de meilleures valeurs	90	10
ecart moyen par rapport aux meilleures valeurs	26	73

PARAMETRES DE LA GENERATION DES ENONCES

dates de disponibilite : valeur minimale valeur maximale
0 400

delais : valeur minimale valeur maximale
0 400

durees : 10 nombre moyen de taches : 40

numero de serie :14 taille de l"echantillon : 10

RESULTATS

	Jackson	DTRU
cout moyen	3	16
somme des retards (moyenne)	3029	3221
pourcentage de meilleures valeurs	90	10
ecart moyen par rapport aux meilleures valeurs	1	193

PARAMETRES DE LA GENERATION DES ENONCES			
dates de disponibilite :	valeur minimale	valeur maximale	
	0	500	
delais :	valeur minimale	valeur maximale	
	0	500	
durees :	10	nombre moyen de taches :	50
numero de serie : 15		taille de l"echantillon :	10

RESULTATS		
	Jackson	DTRU
cout moyen	3	22
somme des retards (moyenne)	5505	5755
pourcentage de meilleures valeurs	100	0
ecart moyen par rapport aux meilleures valeurs	0	249

ANNEXE D

EXPERIENCES PRATIQUES ANTERIEURES EN ORDONNANCEMENT

ANNEXE D

EXPERIENCES PRATIQUES EN ORDONNANCEMENT

L'auteur de cette thèse a commencé des recherches en ordonnancement dès 1975 sous la direction du Professeur ROY dans le cadre de l'équipe "Ordonnancement et gestion de production" du laboratoire LAMSADE à l'université de PARIS IX DAUPHINE.
Dans ce cadre, pendant dix ans, il a encadré chaque année deux à trois mémoires de D.E.A. dont environ la moitié était des applications industrielles réalisées pour et dans les entreprises.

Plus particulièrement, son premier sujet de thèse (abandonné car son intérêt théorique était limité) concernait la construction par ordinateur de menus diététiques personnalisés, en collaboration avec une société de service (APPLEDIM) et plusieurs services médicaux. Ce problème pouvait se modéliser comme un problème d'ordonnancement, mais l'analogie ne débouchait pas sur des résultats constructifs.

Un logiciel d'ordonnancement de tâches manuelles et informatiques, concernant la facturation téléphonique, a été mis au point pour TELESYSTEMES NANCY. Il tenait compte de tâches interrruptibles et non interruptibles ainsi que de ressources limitées. Il utilisait des règles empiriques de placement et des possibilités limitées de retour arrière ou "backtracking" de manière à réduire l'exploration de l'arborescence des solutions. Le nombre de décisions possibles explorées dépendait de paramètres choisis par l'utilisateur.

Un pilotage a été étudié pour RENAULT VEHICULES INDUSTRIELS concernant le schéma initial de l'atelier flexible BOUTHEON (St Etienne). Un ordonnancement utilisant une hiérarchie de règles empiriques a été proposé et étudié par simulation. Un phénomène de blocage ou de "verrou fatal" a été constaté lors des expériences et des propositions ont été faites pour le prévoir et l'éviter.

Pour découper les tours de service des chauffeurs d'une ligne d'autobus (CGFTE Nancy et Paris), une méthode approchée a été mise au point. Elle procède par étapes : élimination des services évidents puis des "tours manuels", construction et élimination des tâches concernant les "tours en une fois" (le chauffeur n'a pas de pause de durée et de tranche horaire réglementaires), construction de demi-tours par des méthodes empiriques et mariage des demi-tours entre eux par des méthodes d'affectation.

L'informatisation de l'ordonnancement à court et à trés court terme de l'atelier "à froid" d'une cristallerie (BACCARAT) a fait l'objet d'une thèse de docteur CNAM à NANCY, thèse qui a été co-encadrée par l'auteur. La solution proposée est une décomposition hiérarchique à deux niveaux avec agrégation des informations du niveau inférieur pour définir le niveau supérieur. Les décisions du niveau haut sont prises à chaque début de période du niveau bas. Elles concernent l'opportunité d'entrer dans l'atelier "à froid" certains ordres de fabrication plutôt que d'autres. Sachant qu'une méthode MRP a préalablement déterminé l'ensemble des ordres de fabrication à lancer pendant la période de niveau haut, il s'agit de définir les instants d'entrée des ordres de fabrication de manière à respecter approximativement (informations agrégées) les limitations de ressources (machines et hommes).
Pour cette partie, on crée des familles d'ordres de fabrication, on les classe par degré d'urgence, on entre les familles les plus urgentes et on essaie plusieurs combinaisons des suivantes de manière à utiliser au mieux le système de production. L'ordonnancement de niveau bas, à très court terme, place les opérations sur les machines, ordre de fabrication par ordre de fabrication, avec des règles empiriques de placement, (au plus tôt, au plus tard, en partant en amont et en aval par rapport à une opération fixée...).

L'exemple de la cristallerie est un très bon exemple où l'on pourrait appliquer, au niveau du très court terme, nos méthodes de décomposition. En effet, dans ce cas, s'il existe bien des sous-ateliers constitués physiquement, par contre, les produits vont et viennent dans tous les sens entre ces sous-ateliers. Un décomposition spatiale devrait permettre de définir des îlots virtuels tels que les flux de produits entre les îlots soient pauvres. En fait, le problème est plus complexe que celui étudié dans cette thèse car, à chaque poste de travail, il y a plusieurs machines en parallèle que l'on peut répartir entre les îlots.

L'approche hiérarchique à deux niveaux a également été utilisée dans le cadre du pilotage d'ateliers flexibles en présence de perturbations longues, mais cette étude, conduite dans le cadre d'ARA (automatique et robotique avancée) n'avait pas de partenaire industriel.

Une dernière application industrielle, récente, est la conception d'un algorithme de "bin packing" pour une station automatique d'emballage qui place des objets parallélépipédiques différents dans un emballage parallélépipédique. Une méthode heuristique a été développée pour la société SYSPRO filiale de PONT-A-MOUSSON S.A.

ANNEXE E

EXEMPLE DE DEROULEMENT DE LA METHODE DE DECOMPOSITION TEMPORELLE

ANNEXE E

EXEMPLE DE DEROULEMENT DE LA METHODE DE DECOMPOSITION TEMPORELLE

L'énoncé est fourni par la figure E 1.

numéro des tâches i	1	2	3	4	5	6
r_i	0	D-1	2 D	3D-2	4 D	5 D-1
d	5D-1	2D-1	6D-1	4D-2	5 D	6 D

D = durée égale des tâches

énoncé d'un problème de 6 tâches

figure E 1

Les itérations successives sont illustrées par la figure E 2.

Le détail des calculs selon l'algorithme simplifié du paragraphe 3.3.3. constitue le texte de cette annexe.

Initialisation

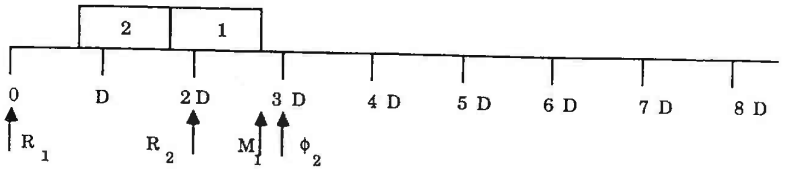
$\varphi_0 := \min_i [r_i] = 0$; $\varphi_1 := \min_i [r_i + D] = D$

$k := 0$; $0 := \emptyset$; $P_0 := \emptyset$; $M_0 := 0$

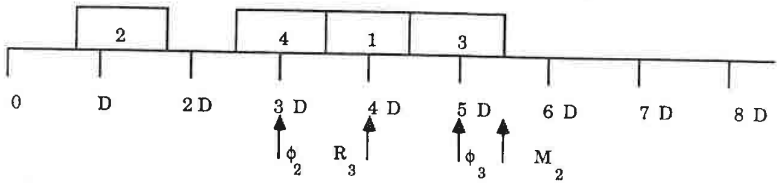
1ère itération

$k := 1$; $\pi_1 := \{i / \varphi_0 \leq r_i < \varphi_1\} = \{i / 0 \leq r_i < D\} = \{1, 2\}$

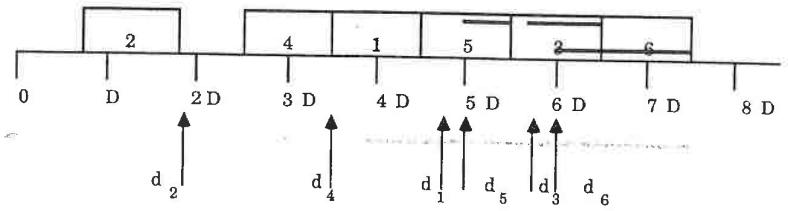
$P_1 := \{1, 2\}$;



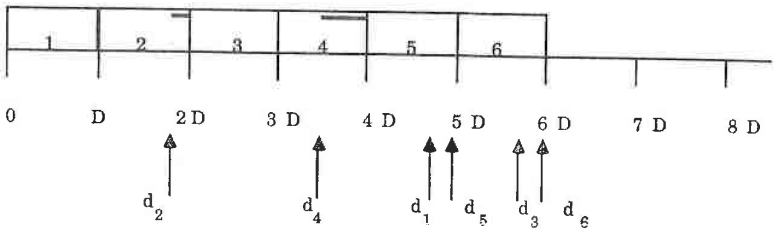
ordonnancement partiel à la fin de la première itération (somme des retards = 0)



ordonnancement partiel à la fin de la deuxième itération (somme des retards=0)



solution trouvée (somme des retards= 4 D - 5)



solution optimale (somme des retards=3)

Déroulement de la méthode de décomposition temporelle sur un exemple de 6 tâches et comparaison avec la solution optimale

figure E 2

Optimisation locale de l'itération 1

$M_0 \leq R_1 = \varphi_0$: $\omega_1 := \emptyset$ (algorithme du cas a))

σ_1 = EDD / FIFO ({1,2}) = (2,1)

(pas d'intervalle sur la machine entre 2 et 1)

$T\sigma_1 := \max [0, r_2 + D - d_2] + \max [0, r_2 + 2D - d_1] =$

$\max [0, D-1 + D - 2D + 1] + \max [0, D-1 + 2D - 5D + 1] = 0$

$d_1 > d_2$ mais $r_1 < r_2$: on examine σ_2 (bien qu'ici on sache déjà que σ_1 est un optimum local puisque sa somme des retards est nulle. Ce qui suit est réalisé par les méthodes de décomposition temporelle DTRM ou DTRUM qui cherche, après la somme des retards, à minimiser la durée totale ou makespan M_k).

$\sigma_2 := (1,2)$

(pas d'intervalle sur la machine entre 1 et2)

$T\sigma_2 := \max [0, r_1 + D - d_1] + \max [0, r_1 + 2D - d_2] =$

$\max [0, D - 5D + 1] + \max [0, 2D - 2D + 1] = 1$

Toute méthode de décomposition temporelle retient donc $\sigma 1 = (2,1)$ pour la première optimisation locale représentée en premier sur la figure E2.

$\varphi_2 := \min_{i \notin P_1} [r_i + D] = 3D$ et $M_1 := c_1 = r_2 + 2D = 3D - 1$

2ème itération

$k := 2$; $\pi_2 := \{i / \varphi_1 \leq r_i < \varphi_2\} = \{i / D \leq r_i < 3D\} = \{3,4\}$

$P_2 := \{1, 2, 3, 4\}$

Optimisation locale de l'itération 2

$M_1 = 3D - 1 > R_2 = \min_{i \in \pi_2} [r_i] = 2D$

d'où $\omega_2 := \{i / r_i < R_2 \text{ et } c_i > R_2\} = \{1\}$

$\pi_2 \cup \omega_2 = \{1,3,4\}$ et $q_2 := c_2 = 2D - 1 = r'_1$

$r'_3 = \max [q_2, r_3] = \max [2D - 1, 2D] = 2D$

$r'_4 = \max [q_2, r_4] = \max [2D - 1, 3D - 2] = 3D - 2$

$\sigma_2 :=$ EDD / FIFO ({1,3,4}) = (4,1,3)
(pas d'intervalle entre les 3 tâches sur la machine car on commence par une tâche de π_k)

$$\begin{aligned} T\sigma_2 := \rho_2 &+ \max [0, r'_4 + D - d_4] \\ &+ \max [0, r'_4 + 2D - d_1] \\ &+ \max [0, r'_4 + 3D - d_3] \\ = 0 &+ \max [0, 3D - 2 + D - 4D + 2] \\ &+ \max [0, 3D - 2 + 2D - 5D + 1] \\ &+ \max [0, 3D - 2 + 3D - 6D + 1] = 0 \end{aligned}$$

(comme précédemment, on poursuit bien que σ_2 soit optimal)

$\sigma^* := \sigma_2$; MT := 0 ; $r_{min} := 3D - 2$;

1. u := 1 (1 placé en tête devant EDD / FIFO - {1})
   q := $c_1 = 3D - 1$

   $(r_u = r_1 = 0) < (r_{min} = 3D - 2)$
   $\mathcal{T} = \{1\} \cup \{3,4\} - \{1\} = \{3,4\}$
   $T := MT - \rho_u = 0$

optloc ($\mathcal{T}$,T,q,s) :

s1 := EDD / FIFO ($\mathcal{T}$) := (4,3)
(pas d'intervalle entre 4 et3)
$r_{min} := r''_4 = \max [r'_4, q] = \max [3D - 2, 3D - 1] = 3D - 1$

$$\begin{aligned} T_{s1} := &\max [0, r''_4 + D - d_4] + \\ &\max [0, r''_4 + 2D - d_3] \\ = &\max [0, 3D - 1 + D - 4D + 2] + \\ &\max [0, 4D - 1 + D - 6D + 1] = 1 \end{aligned}$$

$T_{s1} > T$: s1 n'est pas retenue
$j^* := \emptyset$; $s^* := \emptyset$
j := 3
$r''_3 := \max [r_3, q] = \max [2D, 3D - 1] = 3D - 1$
$r''_3 \geq r_{min}$: la séquence (3,4) n'est pas examinée.

fin de "optloc" avec $s^* = \emptyset$, d'où σ^* reste $\sigma 2$.

2. u := ^ (1 ne sera pas placé en tête).
   q := $q_2 = 2D - 1$;

   $(r_u := R_2 = 2D) < (r_{min} = 3D - 2)$
   $\mathcal{T} := \{1,3,4\}$;
   T := MT = 0

optloc ($\mathcal{T}$, T,q,s) :

$s_1 :=$ EDD / FIFO ($\mathcal{T}$) = σ_2 = (4,1,3) ;
$r_{min} := r''_4 = \max [r_4, q] = \max [3D - 2, 2D - 1] = 3D - 2$
$T_{s1} := 0$ (s_1 déjà retenu) ($j^* := 4$; $s^* := s_1$)

on n'essaie pas j = 1 car 1 ne doit pas être placé en tête.

j := 3
$(r''_3 = \max [r_3, q] = \max [2D, 2D - 1]) < (r''_4 = 3D - 2)$
dans le cas où D > 2
$r_{min} := r''_3 = 2D$
$s_2 :=$ (3,4,1)
(pas d'intervalle entre les tâches)

$$T_{s2} : = \max [0, r''_3 + D - d_3] + \max [0, r''_3 + 2D - d_4] + \max [0, r''_3 + 3D - d_1] = \max [0, 3D - 6D + 1] + \max [0, 4D - 4D + 2] + \max [0, 5D - 5D + 1] = 3$$

$T_{s2} > T$: s_2 n'est pas retenue.

fin de "optloc" avec $s^* = s_1$ (ou ø).

Il n'y a pas d'autre choix pour u.

(4,1,3) est la solution optimale locale retenue à la deuxième itération quelle que soit la méthode de décomposition temporelle utilisée (DTRU ou DTRUM).
La figure E2 présente l'ordonnancement partiel à la fin de cette deuxième itération.

$\varphi_3 : = \min_{i \notin P_2} [r_i + D] = 5D$; $M_2 : = c_3 = r_4 + 3D = 6D - 2$

3ème itération

$k : = 3$; $\pi_3 : = \{i / \varphi_2 \leq r_i < \varphi_3\} = \{i / 3D \leq r_i < 5D\} = \{5,6\}$
$P3 : = \{1,2,3,4,5,6\}$

Optimisation locale de l'itération 3

$$M_2 = 6D - 2 > R_3 = \min_{i \in \pi_3} [r_i] = 4$$

d'où $\omega_3 : = \{i / r_i < R_3 \text{ et } c_i > R_3\} = \{1,3\}$

$\pi_3 \cup \omega_3 = \{1,3,5,6\}$ et $q_3 : = c_4 = 4D - 2$

$r'_1 = 4D - 2$; $r'_3 = 4D - 2$; $r'_5 = 4D$; $r'_6 = 5D - 1$

$\sigma 3 = EDD / FIFO (\{1,3,5,6\}) = (1,5,3,6)$
(pas d'intervalle entre 5, 3 et 6)

$$T\sigma_3 : = \rho_2 + \rho_4 + \max [0, r'_1 + D - d_1] + \max [0, \max [r'_1 + D, r'_5] + D - d_5] + \max [0, \max [r'_1 + D, r'_5] + 2D - d_3] + \max [0, \max [r'_1 + D, r'_5] + 3D - d_6] = \max [0, 5D - 2 - 5D + 1] + \max [0, 6D - 2 - 5D] + \max [0, 7D - 2 - 6D + 1] + \max [0, 8D - 2 - 6D] = (D - 2) + (D - 1) + (2D - 2) = 4D - 5$$

$\sigma^* : = \sigma_3$; $MT : = 4D - 5$; $r_{min} : = r'_1 = 4D - 2$

comme il n'y a pas de tâche telle que $r'_j < r'_1$, et que les tâches de (1,5,3,6) sont contiguës, on ne peut améliorer ni la somme des retards (propriétés de dominance), ni la durée totale.

La solution localement optimale de cette troisième et dernière itération est donc (1,5,3,6).

L'ordonnancement obtenu, ainsi que la solution globalement optimale terminent la figure E2.

Cet exemple illustre le fait que la méthode de décomposition temporelle ne fournit pas toujours la solution globale optimale dans le cas particulier des durées égales. Par exemple, ici, on commet une erreur de $4D - 2$.

C'est donc une méthode polynômiale approchée.

ANNEXE F

GLOSSAIRE DES NOTATIONS UTILISEES

ANNEXE F
Glossaire des notations utilisées
avec les titres des paragraphes où elles sont définies

2. METHODES DE DECOMPOSITON EN ORDONNANCEMENT

2.1. INTRODUCTION

m	nombre de machines
n	nombre de produits
r(i)	date de disponibilité du produit i
d(i)	délai du produit i
q(i)	nombre d'opérations sur le produit i
pp(i)	durée totale minimale de présence dans l'atelier du produit i
c(i)	date planifiée de fin d'exécution du produit i
w_i	pénalité appliquée au produit i par unité de temps de retard
p(i,k)	durée de la k-ième opération sur le produit i
$\mu(i,k)$	machine utilisée pour la k-ième opération sur le produit i
s(i,k)	durée du réglage de la machine $\mu(i,k)$ avant la k-ième opération sur le produit i (le produit i n'est pas nécessairement arrivé)
t(i,k)	date planifiée de début d'exécution de la k-ième opération sur le produit i
$\tau(m_1,m_2)$	durée du transport de la machine m_1 à la machine m_2

2.2. DECOMPOSITION TEMPORELLE

L	nombre d'itérations, choisi ou calculé à partir des données, de la méthode de décomposition temporelle
φ	suite temporelle finie d'instants $\varphi_0, \varphi_1,\ldots, \varphi_{L+1}$
π	suite de sous-ensembles de produits $\pi_0,\pi_1,\ldots,\pi_L$
O_k	ordonnancement partiel obtenu à la fin de la k-ième itération construit à partir de O_{k-1} et π_k

2.3. DECOMPOSITION SPATIALE

n_c nombre de classes dans les partitions

Z^o ensemble des indices des produits

Z ensemble des indices des produits types

N cardinalité de Z = nombre de produits types

P_Z partition de Z en n_c classes : $Z_1, Z_2, \ldots, Z_{nc}$

Y^o ensemble des indices des machines

Y ensemble des indices des machines types

M cardinalité de Y = nombre de machines types

P_Y partition de Y en n_c classes : $Y_1, Y_2, \ldots, Y_{nc}$

$\alpha(i)$ nombre de produits i_o correspondant au produit type i, considéré comme le poids du produit type i

$\beta(j)$ nombre de machines j_o correspondant à la machine type i

$a(i_o j_o)$ temps total de passage du produit i sur la machine j

$a''(i,j)$ temps total de passage d'un produit i_o correspondant au produit type i sur l'une des machines j_o correspondant à la machine type j (ce temps est indépendant de i_o et j_o)

$v(i,j)$ durée, normalisée entre 0 et 1, du passage d'un produit de type i sur une machine de type j

$u(i,j)$ variable bivalente indiquant l'utilisation ou non de la machine type j par le produit type i

$\Delta^{\lambda}_h(P_Z,P_Y)$ évaluation d'une classification croisée définie par le couple (P_Z,P_Y)

h paramètre, compris entre 0 et 1, permettant d'agréger l'évaluation du critère correspondant aux blocs diagonaux avec l'évaluation du critère correspondant à l'extérieur de ces blocs

λ paramètre, compris entre 0 et 1, qui module l'importance donnée aux v(i,j), différents de 0 et de 1, dans l'évaluation

$\Delta u_h(P_Z,P_Y)$ évaluation asymptotique d'une classification croisée définie par le couple de partitions (P_Z,P_Y) à partir des variables bivalentes u(i,j)

ζ^k centre d'inertie de Z_k

$a_s(j)$ participation de la colonne j à l'évaluation du critère si elle est placée dans Y_s

$b_s(i)$ participation de la ligne i à l'évaluation du critère si elle est placée dans Z_s

$m^{oo}(\hat{i})$ nombre de machines dans l'îlot $\hat{i}$

$n^{oo}(\hat{i})$ nombre de sous-produits dans l'îlot $\hat{i}$

$cmm(\hat{i})$ charge moyenne des machines dans l'îlot $\hat{i}$

$mmsp(\hat{i})$ marge moyenne des sous-produits dans l'îlot $\hat{i}$

$\hat{i}am(\hat{o})$ numéro de l'îlot auquel appartient la machine $\hat{o}$

$\hat{i}ap(\hat{u})$ numéro de l'îlot auquel appartient le produit $\hat{u}$ ou "îlot mère" du produit $\hat{u}$

$sp(\hat{u})$ nombre de sous-produits obtenus à partir du produit $\hat{u}$

$marge(\hat{u})$ marge totale du produit $\hat{u}$

$ra(\hat{u},j)$ instant de début de fabrication au plus tôt du j-ème sous-produit du produit $\hat{u}$ (délai externe invariant)

$r^{oo}(\hat{u},j)$ instant de début de fabrication au plus tôt du j-ème sous-produit du produit $\hat{u}$ (délai interne lié au partage des marges et variable avec les itérations des méthodes de résolution)

$da(\hat{u},j)$ instant de fin de fabrication au plus tard du j-ème sous-produit du produit $\hat{u}$ (délai externe)

$d^{oo}(\hat{u},j)$ instant de fin de fabrication au plus tard du j-ème sous-produit du produit $\hat{u}$ (délai interne)

$\hat{i}asp(\hat{u},j)$ numéro de l'îlot où s'exécute le j-ème sous-produit du produit $\hat{u}$

$pp^{oo}(\hat{u},j)$ durée totale minimale de présence dans l'îlot $\hat{i}asp(\hat{u},j)$ du j-ème sous-produit du produit $\hat{u}$

$q^{oo}(\hat{u},j)$ nombre d'opérations du j-ème sous-produit du produit $\hat{u}$

$p^{oo}(\hat{u},j,k)$ durée de la k-ème opération sur le j-ème sous-produit du produit $\hat{u}$

$m^{oo}(\hat{u},j,k)$ machine utilisée pour la k-ème opération sur le j-ème sous-produit du produit $\hat{u}$

$t^{oo}(\hat{u},j,k)$ instant de début planifié pour la k-ème opération du j-ème sous-produit du produit $\hat{u}$ dans l'ordonnancement T^{oo}

2.4. DECOMPOSITION SPATIALE ET TEMPORELLE

$\hat{i}_k$	numéro de l'îlot à considérer à la k-ème itération
SPA_k	ensemble des indices des sous-produits accessibles au début de l'itération k
WE(û,j)	pénalité appliquée au sous-produit (û,j) par unité de retard par rapport aux délais externes
WI(û,j)	pénalité appliquée au sous-produit (û,j) par unité de retard par rapport aux délais internes

NOM DE L'ETUDIANT : Mademoiselle PORTMANN M. Claude

NATURE DE LA THESE : Doctorat d'Etat ès sciences

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le - 7 SEP. 1987 n° 1806

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. [illegible] LE PRESIDENT

Thèse d'état de Marie-Claude PORTMANN
soutenue le 18 septembre 1987
à l'Université de Nancy 1

sujet :

<<Méthodes de décomposition spatiale et temporelle en ordonnancement de la production>>

résumé :

Les problèmes d'ordonnancement sont, pour la plupart, NP-difficiles. Aussi, le coût exponentiel des algorithmes exacts utilisés pour les résoudre explique la prolifération d'algorithmes approchés polynômiaux.

Dans ce travail, nous proposons de nouvelles méthodes approchées par décomposition qui utilisent des algorithmes exacts pour des sous-problèmes du problème initial. On cherche ainsi à minimiser l'écart entre la valeur de l'objectif de la solution trouvée par la méthode approchée et la valeur optimale si on se donnait le temps (astronomique) de l'obtenir par un algorithme exact.

Dans le chapitre 1, nous situons notre travail par rapport aux recherches antérieures.

Dans le chapitre 2, nous présentons nos nouvelles approches par décomposition : temporelle et/ou spatiale avec l'utilisation pour cette dernière d'une méthode de classification croisée.

Dans les chapitres 3 et 4, nous évaluons les méthodes proposées en qualité et en coût d'obtention de la solution. En particulier, nous démontrons des théorèmes de dominance et des théorèmes d'évaluation de l'erreur dans le pire des cas pour un cas particulier de décomposition temporelle (ceci nous amène à définir de nouvelles familles d'ordonnancement et à en analyser certaines propriétés). Nous comparons nos différentes méthodes par décomposition en réalisant des séries d' expériences pour le cas général.

Ces évaluations montrent l'intérêt des nouvelles méthodes proposées.

mots clés :

algorithmique - ordonnancement d'atelier - décomposition spatiale et/ou temporelle - classification croisée - évaluation dans le pire des cas - propriétés de dominance

Thèse d'état de Marie-Claude PORTMANN
soutenue le 18 septembre 1987
à l'Université de Nancy 1

sujet :

<<Méthodes de décomposition spatiale et temporelle en ordonnancement de la production>>

résumé :

Les problèmes d'ordonnancement sont, pour la plupart, NP-difficiles. Aussi, le coût exponentiel des algorithmes exacts utilisés pour les résoudre explique la prolifération d'algorithmes approchés polynômiaux.

Dans ce travail, nous proposons de nouvelles méthodes approchées par décomposition qui utilisent des algorithmes exacts pour des sous-problèmes du problème initial. On cherche ainsi à minimiser l'écart entre la valeur de l'objectif de la solution trouvée par la méthode approchée et la valeur optimale si on se donnait le temps (astronomique) de l'obtenir par un algorithme exact.

Dans le chapitre 1, nous situons notre travail par rapport aux recherches antérieures.

Dans le chapitre 2, nous présentons nos nouvelles approches par décomposition : temporelle et/ou spatiale avec l'utilisation pour cette dernière d'une méthode de classification croisée.

Dans les chapitres 3 et 4, nous évaluons les méthodes proposées en qualité et en coût d'obtention de la solution. En particulier, nous démontrons des théorèmes de dominance et des théorèmes d'évaluation de l'erreur dans le pire des cas pour un cas particulier de décomposition temporelle (ceci nous amène à définir de nouvelles familles d'ordonnancement et à en analyser certaines propriétés). Nous comparons nos différentes méthodes par décomposition en réalisant des séries d' expériences pour le cas général.

Ces évaluations montrent l'intérêt des nouvelles méthodes proposées.

mots clés :

algorithmique - ordonnancement d'atelier - décomposition spatiale et/ou temporelle - classification croisée - évaluation dans le pire des cas - propriétés de dominance