N° d'ordre :

THESE

présentée à

L'UNIVERSITE DE METZ

pour l'obtention du titre de

DOCTEUR DE L'UNIVERSITE DE METZ

Spécialité :

AUTOMATIQUE

par

Chengbin CHU

Sujet de la thèse :

NOUVELLES APPROCHES ANALYTIQUES ET CONCEPT DE MEMOIRE ARTIFICIELLE POUR DIVERS PROBLEMES D'ORDONNANCEMENT

Soutenue le 25 Septembre 1990 devant le Jury composé de :

M.	**Jacques CARLIER**	
M.	**Jacques ERSCHLER**	Rapporteurs
M.	**Yvon GARDAN**	
M.	**Claude LAURENT**	
Mlle	**Marie-Claude PORTMANN**	
M.	**Jean-Marie PROTH**	Examinateurs

REMERCIEMENTS

Ce travail a été effectué dans l'équipe SAGEP de l'INRIA-Lorraine où j'ai pu bénéficier d'un encadrement efficace et d'une ambiance chaleureuse. Que les membres de cette équipe reçoivent toute ma gratitude.

Je tiens à remercier tout particulièrement M. Jean-Marie Proth, Directeur de Recherche à l'INRIA, de m'avoir accueilli dans son équipe, et de m'avoir encouragé tout au long de ce travail. Ses solides connaissances scientifiques m'ont permis de travailler sur de nouvelles voies de recherche.

Je souhaite exprimer mes plus vifs remerciements à Mlle Marie-Claude Portmann, Professeur à l'Ecole des Mines de Nancy, Conseillère Scientifique à l'INRIA, qui m'a suivi tout au long de cette thèse, a consacré de nombreuses heures à l'amélioration de ce document.

Je tiens à exprimer toute ma reconnaissance à M. Jacques Carlier, Professeur à l'Unviversité de Technologie de Compiègne, et M. Jacques Erschler, Professeur à l'Institut National des Sciences Appliquées de Toulouse d'avoir accepté d'être rapporteurs de cette thèse et membres du jury.

Je remercie vivement les personnalités scientifiques qui se sont intéressées à mon travail et qui ont bien voulu me faire l'honneur de participer à ce jury, à savoir:

M. Yvon Gardan, Professeur à l'Univeristé de Metz,

M. Claude Laurent, Professeur à l'Université de Metz.

TABLE DE MATIERES

NOTATIONS

Notations Générales

n: nombre de produits

m: nombre de machines dans l'atelier

r_i: instant d'arrivée au plus tôt du produit i ("ready-time", "release-time" ou "arrival-time")

p_i: durée totale des opérations qui permettent de réaliser le produit i ("processing-time")

d_i: instant de fin souhaitée au plus tard pour le produit i ("due-date")

g_i: nombre d'opérations pour le produit i

C_i: instant de fin du produit i ("completion time")

$\mu_{i,j}$: machine utilisée par l'opération j du produit i

$l_{i,j}$: durée de l'opération j du produit i

$t_{i,j}$: instant de début

F_i: temps de presence de la tâche i, i.e. durée pendant laquelle la tâche i reste dans l'atelier ("flow time")

W_i: durée d'attente de la tâche i ("waiting time")

L_i: écart entre la date de fin réelle et la date de fin souhaitée de la tâche i ("lateness")

T_i: retard de la tâche i ("tardiness")

Chapitre 2

p: dimension de l'espace des occurences d'énoncé après caractérisation des jeux de données numériques pour une application donnée

γ: nombre de critères pour une application donnée

e: indice courant utilisé pour désigner une occurrence d'énoncé d'une application donnée

ω: nombre d'occurrences d'énoncé pour une application donnée lors de la construction de la mémoire artificielle correspondante

E: ensemble des indices des occurrences d'énoncé pour une application donnée lors de la construction de la mémoire artificielle correspondante ($E=\{1,2,\ldots,\omega\}$)

h: indice courant utilisé pour désigner une heuristique pour une application donnée

η: nombre d'heuristiques utilisées pour une application donnée

H: ensemble des indices des heuristiques utilisées pour une application donnée ($H=\{1,2,\ldots,\omega\}$)

v_e^h: valeur normalisée du critère considéré obtenue en appliquant l'heuristique d'indice h à l'occurrence d'énoncé d'indice e

q: nombre de modalités différentes pour le critère considéré

Γ_i^h: classe de modalité i pour le critère considéré et l'heuristique h

s: seuil minimal pour la cardinalité des classes

ME: métrique euclidienne

MD_S: métrique diagonale pour l'ensemble S

MM_S: métrique de Mahalanobis pour l'ensemble S

$\delta_M(x,y)$: distance du point x au point y pour la métrique M

Chapitre 3

N: ensemble des indices de toutes les tâches à ordonnancer

B_i: ensemble des indices des tâches précédant i dans la séquence considérée

A_i: ensemble des indices des tâches suivant i dans la séquence considérée

ξ_i: indice de la dernière tâche de B_i (c'est-à-dire la tâche précédant immédiatement i)

ϕ_i: date d'achèvement de la tâche ξ_i, c'est-à-dire $\phi_i=C_{\xi_i}$ (si $B_i=\emptyset$, on pose $\phi_i=0$)

S: ordonnancement partiel considéré, ou ensemble des tâches de cet ordonnancement partiel

S|i: ordonnancement partiel obtenu en ordonnançant la tâche i immédiatement derrière l'ordonnancement partiel donné S, ou encore ensemble des tâches de cet ordonnancement partiel

$\Sigma(S,i)$: ordonnancement obtenu en ordonnançant optimalement les tâches de N-(S|i) immédiatement derrière S|i

$R(i,\Delta)$: date d'exécution au plus tôt de la tâche i à l'instant Δ, $R(i,\Delta)=\max(r_i,\Delta)$ (pour simplifier, nous notons R_i, lorsqu'il n'y a pas d'ambiguïté)

$\Phi(i,\Delta)$: date de fin au plus tôt de la tâche i à l'instant Δ, $\Phi(i,\Delta)=R(i,\Delta)+p_i$

$X_i(\Delta)$: valeur du critère X, X étant le symbole du critère à optimiser de la tâche i en l'ordonnançant au plus tôt à partir de l'instant Δ (on note X_i lorsqu'il n'y a pas d'ambiguïté)

$X_{i,j}(\Delta)$: valeur du critère X, X étant le symbole du critère à optimiser de deux tâches consécutives i et j (i suivie de j), exécutées au plus tôt à partir de l'instant Δ (on note X_{ij} lorsqu'il n'y a pas d'ambiguïté)

X(i,O): valeur du critère X, X étant le symbole du critère à optimiser de la tâche i dans l'ordonnancement O

Chapitre 4

ν: nombre de classes

M: moyenne du nombre d'occurrences d'énoncé dans une classe

E: écart-type du nombre d'occurrences d'énoncé dans une classe

%q: pourcentage d'occurrences d'énoncé dans la classe de modalité q

ψ: nombre de régions obtenues

ϕ: nombre total de points flous

ϕq: nombre de points flous dans la classe de modalité q

NG: nombre de fois où la mémoire artificielle donne une valeur du critère strictement meilleure que la stratégie habituelle

NP: nombre de fois où la mémoire artificielle donne une valeur du critère strictement moins bonne que la stratégie habituelle

NI: nombre de fois où la mémoire artificielle demande un rafraîchissement

NH: nombre de fois où la mémoire artificielle propose la stratégie habituelle

VH: valeur moyenne du critère fournie par la stratégie habituelle

VM: valeur moyenne du critère fournie par la mémoire artificielle

TH: temps total de calcul en appliquant systématiquement la stratégie habituelle

TM: temps total de calcul en utilisant la mémoire artificielle

TRI: nombre de fois où l'heuristique donne la meilleure valeur au critère, critère principal de tri des heuristiques dans les tableaux et pour l'utilisation de la mémoire

MC: moyenne du critère pour chaque heuristique sur l'ensemble des occurrences d'énoncé

INTRODUCTION

Les applications concrètes donnent naissance à de très nombreux problèmes d'ordonnancement (ordonnancement de la production à moyen, court ou très court terme, ordonnancement de projets, problèmes d'emploi du temps, problèmes de transport ou de tournées ...).

Malheureusement, à part quelques exceptions (ordonnancement de projets à moyens illimités ou problèmes simples d'affectation ou d'atelier, par exemple), il existe très peu de problèmes que l'on peut résoudre de manière exacte par des algorithmes polynomiaux [1] (en effet la plupart de ces problèmes sont NP-difficiles).

En outre, il existe une multitude de problèmes différents selon les hypothèses particulières de fonctionnement des systèmes de production ou de transport.

Aussi, dans la littérature, on trouve, pour chaque famille d'hypothèses étudiées, un ensemble éventuellement vide de méthodes exactes que l'on ne peut utiliser que sur des problèmes de petite taille et une liste de méthodes approchées qui sont plus ou moins bonnes selon les exemples sur lesquelles on les applique.

Le problème que l'on cherche à résoudre dans cette thèse est le suivant:

étant donné un problème d'ordonnancement auquel on peut associer un ou plusieurs critères d'appréciation de la valeur des solutions et étant donné un jeu de valeurs numériques pour les paramètres de ce problème appelé ici "occurrence d'énoncé", comment choisir parmi les nombreuses heuristiques, proposées dans la littérature ou imaginées par la personne qui doit résoudre ce problème particulier, celle qui a le plus de chance de donner de bonnes valeurs aux critères considérés.

C'est pour cela que l'on a construit une mémoire artificielle qui cherche à utiliser l'expérience passée de manière à mieux choisir dans l'avenir.

---

[1] Si l'on prend l'hypothèse généralement admise que P est différent de NP.

Elle repose sur l'hypothèse de base suivante: si deux occurrences d'énoncé numériques correspondant à un problème donné "se ressemblent" alors les heuristiques ont des performances voisines sur ces deux occurrences d'énoncé. Par exemple, si l'objectif est de minimiser un critère donné, alors les heuristiques qui donnent une "bonne" valeur au critère pour ces deux occurrences d'énoncé sont les mêmes.

La conséquence de cette hypothèse est que l'on peut, pour une nouvelle occurrence d'énoncé, chercher sa proximité avec des sous-ensembles d'occurrences d'énoncé antérieures ayant des comportements similaires et en déduire l'heuristique à appliquer pour sa résolution en fonction du critère choisi.

L'idée de la mémoire artificielle a été proposée dans notre équipe (SAGEP de l'INRIA) en 1983 et avait été testée à l'époque sur une petite maquette liée à un problème d'ordonnancement d'un atelier flexible pour des produits en plusieurs exemplaires avec comme critère principal l'équilibre des flux en volume pour les différents produits.

Dans cette thèse, nous appliquons cette idée à divers problèmes d'ordonnancement: problèmes d'ordonnancement d'ateliers de type Job-shop, problèmes d'ordonnancement à une machine; problème de container loading (bin-packing à trois dimensions).

Pour certains de ces problèmes d'ordonnancement, nous ne nous sommes pas contentés des heuristiques données dans la littérature, mais nous avons cherché à construire de nouvelles heuristiques efficaces. En particulier, pour les problèmes d'ordonnancement à une machine, nous avons construit des heuristiques efficaces, basées sur des études théoriques. Ces heuristiques ont ensuite été généralisées aux problèmes d'ordonnancement de type Job-shop.

Pour évaluer les méthodes approchées, nous avons construit des méthodes exactes qui nous ont permis des comparaisons intéressantes pour des problèmes de taille limitée (procédure par Séparation et Evaluation (PSE) explorant seulement des ensembles dominants et utilisant des bornes de bonne qualité).

La présentation de cette thèse est organisée de la manière suivante.

Le chapitre 1, chapitre essentiellement bibliographique, est une introduction générale aux problèmes d'ordonnancement et à leur résolution.

La première partie du chapitre 2 est consacrée aux concepts et aux outils qui permettent de construire une structure générale de mémoire artificielle.

La structure générale de la mémoire artificielle utilise:
- une projection des occurrences d'énoncé dans un espace de caractéristiques lié au problème particulier considéré de manière à rechercher les proximités dans $\mathbb{R}^p$ avec p fixé,
- une simulation de toutes les heuristiques sur un sous-ensemble initial d'occurrences d'énoncé choisies les plus dispersées possibles,
- une échelle de valeurs permettant de recouvrir les occurrences d'énoncé par des classes (classes qui ne sont en général, ni convexes, ni connexes),
- une technique de partitionnement des classes en régions connexes en utilisant la distance de Mahalanobis ou la distance diagonale légèrement moins performante mais considérablement moins coûteuse,
- une technique d'affectation d'une nouvelle occurrence d'énoncé à une ou plusieurs régions de manière à assurer le meilleur choix d'heuristique pour un critère donné.

La deuxième partie du chapitre 2 montre comment construire les modules spécifiques propres aux problèmes considérés dans cette thèse:
- ordonnancement d'ateliers de type job-shop pour différents critères,
- ordonnancement d'ateliers à une machine avec dates d'arrivées des tâches non identiques pour différents critères,
- ordonnancement du placement de boîtes parallélépipédiques dans un emballage parallélépipédique ("bin-packing" ou "container loading").

Le chapitre 3 est consacré aux études théoriques concernant des problèmes à une machine avec des dates d'arrivée des tâches différentes: minimisation de la somme des retards, minimisation de la somme des temps de présence (flow-times). Pour ces deux problèmes, nous définissons des ensembles

dominants: T-actif pour la minimisation de la somme des retards, F-actif pour la minimisation de la somme des temps de présence. Plusieurs heuristiques efficaces sont construites qui fournissent une solution dans cet ensemble dominant. En utilisant ces ensembles dominants, et des bornes efficaces, nous construisons aussi des algorithmes exacts pour ces deux problèmes.

Le chapitre 4 est consacré aux expériences réalisées pour les trois applications et à l'évaluation des mémoires artificielles dans le cas monocritère.

Chapitre 1

PROBLEMES D'ORDONNANCEMENT

1.1. INTRODUCTION

Dans ce chapitre, nous présentons les problèmes que nous abordons dans cette thèse dans leur contexte général et nous précisons les hypothèses particulières que nous retenons. Nous nous intéressons à deux grandes familles de problèmes d'ordonnancement: les problèmes classiques d'ordonnancement d'atelier et un problème moins classique de "bin-packing". Le problème de bin-packing considéré est un problème de "container-loading": il s'agit d'ordonnancer le placement d'objets parallélépipédiques dans un emballage parallélépipédique.

1.2. PROBLEMES D'ORDONNANCEMENT D'ATELIER

Les problèmes d'ordonnancement sont importants dans la vie économique. Pour produire plus ou pour faire plus de bénéfice à un prix aussi bas que possible, il convient d'optimiser le processus de production. C'est pour cette raison que nous nous intéressons aux problèmes d'ordonnancement.

Bien que des problèmes d'ordonnancement se posent dans de nombreux domaines (ordonnancement d'atelier, ordonnancement de projets, ordonnancement de tâches dans les services publics, établissement d'un emploi du temps, etc...), nous nous intéressons ici uniquement aux problèmes d'ordonnancement d'ateliers dans les entreprises manufacturières.

Nous allons définir ce qu'est un problème d'ordonnancement d'atelier ainsi que les solutions de ce problème appelées "ordonnancements", et rappeler les résultats obtenus par d'autres chercheurs.

1.2.1. DEFINITIONS

Un **problème d'ordonnancement** consiste à affecter des tâches à des moyens de fabrication au cours du temps pour effectuer un ensemble de travaux de manière à optimiser certain(s) critère(s), tout en respectant les contraintes techniques de fabrication.

Dans ce type de problème, un **travail** correspond au traitement d'un produit par l'atelier. Un travail est constitué d'une succession d'**opérations**. Les opérations sont effectuées par des machines. Certains auteurs ([Blazewicz et

al. 1987]) définissent plusieurs types de ressources (consommables, renouvelables et doublement limitées (voir le paragraphe 1.2.1.2. pour les définitions)). Dans le cas d'une machine (voir le chapitre 3), un travail se réduit à une opération. Nous utilisons alors le terme de "tâche" pour le désigner.

La solution d'un problème d'ordonnancement général doit répondre à deux questions:
-quand?
-avec quels moyens?

Une solution répondant à ces questions est appelée **ordonnancement**. Une méthode permettant de construire un ordonnancement est appelée **algorithme ou méthode de résolution. Un ordonnancement réalisable** est un ordonnancement qui respecte toutes les contraintes du problème. Nous utilisons le terme "ordonnancement", pour simplifier, pour représenter un ordonnancement réalisable (sauf indication contraire). Un ordonnancement est souvent représenté par un diagramme de Gantt (voir la figure 1.1 où l'opération j du produit i est désignée par i,j).

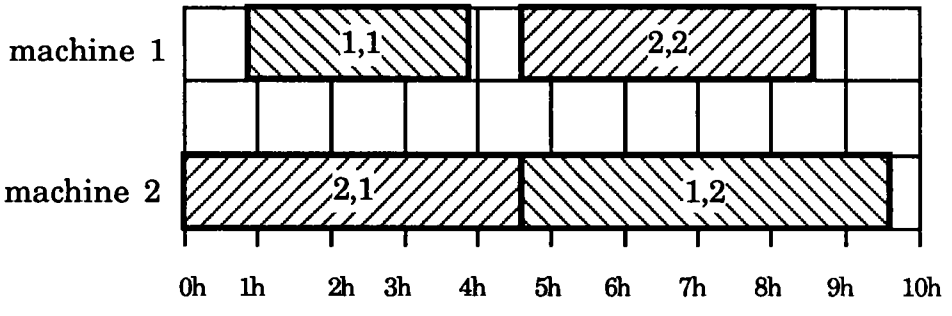


Fig. 1.1. Diagramme de Gantt

L'énoncé général d'un problème d'ordonnancement d'atelier comporte quatre types d'informations:
1) Les produits et les opérations à effectuer.
2) Les quantités et les types de ressources.
3) Les contraintes endogènes (gammes, capacités) et exogènes (délais).
4) Les critères pour évaluer la solution.

Les différents types de contraintes et de critères font apparaître une diversité de problèmes d'ordonnancement. Les hypothèses potentielles et les hypothèses retenues sont présentées dans la section 1.2.1.2.

1.2.2. DIVERSITE DES PROBLEMES ET HYPOTHESES RETENUES

a) Les produits

Les informations sur les produits sont les **dates d'arrivée au plus tôt** dans l'atelier, les **durées** des opérations et les **gammes**, ainsi que les **dates de fin au plus tard**.

Les produits peuvent arriver dans l'atelier en même temps, ou au fur et à mesure que le temps progresse. Dans le premier cas, les arrivées sont dites **identiques**, alors que dans le deuxième cas, elles sont dites **dynamiques**. Nous nous plaçons dans le cas le plus général où les arrivées sont dynamiques.

En ce qui concerne les dates de fin au plus tard, deux cas sont possibles: ou bien il s'agit de **délais impératifs** (en anglais "deadlines"), et il est alors possible qu'il n'existe aucun ordonnancement réalisable ([Carlier 1984], [Bagchi & Ahmadi 1987], [Potts & Van Wassenhove 1983]), ou bien il s'agit de dates de fin souhaitées (en anglais "due dates") que nous désignons ici par **délais**, le dépassement autorisé pouvant entraîner des pénalités. C'est ce dernier cas que nous considérons.

Les produits s'obtiennent par l'exécution d'une séquence d'opérations. Chaque opération doit passer sur une machine et rester sur cette machine pendant un certain temps appelé durée de l'opération. En général le passage d'une opération à l'autre est spécifié. L'ensemble de ces passages est appelé **gamme**. Nous allons présenter les différents types d'opérations et de gammes.

**Les opérations*

Une opération est une succession d'actions élémentaires réalisables de manière consécutive sur une machine. C'est une phase de fabrication du produit que l'on ne souhaite pas détailler plus finement au niveau du problème d'ordonnancement considéré. Un produit est terminé si et seulement si toutes les opérations de la gamme sont réalisées.

On distingue trois types d'opérations:

Les **opérations non interruptibles** – Une fois qu'une opération commence à être exécutée sur une machine, l'exécution ne peut pas être interrompue avant qu'elle ne soit terminée (par exemple, une opération faisant intervenir conjointement des traitements chimiques, thermiques et mécaniques).

Les **opérations interruptibles** – L'interruption est autorisée. Pour finir cette opération, il faut qu'elle se poursuive à partir de l'état où elle a été interrompue. ([Kämpke 1987]) (par exemple, lorsque le produit est en réalité un lot de produits élémentaires identiques traités conjointement ou lorsque l'opération est une tâche utilisant le CPU d'un ordinateur).

Les **opérations préemptives répétées** – L'interruption est autorisée, mais l'exécution d'une opération doit être recommencée depuis le début. C'est-à-dire que le travail déjà fait avant l'interruption a été perdu. ([Chandra 1979]) (par exemple, réchauffement d'une pièce avant une opération de traitement à chaud).

Dans ce travail, nous ne considérons que des opérations non interruptibles.

**Les gammes*

Si le choix de types de machines pour les opérations n'est pas imposé, on dit que la gamme est une **gamme logique**, sinon on dit que la gamme est une **gamme technique**. Dans une gamme logique, les entités qui interviennent sont les opérations. Dans une gamme technique, machines et opérations se confondent.

Dans le cas de gammes techniques, si l'ordre suivant lequel on doit effectuer les opérations pour terminer un produit est totalement libre, on dit que c'est une **gamme libre**, si l'ordre est entièrement déterminé, on dit que c'est une **gamme linéaire**. Pour un ordre partiellement déterminée on peut utiliser le terme de **gammes mixtes**.

Un atelier ne comportant que des gammes libres est appelé "**open-shop**" (voir par exemple [Achugbue & Chin 1982]). Parmi les problèmes à gammes linéaires, on peut distinguer deux grandes familles de problèmes: les "flow-shops" et les "job-shops". Un problème de type "**flow-shop**" est un problème

dans lequel les ordres de passage de tous les produits sur les machines sont exactement les mêmes, sinon, le problème est un problème de type "**job-shop**".

Dans la suite nous ne considérons que des problèmes de type "job-shop". Nous appliquons également nos méthodes à des problèmes de "flow-shop" comme cas particuliers de "job-shop". Il est à noter que pour les "flow-shops", il existe des résultats théoriques complémentaires.

b) Les machines

Le cas le plus simple parmi les problèmes d'ordonnancement est le problème d'ordonnancement à une machine. Nous y reviendrons au chapitre 3.

Si plusieurs machines sont capables de réaliser un même ensemble d'opérations, on parle de **machines en parallèle**. Des machines visitées successivement par les opérations de la gamme d'un produit sont dites **en série**.

En ce qui concerne les machines en parallèle, il existe trois types de parallélismes.

**Machines identiques*

Dans le cas de machines identiques (par exemple, un ordinateur biprocesseur.), toute opération peut être effectuée sur n'importe quelle machine parmi les machines parallèles. Les durées de l'opération sur les machines sont égales. Ce cas est considéré dans les études de [Eastman et al. 1964], [Tovey 1986], [Sundaraghavan & Ahmed 1984], [Sanhi 1976], [Lawler 1964], [Luh et al. 1988].

**Machines uniformes ou à vitesses proportionnelles*

Dans ce cas, les machines peuvent effectuer les mêmes types d'opérations avec des performances différentes: les temps nécessaires pour effectuer une opération sont différents d'une machine à l'autre. Un coefficient est attaché à chaque machine et le temps nécessaire pour effectuer une opération sur cette machine est le produit de ce coefficient par le temps nécessaire sur la machine la plus performante (par exemple, des imprimantes à 600 et à 1200 lignes à la minute). Ce cas a été considéré par [Emmons 1987], [Coffman et al. 1987].

**Machines non reliées*

Il s'agit de machines polyvalentes. Les machines sont partiellement interchangeables. Par exemple, une machine peut être conçue spécialement pour certains types d'opérations, mais elle peut aussi en effectuer d'autres avec une performance moins bonne que les machines dédiées à ces opérations. Il existe alors une matrice $[t_{i,j}]$, où $t_{i,j}$ est le temps nécessaire pour effectuer l'opération i sur la machine j ([Lawler & Labetoulle 1978]).

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

Nous nous limitons au cas où une opération ne peut passer que sur une machine. Nous supposons que, lorsque plusieurs machines sont en mesure d'effectuer la même opération, alors un choix a été fait antérieurement de façon à équilibrer la charge sur le moyen terme.

Nous supposons que les machines sont continuellement disponibles. Leur capacité reste la même au cours du temps. Le cas où la disponibilité des machines change au cours du temps a été considéré dans [Baker & Nuttle 1980].

c) Les ressources ([Blazewicz et al. 1987])

Les **ressources "renouvelables"** sont des ressources réutilisables (machines, ouvriers spécialisés, régleurs...). A chaque instant, la somme des quantités utilisées dans l'atelier ne doit pas dépasser la quantité disponible.

Les **ressources consommables** sont des ressources qui disparaissent au fur et à mesure de leur utilisation (vis, boulons, petites pièces...). La consommation cumulée de ces ressources sur un intervalle de temps compris entre deux approvisionnements ne doit pas dépasser la quantité initialement disponible.

Les **ressources doublement limitées** sont des ressources qui doivent vérifier simultanément les deux contraintes précédentes : limitation instantanée et limitation de la consommation cumulée (matières énergétiques: charbon, pétrole, électricité...).

Les seules ressources que nous considérons sont les machines qui sont de type renouvelable.

d) Les contraintes

Les contraintes sont des conditions à respecter par un ordonnancement. A part les contraintes exprimant les types de gammes, les types d'opérations, les types de ressources, les niveaux des stocks qui ne peuvent pas être dépassés, il peut exister des contraintes spécifiques comme, par exemple, le retard maximal admissible pour un produit ([Chand & schneeberger 1986]). Les contraintes peuvent également traduire des durées minimales de transport entre les machines ou des durées de réglage des machines (le problème est d'autant plus difficile que la durée de réglage dépend de la succession des produits sur la machine).

Il y a également des chercheurs qui considèrent les problèmes où il existe des contraintes de précédence entre les produits (voir, par exemple, [Lawler 1973, 1978], [Schrage & Baker 1978], [Sidney 1975]).

Nous ne considérerons pas ici ces contraintes particulières.

e)Les critères

Les critères sont utilisés pour évaluer la qualité d'un ordonnancement obtenu.

Dans la littérature, les chercheurs utilisent souvent des mesures régulières comme critères. Une mesure est **régulière** ou un critère est **régulier**, si la valeur à minimiser est une fonction non décroissante des dates de fin des produits. Les critères réguliers les plus utilisés sont le maximum ou la moyenne (éventuellement pondérée) des dates d'achèvement des produits ("completion time"), des durées de présence des produits dans l'atelier ("flow time"), des avances-retards ("lateness"), ou des retards "vrais" ("tardiness") ou encore le nombre de produits en retard.

Ces critères servent à exprimer de différentes façons deux sources d'insatisfaction: celle de l'entreprise liée aux volumes trop importants d'en-cours, et celle des clients liée au non respect des délais. Certains d'entre eux définissent les mêmes sous-ensembles de solutions optimales (par exemple, la somme ou la moyenne des retards!).

Un critère **non régulier** est un critère qui peut augmenter lorsque l'on termine un travail plus tôt. En particulier, si le fait de terminer un travail avant sa date de fin souhaitée ("earliness") implique un coût, alors tout critère incluant ce

coût n'est pas régulier. Des problèmes d'ordonnancements utilisant des critères de ce type sont considérés dans les études de Lakshminarayan et al. (1978), de Bagchi et al. (1987) et de Ow & Morton (1989).

Nous utilisons dans ce travail plusieurs critères. Ils sont tous réguliers.

Pour un critère d'évaluation donné, un **ordonnancement est optimal** si la valeur du critère est optimale (minimale ou maximale selon le critère).

Si l'algorithme peut fournir des solutions non optimales ou si on ne peut pas prouver qu'il conduit, pour tous les énoncés du problème considéré, à un ordonnancement optimal, alors on l'appelle **algorithme approché** ou **méthode approchée**, ou encore **heuristique**. Dans le cas contraire, on dira que l'**algorithme est optimal** ou que **la méthode est exacte**.

Une solution O domine une autre solution O1 si et seulement si elle donne une valeur du critère au moins aussi bonne que celle fournie par O1. **Une propriété de dominance** est une propriété indiquant qu'une solution domine une autre. **Un ensemble dominant** est un ensemble dans lequel il existe au moins une solution optimale.

Nous venons de présenter les différents aspects des problèmes d'ordonnancement et les hypothèses retenues dans ce travail. Comme dans la littérature, nous utilisons la notation A/B/C/D pour caractériser tout problème d'ordonnancement d'atelier ([Conway et al. 1967] [Rinnooy Kan 1976]), où:

A: décrit le processus d'arrivée des tâches. Pour des problèmes stochastiques, A identifie la loi de distribution des probabilités en fonction du temps entre les arrivées des tâches. Pour les problèmes déterministes, A fournit le nombre de produits à ordonnancer, la valeur de A peut être une valeur entière précise (par exemple 3) ou la lettre n qui désigne un nombre arbitraire mais fini de produits.

B: décrit le nombre de machines dans l'atelier. Il s'agit d'un entier (en général 1, 2 ou 3) ou de la lettre m qui désigne un nombre arbitraire mais fini de machines.

C: décrit les propriétés particulières des gammes des produits. Par exemple: F pour les flow-shops, J ou G (pour général) pour les job-shops. Dans le cas d'une seule machine où il n'y a pas de gamme, on utilise ce terme pour

décrire des hypothèses particulières comme les arrivées dynamiques ou non des tâches ($r_i \geq 0$ ou rien), l'existence de contraintes de précédence entre les tâches (prec ou rien) ou encore le fait que les tâches sont interruptibles ou non (preempt ou rien).

D: décrit le critère utilisé pour évaluer l'ordonnancement:
Une lettre désigne la mesure de base entrant dans la composition du critère (C pour "completion time", F pour "flow time", L pour "lateness", T pour "tardiness", E pour "earliness", U pour "number of tardy jobs").
Elle est précédée de Σ ou de Σw_i et suivie de l'indice i pour les moyennes et les moyennes pondérées de cette mesure, où i désigne les produits. Elle est simplement suivie de l'indice max pour les critères construits avec les maxima.

Par exemple n/2/F/Fmax caractérise un problème d'ordonnancemnt du type flow-shop avec un nombre arbitraire de produits et 2 machines pour minimiser la durée maximale de présence des produits dans l'atelier.

1.2.3. MODELISATION

En résumé, nous avons retenu les hypothèses suivantes:
- Les dates d'arrivée des produits peuvent être différentes.
- Il n'y a pas de délais impératifs pour les produits.
- Les gammes sont linéaires, c'est-à-dire que l'ordre des opérations est imposé par la gamme.
- Chaque opération ne peut être exécutée que sur une machine.
- Toutes les opérations sont non interruptibles.
- Chaque produit ne peut se trouver que sur une machine à un instant quelconque. Autrement dit, une opération ne peut être commencée avant que l'opération qui la précède dans la gamme ne soit terminée.
- Chaque machine ne peut exécuter qu'une opération à la fois.
- Les ressources (sauf les machines) sont en quantité infinie.
- Les critères à optimiser sont des critères réguliers.

Pour ce qui concerne la modélisation d'un problème d'ordonnancement, différentes approches sont envisageables.

Dans la littérature, on peut trouver la méthode des Réseaux de Petri (voir [Chrétienne 1983], [Hillion & Proth 1988, 1989]), le graphe potentiel-tâche ([Roy 1970]) complété par un graphe disjonctif ([Roy & Sussman 1962]).

La modélisation la plus courante utilise des équations mathématiques, et on obtient en général des problèmes de programmation linéaire en nombres entiers. Dans notre étude, nous utilisons cette modélisation.

Utilisant des critères réguliers qui sont des fonctions des dates d'achèvement C_i, nous utiliserons la notation $\mathcal{F}(C_i)$ pour désigner la valeur du critère considéré quelque soit celui-ci.

Le problème se modélise alors sous la forme:

Min $(\mathcal{F}(C_i))$

sous les contraintes:

$\forall i,\ t_{i,1} \geq r_i$ (1)

$\forall i,\ \forall 1<j\leq g_i\ t_{i,j} \geq t_{i,j-1}+l_{i,j-1}$ (2)

$\forall i, \forall 1\leq j\leq g_i, \forall i1, \forall 1\leq j1\leq g_{i1}$, si $\mu_{i,j}=\mu_{i1,j1}$ alors $t_{i,j}-t_{i1,j1}>l_{i1,j1}$ ou $t_{i1,j1}-t_{i,j}>l_{i,j}$ (3)

où les notations ont les significations suivantes (les i représentent les produits, les j représentent les opérations):

r_i: instant d'arrivée au plus tôt du produit i ("ready-time", "release-time" ou "arrival-time").

p_i: durée totale des opérations qui permettent de réaliser le produit i ("processing-time")

d_i: instant de fin souhaitée au plus tard pour le produit i ("due-date").

g_i: nombre d'opérations à effectuer pour le produit i.

C_i: instant de fin du produit i ("completion time", inconnue du problème).

et pour chaque opération j effectuée sur le produit i, on a:

$\mu_{i,j}$: machine utilisée.

$l_{i,j}$: durée.

$t_{i,j}$: instant de début ("beginning time", inconnue du problème).

Pour faciliter la lecture de ce document, les notations sont introduites au fur et à mesure de leur utilisation. Un glossaire en début de document réunit toutes les notations générales et celles qui sont spécifiques à chaque chapitre.

Les familles de contraintes (1) et (2) expriment la disponiblité des opérations. Une opération ne peut être exécutée sur un produit que si le produit est bien arrivé dans l'atelier et que les opérations précédentes dans la gamme ont été réalisées.

La famille de contraintes (3) exprime les contraintes de disponibilité des machines: une machine ne peut exécuter qu'une opération à la fois.

1.2.4. NOTION DE COMPLEXITE

* *Présentation générale*

Dans ce paragraphe, nous allons présenter brièvement la notion de complexité afin d'éclaircir les termes que nous utilisons tout au long de cette thèse.

La **complexité** analyse le temps nécessaire pour obtenir une solution (on peut également s'intéresser à la mémoire nécessaire, mais nous ne nous intéresserons pas ici à cet aspect de la complexité). On peut considérer la durée moyennne ou la durée dans le pire des cas. Nous traitons essentiellement ce dernier cas.

La théorie de la complexité conduit à distinguer entre la complexité des problèmes et la complexité des algorithmes utilisés pour les résoudre.

Pour obtenir la complexité d'un algorithme, il faut exprimer sa durée en fonction de la taille de l'énoncé du problème notée n : f(n) (par exemple ici, le nombre de produits ou le nombre de machines ou le nombre d'opérations ou une combinaison de ces quantités). Lorsque n devient grand, on cherche à majorer cette fonction f par une formule de la forme $c \cdot |(g(n))|$, c'est-à-dire que l'on cherche g de telle sorte que f soit en O(g(n)) (notation classique en mathémathique).

La **fonction complexité** d'un algorithme est la «meilleure» fonction g(n) que l'on puisse trouver pour majorer f.

Si f est en O(g(n)) et que g(n) est un polynôme, alors on dit que l'algorithme est **polynomial** et il est d'autant meilleur pour n grand que le meilleur degré possible pour le polynôme est faible.

Les algorithmes pour lesquels il n'existe aucune fonction g polynomiale telle que f soit en O(g(n)) sont dits **exponentiels** (on le démontre en minorant f par une fonction exponentielle de n).

Un problème peut être résolu par différents algorithmes et en général on n'est pas sûr d'avoir trouvé l'algorithme de meilleure complexité. L'analyse de la complexité des problèmes consiste à les ranger dans des classes de problèmes de complexité «analogue».

On distingue tout d'abord entre la classe des problèmes de décision où le résultat s'exprime par un booléen «vrai» ou «faux» ("Existe-t-il un objet dans domaine de définition donné possédant une propriété donnée?") et la classe des problèmes d'optimisation (où l'on cherche une solution optimale dans un domaine donné). Les problèmes d'optimisation sont plus difficiles que leurs homologues exprimés sous forme de problème de décision comme on le verra plus loin.

Considérons tout d'abord les problèmes de décision.

On distingue une classe de problèmes très difficiles, mais comportant peu d'éléments: ce sont les problèmes indécidables pour lesquels il n'existe pas d'algorithme.

On distingue une classe de problèmes dits "faciles", la classe P ou classe des problèmes polynomiaux, pour lesquels il existe au moins un algorithme de résolution polynomial.

On distingue une classe NP, qui contient la classe P, constituée des problèmes que l'on peut résoudre en temps polynomial sur des ordinateurs non déterministes (intuitivement, ce sont les problèmes qui seraient résolus en temps polynomial si l'on disposait d'une infinité de processeurs en parallèle).

Dans la classe NP, outre la classe P, on définit une classe de problèmes difficiles, la classe des problèmes NP-complets.

Pour constituer des classes de problèmes, on utilise une relation d'ordre "au moins aussi difficile que":

Un problème de décision P1 est "au moins aussi difficile" qu'un problème P2 si une "transformation polynomiale" (ou algorithme polynomial) peut

transformer tout énoncé du problème P2 en un énoncé du problème P1 et si les décisons (i.e. les réponses "vrai" ou "faux") coïncident pour les énoncés initiaux et pour les énoncés transformés.

Si P1 est polynomial, alors P2 sera polynomial, et tout algorithme qui résoud P1 résoud P2 moyennant une transformation polynomiale de l'énoncé.

Cook (1971) a démontré qu'un problème particulier de NP appelé SAT (ou SATISFAISABILITE) était au moins aussi difficile que n'importe quel problème de NP. Puis il a été démontré que de nombreux problèmes de NP étaient au moins aussi difficile que SAT. Il s'agit donc des problèmes les plus difficiles de NP, on les appelle les problèmes NP-complets.

Il existe des listes de problèmes déjà démontrés comme étant NP-complets ([Garey & Johnson 1979]). Pour montrer qu'un problème est NP-complet, il suffit de montrer qu'il est au moins aussi difficile que l'un quelconque d'entre eux.

De même, il existe des listes de problèmes polynomiaux. Pour montrer qu'un problème est polynomial, il suffit de montrer que l'un quelconque d'entre eux est au moins aussi difficile que lui (ou de savoir le résoudre par un algorithme polynomial).

Actuellement, on se demande si P=NP, c'est-à-dire que tous les problèmes de NP seraient polynomiaux ou si P≠NP, c'est-à-dire que les problèmes NP-complets ne pourraient pas être résolus par des algorithmes polynomiaux.

On suppose, sans aucune preuve, que c'est la dernière conjecture qui est la bonne et c'est pourquoi on affirme que les problèmes NP-complets ne peuvent probablement pas être résolus par des algorithmes polynomiaux.

On peut associer à tout problème d'optimisation un problème de décision:

le problème d'optimisation étant de la forme:
«trouver dans un domaine D un objet o qui minimise une fonction F(o)»,

le problème de décision correspondant est de la forme:
«étant donné un paramètre Φ existe-t-il dans le domaine D un objet o tel que $F(o) \leq \Phi$?».

Soit I_Φ l'intervalle de définition du paramètre Φ. Pour résoudre le problème de minimisation, il s'agit de trouver la plus petite valeur Φ^* du paramètre Φ telle qu'il existe un objet o* tel que $F(o^*) \leq \Phi^*$. Pour obtenir cette valeur F*, il suffit d'effctuer une recherche dichotomique sur l'intervalle I_Φ utilisant la résolution du problème de décision.

Par exemple, si I_Φ est l'intervalle fini $[\Phi_A, \Phi_B]$ on applique l'algorithme suivant:

$\Phi 1 = \Phi_A$; $\Phi 2 = \Phi_B$;
Tantque $|\Phi 1 - \Phi 2| > \varepsilon$ **faire**
    $\Phi = (\Phi 1 + \Phi 2)/2$
    S'il existe une solution pour le problème de décision avec Φ comme paramètre **alors** $\Phi 2 = \Phi$;
    sinon $\Phi 1 = \Phi$;
fintantque
$\Phi^* = \Phi$.

On a alors la valeur optimale Φ^* du problème d'optimisation. On ne dispose de l'objet optimal o* que si le problème de décision, non seulement répond à la question posée mais encore fournit un objet permettant de répondre «oui» à la question.

Un problème d'optimisation est donc effectivement plus difficile que le problème de décision correspondant. On dit qu'un problème d'optimisation est **NP-difficile** si le problème de décision qui lui correspond est NP-complet. Un problème d'optimisation sera polynomial si on peut le résoudre par un algorithme polynomial.

La notion de complexité permet d'informer les chercheurs et leur évite de perdre du temps à essayer de construire un algorithme polynomial pour obtenir une solution optimale alors que le problème a été démontré NP-difficile.

Conway et al. (1967) insistent sur le fait que des problèmes qui paraissent simples sont en fait NP-difficiles et qu'avant la théorie de la complexité des chercheurs s'acharnaient sur ces problèmes en voulant les résoudre polynomialement.

* *Résultats connus de complexité en ordonnancement*

La complexité des problèmes à une machine est présentée dans le chapitre 3. Nous ne discutons ici que de la complexité des problèmes comportant plusieurs machines. La plupart de ces problèmes ont été démontrés NP-difficiles.

Quelques-uns seulement sont polynomiaux. Nous les présentons ci-dessous.

La méthode de Johnson ([Johnson 1954]) permet de résoudre le problème n/2/F/Fmax (minimisation de la durée totale de fabrication pour un flow-shop à deux machines). En modifiant légèrement cet algorithme, Jackson (1956) a construit un algorithme pour le problème n/2/G/Fmax (minimisation de la durée totale de fabrication pour un job-shop à deux machines) sous une hypothèse restrictive: chaque produit passe au plus une fois sur chaque machine. Pour le problème n/2/G,$l_{i,j}$=1/Fmax (même problème sous l'hypothèse que les durées de toutes les opérations sont unitaires, mais sans la restriction de Jackson), Hefetz & Adiri (1982) ont construit un algorithme polynomial très simple ("Longest Remaining First" ou l'opération la plus prioritaire est celle qui appartient au travail pour lequel il reste le plus d'opérations à effectuer).

Le seul problème polynomial pour trois machines est le problème n/3/F/Fmax (minimisation de la durée totale de fabrication pour un flow-shop comportant trois machines) sous des hypothèses restrictives ([Johnson 1954]):

$$\min_{i=1,\ldots,n} (l_{i,1}) \geq \max_{i=1,\ldots,n} (l_{i,2})$$

ou (inclusif)

$$\min_{i=1,\ldots,n} (l_{i,3}) \geq \max_{i=1,\ldots,n} (l_{i,2})$$

qui signifient que:
ou bien la plus courte opération sur la machine 1 est plus longue que la plus longue opération sur la machine 2,
ou bien la plus courte opération sur la machine 3 est plus longue que la plus longue opération sur la machine 2.

Lorsque le nombre de machines est strictement supérieur à 3, tous les problèmes traités dans la littérature sont NP-difficiles, quel que soit le critère,

mais il existe des propriétés de dominance qui permettent de réduire l'ensemble de solutions à considérer. Nous donnons brièvement ci-dessous les problèmes pour lesquels il existe des propriétés de dominance.

1.2.5. PROPRIETES DE DOMINANCE, ENSEMBLES PARTICULIERS D'ORDONNANCEMENTS

** Propriétés de dominance*

1° Pour les problèmes de flow-shop, lorsque tous les produits arrivent en même temps dans l'atelier, il suffit, pour minimiser des mesures régulières, de considérer les ordonnancements dans lesquels les produits sont exécutés dans le même ordre sur les deux premières machines.

2° Pour les problèmes de type n/m/F/Fmax, lorsque tous les produits arrivent en même temps dans l'atelier, il suffit de considérer les ordonnancements dans lesquels les produits sont exécutés dans le même ordre sur les deux premières machines et dans le même ordre sur les machines m-1 et m (ces deux ordres pouvant être distincts pour m>3).

3° Pour minimiser des mesures régulières, il suffit de considérer les ordonnancements **semi-actifs** ou les ordonnancements **actifs**. Les ensembles semi-actifs et actifs sont des ensembles dominants pour les critères réguliers. Ils sont présentés au paragraphe suivant.

4° L'ensemble des ordonnancements **sans délai**, que nous définissons également au paragraphe suivant, n'est malheureusement pas dominant pour les critères réguliers. Il est cependant beaucoup utilisé en pratique.

** Ensembles d'ordonnancements*

Différents ensembles d'ordonnancements sont définis dans la littérature. Les plus connus sont ceux introduits au paragraphe précédent. Ils sont très utiles pour démontrer des propriétés de dominance ou pour construire des algorithmes à partir de règles d'ordonnancement.

Un **ordonnancement semi-actif** ou un **ordonnancement calé à gauche** est un ordonnancement dans lequel aucune opération ne peut être exécutée plus tôt si l'on maintient l'ordre des opérations sur toutes les machines.

Dans cette thèse nous ne considérons que des ordonnancements semi-actifs, sauf indication contraire.

Un **ordonnancement actif** est un ordonnancement dans lequel aucune opération ne peut être exécutée plus tôt sans retarder au moins une autre opération. Tout ordonnancement actif est semi-actif.

Un **ordonnancement sans délai** est un ordonnancement dans lequel aucune machine n'est laissée inoccupée pendant une période de temps où il y a des opérations disponibles attendant pour être exécutées sur cette machine.

1.2.6. METHODES DE RESOLUTION

Comme on l'a rappelé dans le paragraphe sur la complexité, la plupart des problèmes d'ordonnancement sont NP-difficiles. On ne peut donc pas espérer trouver un ordonnancement optimal en un temps raisonnable pour des problèmes de taille industrielle. Les méthodes utilisées sont donc des méthodes approchées. On compare deux méthodes approchées destinées à résoudre le même problème selon deux critères principaux: la complexité de l'algorithme (en durée de calcul) et la qualité de la solution obtenue (écart absolu ou relatif par rapport à la solution optimale ou par rapport à la meilleure solution trouvée).

Pour des problèmes de petite taille, nous pouvons obtenir une solution exacte. Une méthode exacte peut servir pour résoudre des sous-problèmes d'un problème de grande taille de manière optimale.

• *Méthodes approchées*

Les méthodes approchées peuvent être regroupées en deux catégories:

La première catégorie consiste à utiliser des connaissances acquises au cours d'ordonnancements précédents ou à l'occasion de simulations. Dans cette catégorie, nous trouvons:

- systèmes experts,
- mémoire artificielle.

Les principaux systèmes experts développés en ordonnancement sont ISIS ([Fox & Smith 1984]), SOJA ([Lepape 1985], [Lepape & Sauve 1985]) et OPAL ([Bensana et al. 1988]).

Le chapitre 2 de cette thèse est consacré à la conception et à l'utilisation d'une mémoire artificielle.

La deuxième catégorie consiste à diminuer le volume des calculs. Dans cette catégorie, nous trouvons:

- ajout de contraintes pour diminuer le cardinal de l'ensemble des ordonnancements réalisables ([Potts & Van Wassenhove 1982]);
- relaxation des contraintes de façon à pouvoir construire un modèle dont la solution est connue;
- linéarisation du critère lorsque le critère n'est pas linéaire;
- décomposition du problème initial en sous-problèmes de taille plus petite et aussi indépendants les uns des autres que possible (décomposition temporelle par paquets de produits ([Yamamoto 1977]), décomposition spatiale et/ou temporelle ([Portmann 1987]);
- utilisation d'une approche hiérarchique;
- construction progressive d'un ordonnancement par l'application de règles;
- amélioration progressive des ordonnancements par des méthodes de voisinages (plus forte pente, recuit simulé, méthode TABU).

Dans ce travail, nous utilisons essentiellement la construction progressive d'un ordonnancement par l'application de règles.

• *Méthodes exactes*

Les méthodes exactes utilisent surtout deux approches de résolution très connues: la programmation dynamique et les procédures par séparation et évaluation ("branch and bound"). Ces méthodes sont souvent utilisées pour résoudre les problèmes combinatoires de manière exacte, en ordonnancement tout particulièrement. Ce sont des méthodes d'énumération implicite:

L'énumération explicite construit toutes les solutions réalisables et retient une parmi les meilleures.

L'énumération implicite consiste à explorer l'ensemble de toutes les solutions réalisables en éliminant des sous-ensembles de solutions moins intéressants sans avoir à les construire.

*Programmation dynamique

La méthode de programmation dynamique est basée sur le fameux principe d'optimalité de Bellman (1954):

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).

Ce principe permet de calculer les plus courts (resp. longs) chemins d'un graphe de manière récurrente (étape par étape) en utilisant la connaissance des sous-chemins optimaux des étapes précédentes pour calculer les sous-chemins optimaux de l'étape en cours.

Cette technique peut s'étendre à des séquences de tâches dans les problèmes d'ordonnancement (par exemple [Srinivasan 1971]).

*Procédure par Séparation et Evaluation

Dans cette méthode, l'ensemble des solutions du problème est décomposé en plusieurs sous-ensembles de solutions de taille plus petite. Ces nouveaux sous-ensembles sont eux-mêmes décomposés, et ainsi de suite, jusqu'à obtenir des sous-ensembles dont on sait extraire la solution optimale (ou bien parce que le sous-ensemble est réduit à une seule solution, ou bien parce qu'un algorithme rapide permet d'obtenir la solution optimale du sous-ensemble).

Des évaluations associées aux sous-ensembles permettent d'éliminer des sous-ensembles vides des solutions réalisables et d'éliminer sans les explorer des sous-ensembles dominés: en particulier, l'évaluation principale, appelée **borne**, est un minorant (resp. majorant) pour une recherche de minimum (resp. maximum) (calculé le plus simplement possible [1]) de la valeur du

[1] Parmi les techniques permettant de calculer les bornes figurent en particulier des méthodes générales de relaxation comme la relaxation lagrangienne ([Fisher 1976]) ou la relaxation "surrogate" ([Glover 1975]) et des méthodes de relaxation propres aux problèmes d'ordonnancement, comme par exemple de supposer que les opérations sont interruptibles.

critère dans le sous-ensemble considéré, si ce minorant (resp. majorant) est plus grand (resp. petit) qu'une solution réalisable déjà trouvée, alors le sous-ensemble est sans intérêt.

Ces minorants (ou majorants) seront d'autant plus efficaces qu'ils sont proches de la valeur optimale du critère pour le sous-ensemble considéré.

Outre l'utilisation de la borne, des propriétés de dominance démontrées entre des sous-ensembles permettent d'éliminer des sous-ensembles supplémentaires.

La succession des séparations effectuées au cours d'une PSE peut être représentée par une arborescence. A chaque séparation, le choix du sous-ensemble de solutions à séparer est lié à l'ordre d'exploration de cette arborescence. On peut ainsi distinguer des PSE en profondeur d'abord lorsque l'on sépare toujours le premier des sous-ensembles obtenus lors de la dernière séparation, des PSE "progressives" lorsque l'on sépare toujours le sous-ensemble de meilleure borne, ou tout autre ordre d'exploration. Selon l'ordre d'exploration choisie, la mémoire ou les temps de calcul varient.

Nous présenterons en détail au chapitre 3 une PSE que nous avons construite pour des problèmes à une machine.

** Méthodes exactes développées par les chercheurs*

Dans la littérature, les chercheurs s'intéressent plutôt au critère "minimisation de la durée totale de fabrication" (Fmax ou "makespan" en terme anglo-saxon). D'autres critères ont été étudiés par Grabowski et al. (1983), Townsend (1977) et Vepsalainen & Morton (1987).

Pour les problèmes de flow-shop, les chercheurs ont construit des méthodes exactes essentiellement pour les "flow-shops de permutation", c'est-à-dire des flow-shops où les produits ne peuvent pas se dépasser: ils doivent être exécutés dans le même ordre sur toutes les machines. L'arrivée simultanée de tous les produits en début d'ordonnancement est une hypothèse quasi universelle. Les auteurs utilisent des ordonnancements semi-actifs (voir par exemple [Lageweg et al. 1978]). Des propriétés de dominance démontrées par certains auteurs permettent de réduire l'ensemble de solutions à explorer (voir [Bagga &

Chakravarti 1968], [Dudek & Teuton 1964], [Smith & Dudek 1969], [Szwarc 1971, 1973, 1978]).

En ce qui concerne les problèmes de job-shop et le critère "Makespan", on trouve énormément d'articles. Dans leur livre "Industrial Engineering", Muth & Thampson (1969) ont donné trois exemples. Ces trois exemples sont repris par de nombreux chercheurs. C'est seulement vers 1970 que deux de ces exemples sont résolus ([Carlier 1978], [McMahon & Florian 1975]). Beaucoup de chercheurs testent leurs algorithmes sur le troisième exemple ([Adams et al. 1988], [Barker & McMahon 1988], [Lageweg et al. 1977]). C'est seulement récemment que Carlier & Pinson (1989) ont réussi à résoudre le troisième problème.

1.3. PROBLEMES DE BIN-PACKING

Dans cette section, nous allons définir les problèmes de bin-packing à une, deux ou trois dimensions et essayer de résumer les résultats connus dans ce domaine.

1.3.1. DEFINITIONS

Un problème concret de bin-packing se pose lorsque l'on cherche à remplir un ou plusieurs contenants avec un ensemble fini d'objets ou lorsque l'on cherche à obtenir un ensemble fini d'objets en découpant un ou plusieurs objets de taille supérieure. Dans le premier cas, il s'agit d'un **problème de remplissage** ou de "**container loading**" et dans le deuxième cas il s'agit d'un **problème de découpe**. Ces problèmes sont très fréquents dans le monde industriel.

Il y a trois grandes familles de problèmes selon le nombre de dimensions des objets.

- problèmes à une dimension. Le cas le plus fréquent consiste à utiliser des barres de métal existant dans une entreprise pour satisfaire des demandes en barres de longueurs différentes. On cherche à minimiser la longueur des chutes non réutilisables (inférieures à une longueur minimale donnée).

- problèmes à deux dimensions. Ces problèmes se posent en particulier chez les fabricants de verre, de tôles, et chez les fabricants de vêtements. Il s'agit de placer des commandes rectangulaires sur des formes rectangulaires de formats standards en minimisant le nombre de formes à découper et/ou la surface des chutes non réutilisables obtenues.

- problèmes à trois dimensions. Ce type de problème surgit lorsqu'il s'agit de remplir des colis, des containers, des camions, des wagons, des navires....

C'est le problème à trois dimensions qui est considéré dans ce travail.

On peut distinguer deux familles de problèmes selon le critère considéré:

- minimisation de la hauteur de l'emballage. On utilise dans ce cas des emballages de fermeture "à l'américaine" constitués de deux demis emballages qui s'emboîtent de telle sorte que la hauteur de l'emballage peut varier entre deux valeurs extrêmes h1 et h2. Dans ce cas, on minimise le volume de l'emballage en cherchant à mettre tous les objets tout en utilisant une hauteur minimale (supérieure ou égale à h1et inférieure ou égale à h2).

- minimisation du nombre d'emballages utilisés et de leur volume total.

Les contraintes varient beaucoup suivant les types d'objets à placer dans l'emballage. Il se peut qu'il y ait des objets fragiles sur lesquels il ne faut pas mettre d'objets lourds. Dans certains cas, il faut prendre en considération le centre de gravité des objets et les problèmes d'équilibre au fur et à mesure du remplissage.

1.3.2. APPROCHES DE RESOLUTION

Les problèmes de bin-packing sont NP-difficiles. Aussi les méthodes exactes ne peuvent être utilisées que pour des exemples de très petite taille ou dans des cas très particuliers (beaucoup d'objets identiques par exemple: problèmes de palétisation). Le fait que, dans les services de distribution des entreprises, ce sont des robots qui commandent le remplissage ou la découpe, impose des méthodes de résolution rapides, voire en temps réel.

Dans le domaine de bin-packing, les algorithmes utilisés sont essentiellement des règles de priorité.

Pour évaluer la qualité d'un algorithme, les chercheurs démontrent des théorèmes sur les performances dans le pire des cas.

Dans [Garey & Johnson 1981], les auteurs ont passé en revue des méthodes de "type glouton" (c'est-à-dire où l'on place de manière définitive les objets les uns après les autres) pour les problèmes à une dimension en considérant comme critère la minimisation du nombre d'emballages. Dans cet article, on peut trouver les performances relatives à différentes règles de priorité. Ces performances sont mesurées par le rapport du nombre d'emballages obtenus en utilisant une règle de priorité sur le nombre d'emballages utilisés dans la solution optimale.

Pour les problèmes de deux dimensions, nous pouvons citer les études de Berkey & Wang (1987), de Coffman et al. (1980), Baker et al. (1980), de Rode & Rosenberg (1987) et de Wang (1982) qui fournissent des méthodes et des évaluations de performances dans le pire des cas.

Récemment Han et al. (1989) ont étudié un problème de "cargo loading", mais dans leur travail, tous les objets sont identiques. Dans [Bischoff & Marriott 1987], les auteurs ont comparé 14 heuristiques basées sur la méthode de George & Robinson (1980) et celle de Bischoff & Dowsland (1982). Dowsland (1987) a décrit des conclusions tirées des travaux par l'auteur et par d'autres chercheurs sur les problèmes de palletisation et de container loading.

Chapitre 2

MEMOIRE ARTIFICIELLE

2.1. INTRODUCTION

En général, les problèmes industriels sont fondamentalement multicritères et l'importance relative des différents critères évolue dynamiquement au cours du temps. La mémoire artificielle permet de considérer simultanément plusieurs critères et peut fournir à l'utilisateur un outil d'aide à la décision multicritère. Cette décision multicritère repose sur l'ensemble des "versions" de la mémoire artificielle: chaque version est dédiée à un critère particulier et est indépendante des autres versions. Nous présentons dans ce chapitre la construction d'une version de la mémoire artificielle pour l'un quelconque des critères. L'utilisation de la mémoire artificielle, par contre, sera, soit monocritère (utilisant une seule version), soit multicritère.

2.2. PRESENTATION GENERALE DE LA MEMOIRE ARTIFICIELLE

Pour un critère donné, on cherche à résoudre un problème d'optimisation NP-difficile et il est impossible d'utiliser des méthodes exactes pour des problèmes de taille industrielle. Aussi, la littérature présente une multitude de méthodes heuristiques qui fournissent souvent des solutions proches de l'optimum.

En présence de plusieurs algorithmes heuristiques destinés à résoudre un problème donné, lequel choisir? Il est très rare qu'un algorithme soit systématiquement meilleur que les autres dans tous les cas de figure. Si l'on veut obtenir la meilleure solution (sans garantie d'optimalité), il faut appliquer tous les algorithmes, comparer les solutions obtenues, et retenir la meilleure. On perd alors l'avantage des méthodes approchées — la rapidité — si le nombre d'algorithmes candidats est très important.

Nous faisons l'hypothèse suivante: «si deux jeux de données numériques correspondant à un problème donné sont "proches", alors les algorithmes ont des performances voisines sur ces deux jeux».

Nous sommes tentés d'en tirer la conséquence suivante: lorsqu'un jeu de données se présente, on examine les expériences faites sur des jeux de données antérieurs proches et l'on suppose que les algorithmes disponibles vont avoir,

sur le nouveau jeu de données, le même effet que sur les jeux proches testés antérieurement.

C'est l'idée de base du fonctionnement de la mémoire artificielle.

L'idée de la mémoire artificielle a été proposée dans notre équipe (SAGEP de l'INRIA) en 1983 et avait été testée à l'époque sur une petite maquette liée à un problème d'ordonnancement d'un atelier flexible pour des produits en plusieurs exemplaires avec comme critère principal l'équilibre des flux en volume pour les différents produits.

L'objet de cette thèse est de construire une structure de mémoire artificielle indépendante des applications complétée par des modules spécifiques qui permettent d'adapter la mémoire à de nombreux problèmes d'ordonnancement.

Le chapitre 2 comprend deux grandes parties. La section 2.3. est consacrée aux concepts et aux outils qui permettent de construire une structure générale de mémoire artificielle. La section 2.4 montre comment construire les modules spécifiques propres aux problèmes considérés dans cette thèse.

La figure 2.1 résume les principales phases de conception de la mémoire artificielle pour un problème donné. Les cases entourées d'un trait simple correspondent à des modules généraux de la mémoire indépendants des applications. Les cases entourées d'un trait discontinu correspondent à des modules spécifiques à chaque application, mais indépendants des contextes industriels particuliers. Les cases entourées d'un trait gras correspondent à des modules spécifiques de l'application et du contexte industriel (par exemple: nombre de machines dans l'atelier, choix du catalogue limité de produits ...). Une case entourée de crochets est une étape facultative.

La phase de préparation commence par une étape qui consiste à analyser et à modéliser le problème.

Plusieurs étapes peuvent ensuite être effectuées en parallèle.

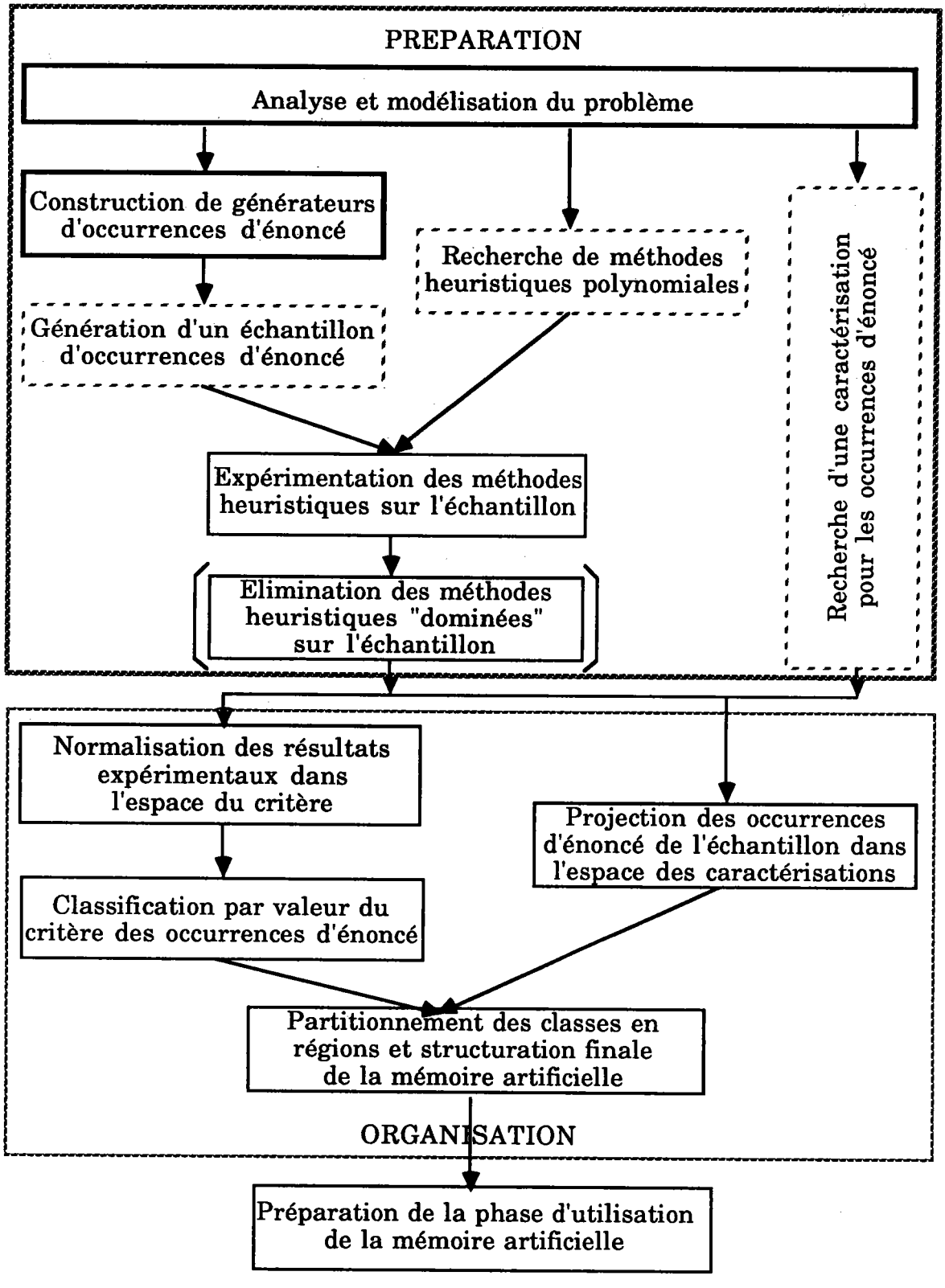


Fig. 2.1. Phases de conception générale de la mémoire artificielle

La mémoire nécessite un ensemble de jeux de données (appelés dans cette thèse occurrences d'énoncé) les plus dispersés et les plus caractéristiques possibles pour une application particulière (un contexte industriel donné par exemple) du problème considéré. Ceci demande la construction de générateurs de jeux de données. Ces générateurs seront présentés respectivement aux paragraphes 2.4.1.1, 2.4.2.1 et 2.4.3.1 pour chacune des applications. Le choix des paramètres de ces générateurs permet de construire l'échantillon souhaité à l'étape suivante.

L'étape de recherche de méthodes heuristiques existantes ou de construction de nouvelles méthodes pour résoudre le problème peut être menée en parallèle avec la génération d'occurrences d'énoncé. Ces méthodes sont présentées dans la partie 2.4 pour chacune des applications: ordonnancement d'atelier de type job-shop pour différents critères (section 2.4.1), ordonnancement d'atelier à une machine avec dates d'arrivées des tâches non identiques pour différents critères (section 2.4.2), placement de boîtes parallélépipédiques dans un emballage parallélépipédique ("bin-packing" ou "container loading") pour deux critères (section 2.4.3).

L'étape suivante consiste à expérimenter les méthodes sur l'échantillon généré et à retenir les performances obtenues pour différents critères. Cette étape est donc commune à toutes les versions de la mémoire artificielle.

Nous allons expliquer le fonctionnement de la mémoire artificielle pour une version considérée, c'est-à-dire pour un seul critère.

Pour le critère choisi et si l'utilisation de la mémoire artificielle est monocritère, on élimine les méthodes heuristiques "dominées", c'est-à-dire moins bonnes que les autres sur tous les exemples (cette phase sera illustrée au chapitre 4).

En parallèle avec les étapes d'expérimentation, il faut rechercher des caractérisations des occurrences d'énoncé. il s'agit d'une projection des occurrences d'énoncé dans un espace de caractéristiques lié au problème particulier considéré de manière à rechercher les proximités dans $\mathbb{R}^p$ avec p fixé, p étant la dimension de l'espace considéré (paragraphe 2.3.1.1).

La phase d'organisation a pour but de structurer la mémoire à partir des informations recueillies au cours de la phase de préparation. Une première

étape de cette phase projette les occurrences d'énoncé dans l'espace des caractérisations, c'est-à-dire dans l'espace dont les composantes sont les caractéristiques.

En parallèle, on s'intéresse à la valeurs du critère fournie par les différentes heuristiques appliquées à toute occurrence d'énoncé.

On normalise ces valeurs dans l'espace des critères (paragraphe 2.3.1.1).

L'introduction d'une échelle de valeur permet de construire un recouvrement de l'ensemble des occurrences d'énoncé par des classes (de modalité donnée qui indique si l'heuristique considérée donne de bons ou de mauvais résultats pour la classe considérée). Les classes ne sont en général, ni convexes, ni connexes (paragraphe 2.3.1.2).

En travaillant simultanément sur l'espace des caractérisations et sur l'espace des critères on cherche à partitionner les classes de l'espace des critères en régions connexes "homogènes" de l'espace des caractérisations (paragraphe 2.3.1.3).

La structuration finale de la mémoire artificielle consiste à retenir les informations essentielles concernant la position des différentes régions et leur forme.

La dernière phase de conception de la mémoire artificielle consiste à préparer le module d'utilisation de la mémoire: étant donné une nouvelle occurrence d'énoncé, il s'agit de trouver rapidement à partir du contenu de la mémoire la meilleure heuristique à lui appliquer (section 2.3.2).

2.3. MISE EN PLACE D'UNE STRUCTURE GENERALE DE MEMOIRE ARTIFICIELLE

Pour la construction et l'utilisation de la mémoire artificielle, nous avons étendu les méthodes proposées par Bonneau & Proth (1985a, 1985b). Nous présentons ici de manière détaillée les concepts correspondant au développement de chaque phase, lorsqu'ils sont indépendants des applications. Nous reportons dans la section suivante les concepts spécifiques.

Nous terminons par une évaluation des coûts de construction de la mémoire artificielle en fonction des choix retenus (choix des distances, essentiellement). La comparaison entre le coût d'utilisation d'une mémoire artificielle et le coût des méthodes classiques sera présentée avec les expériences du chapitre 4.

2.3.1. CONSTRUCTION INITIALE DE LA MEMOIRE ARTIFICIELLE

Les premiers modules non spécifiques à une application donnée sont les modules "expérimentation des heuristiques sur un échantillon" et "élimination des heuristiques dominées". Ils utilisent les résultats obtenus dans les modules précédents spécifiques et ne nécessitent qu'un environnement informatique structurant les informations. N'impliquant aucun développement théorique, ils ne seront pas détaillés dans cette thèse.

2.3.1.1. Normalisation des résultats expérimentaux

Lorsque l'on utilise des techniques d'analyse des données, on ressent le besoin de normaliser les quantités pour donner le même poids aux différents paramètres retenus. Nous ferons de même ici pour les occurrences d'énoncé.

Pour simplifier la présentation, nous supposons dans toutes les formules du chapitre 2 que l'on cherche à minimiser la valeur des critères considérés (on passe évidemment d'une maximisation à une minimisation en multipliant la valeur du critère par -1 et en ajoutant éventuellement une constante adéquate si on souhaite travailler sur des valeurs positives).

Il y a plusieurs manières de normaliser des données quantitatives. Nous donnons le choix aux utilisateurs de la mémoire artificielle entre trois types de normalisations pour la valeur des critères.

En notant v'^{h}_{e} la valeur du critère considéré obtenue en appliquant l'algorithme heuristique d'indice h (h appartenant à l'ensemble des indices d'heuristiques $H=\{1,2,...\eta\}$) à l'occurrence d'énoncé d'indice e (e appartenant à l'ensemble des indices d'occurrences $E=\{1,2,...,\omega\}$) et v^{h}_{e} la valeur normalisée par l'un quelconque des procédés de normalisation, on peut définir les trois procédés de normalisation suivants:

1° Valeur initiale

Pas de normalisation:
$v_e^h = v'^h_e$

2° Translation sans réduction

On effectue une translation propre à chaque occurrence d'énoncé de manière à ramener l'origine à la meilleure valeur trouvée pour l'ensemble des heuristiques:

En posant $v1_e = \min_{h \in H} (v'^h_e)$, on définit la valeur normalisée par:

$v_e^h = v'^h_e - v1_e$

3° Translation avec réduction

On effectue une translation propre à chaque occurrence d'énoncé de manière à ramener l'origine à la meilleure valeur trouvée pour l'ensemble des heuristiques, puis on divise par la longueur de l'intervalle compris entre la meilleure valeur et la valeur la plus mauvaise de manière à ramener toutes les valeurs entre 0 et 1 (si la longueur de l'intervalle est nulle, on pose arbitrairement la valeur normalisée à 0):

En posant:

$v1_e = \min_{h \in H} (v'^h_e)$ et $v2_e = \max_{h \in H} (v'^h_e)$

on définit la valeur normalisée par:
$v_e^h = (v'^h_e - v1_e)/(v2_e - v1_e)$

Nous notons [INF, SUP] l'intervalle des valeurs normalisées extrêmes prises par le critère:

$INF = \min_{e,h}(v_e^h)$ et $SUP = \max_{e,h}(v_e^h)$

2.3.1.2. Recouvrement des occurrences d'énoncé par des classes

Pour chaque heuristique, cette partie consiste à recouvrir l'ensemble des occurrences d'énoncé en classes. Il s'agit d'un recouvrement et non pas d'un partitionnement: l'intersection de deux classes pour une heuristique donnée peut ne pas être vide.

Pour chaque application, on demande au concepteur de la mémoire artificielle de fournir un paramètre q représentant le nombre maximal de classes à construire pour chaque heuristique.

Remarque:

Nous avons essayé plusieurs procédés pour construire les classes. Il s'agit d'un des points les plus délicats rencontrés lors de la construction de la mémoire artificielle.
Nos premiers essais ont consisté à effectuer des partitionnements de l'ensemble des occurrences d'énoncé en classes disjointes. Certains utilisaient un découpage de l'intervalle [INF, SUP] en q parties égales. D'autres cherchaient les points de coupure de l'intervalle [INF, SUP] de manière à équilibrer les classes de la "meilleure" heuristique (celle qui fournit le plus souvent la meilleure valeur du critère). Ces méthodes créaient dans la mémoire artificielle plus de 30% de points "flous", c'est-à-dire des points qui ne sont pas rattachés à des régions par la méthode de partitionnement des classes en régions proposée par Bonneau et Proth (1985) (cf. paragraphe 2.3.1.3). Cette présence importante de points "flous" rendait inefficace l'utilisation de la mémoire artificielle.

La version actuelle de la mémoire artificielle utilise une technique de recouvrement en classes. Elle consiste à demander un coefficient de chevauchement α ($0<\alpha<0,5$) et à recouvrir l'intervalle [INF, SUP] en q parties comme l'illustre la figure 2.2.

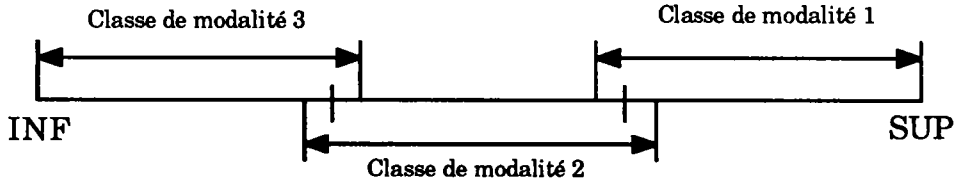


Fig. 2.2. Recouvrement de l'intervalle des valeurs normalisées du critère

Pour chaque heuristique h, on attribue à une occurrence d'énoncé e une ou deux **modalités** de classes selon l'appartenance de v_e^h à un ou deux intervalles du recouvrement. Une occurrence d'énoncé est donc présente dans une ou

deux classes pour chaque heuristique et à chaque classe est associée une modalité: q pour la meilleure classe, q-1 pour la suivante, etc. selon l'ordre inverse de la suite d'intervalles considérés.

En cas d'absence de chevauchement des intervalles, la séparation était arbitraire entre les classes et pour une petite variation de la valeur du critère un point pouvait se voir entourer de points proches dans l'espace des caractérisations mais appartenant à une classe différente, d'où la prolifération des points flous.

Avec chevauchement, on laisse la possibilité aux points appartenant simultanément à deux classes C1 et C2, ou bien d'appartenir seulement à une région de C1 ou de C2, ou bien d'appartenir simultanément à une région de C1 et à une région de C2 ou encore dans un faible pourcentage de cas à rester dans l'ensemble des points flous. Cette possibilité sera examinée lors de la construction des régions.

A la fin de la phase de recouvrement en classes, on pratique un éventuel regroupement des classes afin de faire disparaître les classes trop petites qui ne pourraient pas être partitionnées par des régions de densité suffisante.

Regroupement de classes:

On demande à l'utilisateur de fournir le paramètre s, seuil minimal accepté pour la cardinalité de toute classe.

On traite le regroupement des classes pour chaque heuristique.

Pour une heuristique donnée, l'algorithme est itératif. Il s'arrête lorsqu'il n'y a plus de classe de cardinalité strictement inférieure au seuil s.

En début d'itération, on considère une suite finie de classes dont les modalités sont $i_1, i_2, \ldots, i_{q'}$, avec
$i_1<i_2<\ldots<i_{q'}$ et $q'\leq q$.

Soit i_j la modalité d'une des classes de cardinalité minimale, celle de plus petite modalité s'il y en a plusieurs. La classe de modalité i_j est absorbée par la classe de modalité i_k où k est déterminé par la table de décision suivante où les tests sont effectués dans l'ordre de présentation et où k prend la première

valeur pour laquelle la condition est vérifiée. Dans la table, φ_i est la cardinalité de la classe de modalité i.

Table 2.1. Détermination de la valeur de k

Conditions testées successivement	valeur prise par k
j=1	2
j=q'	q'-1
$i_j-i_{j-1}<i_{j+1}-i_j$ ou $i_j-i_{j-1}=i_{j+1}-i_j$ et $\varphi_{i_{j-1}}\leq\varphi_{i_{j+1}}$	j-1
$i_j-i_{j-1}>i_{j+1}-i_j$ ou $i_j-i_{j-1}=i_{j+1}-i_j$ et $\varphi_{i_{j-1}}>\varphi_{i_{j+1}}$	j+1

La classe de modalité i_j est absorbée par la classe de modalité i_k et la suite finie de classes restante est remise à jour pour l'itération suivante.

Après absorption, les points prennent la modalité de la classe absorbante.

2.3.1.3. Partitionnement des classes en régions et structuration finale de la mémoire

Les classes obtenues précédemment ne sont ni connexes, ni convexes comme le montre la figure 2.3 dans la page suivante.

Aussi, lors de l'utilisation de la mémoire artificielle, il est difficile d'utiliser directement les classes pour voir si un point peut être rattaché à l'une d'entre elles. C'est pourquoi, comme Bonneau & Proth (1985b) l'avaient fait pour des classes disjointes, nous avons été amenés à construire un partitionnement des classes en régions. Nous présentons maintenant une partie des notions définies par Bonneau & Proth (1985b).

Pour rattacher, ou affecter, une nouvelle occurrence d'énoncé à une classe lors de la phase d'utilisation, on utilise la notion de proximité. Ceci nécessite de choisir une distance dans l'espace des caractérisations. Nous utilisons les distances construites à partir d'une métrique, la distance entre deux points x et y est définie de la manière suivante:
$\delta_M(x,y)=[{}^t(P_x-P_y)M(P_x-P_y)]^{1/2}$
où M est une métrique.

M est une matrice carrée (pxp) où p est le nombre de caractéristiques ou coordonnées. P_x est le vecteur composé des coordonnées de x. Selon cette

définition, il y a autant de distances différentes que de métriques. Examinons les métriques intéressantes pour notre problème.

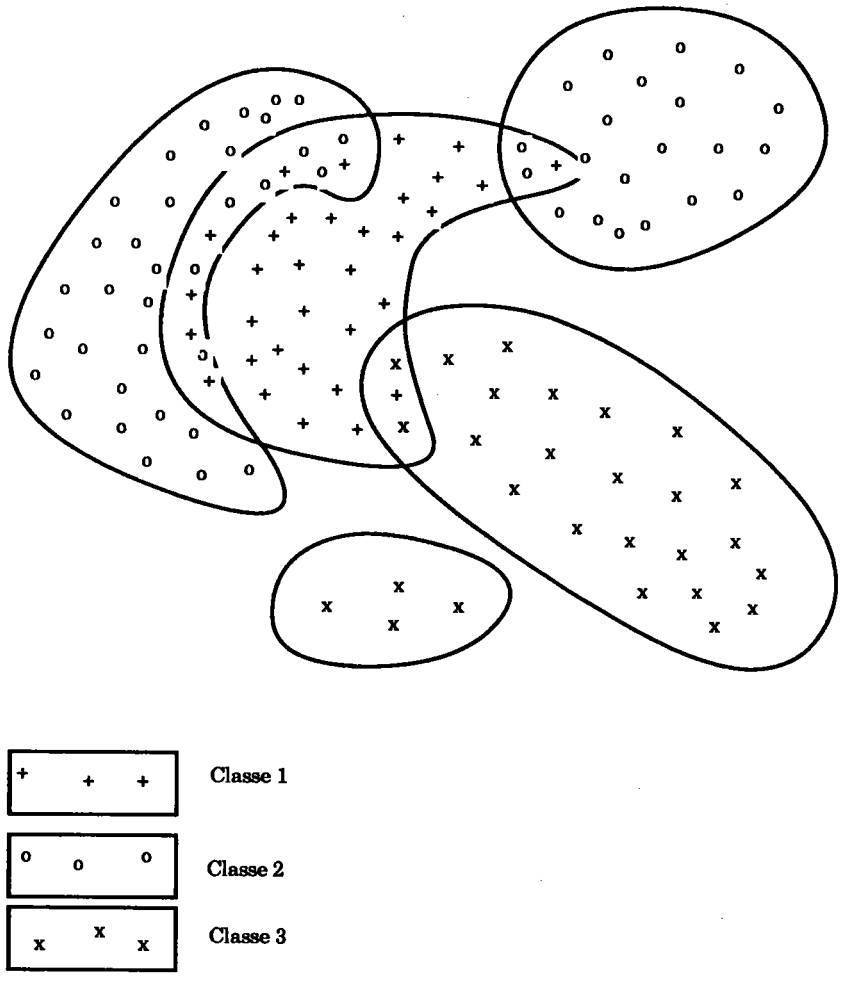


Fig. 2.3. Exemple de classes

1° Métrique euclidienne, ME.

C'est la métrique la plus courante dans le calcul des distances dans l'espace.

Dans ce cas, M=ME $=I_p$ = matrice identité de dimensions (pxp).

2° Métrique diagonale, MD_S.

Dans ce cas, $M=MD_S=(\prod_{i=1}^{p}\varpi_{i,S})^{1/p}\text{diag}(1/\varpi_{1,S}, 1/\varpi_{2,S}, \ldots, 1/\varpi_{p,S})$

où $\varpi_{i,S}$, i=1, 2, ..., p, est la variance du vecteur obtenu en considérant la $i^{ème}$ caractéristique (ou coordonnée) des occurrences d'énoncé de l'ensemble S.

3° Métrique de Mahalanobis, MM_S.

Dans ce cas, $M=MM_S=[\det(VC_S)]^{1/p}(VC_S)^{-1}$
où VC_S est la matrice de variance-covariance associée aux vecteurs-lignes de la matrice des caractéristiques des occurrences d'énoncé de l'ensemble S.

Il est à noter que l'on peut associer une distance diagonale ou une distance de Mahalanobis à tout sous-ensemble $\mathcal{X}$ de $\mathcal{E}$. Nous utilisons pour la construction de la mémoire, une distance diagonale ou de Mahalanobis générale associée à tous les points de $\mathcal{E}$ tant que nous n'avons pas construit le partitionnement en régions. Une fois le partitionnement en régions terminé, nous utilisons une métrique propre à chaque région qui tient mieux compte de la forme de la région comme nous l'illustrons ci-après.

Pour le partitionnement en régions (distance générale), l'utilisateur a le choix entre les trois métriques. Pour la structuration finale de la mémoire, nous imposons dans la version actuelle de la mémoire la métrique diagonale en raison de son rapport qualité-temps de calcul.

La métrique de Mahalanobis appliquée à des ensembles S de forme allongée donne des distances de mêmes ordres de grandeur pour les points les plus éloignés du centre de gravité, c'est-à-dire qu'elle transforme un ovoïde en sphère pour le calcul des distances entre les points de l'ovoïde. Elle permet donc de tenir compte de la forme des ensembles (classes ou régions) auxquels on cherche à affecter un point.

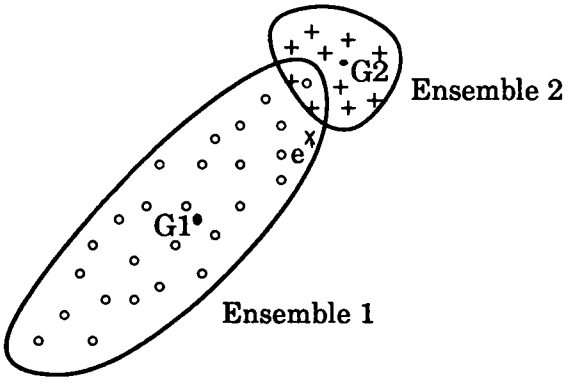


Fig. 2.4. Problème lié à l'affectation

On voit sur la figure 2.4 dans la page suivante, sur un exemple à 2 dimensions, que si l'on utilise la métrique euclidienne, e' sera affecté à l'ensemble 2, alors

qu'en fait, il est à l'intérieur des frontières à l'ensemble 1. Avec la métrique de Mahalanobis associée à chaque ensemble, ce problème ne se pose plus.

Malheureusement, la métrique de Mahalanobis coûte très cher. Aussi on peut se contenter d'utiliser la métrique diagonale. C'est un compromis entre la métrique de Mahalanobis et la métrique euclidienne, car la métrique diagonale transforme moins bien les ovoïdes en sphères, mais les calculs correspondants sont nettement moins coûteux.

Nous détaillons maintenant la manière de définir puis de construire les régions en utilisant l'une quelconque des métriques.

A) DEFINITIONS

Pour définir la notion de région, on a besoin de définir tout d'abord la notion de boule et la notion d'ensemble connexe.

Définition 2.1:

Par analogie avec la topologie, on définit une **boule** dans l'ensemble d'occurrences d'énoncé $\mathcal{E}$, de centre x et de cardinalité s+1 comme un des sous-ensembles de points de $\mathcal{E}$ formé par x et ses s plus proches voisins.

Définition 2.2:

Un **ensemble connexe** $\mathcal{X}$ de $\mathcal{E}$ est un sous-ensemble vérifiant la propriété suivante:
Pour tout couple de points (x, y) de $\mathcal{X}$, il existe une suite de boules de $\mathcal{E}$ incluses dans $\mathcal{X}$, B_1, B_2, ..., B_m telles que:
$\forall i = 1, 2, ..., m-1$, $B_i \cap B_{i+1} \neq \emptyset$, et $x \in B_1$, $y \in B_m$.

Définition 2.3:

Un sous-ensemble $\mathcal{A}$ de $\mathcal{E}$ est **d'épaisseur au moins s0** s'il existe une boule de $\mathcal{E}$ de cardinalité s0+1 incluse dans $\mathcal{A}$. On dit qu'il est **de taille au moins s1** si son cardinal est supérieur ou égal à s1.

Définition 2.4:

On définit une **région** R(s0,s1) de $\mathcal{A}$ (sous-ensemble de $\mathcal{E}$) comme un sous-ensemble de $\mathcal{A}$ connexe d'épaisseur au moins s0 et de taille au moins s1.

Remarque:

La notion de région telle que nous l'avons définie est beaucoup moins forte que la convexité, ces régions pouvant avoir toutes les formes possibles pourvu qu'elles restent connexes. Nous n'avons pas cherché à définir des régions à la fois connexes et convexes pour deux raisons:

1° Algorithmiquement il est très difficile de reconnaître les zones convexes.

2° La convexité est une hypothèse agréable mais exigeante et l'on risque d'obtenir des régions de cardinal bien trop faible.

Définitions 2.5:

1° Le **centre de gravité** G_R de la région R est le point dont les coordonnées sont les coordonnées moyennes des points de la région.

2° La métrique diagonale associée à la région R est notée M_R.

3° Le **rayon maximal** $Rmax_R$ de la région R est la distance maximale entre un élément de la région et le centre de gravité pour la métrique M_R.

$$Rmax_R = \max_{e \in R}(\delta_{M_R}(G_R, e))$$

4° Le **rayon minimal** $Rmin_R$ de la région R est la distance minimale entre un élément en dehors de la région et le centre de gravité pour la métrique M_R.

$$Rmin_R = \min_{e \notin R}(\delta_{M_R}(G_R, e))$$

Ces caractéristiques vont servir pour la phase d'utilisation. Il faudra donc stocker ces informations dans la mémoire.

Commentaire:

Rmax et Rmin définissent deux sphères avec G comme centre. La sphère définie par Rmin est la plus grande sphère possible ne contenant que les points de la région. La sphère définie par Rmax est la plus petite sphère possible contenant tous les points de la région (voir la figure 2.5). Il est possible que Rmin soit plus grand que Rmax.

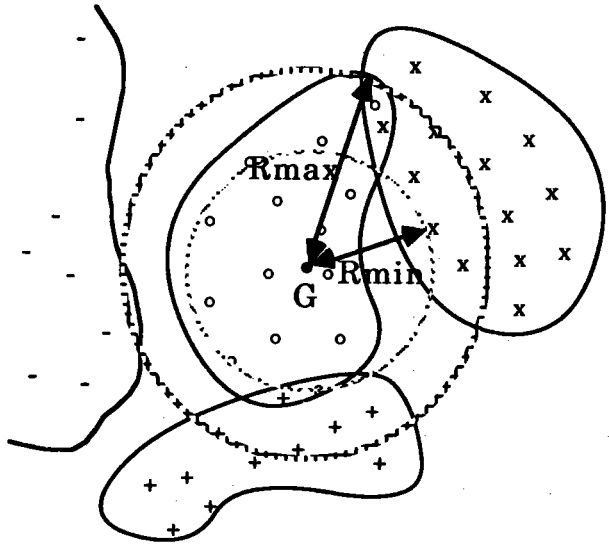


Fig. 2.5. Signification de Rmin et Rmax

B) PARTITIONNEMENT EN REGIONS

Définition 2.6:

Soit $\mathcal{A}$ un sous-ensemble d'un ensemble fini $\mathcal{E}$, on dira que R_1, ..., R_l, $\mathcal{Z}$ est **un partitionnement en régions (s0, s1)** de $\mathcal{A}$ si les ensembles R_1, ..., R_l, $\mathcal{Z}$ vérifient:

1) $\forall i \leq l$, R_i est une région de taille au moins s1 et d'épaisseur au moins s0,

2) $\forall i$, $\forall j \neq i$, l'ensemble $R_i \cup R_j$ n'est plus une région,

3) le sous-ensemble $\mathcal{Z}$ est tel que:

- pour tout $x \in \mathcal{Z}$ et pour tout i, l'ensemble $R_i \cup \{x\}$ n'est plus une région,
- il n'existe pas de région (s0,s1) incluse dans $\mathcal{Z}$.

$\mathcal{Z}$ est appelé **zone trouble**.

Bonneau & Proth (1985b) ont démontré le théorème suivant:

Théorème 2.1:

Pour tout s0 et s1 et pour tout sous-ensemble $\mathcal{A}$ de $\mathcal{E}$, il existe un et un seul partitionnement en régions (s0,s1) de $\mathcal{A}$.

Algorithme de construction des régions:

On se donne (s0,s1) et une classe $\mathcal{A} = \{a_1, a_2, ..., a_\omega\}$ (où a_j est le point défini par le vecteur des caractéristiques d'une occurrence d'énoncé de la classe $\mathcal{A}$) que l'on veut partitionner en régions (s0,s1):

1. $\mathcal{Z}$:= $\mathcal{A}$, k := 0, FINI := faux;
2. Tantque FINI = faux faire
   2.1. FINI = vrai
   2.2. Pour chaque x∈ $\mathcal{Z}$ faire
      2.2.1 Calculer la plus grande boule B_x de centre x, incluse dans $\mathcal{A}$ (1);
      2.2.2 INT := {j / j≤k et $R_j \cap B_x \neq \emptyset$};
      2.2.3 Si card(B_x)≥s0+1 et INT=Ø alors
            k := k+1, R_k := B_x, FINI:=faux, $\mathcal{Z}$:= $\mathcal{Z}$ - {x};
            sinon Si INT≠Ø alors
                  k := k+1, R_k:=$B_x \cup_{i \in INT} R_i$;
                  FINI := faux, $\mathcal{Z}$:= $\mathcal{Z}$ - {x};
                  Supprimer les régions de INT, et renuméroter les régions de manière contiguë, recalculer k;
               Finsi.
            Finsi.
      Finpour.
   Fintantque.
3. Remettre les régions de taille inférieure à s1 dans $\mathcal{Z}$.

Comme le partitionnement est unique (théorème 2.1), on peut considérer les éléments de $\mathcal{Z}$ dans n'importe quel ordre.

Nous notons Γ_i^h la classe de modalité i pour l'heuristique h. Cet algorithme de partitionnement en régions est utilisé pour le critère considéré pour chaque heuristique h et en remplaçant $\mathcal{A}$ par Γ_i^h pour i = 1, 2,..., q. On obtient ainsi pour chaque classe Γ_i^h, une zone trouble $\mathcal{Z}_i^h$ et un ensemble de régions $\mathcal{R}_i^h = \{R_{i,1}^h, R_{i,2}^h, ..., R_{i,r_i^h}^h\}$, où $r_i^h = \text{card}(\mathcal{R}_i^h)$ est le nombre de régions pour la classe Γ_i^h.

1: Lorsqu'un point de $\mathcal{A}$ est aussi dans une autre classe en raison des chevauchements, on suppose ici qu'il est uniquement dans $\mathcal{A}$.

2.3.1.4. Coût de la construction

La construction de la mémoire artificielle demande beaucoup de calculs et utilise beaucoup de mémoire centrale et d'espace disque.

Si on veut économiser de la place mémoire ou de l'espace disque, on perd du temps pour reconstruire des informations utiles non conservées. Si ces informations sont souvent utilisées ou longues à obtenir, le temps de reconstruction est très long. Mais, par ailleurs, comme on n'a pas assez de mémoire centrale pour stocker toutes ces informations, on est obligé d'utiliser le disque. On utilise aussi le disque parce que la mémoire artificielle est composée de plusieurs modules indépendants qui communiquent, pour la plupart d'entre eux, par des fichiers.

Les coût de la phase de préparation ne dépendent absolument pas des choix faits pour la construction de la mémoire artificielle. Les coûts sont liés à l'application considérée, à la taille moyenne et au nombre total des occurrences d'énoncé générées, ainsi qu'au nombre d'heuristiques et à leur complexité moyenne. Il n'existe pas de formules donnant une expression analytique de ces coûts, aussi nous contenterons nous de fournir les valeurs pour les expériences réalisées au chapitre 4.

Par contre, le coût de la phase d'organisation n'est pas lié à l'application retenue. Il est lié au problème considéré par le nombre p de caractéristiques retenues pour chaque occurrence d'énoncé, le nombre γ de critères retenus (pour une version de la mémoire artificielle, γ=1) et le nombre η d'heuristiques retenues (qui peuvent varier selon le problème), d'où, pour cette phase, une évaluation générale du coût indépendante des problèmes, mais dépendante des métriques choisies.

A) COUT EN MEMOIRE CENTRALE DE LA PHASE D'ORGANISATION

On utilise la mémoire centrale pour stocker les caractéristiques des occurrences d'énoncé. Cette partie de mémoire est donc proportionnelle au nombre ω d'occurrences d'énoncé et au nombre p de caractéristiques. On utilise aussi la mémoire pour stocker, pour chaque heuristique et chaque critère, l'appartenance à une classe d'une occurrence d'énoncé. Cette partie de mémoire est proportionnelle au nombre ω d'occurrences d'énoncé et au nombre η d'heuristiques.

B) COUT EN ESPACE DISQUE PENDANT LA PHASE D'ORGANISATION

On utilise au moins deux fois la distance entre tout couple d'occurrences d'énoncé. Le calcul de cette distance est très coûteux comme on le verra au paragraphe D. C'est pourquoi on stocke le calcul de ces distances sur disque, car d'une part la mémoire principale est insuffisante pour conserver $\frac{\omega(\omega-1)}{2}$ distances et, d'autre part, lors de la mise au point de la mémoire artificielle, cela nous évitait de recommencer cette phase coûteuse précédant le partitionnement en régions.

Remarques:

1° On devrait ne stocker que $\omega(\omega-1)/2$ distances, en raison de la symétrie des distances, mais ceci augmenterait considérablement le coût du partitionnement en régions en ajoutant de nombreuses lectures d'informations inutiles dûes à la lecture des informations par blocs et à l'absence, pour cette solution qui économise la place, de séries contigües pour les distances entre un point x et tous les autres points, c'est pourquoi on a stocké en fait ω^2 distances.

2° Ce fichier des distances n'est plus utile après le partitionnement en régions, il ne figure donc pas dans l'état final de la mémoire artificielle.

On utilise également le disque pour stocker les informations pendant le partitionnement en régions. Ceci demande une place sur disque proportionnelle au nombre ω d'occurrences d'énoncé et au nombre η d'heuristiques, puisqu'on stocke seulement le numéro de région pour chaque occurrence d'énoncé et chaque heuristique.

Un autre fichier contient les grandeurs caractéristiques des régions. Son encombrement est proportionnel au nombre total de régions et au nombre de caractéristiques d'une occurrence d'énoncé.

C) COUT EN TEMPS DE LA PHASE D'ORGANISATION

Comme on l'a vu ci-dessus, le coût en espace mémoire et en espace disque de la phase d'organisation est pratiquement indépendant de la métrique. Il n'en est pas de même pour le temps de calcul, et les différences sont considérables. Ce temps peut être décomposé en deux parties: partitionnement en régions, et calcul des grandeurs caractéristiques des régions. Nous exprimons les coûts en nombre d'opérations élémentaires.

Temps de partitionnement en régions

En ce qui concerne ce temps, c'est le calcul des distances qui coûte très cher. Il faut calculer $(\omega-1)\omega/2$ distances. Le calcul d'une distance coûte plus ou moins cher suivant le choix de la métrique et le nombre p de caractéristiques pour les occurrences d'énoncé.

Pour calculer la distance entre deux points e1 et e2, il faut tout d'abord calculer l'écart des coordonnées entre ces points

$$\forall i,\ dif_i(e1,e2)=x_{i,e1}-x_{i,e2};$$

Le volume de calculs est proportionnel à p.

Le temps du calcul suivant est différent selon la métrique. On peut distinguer trois cas (en supposant la métrique déjà calculée):

1° La métrique est une matrice carrée non creuse (métrique de Mahalanobis par exemple):

Dans ce cas, on a:

$$\delta(e1,e2) = \sqrt{\sum_{i=1}^{p}\sum_{j=1}^{p} dif_i(e1,e2)\cdot m_{i,j}\cdot dif_j(e1,e2)}$$

où $m_{i,j}$ est l'élément de i-ième ligne et j-ième colonne de la métrique. Ceci coûte alors $(3p^2+1)$ lorsque, pour tout i, $dif_i(e1,e2)$ a été calculé.

Le nombre d'opérations nécessaires pour calculer $\omega(\omega-1)/2$ distances est donc

$$\frac{\omega(\omega-1)}{2}[3p^2+p+1]$$

En prenant le monôme de plus grands degrés de ce polynôme en p et ω, on obtient un ordre de grandeur du coût proportionnel à ω^2p^2.

2° La métrique est une matrice diagonale mais non euclidienne

Dans ce cas, on a:

$$\delta(e1,e2) = \sqrt{\sum_{i=1}^{p} m_i\cdot dif_i^2(e1,e2)}$$

où m_i est le i-ième élément de la diagonale de la métrique. Ceci coûte donc $(3p+1)$ lorsque, pour tout i, $dif_i(e1,e2)$ a été calculé.

Le nombre d'opérations nécessaires pour calculer $\omega(\omega-1)/2$ distances est alors $\frac{\omega(\omega-1)}{2}[4p+1]$

L'ordre de grandeur du coût est donc proportionnel à $\omega^2 p$.

3° La métrique euclidienne

Dans ce cas, on a:

$$\delta(e1,e2) = \sqrt{\sum_{i=1}^{p} dif_i^2(e1,e2)}$$

Ceci coûte (2p+1) lorsque, pour tout i, $dif_i(e1,e2)$ a été calculé.

Le nombre d'opérations nécessaires pour calculer $\omega(\omega-1)/2$ distances est alors $\frac{\omega(\omega-1)}{2}(3p+1)/2$.

L'ordre de grandeur de ce coût est donc proportionnel à $\omega^2 p$.

Il faut encore prendre en considération les temps de calcul de la métrique.

1° Utilisation de la métrique de Mahalanobis.

Si on utilise la métrique de Mahalanobis pour déterminer les distances entre les points pour le partitionnement en régions, il faut tout d'abord calculer la métrique de Mahalanobis associée à tout l'ensemble, c'est-à-dire calculer la matrice de variances-covariances associée à cet ensemble, et l'inverser.

Afin de calculer la matrice de variances-covariances, il faut calculer la moyenne pour chaque variable. La moyenne de la caractéristique i est calculée en utilisant la formule suivante:

$$\bar{x}_i = \frac{\sum_{k=1}^{\omega} x_{i,k}}{\omega}$$

Il faut donc $(\omega+1)$ opérations.

Il faut au total $p\cdot(\omega+1)$ pour calculer les moyennes des variables.

La covariance entre les variables i et j est calculée par

$$cov_{i,j} = \frac{\sum_{k=1}^{\omega}(x_{i,k} - \bar{x}_i)(x_{j,k} - \bar{x}_i)}{\omega}$$

Le coût de calcul d'une covariance est alors de $(4\omega+1)$ opérations (deux soustractions, une multiplication, ensuite une addition, pour un k, et enfin une division pour calculer la covariance). Le nombre d'opérations nécessaires pour calculer la matrice de variances-covariances est donc de
$\frac{p(p+1)}{2}(4\omega+1)+(p\omega+1)$

L'ordre de grandeur du coût est donc proportionnel à $p^2\omega$.

En ce qui concerne l'inversion d'une matrice, on utilise la méthode de Gauss.

Supposant que la matrice originale et inverse sont respectivement X et Y, et $x_{i,j}$ et $y_{i,j}$ sont respectivement l'élément de i-ième ligne et j-ième colonne de X et de Y, une version simplifiée de cette méthode (où on néglige la transposition de deux lignes lorsque le "pivot" est nul) peut fournir une évaluation approchée de son coût. Cette version simplifiée s'écrit:

1. Initialiser Y à la matrice identité dans p.
2. Mettre X sous forme d'une matrice triangulaire supérieure et de diagonale unitaire: c'est-à-dire: $\forall i,\ x_{i,i}=1$, et $\forall j>i,\ x_{j,i}=0$;

```
Pour i de 1 à p-1 faire
  Pour k de 1 à p faire
        x[i,k]:=x[i,k]/x[i,i];
        y[i,k]:=y[i,k]/x[i,i];
  Finpour
  Pour j de i+1 à p faire
        coef:=x[j,i];
        Pour k de 1 à p faire
              x[j,k]:=x[j,k]-coef·x[i,k];
              y[j,k]:=y[j,k]-coef·y[i,k];
        Finpour
  Finpour
```

Finpour

3. Mettre X sous la forme d'une matrice identité

Pour i de p à 2 faire

Pour j de i-1 à 1 faire

Pour k de 1 à p faire

$x_{j,k}:=x_{j,k}-x_{j,i}\cdot x_{i,k}$;

$y_{j,k}:=y_{j,k}-x_{j,i}\cdot y_{i,k}$;

Finpour

Finpour

Finpour

La matrice Y devient la matrice inverse de X.

La troisième partie rend $\forall j\neq i$, $x_{j,i}=0$ et $x_{i,i}=1$. La ligne encadrée n'est pas nécessaire, puisque seules les données des colonnes inférieures à i sont utiles pour les calculs ultérieurs. Qu'il y ait cette instruction ou non, les colonnes d'indice inférieur à i ne sont pas changées. Cette instruction peut donc être éliminée.

La deuxième partie nécessite 2p(p-1) divisions plus p(p-1)·p multiplications et soustractions, il faut donc au total des opérations dont le nombre est $2p(p-1)+p(p-1)\cdot p+p(p-1)\cdot p=2p(p-1)(p+1)$

La troisième partie a besoin de $p^2(p-1)/2$ multiplications et soustractions, soit un ordre de grandeur de p^3.

Il faut au total $p(3p^2-p-2)$ opérations pour calculer la matrice inverse, c'est-à-dire de l'ordre de p^3 opérations.

Le déterminant de la matrice peut être calculé en parallèle avec l'inversion.

Le coût du calcul d'une métrique de Mahalanobis a alors un ordre de grandeur proportionnel à $(2\omega+3p)p^2$, qui se simplifie ici en ωp^2 (car on a toujours $p\leq\omega$).

2° Utilisation d'une métrique diagonale

Pour déterminer une métrique diagonale, il faut tout d'abord calculer les écart-types des variables. Pour cela, le calcul des moyennes des variables est nécessaire.

L'écart-type de la variable i peut être calculé en utilisant la formule suivante:

$$\sigma(x_i) = \sqrt{\frac{\sum_{k=1}^{\omega} \left(x_{i,k} - \bar{x}_i\right)^2}{\omega}} = \sqrt{\frac{\sum_{k=1}^{\omega} \left(x_{i,k}^2 - 2x_{i,k}\bar{x}_i + \bar{x}_i^2\right)}{\omega}}$$

$$= \sqrt{\frac{\sum_{k=1}^{\omega} x_{i,k}^2 - 2\omega \cdot \bar{x}_i \cdot \bar{x}_i + \omega \cdot \bar{x}_i^2}{\omega}} = \sqrt{\frac{\sum_{k=1}^{\omega} x_{i,k}^2}{\omega} - \bar{x}_i^2}$$

On adopte ainsi l'algorithme suivant pour le calcul d'un écart-type

$X1_i:=0$, $X2_i:=0$;
Pour j de 1 à ω faire
    $X1_i:=X1_i+x_{i,j}$;
    $X2_i:=X2_i+x_{i,j}^2$;
Finpour

$$X2_i:=\sqrt{\frac{X2_i}{\omega} - \left(\frac{X1_i}{\omega}\right)^2}$$

$X2_i$ est alors l'écart-type de la variable x_i.

Pour calculer un écart-type, il faut 2ω additions, ω+3 multiplications, une soustraction, puis le calcul d'une racine carrée. C'est-à-dire (3ω+5) opérations.

Pour calculer la métrique, il faut des opérations dont le nombre est
$p\cdot(3\cdot\omega+5)+p=p\cdot(3\cdot\omega+6)$

L'ordre de grandeur est proportionnel à ωp opérations.

Sur un SUN 3/110, pour 600 exemples et 90 caractéristiques, il faut 40 heures pour calculer les distances en utilisant la métrique de Mahalanobis. Ce temps se réduit à une heure et demie si l'on utilise la métrique diagonale.

• *Le temps de calcul des grandeurs caractéristiques*

Pour calculer les grandeurs caractéristiques, il faut calculer, pour chaque région, la métrique qui lui est associée. La formule de calcul du temps de ce traitement a été établie plus haut: il suffit de remplacer ω par le nombre

d'occurrences d'énoncé de la région (on ne peut plus simplifier la formule $(2\omega+3p)p^2$).

Remarque:

Les centre de gravité des régions peuvent être calculés pendant le calcul des métriques de Mahalanobis ou diagonale.

Pour calculer les rayons Rmax et Rmin, il faut calculer les distances entre les centres de gravité et toutes les occurrences d'énoncé.

En notant ρ le nombre total de régions pour toutes les heuristiques et υ_i le nombre d'occurrences d'énoncé dans la région i, on a: $\sum_{i=1}^{\rho}\upsilon_i$ qui est du même ordre de grandeur que $\eta\cdot\omega$.

Les temps de calcul des grandeurs caractéristiques en nombre d'opérations élémentaires sont respectivement des ordres suivants selon le choix de métrique.

1° Pour la métrique de Mahalanobis

$$[3\rho\omega p^2+\sum_{i=1}^{\rho}(2\upsilon_i+3p)p^2]\equiv p^2(3\rho\omega+3\rho p+2\eta\omega)$$

2° Pour la métrique diagonale

$$[3\rho\omega p+\sum_{i=1}^{\rho}3\upsilon_i p]\equiv 3\omega p(\rho+\eta)$$

3° Pour la métrique euclidienne

$$2\omega\rho p+\sum_{i=1}^{\rho}\upsilon_i p\equiv\omega p(2\rho+\eta)$$

En résumé nous obtenons un tableau récapitulatif suivant:

Table 2.1. Co-ût d'organisation

	Mahalanobis	Diagonale	Euclidienne
Recouvrement	$3\omega^2p^2/2$	$2\omega^2p$	$3\omega^2p/2$
Structuration	$p^2(3\rho\omega+3\rho p+2\eta\omega)$	$3\omega p(\rho+\eta)$	$\omega p(2\rho+\eta)$

Le coût de construction de la mémoire artificielle dépend fortement du choix de la métrique. D'après les analyses de ce coût, on remarque que l'utilisation de la métrique diagonale coûte beaucoup moins cher que la métrique de Mahalanobis et qu'elle coûte à peine plus cher que la métrique euclidienne. C'est pourquoi les expériences que nous présentons au chapitre 4 utilisent la métrique diagonale. Par ailleurs, pendant l'affectation, l'utilisation d'une métrique de Mahalanobis coûte très cher. Afin de limiter le temps d'affectation, nous utilisons exclusivement la métrique diagonale au cours de l'utilisation de la mémoire artificielle.

2.3.2. UTILISATION DE LA MEMOIRE: AFFECTATION

Nous présentons l'utilisation de la mémoire dans le cas monocritère, nous donnerons simplement à la fin de ce paragraphe quelques idées pour une utilisation interactive multicritère.

La mémoire étant construite, le problème qui nous intéresse maintenant est de considérer une nouvelle occurrence d'énoncé e et, en utilisant la mémoire, de trouver le plus rapidement possible l'heuristique à lui appliquer afin d'essayer de minimiser un critère donné.

Ceci demande de chercher à attacher la nouvelle occurrence d'énoncé e, pour chaque heuristique, à des régions. On suppose alors qu'en appliquant cette heuristique, e obtiendrait la même modalité que celle des occurrences d'énoncé de cette région. Nous obtenons ainsi une modalité supposée pour chaque heuristique. En comparant ces modalités, nous pouvons décider d'appliquer une des heuristiques qui fournissent la meilleure modalité. Pour éliminer les ex-æquos, lorsque plusieurs heuristiques fournissent cette meilleure modalité, nous nous donnons un ordre de préférence. Cette ordre est défini par une hiérarchie de type suivant: une heuristique est préférée si, pendant l'expérimentation,
1° elle est celle qui donne le plus souvent la meilleure valeur au critère;
2° à égalité, elle donne strictement la meilleure valeur le plus souvent;
3° à égalité, elle donne la meilleure valeur moyenne pour le critère.

Nous examinons les heuristiques dans cette ordre, c'est-à-dire dans l'ordre h_1, h_2, ..., h_η, où h_1 est celle qui est préférable, h_2 est la suivante dans l'ordre de préférence, etc.

Avant de présenter en détail l'algorithme d'affectation, nous définissons les notion de faible attachement et de fort attachement.

Définitions 2.7:

Pour une région R d'une heuristique h, on dira qu'une occurrence d'énoncé e est **faiblement attachée** à la région R si on a $d_{M_R}(G_R,e) \leq Rmax_R$ (cette relation signifie que la nouvelle occurrence d'énoncé se trouve à l'intérieur de la plus petite zone contenant toutes les occurrences d'énoncé de la région R) et que l'occurrence d'énoncé e est **fortement attachée** à la région R si on a $d_{M_R}(G_R,e) \leq Rmin_R$ (cette relation signifie que la nouvelle occurrence d'énoncé se trouve à l'intérieur d'une zone ne contenant que les occurrences d'énoncé de la région R).

L'algorithme est de forme itérative et s'arrête dès que l'on trouve une heuristique de meilleure modalité possible à laquelle l'occurrence d'énoncé peut être attachée le plus fortement possible (à égalité, on retient l'heuristique la mieux classée dans l'ordre des heuristiques défini ci-dessus).

De manière plus détaillée:

On examine successivement les modalités par valeurs décroissantes (jusqu'à la valeur 2).

Pour une valeur de modalité donnée, on examine les possibilités d'attachement de l'occurrence d'énoncé à des régions associées à chaque heuristique, d'abord en considérant les attachements forts, puis en considérant les attachements faibles.

Pour une modalité donnée k et pour un type d'attachement (fort ou faible), on examine les heuristiques dans l'ordre de leur classement et on cherche pour chacune d'elle la meilleure modalité d'une des régions auxquelles l'occurrence d'énoncé peut être attachée pour le type d'attachement considéré.

Si la meilleure modalité trouvée est égale à la modalité considérée k, alors on retient l'heuristique correspondante et on arrête la recherche.

Si les itérations se terminent sans que l'on ait choisi une heuristique, alors on demande un raffraîchissement de la mémoire car l'occurrence d'énoncé est probablement dans une région pauvre en expériences de la mémoire ; mais on applique l'heuristique numéro 1, pour fournir immédiatement un résultat pas trop mauvais à l'utilisateur.

Signalons qu'un autre chercheur de notre laboratoire ([Voyiatzis 1987]) a également construit une mémoire artificielle pour piloter un système de production. Il utilise une méthode de classification automatique hiérarchique pour construire la mémoire et une méthode d'affectation hiérarchique basée sur des conditions de proximité. Dans son travail, les classes ne sont pas obtenues à partir des valeurs du critère mais à partir des caractéristiques des occurrences d'énoncé ; mais à chaque classe est associée un vecteur d'informations sur le comportement global de chaque heuristique pour la classe (par exemple, le pourcentage d'occurrences d'énoncé dans la classe qui obtient la meilleure modalité pour un critère donné). Il utilise ces valeurs pour choisir l'heuristique après affectation de la nouvelle occurrence d'énoncé à la classe.

Pour une utilisation multicritère de la mémoire artificielle, on peut utiliser un tableau d'informations construit pour l'utilisation monocritère (mais non détaillé dans l'algorithme présenté ci-dessus sous forme littéraire).

A partir d'une occurrence d'énoncé, on peut construire, pour chaque heuristique et pour chaque critère, la liste éventuellement vide, contenant la référence et la modalité, des régions auxquelles l'occurrence est fortement (ou faiblement) attachée.

De même que dans le cas monocritère, on utilise cette liste, mais de manière interactive avec l'utilisateur.

On peut dans un premier temps lui fournir pour chaque critère, la liste des modalités possibles que l'on peut espérer obtenir avec certaines des heuristiques et le meilleur attachement correspondant.

En conversationnel, il peut ou bien analyser le résultat donné par une des heuristiques et la retenir éventuellement, ou bien limiter le champ des modalités possibles pour chaque critère et avoir des conclusions sur les conséquences pour les autres critères; ainsi en modifiant de manière dynamique les modalités minimales souhaitées pour chaque critère, il finira par déduire l'heuristique qui lui convient le mieux.

Une autre façon plus simple de travailler dans un atelier de manière multicritère consiste à choisir son critère au moment où l'on considère la nouvelle occurrence d'énoncé (qui peut être un sous ensemble non encore terminé de l'occurrence d'énoncé précédente dont on recalcule l'ordonnancement en raison d'aléas).

2.4. REALISATION DES MODULES SPECIFIQUES A CHAQUE APPLICATION

2.4.1. PROBLEMES D'ORDONNANCEMENT DE TYPE JOB-SHOP

Dans cette section, nous allons montrer comment réaliser les modules de la mémoire artificielle en prenant comme application les problèmes d'ordonnancement de type Job-shop.

2.4.1.1. Génération aléatoire des occurrences d'énoncé

Nous supposons que nous utilisons la mémoire artificielle pour un atelier particulier. Aussi, les types de machines et les types produits pouvant être fabriqués dans l'atelier sont connus et décrits par des paramètres. Les nombres de types de machines et de produits sont respectivement M et N. Il y a donc autant de réalisations de mémoire artificielle possibles que d'ateliers, car la dispersion des occurrences d'énoncé sera propre à l'atelier.

Pour chacun des N types de produits, on génère le nombre d'opérations à effectuer, les machines utilisées par les opérations ainsi que les durées opératoires correspondantes.

Pour générer les gammes de produits, on se donne les intervalles suivants:

[on,om] fourchette (2) du nombre d'opérations d'un produit;

[ln,lm] fourchette de la plus petite durée des opérations d'un produit;

[rn,rm] fourchette du rapport de la durée la plus grande des opérations sur la durée la plus petite des opérations.

---

2 Nous utilisons le mot "fourchette" pour désigner l'intervalle de définition des données correspondantes lors de la génération.

Algorithme de génération de N gammes:

Pour i de 1 à N faire
    Générer un nombre d'opérations g_i sur [on,om];
    Générer la plus petite durée des opérations ℓ sur [ln,lm];
    Générer un entier j1 sur $[1,g_i]$;
    $l_{i,j1}:=\ell$;
    Pour chaque $j\neq j1$ et $1\leq j\leq g_i$
        Générer une durée opératoire $l_{i,j}$ sur $[\ell\cdot rn,\ell\cdot rm]$;
    finpour
    Pour j de 1 à g_i faire
        Générer un numéro de machine $\mu_{i,j}$ sur [1,M] tel que $\mu_{i,j}\neq\mu_{i,j-1}$
    finpour
finpour

A ce niveau, on a généré un catalogue de produits. Pour générer une occurrence d'énoncé on tire de manière aléatoire une suite de produits dans ce catalogue et pour chacun d'eux on génère une date d'arrivée et une date de fin souhaitée.

Nous supposons qu'il existe des familles d'occurrences d'énoncé ayant des caractéristiques voisines que nous générons avec les mêmes paramètres, et d'autres familles définies par d'autres valeurs des paramètres. Ce sont ces paramètres qui rendent les occurrences d'énoncé différentes les unes des autres.

A chaque famille d'occurrences d'énoncé sont associés six paramètres:

[nn,nm]: intervalle contenant le nombre de produits à fabriquer.
$\alpha 1$: coefficient utilisé pour la génération des dates d'arrivées des produits;
$\alpha 2$, $\alpha 3$, $\alpha 4$: coefficients utilisés pour générer les marges des produits.

Algorithme de génération d'une commande ou occurrence d'énoncé

Générer un nombre de produits n sur [nn,nm];
1° Pour i de 1 à n faire
    Générer un numéro de type de produit τ sur [1,N];
    Attribuer au produit i les caractéristiques de type τ.

$g_i := g'_x$;
Pour j de 1 à g_i faire
$l_{i,j} := l'_{x,j}$;
$\mu_{i,j} := \mu'_{x,j}$;
finpour
finpour.

2° Calculer la charge W de la machine la plus chargée:

$$W = \max_{\mu}\left(\sum_{\mu_{i,j}=\mu} l_{i,j}\right)$$

Génération des dates d'arrivées et des dates de fin souhaitées des produits:
3° Pour i de 1 à n faire
Générer la date d'arrivée sur r_i sur $[0,\alpha 1 \cdot W]$
Générer le délai d_i sur $[r_i+p_i, r_i+p_i+\alpha 2 \cdot W+\alpha 3 \cdot p_i+\alpha 4 \cdot g_i]$
finpour

2.4.1.2. Caractérisation des occurrences d'énoncé

Comme, pour un ensemble de commandes, le nombre de machines a été fixé pour l'atelier considéré, mais que le nombre de produits et le nombre d'opérations sont variables et en outre relativement grands, il faut remplacer les jeux de données numériques correspondant à une commande par un vecteur de caractéristiques de $\mathbb{R}^p$ avec p fixé. **La phase de caractérisation est un des aspects les plus difficiles de la construction de la mémoire artificielle.** Elle doit vérifier les deux conditions suivantes:
1° Bien représenter les informations des jeux de données.
2° Limiter le nombre de caractéristiques retenues.

Pour les problèmes d'ordonnancement de type Job-shop, nous proposons la caractérisation suivante.

Nous utilisons deux scalaires sans transformation:
n: Nombre de produits dans la commande.
NOP: Nombre total d'opérations dans la commande.

Nous utilisons également les moyennes et les écart-types des vecteurs suivants:
1° Dates d'arrivée des produits.

2° Durées de fabrication des produits.
3° Délais des produits
4° Marges absolues des produits (d_i-(r_i+p_i))
5° Ratios de marges des produits ([d_i-(r_i+p_i)]/(d_i-r_i))
6° Nombres d'opérations des produits.
7° Charges des machines
8° Nombres d'opérations sur les machines
9° Durées des opérations.

Cela fournit au total 20 caractéristiques. La caractérisation retient les informations agrégées qui nous semblent les plus importantes et les plus significatives pour tester les ressemblances entre deux jeux de données.

2.4.1.3. Heuristiques utilisées

Dans la littérature, on trouve essentiellement deux types d'algorithmes approchés: les méthodes énumératives tronquées et les constructions de solutions obtenues en appliquant des règles de priorité ([Panwalker & Iskander 1977]). Les premières méthodes ne sont pas applicables lorsque la taille des problèmes devient grande, surtout lorsque le nombre d'occurrences d'énoncé est important.

Dans notre étude, nous nous limitons donc aux algorithmes utilisant des règles de priorité pour choisir entre plusieurs opérations disponibles au pied de la machine ou sur le point d'arriver.

Nous utilisons pour certaines de ces règles des ordonnancements sans délais, (c'est-à-dire les ordonnancements dans lesquels on ne laisse jamais la machine inoccupée s'il existe au moins un produit au pied de la machine qui attend pour subir une opération), ou des ordonnancements actifs, où on peut laisser la machine inoccupée pour attendre une opération urgente qui n'est pas encore au pied de la machine (mais il ne doit pas exister de produits au pied de la machine que l'on pourrait exécuter sur l'intervalle où on laisse la machine inoccupée).

Chaque règle donne naissance à une ou plusieurs heuristiques. Les règles utilisées dans notre approche sont:

1. Règles tenant compte des durées d'exécution:

(1) SPT: priorité à l'opération ayant la plus petite durée d'exécution.

(2) LPT: priorité à l'opération ayant la plus grande durée d'exécution.

(3) SR: priorité à l'opération d'un travail correspondant à un minimum de durées opératoires restant à réaliser (la plus petite somme de durées pour toutes les opérations non encore exécutées).

(4) LR: similaire à (3) en remplaçant "minimum" par "maximum".

(5) LRM: similaire à (4), mais le travail restant n'inclut pas la durée de l'opération envisagée.

(6) LSPON: priorité à l'opération telle que l'opération suivante dans la gamme a la plus grande durée opératoire.

2. Règles tenant compte des dates de fin au plus tard:

(7) EDD: priorité à l'opération appartenant à un produit ayant la plus petite date de fin au plus tard.

(8) OPNDD: priorité à l'opération ayant la plus petite date de fin au plus tard par opération. Les dates de fin au plus tard par opération sont calculées préalablement pour chaque opération de chaque produit suivant un principe de répartition de la marge entre les opérations. La marge est ou bien partagée également entre les opérations, ou encore partagée proportionnellement à la durée de chaque opération, d'où (8') et (8").

3. Règles tenant compte du nombre d'opérations :

(9) FOPNR: priorité à l'opération qui appartient à un produit dont le nombre d'opérations restant à effectuer est minimal.

(10) MOPNR: similaire à (9) en remplaçant "minimal" par "maximal".

(11) LHALF: priorité à l'opération d'un produit dont au plus la moitié des opérations restent à exécuter.

(12) FHALF: similaire à (11) en remplaçant "au plus" par "au moins".

4. Règles tenant compte des dates d'arrivée au plus tôt et/ou des dates de fin au plus tard:

(13) FIFO: priorité à l'opération qui est arrivée le plus tôt au pied de la machine.

(14) FASFO: priorité à l'opération appartenant à un produit arrivé le plus tôt dans l'atelier.

(15) LIFO: priorité à l'opération arrivée le plus tard au pied de la machine.

(16) S_1: priorité à l'opération appartenant à un produit ayant le moins de marge restante (date de fin au plus tard moins durée opératoire totale restante).

(17) S_2: priorité à l'opération appartenant à un produit ayant le moins de marge statique (date de fin au plus tard moins durée totale d'exécution, moins date d'arrivée au plus tôt du produit).

(18) S_1/OP: priorité à l'opération appartenant au produit ayant le plus petit ratio défini par la marge restante divisée par le nombre d'opérations restantes.

(19) S_2/OP: priorité à l'opération appartenant à un produit ayant le plus petit ratio défini par la marge statique divisée par le nombre d'opérations restantes.

(20) JSR: priorité à l'opération appartenant au produit ayant le plus petit ratio défini par la marge restante divisée par le temps restant (intervalle entre l'instant donné et la date de fin au plus tard).

(21) RSPT1: priorité à l'opération appartenant à un produit ayant le plus petit ratio défini par la marge restante divisée par la somme des durées de toutes les opérations non encore terminées.

5. Règles tenant compte des machines:

(22) NINQ: priorité à l'opération dont le successeur immédiat va aller sur une machine ayant le moins d'opérations en attente au pied de la machine.

(23) WINQ: priorité à l'opération dont le successeur immédiat va aller sur une machine ayant la plus petite somme de durées opératoires pour les opérations en attente au pied de la machine.

Nous utilisons également les règles de priorité PRMT, PRTT, PRTF définies dans le chapitre 3 et appliquées aux délais par opération obtenus en répartissant la marge entre les opérations.

2.4.2. PROBLEME D'ORDONNANCEMENT A UNE MACHINE

Pour expérimenter l'approche mémoire artificielle, nous construisons aussi une mémoire artificielle pour le problème d'ordonnancement à une machine.

2.4.2.1. Génération des occurrences d'énoncé

On se donne des intervalles et des coefficients suivants:

[nn, nm] Intervalle contenant le nombre de tâches.

[l1, l2] Intervalle contenant la durée des tâches.

$\alpha 1$: Coefficient pour calculer les dates d'arrivées des tâches

$\alpha 2$, $\alpha 3$ ($\alpha 3 \geq 1$, $\alpha 3 > \alpha 1$): Coefficients pour calculer les délais des tâches

Pour chaque série d'exemples, on change ces paramètres.

L'algorithme utilisé pour la génération des occurrences d'énoncé est le suivant:

Générer n sur [nn,nm];
PT:=0;
Pour i de 1 à n faire
    Générer la durée p_i pour la tâche i sur [l1,l2];
    PT:=PT+p_i;
finpour
Pour i de 1 à n faire
    Générer la date d'arrivée r_i sur [0,$\alpha 1 \cdot$PT];
    Générer la marge μ_i sur [0,$\alpha 2 \cdot (\alpha 3 - \alpha 1) \cdot$PT];
    $d_i := r_i + p_i + \mu_i$
finpour

2.4.2.2. Caractérisation des occurrences d'énoncé

Comme pour le problème de Job-shop, nous caractérisons les occurrences d'énoncé par:
n: nombre de tâches
et les moyennes et les écart-types des vecteurs suivants:
r: vecteur des dates d'arrivées
p: vecteur des durées
d: vecteur des délais
μ: vecteur des marges

Il y a donc au total 9 variables.

Toutes les moyennes et tous les écart-types sont divisés par la durée moyenne des tâches. Aussi, la variable indiquant la moyenne des durées des tâches prend toujours la valeur 1 et on peut la supprimer. Le nombre de variables devient ainsi 8.

2.4.2.3. Heuristiques utilisées

Nous utilisons des règles de priorité existantes, et les règles de priorité que nous présentons dans le chapitre 3. Les algorithmes sont construits en combinant les règles de priorité avec des générateurs d'ordonnancements.

Certaines sont plus sophistiquées. Tous ces algorithmes sont présentés dans chapitre 3.

2.4.3. PROBLEMES DE CONTAINER LOADING

Dans ce paragraphe, nous nous intéressons à l'utilisation de la mémoire artificielle pour un problème de "container loading". Cette mémoire a pour but de choisir la meilleure stratégie parmi celles proposées par le progiciel C.I.A.P. (Computer Integrated Automatic Packing) développé dans notre laboratoire. Nous présentons, comme pour les autres applications, les modules spécifiques qui permettent d'intégrer le progiciel C.I.A.P. dans une mémoire artificielle.

2.4.3.1. Génération des occurrences d'énoncé

Nous avons programmé un générateur de jeux de données. Pour chaque commande, on a des objets à placer dans un emballage proposé. Il faut générer les dimensions des objets et de l'emballage ainsi que les hauteurs autorisées (en supposant que l'objet n'est pas obligatoirement posé sur sa plus grande surface).

A chaque objet sont associés trois dimensions et trois booléens indiquant si une dimension est autorisée comme hauteur ou non. Un emballage est défini par trois dimensions: longueur, largeur et hauteur.

En ce qui concerne la génération, nous distinguons trois familles d'objets selon leur volume: des objets gros, des objets moyens et des objets petits. Nous distinguons également quatre familles différentes d'objets selon leur forme: des objets cubiques, des objets plats, des objets allongés et des objets allongés plats.

On se donne les paramètres suivants pour générer une commande, ou plutôt une famille de commandes:

- Nombre minimal et nombre maximal d'objets dans une commande (2 paramètres).
- Volumes minimaux et maximaux pour chaque catégorie d'objets: gros, moyens ou petits (6 paramètres).
- Fourchettes des rapports entre tout couple de dimensions pour définir des objets plutôt cubiques, plats, allongés ou allongés et plats (16 paramètres).

- Pourcentages des objets gros, moyens et petits (2 paramètres).
- Pourcentages d'objets de forme cubique, plate, allongée, allongée et plate pour chaque catégorie d'objets gros, moyens ou petits. (12 paramètres)
- Fourchette du rapport du volume de l'emballage sur le volume total des objets.
- Fourchette du nombre d'emballages proposés.

Nous générons tout d'abord le nombre d'objets à l'intérieur de la fourchette correspondante. Ensuite, nous générons respectivement le nombre d'objets gros, le nombre d'objets moyens, et le nombre d'objets petits, selon les pourcentages pour chaque type d'objets. Puis les nombres d'objets gros cubiques, moyens cubiques, etc. sont déterminés suivant les pourcentages pour chaque forme et chaque type d'objets. Les volumes sont générés à l'intérieur de la fourchette du volume pour chaque cas. Les rapports des dimensions sont ensuite générés à l'intérieur de leurs fourchettes. Les dimensions de chaque objet sont calculées en considérant le volume et les rapports entre les dimensions. Les trois booléens indiquant l'autorisation des dimensions comme hauteur sont générés (avec une, deux ou trois hauteurs autorisées [3]) selon la forme de l'objet.

2.4.3.2. Caractérisations des occurrences d'énoncé

Pour les problèmes de bin-packing, nous avons prévue deux caractérisations possibles. La première consiste à utiliser les caractéristiques brutes (voir A). La deuxième consiste à transformer les données brutes comme pour les problèmes d'ordonnancement de job-shop ou à une machine. Les caractéristiques obtenues sont appelées caractéristiques transformées.

A) CARACTERISTIQUES BRUTES

Dans le cadre de l'utilisation initiale prévue du progiciel C.I.A.P., le nombre d'objets ne dépasse jamais 30. Ceci nous permet de travailler sur les données brutes sans avoir à les agréger en caractéristiques. Les objets sont supposés numérotés dans l'ordre des volumes décroissants. En cas d'ex æquo, on prend successivement les trois dimensions de l'objets pour éliminer les cas ex æquo. Si une commande n'a pas 30 objets à placer, les dimensions restantes sont remplacées par des 0.

---

3 On ne permet pas, par exemple, de redresser les objets plats alors qu'un objet plutôt cubique peut avoir de une à trois hauteurs autorisées.

Les caractéristiques sont les rapports des paramètres suivants sur la hauteur de l'emballage de manière à supprimer l'indétermination liée à l'unité de dimensions ou à des transformations homothétiques sur les occurrences d'énoncé. Ce rapport est multiplié par -1 si la dimension correspondante n'est pas une hauteur autorisée.

Le nombre d'objets à placer
La longueur de l'emballage
La largeur de l'emballage
La plus grande dimension de l'objet n° 1
La dimension intermédiaire de l'objet n° 1
La plus petite dimension de l'objet n° 1
La plus grande dimension de l'objet n° 2
.
.
.
La plus grande dimension de l'objet n° 30
La dimension intermédiaire de l'objet n° 30
La plus petite dimension de l'objet n° 30

Il y a donc au total 93 variables.

B) CARACTERISTIQUES TRANSFORMEES

Nous caractérisons les occurrences d'énoncé par les valeurs suivantes.

Valeurs scalaires:

- NO: nombre d'objets à placer;
- NI: nombre maximal d'objets identiques dans la commande;
- ND: nombre de valeurs de dimensions différentes dans la commande;
- NDO: nombre d'objets différents dans la commande;
- NHD1: nombre d'objets pour lesquels la plus petite dimension d1 est une hauteur autorisée;
- NHD2: nombre d'objets pour lesquels la dimension intermédiaire d2 est une hauteur autorisée;
- NHD3: nombre d'objets pour lesquels la plus grande dimension d3 est une hauteur autorisée;
- NH1: nombre d'objets pour lesquels une seule hauteur est autorisée;
- NH2: nombre d'objets pour lesquels deux hauteurs sont autorisées;

rve: taux de remplissage demandé;
HLOE: le rapport de la hauteur de l'emballage sur sa longueur;
HLAE: le rapport de la hauteur de l'emballage sur sa largeur;

Outre ces valeurs scalaires, nous retenons encore les moyennes et les écart-types des vecteurs suivants:

D1: vecteur de plus petites dimensions des objets;
D2: vecteurs des dimensions intermédiaires des objets;
D3: vecteur de plus grandes dimensions des objets;
DG: vecteur de toutes les dimensions des objets (concaténation des vecteurs D1, D2 et D3);
DS123: vecteur des sommes des dimensions des objets (D1+D2+D3);
V: vecteur des volumes des objets;
RD12: rapport de la plus petite dimension sur la dimension intermédiaire;
RD23: rapport de la dimension intermédiaire sur la plus grande dimension;
HLO: rapport de la hauteur sur la longueur pour les seuls objets pour lesquels une seule hauteur est autorisée;
HLA: rapport de la hauteur sur la largeur pour les seuls objets pour lesquels une seule hauteur est autorisée;
HLOM: rapport de la hauteur sur sa longueur pour les seuls objets qui ont deux ou trois hauteurs autorisées;
HLAM: rapport de la hauteur sur la largeur pour les seuls objets qui ont deux ou trois hauteurs autorisées;
REP: vecteur de NI entiers qui représentent le nombre d'objets existants respectivement en 1, 2, ..., NI exemplaires identiques dans la commande;
SIMH: vecteur de ND entiers qui représentent le nombre de fois où une valeur de dimension comme hauteur autorisée apparaît dans la commande;
SIMO: vecteur de NDO entiers qui représentent le nombre de fois où un objet apparaît dans la commande;

Il y a donc au total 42 variables (12 valeurs scalaires et les moyennes et les écart-types de 15 vecteurs).

2.4.3.3. Heuristiques utilisées: présentation du Progiciel C.I.A.P.

Le progiciel C.I.A.P. place des objets parallélépipédiques dans un emballage parallélépipédique en tenant compte de contraintes d'équilibre et de fragilité des objets. L'objectif est de placer un maximum d'objets dans un emballage donné ou de limiter la hauteur de l'emballage pour une surface de base donnée (boîte à hauteur variable composée de deux demi-boîtes qui s'emboîtent l'une dans l'autre).

Comme pour les autres heuristiques présentées pour les autres applications, le progiciel utilise une heuristique "gourmande" ou "gloutonne". Elle consiste à examiner les objets suivant un ordre donné et à placer à un niveau considéré (hauteur pour la base des objets) le premier objet possible en le plaçant localement le mieux possible. Elle ne remet jamais en cause un placement décidé antérieurement. L'algorithme se résume donc à une suite de placements localement "optimaux".

Le résultat dépend de l'ordre dans lequel on considère les objets et des critères retenus pour choisir l'emplacement de l'objet dans l'emballage. Le choix des informations fournissant un ordre sur les objets et des critères de placement définit une stratégie (ou heuristique de la mémoire artificielle).

Dans le progiciel, sont définis trois indices: un indice de blocage, un indice de surplomb ou équilibre et un indice de nivellement. Un emplacement est considéré comme d'autant meilleur que la somme pondérée de ces trois indices est plus grande. En outre, le manque de place au niveau considéré pour placer l'objet ou une condition d'équilibre minimum non vérifiée peut interdire le placement d'un objet (il sera reconsidéré à un niveau supérieur).

Les trois indices traduisent les règles de bon sens suivantes:

a) Un emplacement est d'autant meilleur que l'objet est mieux encastré entre des objets déjà placés ou contre les parois de l'emballage. Cette qualité est mesurée par un indice de blocage. Il agrège le nombre de degrés de liberté absents (face s'appuyant contre un autre objet ou une paroi) et la longueur du périmètre de base de l'objet placé qui vient en contact avec d'autres objets déjà placés ou avec les parois de l'emballage.

b) Un emplacement est d'autant meilleur qu'il prépare une surface plane connexe plus importante pour les niveaux supérieurs de manière à pouvoir poser ultérieurement de nouveaux objets en équilibre sur plusieurs objets. Cette qualité est mesurée par un indice de nivellement qui calcule la longueur du périmètre de base de la face supérieure de l'objet venant s'aligner contre des objets dont la face supérieure est à la même hauteur.

c) Un emplacement admissible est d'autant meilleur que l'objet a un meilleur équilibre, c'est-à-dire que la surface sur laquelle il repose est plus importante. Cette quantité est mesurée par un indice de surplomb.

La version actuelle comprend 2 listes complexes et six listes simples (voir ci-après). Les listes complexes servent éventuellement à choisir une première série d'objets qui viendront en priorité au fond de l'emballage selon le critère plus grand nombre d'objets (ou plus grande surface de base couverte par les objets) ayant tous une même valeur pour une de leurs hauteurs autorisées. On poursuit ensuite avec les listes simples qui fournissent l'ordre des objets pour le reste du remplissage.

Une liste complexe ne peut être utilisée qu'avec une liste simple. Une liste simple, par contre, peut être utilisée toute seule. Grâce à ces combinaisons, nous obtenons au total 18 stratégies différentes.

Les critères de tri des listes complexes sont les suivants:

Liste complexe n° 1:
Elle est partitionnée en classes selon les hauteurs autorisées des objets. A l'intérieur d'une classe, on trie hiérarchiquement selon
- Volume décroissant
- Longueur correspondante décroissante

Liste complexe n° 2:
Elle est partitionnée en classes selon les hauteurs autorisées des objets. A l'intérieur d'une classe, on trie hiérarchiquement selon
- Maximum des dimensions décroissant
- Volume décroissant

Les tris des listes simples utilisent les hiérarchies de critères suivantes:

Liste simple n° 1:
- Volume décroissant
- Surface porteuse minimale croissante
- Longueur correspondante décroissante

Liste simple n° 2:
- Volume décroissant
- Plus grande dimension décroissante
- Surface porteuse maximale décroissante

Liste simple n° 3:
- Surface porteuse minimale décroissante
- Volume décroissant
- Longueur correspondante décroissante

Liste simple n° 4:
- Surface porteuse maximale décroissante
- Volume décroissant
- Longueur correspondante décroissante

Liste simple n° 5:
- Volume décroissant
- Surface porteuse minimale décroissante
- Longueur correspondante décroissante

Liste simple n° 6:
- Volume décroissant
- Surface porteuse maximale décroissante
- Longueur correspondante décroissante

Une différence importante existe entre la manipulation des listes simples selon que leur numéro est pair ou impair. Pour les listes de numéro impair, les objets sont toujours considérés a priori comme dressés sur leur plus petite surface porteuse autorisée (de manière à mieux s'introduire entre des objets déjà placés) et on abandonne cette position seulement quand le niveau de remplissage de l'emballage ne permet plus que l'on dresse ainsi l'objet. Au contraire, dans les listes paires, les objets sont triés en les supposant a priori posés sur leur plus

grande surface porteuse autorisée, mais à chaque fois, on examine si les autres positions de l'objet ne donne pas une meilleure valeur au critère de placement local (on est ici obligé de tester toutes les positions avant de rejeter l'objet, car sinon on risquerait de quitter un niveau de placement sans placer un objet qui peut l'être).

Les noms des algorithmes sont codés de façon suivante: pour les algorithmes utilisant uniquement une liste simple, le nom de l'algorithme est fourni par ce numéro; pour les algorithme utilisant une liste complexe, son nom est donné par le numéro de la liste simple suivi du numéro de la liste complexe.

Chapitre 3

ORDONNANCEMENT A UNE MACHINE

3.1. GENERALITE

3.1.1. INTRODUCTION

Les problèmes d'ordonnancement à une machine ont intéressé de nombreux chercheurs, comme nous le verrons dans la suite de ce chapitre.

Ces problèmes sont importants essentiellement pour deux raisons.

D'abord, les résultats auxquels ils conduisent sont utilisés dans le cas d'ordinateurs mono-processeurs et multi-usagers.

Ensuite, ces mêmes résultats sont utilisés pour la résolution de problèmes plus généraux. En présence de plusieurs machines il est possible, dans certaines situations, de n'ordonnancer que la machine goulot ou la machine la plus chère. Si une machine devient une machine goulot, plusieurs solutions sont envisageables.

* Les solutions "matérielles" consistent:

- soit à acheter une autre machine;
- soit à échanger cette machine contre une autre machine plus rapide;
- soit à modifier les gammes de façon à éviter cette machine.

* Les solutions "logicielles" consistent à améliorer l'ordonnancement de cette machine [1].

Lorsque les solutions matérielles sont difficiles à mettre en œuvre, les méthodes d'ordonnancement à une machine sont utilisables.

Souvent, une machine très chère est goulot d'étranglement car il n'est pas rentable de la laisser inactive et il est trop coûteux de la dupliquer.

Il est parfois également possible qu'une chaîne entière puisse être traitée comme une seule machine du point de vue de l'ordonnancement.

1: En G.P.A.O. OPT(Optimized Production Technology) considère qu'on ne doit ordonnancer que les machines "goulot" en "traduisant" l'influence des autres machines sous forme de contraintes ([Mollet 1989]).

Les problèmes d'ordonnancement sont presque tous NP-difficiles. Parmi ces problèmes, les problèmes à une machine restent les plus faciles à résoudre.

Dans la littérature, on trouve beaucoup d'algorithmes traitant ces problèmes. Il existe notamment des algorithmes polynomiaux pour résoudre certains problèmes particuliers. Mais, malheureusement, même pour les problèmes à une machine, beaucoup de problèmes sont NP-difficiles. Nous allons dans la suite présenter brièvement la complexité des problèmes, proposer des résultats théoriques que nous avons obtenus, et proposer des algorithmes exacts et approchés pour des problèmes NP-difficiles.

3.1.2. COMPLEXITE DES PROBLEMES

Parmi les problèmes d'ordonnancement à une machine, certains peuvent être résolus par un algorithme polynomial, certains autres sont démontrés NP-difficiles. Certains ne sont ni démontrés NP-difficiles, ni polynomialement résolvables. Ils restent donc ouverts. Nous rappelons les résultats connus en nous limitant aux cas où les tâches ne sont pas interruptibles.

Problèmes polynomiaux

Dans la littérature, les chercheurs ont résolu certains problèmes particuliers par un algorithme polynomial. Parmi les plus connus, on trouve les problèmes suivants ([Jackson 1955], [Rinnooy Kan 1976], [Smith 1956], [Moore 1968]). Nous rappelons les principaux résultats en utilisant les notations classiques en ordonnancement introduites au chapitre 1[(2)]:

1° Le problème $n/1/r_i=0/Cmax$ peut être résolu en ordonnançant les tâches dans n'importe quel ordre.

2° $n/1/r_i=0/Lmax$ peut être résolu en ordonnançant les tâches dans l'ordre des délais non décroissants (EDD: Earliest Due Date).

3° $n/1/r_i=0$, $p_i=D/\Sigma T_i$ peut être résolu en ordonnançant les tâches dans l'ordre des délais non décroissants (EDD).

---

[2] Les paramètres utilisés pour caractériser les problèmes signifient, dans l'ordre *(nombre de travaux / nombre de machines / conditions particulières / critère)*. Ils ont été présentés enb détail dans le chapitre 1.

4° $n/1/r_i \geq 0, p_i = 1/\sum T_i$ ou $n/1/r_i \geq 0$, $p_i = D/\sum T_i$, D divisant le PGCD (Plus Grand Commun Diviseur) des r_i, peut être résolu par la méthode de Jackson.

5° $n/1/r_i \geq 0/Cmax$ peut être résolu en ordonnançant les tâches dans l'ordre des dates d'arrivée croissantes (FIFO: First In First Out).

6° $n/1/r_i = 0/\sum w_i C_i$ peut être résolu en ordonnançant les tâches dans l'ordre des p_i/w_i non décroissants (SWPT: Shortest Weighted Processing Time).

7° $n/1/r_i = 0/\sum U_i$ est résolu par l'algorithme de Hodgson appelé aussi l'algorithme de Moore.

Problèmes NP-difficiles

Même dans le cas d'une machine, il existe encore des problèmes NP-difficiles. Dans [Rinnooy Kan 1976], l'auteur a démontré que les problèmes suivants sont NP-difficiles.

1° $n/1/r_i \geq 0/\sum C_i$
2° $n/1/r_i \geq 0/\sum w_i C_i$
3° $n/1/r_i \geq 0/\sum T_i$
4° $n/1/r_i = 0/\sum w_i T_i$
5° $n/1/r_i \geq 0/Lmax$
6° $n/1/r_i \geq 0/Tmax$

Récemment, il a été démontré ([Du & Leung 1990]) que $n/1/r_i = 0/\sum T_i$ est aussi NP-difficile.

Problèmes ouverts

Le problème suivant reste ouvert du point de vue de sa complexité:
$n/1/r_i \geq 0$, $p_i = D/\sum T_i$, D ne divisant pas le PGCD des r_i

Pour ce problème, Portmann (1987) a défini un sous-ensemble dominant, le sous-ensemble u-actif, et calculé une borne supérieure de l'écart entre la valeur du critère de toute solution de ce sous-ensemble et celle de l'optimum.

3.1.3. HYPOTHESES DES PROBLEMES TRAITES DANS CE CHAPITRE

Dans tout ce chapitre, on ne traite que les problèmes ayant les caractéristiques suivantes:

1° Les tâches n'arrivent pas en même temps dans l'atelier ($r_i \geq 0$).
2° Dans tous les critères considérés, nous accordons la même importance à toutes les tâches (les coefficients de pénalisation sur les tâches utilisés par d'autres auteurs sont donc ici égaux, par exemple: $w_i=1$).
3° Toutes les tâches sont non interruptibles, c'est-à-dire qu'une fois qu'une tâche est affectée à une machine, la machine ne sera à nouveau libre que lorsque cette tâche sera finie.
4° Une machine ne peut effectuer qu'une tâche à la fois.
5° Toutes les tâches sont indépendantes les unes des autres. C'est-à-dire qu'il n'y a pas de contraintes de précédence entre les tâches.

3.1.4. OBJECTIF ET RESULTATS DE CE CHAPITRE

Dans ce chapitre, nous ne traitons que des problèmes NP-difficiles. Nous nous intéressons à des conditions d'optimalité locale. L'**optimalité locale** est l'optimalité de deux tâches consécutives dans un ordonnancement donné, sans tenir compte d'éventuelles conséquences sur les tâches qui suivent. Une **condition suffisante d'optimalité locale** est une condition qui assure l'optimalité locale pour deux tâches consécutives.

Nous proposons des conditions suffisantes d'optimalité locale pour les problèmes $n/1/r_i \geq 0/\Sigma T_i$ et $n/1/r_i \geq 0/\Sigma F_i$, et nous utilisons des conditions démontrées par d'autres pour le problème $n/1/r_i \geq 0/Lmax$. En nous basant sur ces conditions, nous construisons quatre schémas d'algorithmes approchés: algorithmes "sans délais", "à règle pure", "à choix alternatif" et "avec phase d'insertion". Ces méthodes approchées ont été élaborées avec pour objectif de construire une mémoire artificielle pour les problèmes d'ordonnancement à une machine. Les conditions suffisantes d'optimalité locale seront généralisées comme règles de priorité pour les ordonnancements de type Job-shop utilisant la mémoire artificielle. D'autres conditions que nous démontrons ou que nous

trouvons dans la littérature nous ont également permis de construire de nouveaux algorithmes exacts de résolution de type P.S.E.

Dans la littérature, les problèmes de minimisation du retard maximal et de minimisation des temps de présence sont largement étudiés. Mais relativement peu de résultats ont été publiés en ce qui concerne le problème de la minimisation de la somme des retards. C'est pourquoi l'accent est plutôt mis sur ce critère.

Dans la section 3.2, nous présentons quatre schémas de construction d'algorithmes qui seront utilisés dans les sections suivantes. Dans la section 3.3, nous parlons brièvement du problème de la minimisation du retard maximal, car nous proposons seulement une méthode approchée basée sur une condition suffisante d'optimalité locale démontrée par d'autres chercheurs. La section 3.4, est consacrée à la minimisation de la somme des retards. Dans la section 3.5, la minimisation des temps de présence est examinée. Une condition nécessaire et suffisante d'optimalité locale sera donnée, ainsi que des algorithmes approchés et un algorithme exact.

3.2. SCHEMAS DE CONSTRUCTION D'ALGORITHMES

Dans cette section, nous présentons quatre schémas de construction d'algorithmes. Pour passer d'un schéma à un algorithme spécifique il suffit de préciser les paramètres soulignés dans chaque schéma. Dans ces schémas, le terme "hiérarchie de règles" signifie que l'on applique les règles de manière hiérarchique: si la première règle ne suffit pas pour sélectionner une tâche, plusieurs tâches étant ex-æquo, on utilise dans l'ordre les règles suivantes jusqu'à sélectionner une seule tâche. On termine si nécessaire par la règle qui consiste à choisir la tâche de plus petit indice (toujours discriminante).

1° Règle pure

A chaque itération, on ordonnance définitivement une tâche prioritaire. La priorité des tâches est totalement définie par le choix de la hiérarchie de règles.

2° Sans délai

A chaque itération, on ordonnance définitivement une tâche prioritaire disponible à l'instant considéré (plus petit instant où une tâche peut être placée). La priorité est totalement définie par le choix de la hiérarchie de règles.

3° Insertion

A chaque itération, on ordonnance définitivement une tâche prioritaire λ choisie selon une première hiérarchie de règles. On cherche ensuite s'il existe des tâches pouvant être terminées avant le début de la tâche λ. Des itérations secondaires placent définitivement un sous-ensemble de ces tâches en utilisant une seconde hiérarchie de règles.

4° Choix alternatif

A chaque itération, on sélectionne une tâche α selon une première hiérarchie de règles et une tâche β selon une seconde hiérarchie de règles.
Si une condition est vérifiée, on ordonnance définitivement α sinon β.

3.3. MINIMISATION DU RETARD MAXIMAL

Ce problème a été largement étudié dans la littérature. Citons en particulier les articles [Dessouky & Margenthaler 1972], [Shwimer 1972], [Bratley et al. 1973], [Baker & Su 1974]), [McMahon & Florian 1975], [Lageweg et al. 1976], [Potts 1980], [Carlier 1982], [Larson et al. 1985], [Grabowski et al. 1986], [Hall & Rhee 1986].

Dans [Dessouky & Margenthaler 1972], les auteurs ont démontré une condition suffisante d'optimalité locale analogue à celles que nous démontrons pour d'autres critères dans les sections suivantes. Les méthodes exactes proposées par certains auteurs ([Grabowski et al. 1986] par exemple) ne peuvent pas être utilisées en temps réel. Aussi, dans la mémoire artificielle, nous avons introduit des méthodes approchées incluant la propriété de Dessouky & Margenthaler selon les quatre types de schéma présentés au 3.2.

Des résultats expérimentaux sont présentés au chapitre 4.

Définition 3.1

On définit une fonction PRMT [3] d'une tâche i à l'instant Δ par:
$PRMT(i,\Delta)=R(i,\Delta)+d_i$
où $R(i,\Delta)$ est la date d'exécution au plus tôt de la tâche i à l'instant Δ (pour simplifier, on note R_i lorsqu'il y a pas d'ambiguïté).

Cette fonction est la somme de la date d'exécution au plus tôt d'une tâche i à l'instant Δ et de son délai. Cette fonction est indépendante de la durée opératoire de la tâche.

Dessouky et Margenthaler ont démontré le théorème suivant. Nous l'écrivons en italiques: *dans tout ce chapitre, nous utiliserons des italiques chaque fois que nous rappellerons des résultats démontrés par d'autres auteurs.*

Théorème 3.1 ([Dessouky & Margenthaler, 1972])

Une condition suffisante d'optimalité locale pour obtenir $Tmax_{ij} \leq Tmax_{ji}$ *à partir de l'instant* Δ *est*
$PRMT(i,\Delta) \leq PRMT(j,\Delta)$
où $Tmax_{ij}$ est le retard maximal des tâches i et j en ordonnançant i avant j.

3.4. MINIMISATION DE LA SOMME DES RETARDS

Cette section est consacrée au problème de minimisation de la somme des retards. Ce problème a été démontré NP-difficile ([Rinnooy Kan 1976]) même pour le cas particulier où les tâches arrivent en même temps dans l'atelier ([Du & Leung 1990]).

Le problème avec des dates d'arrivées identiques a beaucoup attiré l'attention des chercheurs. Les travaux les plus connus sont [Smith 1956], [Emmons 1969], [Lawler 1977, 1982], [Potts & Van Wassenhove 1982], [Fisher 1981]. Dans ces articles, les auteurs ont démontré des propriétés de dominance, et proposé des algorithmes performants.

Dans les revues internationales, nous n'avons pas trouvé d'articles proposant des résultats lorsque l'on traite des ordonnancements à une machine comportant des arrivées de tâches échelonnées dans le temps et utilisant

---

3: Sigle de l'expression anglaise "Priority Rule for Maximum Tardiness"

comme critère la somme des retards. Depuis quelques années, notre laboratoire travaille sur ce sujet ([Portmann 1987], [Chu & Portmann 1989, 1990a, 1990b, 1990c]).

Nous allons présenter une condition suffisante d'optimalité locale (Théorème 3.2), et proposer de nouveaux algorithmes approchés efficaces construits à partir de cette condition (Paragraphe 3.4.2). Cette condition conduit aussi à définir un ensemble dominant (Définition 3.4 et Corollaire 3.1). Nous présentons également une autre propriété de dominance (Théorème 3.4) ainsi qu'un minorant (Théorème 3.5) qui nous ont permis de construire une procédure par séparation et évaluation (PSE: voir paragraphe 3.4.2.2). Dans cette procédure, les propriétés de dominance sont utilisées pour éliminer des branches de l'arbre de recherche. Nous présentons encore une majoration de l'erreur dans le pire des cas pour un des algorithmes approchés (Théorème 3.7).

3.4.1. PROPRIETES DE DOMINANCE

Une condition suffisante d'optimalité locale est obtenue grâce à une nouvelle règle de priorité de chaque tâche face aux délais que nous appelons PRTT [4]. Comme PRMT, c'est une fonction de l'instant Δ et des paramètres de la tâche considérée.

Définition 3.2

On définit la fonction *"priorité face aux délais"* PRTT(i,Δ) pour la tâche i commençant au plus tôt à l'instant Δ par

$PRTT(i,\Delta)=R(i,\Delta)+\max(\Phi(i,\Delta),d_i)$

où $\Phi(i,\Delta)$ est la date d'achèvement au plus tôt de la tâche i à l'instant Δ.

Cette fonction est une fonction continue par morceaux. Elle peut être interprétée comme la somme de la date d'exécution au plus tôt et du délai modifié. Le délai modifié d'une tâche i à l'instant Δ est par définition $\max(\Phi(i,\Delta),d_i)$.

4: Sigle de l'expression anglaise "Priority Rule for Total Tardiness"

Nous allons montrer que cette fonction peut être considérée comme une règle dynamique de priorité des tâches. Une tâche est d'autant plus prioritaire que la valeur de la fonction PRTT correspondante est petite. Cette priorité évolue au cours du temps. On trouvera une étude de l'évolution de la priorité relative de deux tâches dans l'annexe A.

Théorème 3.2

Nous considérons le problème partiel d'ordonnancement de deux tâches i et j sur une machine disponible à partir de l'instant Δ.
Si nous avons $PRTT(i,\Delta) \leq PRTT(j,\Delta)$ alors placer i avant j minimise la somme des retards des deux tâches i et j.
C'est-à-dire:
$PRTT(i,\Delta) \leq PRTT(j,\Delta) \Rightarrow T_{ij} \leq T_{ji}$
où T_{ij} est la somme des retards des tâches i et j en ordonnançant i avant j.

Démonstration

En tenant compte des définitions et des contraintes du problème, on a:
$T_{ij}=\max(R_i+p_i-d_i,0)+\max[\max(R_i+p_i,R_j)+p_j-d_j,0]$
que l'on peut réécrire:
$T_{ij}=\max(R_i+p_i,d_i)+\max[\max(R_i+p_i+p_j,R_j+p_j),d_j]-(d_i+d_j)$
ce qui donne en utilisant l'associativité de l'opération "max":
$T_{ij}=\max(R_i+p_i,d_i)+\max[R_i+p_i+p_j,\max(R_j+p_j,d_j)]-(d_i+d_j)$
$=\max[R_i+\max(R_i+p_i,d_i)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$
$=\max[PRTT(i,\Delta)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$

De manière symétrique, on obtiendrait:
$T_{ji}=\max[PRTT(j,\Delta)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$

Posons:
$P_{ij}=p_i+p_j$,
$Q_{ij}=\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)$
$D_{ij}=d_i+d_j$

Ces trois expressions sont symétriques en i et en j et on a:
$T_{ij}=\max(PRTT(i,\Delta)+P_{ij},Q_{ij})-D_{ij}$ (3.1)
$T_{ji}=\max(PRTT(j,\Delta)+P_{ij},Q_{ij})-D_{ij}$ (3.2)
d'où $PRTT(i,\Delta) \leq PRTT(j,\Delta) \Rightarrow T_{ij} \leq T_{ji}$

Q.E.D.

La condition que nous venons de démontrer est une condition suffisante d'optimalité locale. Il est possible de montrer que cette condition est également nécessaire lorsque des conditions que nous allons préciser sont réunies.

Pour démontrer cette réciproque, nous allons introduire la notion de "délai corrigé" et montrer que l'ensemble des solutions optimales pour les délais corrigés est analogue à l'ensemble des solutions optimales pour les délais originaux, ce qui nous conduit à affirmer que toutes les méthodes d'ordonnancement qui cherchent à minimiser la somme des retards pourraient n'utiliser que les délais corrigés.

Définition 3.3:

On définit la notion de *"délai corrigé"* d_i^c de la tâche i par

$$d_i^c=\max(r_i+p_i,d_i) \qquad (3.3)$$

Cette quantité est le maximum entre le délai et l'instant de fin de fabrication au plus tôt.

Note: Si ce maximum est r_i+p_i, cela signifie que le délai imposé à la tâche i n'avait pas de sens. En d'autres termes, le délai corrigé est, dans la pratique, toujours égal au délai.

Lemme 3.1

Si l'on considère un problème P de minimisation de la somme des retards et le problème P' obtenu en remplaçant les délais d_i par les délais corrigés d_i^c, alors les problèmes P et P' sont équivalents.

Démonstration

On note T'_i le retard calculé avec comme délai d_i^c pour la tâche i, et T' la somme des retards calculés avec les "délais corrigés". Nous allons démontrer ce lemme en démontrant

$\forall i,\ T_i-T'_i=d_i^c-d_i$

Dans ce but, on distingue les deux seuls cas possibles selon que le retard de la tâche i est inévitable ou non.

1) cas où $r_i+p_i \geq d_i$

$(3.3) \Rightarrow d_i^c = r_i + p_i$ (3.4)

$t_i \geq r_i$ et $(3.4) \Rightarrow t_i+p_i \geq r_i+p_i = d_i^c \geq d_i$

D'où $T_i - T'_i = \max(t_i+p_i-d_i,0) - \max(t_i+p_i-d_i^c,0) = t_i+p_i-d_i-(t_i+p_i-d_i^c) = d_i^c - d_i$

2) cas où $r_i+p_i<d_i$

$(3.3) \Rightarrow d_i^c = d_i$

D'où $T_i - T'_i = \max(t_i+p_i-d_i,0) - \max(t_i+p_i-d_i^c,0) = 0 = d_i^c - d_i$

Dans tous les cas, on a:

$T_i - T'_i = d_i^c - d_i$

$$\Rightarrow T-T' = \sum_{i=1}^{n} T_i - \sum_{i=1}^{n} T'_i = \sum_{i=1}^{n} (T_i - T'_i) = \sum_{i=1}^{n} (d_i^c - d_i) = \text{CONSTANTE}$$

Q.E.D.

Théorème 3.3

Nous considérons le problème partiel d'ordonnancement de deux tâches i et j sur une machine disponible à partir de l'instant Δ.
Si nous travaillons avec des "délais corrigés" notés d_i et d_j et si nous avons $T_{ji}>0$ alors
$T_{ij} \leq T_{ji} \Rightarrow PRTT(i,\Delta) \leq PRTT(j,\Delta)$

Démonstration

Comme nous travaillons sur des délais corrigés nous avons:
$\forall k, \quad d_k \geq r_k + p_k$ (3.5)

Et nous allons montrer que:
$PRTT(i,\Delta) > PRTT(j,\Delta) \Rightarrow T_{ij} > T_{ji}(\Delta)$ ou $T_{ij} = T_{ji} = 0$

Pour cela, nous considérons à nouveau les formules (3.1) et (3.2) :
$T_{ij} = \max(PRTT(i,\Delta)+P_{ij}, Q_{ij}) - D_{ij}$
$T_{ji} = \max(PRTT(j,\Delta)+P_{ij}, Q_{ij}) - D_{ij}$

Et nous constatons qu'il faut considérer plusieurs cas en raison de la présence du maximum.

a) $PRTT(i,\Delta)+P_{ij} > Q_{ij}$

Alors on a:

$PRTT(i,\Delta) > PRTT(j,\Delta)$
$\Rightarrow PRTT(i,\Delta)+P_{ij} > \max (PRTT(j,\Delta)+P_{ij} , Q_{ij})$
$\Rightarrow \max (PRTT(i,\Delta)+P_{ij} , Q_{ij}) > \max (PRTT(j,\Delta)+P_{ij} , Q_{ij})$
$\Rightarrow \max (PRTT(i,\Delta)+P_{ij} , Q_{ij}) - D_{ij} > \max (PRTT(j,\Delta)+P_{ij} , Q_{ij}) - D_{ij}$
$\Rightarrow T_{ij} > T_{ji}$

b) $PRTT(i,\Delta)+P_{ij} \leq Q_{ij}$

En combinant avec $PRTT(i,\Delta) > PRTT(j,\Delta)$ on obtient:
$T_{ij}=T_{ji}= Q_{ij} - D_{ij}$

et on voudrait alors montrer que $Q_{ij} = D_{ij}$.

$PRTT(i,\Delta) > PRTT(j,\Delta) \Rightarrow PRTT(i,\Delta)+P_{ij} > PRTT(j,\Delta)+P_{ij}$
et $PRTT(i,\Delta)+P_{ij} \leq Q_{ij} \Rightarrow Q_{ij} \geq PRTT(i,\Delta)+P_{ij} > PRTT(j,\Delta)+P_{ij}$

En développant ces dernières égalités, elles s'écrivent:
$\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j) \geq R_i+\max(R_i+p_i,d_i)+P_{ij} > R_j+\max(R_j+p_j,d_j)+P_{ij}$

De la première inégalité on déduit:
$\max(R_j+p_j,d_j) \geq R_i+P_{ij} = R_i+ p_i +p_j$ (3.6)

Des deux termes extrêmes on déduit:
$\max(R_i+p_i,d_i) > R_j+P_{ij} = R_j+ p_i +p_j$ (3.7)

On étudie les différents cas possibles.

***b1)* $R_j+p_j \geq R_i+p_i+p_j$**

Dans ce cas, on a:
$R_j+p_j+p_i \geq R_i+p_i +p_j+p_i > R_i+p_i$

De (3.7) on déduit que:
$d_i > R_i+p_i$ (3.8)

Avec l'inéquation du cas b1) on déduit encore:
$R_j \geq R_i + p_i$
$\Rightarrow r_j > \Delta$ (car sinon on aurait: $R_j = \max(r_j,\Delta) = \Delta \leq \max(r_i,\Delta) = R_i < R_i + p_i$)
$\Rightarrow r_j = R_j$

et avec (3.5) on obtient:
$d_j \geq R_j + p_j$ (3.9)

De (3.8) et (3.9) en revenant à la définition de Q_{ij} et de D_{ij} on obtient:
$Q_{ij} = \max(R_i+p_i,d_i) + \max(R_j+p_j,d_j) = d_i + d_j = D_{ij}$

b2) **$R_j+p_j < R_i+p_i+p_j$**

De (3.6), on retrouve immédiatement (3.9).

On étudie les deux sous-cas issus de (3.7):

b2.1) $R_i+p_i \geq R_j+p_i+p_j$

Ce cas est symétrique au cas b1) en échangeant les indices i et j (car (3.6) et (3.7) ainsi que les autres éléments de la démonstration sont symétriques en i et j).

b2.2) $R_i+p_i < R_j+p_i+p_j$

De (3.7), on retrouve immédiatement (3.8).

(3.8) et (3.9) retrouvés dans ce cas nous donnent encore $Q_{ij} = D_{ij}$.

Q.E.D.

Nous utilisons maintenant le théorème d'optimalité locale pour définir un nouveau sous-ensemble dominant de solutions pour le critère somme des retards.

Définition 3.4

Considérons un ordonnancement actif O pour un problème du type $n/1/r_i/\Sigma T_i$.
Si pour tout couple (i,j) de tâches consécutives (j suivant i) on a:

$R(i,\phi_i) < R(j,\phi_i)$

ou $PRTT(i,\phi_i) \leq PRTT(j,\phi_i)$

où ϕ_i est la date de fin de la tâche précédant immédiatement la tâche i dans l'ordonnancement O (si i est la première tâche dans O, $\phi_i=0$), alors on dit que **l'ordonnancement O est T-actif.**

Le théorème 3.2 nous permet d'obtenir le corollaire suivant:

Corollaire 3.1

Dans le cas du problème à une machine, le sous-ensemble des ordonnancements T-actifs est dominant pour le critère "somme des retards".

Démonstration

La démonstration est faite par "échange de paires consécutives".

Supposons qu'il existe une solution optimale O pour un problème donné, et que O ne soit pas T-actif. Alors il existe dans O au moins une paire de tâches consécutives qui ne vérifient aucune des deux conditions de la définition de T-activité. Soit i_1 et j_1 une telle paire de tâches, alors j_1 a une plus petite priorité face au délai que i_1 et ou bien j_1 arrive avant i_1 ou bien les deux tâches sont disponibles à l'instant Δ. Selon le théorème 3.2, mettre j_1 avant i_1 n'augmente pas la somme des retards des deux tâches et de même la somme des retards des tâches suivantes n'augmente pas car la fin des deux tâches n'est pas retardée. On construit une suite finie [5] d'ordonnancements en échangeant la première paire de tâches consécutives qui ne vérifie pas la T-activité dans l'ordonnancement précédent et l'on arrête lorsque toutes les paires consécutives vérifient la T-activité. On obtient un ordonnancement O' T-actif dont la somme des retards est au plus aussi grande que celle de O.

Q.E.D.

Remarque

Si les tâches arrivent toutes en même temps ou sont toutes arrivées à l'instant où l'on commence à ordonnancer (par exemple à l'instant 0), la condition suffisante d'optimalité locale conduisant à mettre i avant j à partir de l'instant Δ devient:

$\max(\Delta+p_i,d_i) \leq \max(\Delta+p_j,d_j)$

C'est bien la condition suffisante proposée dans [Smith 1956].

Nous allons démontrer une autre propriété de dominance. Bien qu'elle décrive un cas restrictif, elle est utile pour éliminer des branches de l'arbre de

5: Parce que l'ensemble d'ordonnancements est fini et que l'on ne considère jamais deux fois le même.

recherche. Avant de démontrer cette propriété, nous démontrons un lemme qui va servir à plusieurs démonstrations.

Lemme 3.2

Soient $(x_1, x_2, \ldots, x_n)$ et $(y_1, y_2, \ldots, y_n)$ deux suites quelconques de n nombres réels. Soit $(x'_1, x'_2, \ldots, x'_n)$ une suite ordonnée par valeurs non décroissantes de n nombres réels telle que $\forall i,\ 1\le i\le n,\ x'_i\le x_i$.
Si $(y'_1, y'_2, \ldots, y'_n)$ est la suite de nombres réels obtenue en triant $(y_1, y_2, \ldots, y_n)$ par valeurs non décroissantes, alors on a:

$$\sum_{i=1}^{n}\max(x_i-y_i,0)\ge\sum_{i=1}^{n}\max(x'_i-y'_i,0)$$

Démonstration:

Nous démontrons ce lemme par récurrence sur la longueur des suites.

Lorsque n=1, le lemme est évidemment juste.

Démontrons le aussi pour n=2.

Dans ce cas, nous avons $y'_2\ge y_2$ et $y_1+y_2=y'_1+y'_2$

$$\begin{aligned}
&\max(x_1-y_1, 0)+\max(x_2-y_2, 0)\\
=&\max(x_1-y_1+x_2-y_2, x_1-y_1, x_2-y_2, 0)\\
=&\max(x_1-y'_1+x_2-y'_2, x_1-y_1, x_2-y_2, 0) && (y_1+y_2=y'_1+y'_2)\\
\ge&\max(x'_1-y'_1+x'_2-y'_2, x'_1-y_1, x'_2-y_2, 0) && (x'_1\le x_1, x'_2\le x_2)\\
=&\max(x'_1-y'_1+x'_2-y'_2, x'_1-y_1, x'_1-y_2, x'_2-y_2, 0) && (x'_2\ge x'_1)\\
=&\max(x'_1-y'_1+x'_2-y'_2, x'_1-y'_1, x'_2-y_2, 0) && (y'_1=\min(y_1,y_2))\\
\ge&\max(x'_1-y'_1+x'_2-y'_2, x'_1-y'_1, x'_2-y'_2, 0) && (y'_2\ge y_2)\\
=&\max(x'_1-y'_1, 0)+\max(x'_2-y'_2, 0)
\end{aligned}$$

Supposons maintenant que le lemme soit juste pour $n\le k$, nous démontrons qu'il est aussi juste pour n=k+1.

Soit $(y''_1, y''_2, \ldots, y''_k)$ la suite de nombres réels obtenue en triant par valeurs non décroissantes la sous suite $(y_1, y_2, \ldots, y_k)$ (On a $y''_k=\max_{1\le i\le k}(y_i)$).

D'après l'hypothèse de récurrence au rang k, nous avons

$$XY=\sum_{i=1}^{k+1}\max(x_i-y_i,0)\ge\sum_{i=1}^{k}\max(x'_i-y''_i,0)+\max(x_{k+1}-y_{k+1},0)$$

D'après l'hypothèse du lemme au rang k+1, on a: $x_{k+1} \geq x'_{k+1}$, d'où

$$XY \geq \sum_{i=1}^{k} \max(x'_i - y''_i, 0) + \max(x'_{k+1} - y_{k+1}, 0)$$

$$= \sum_{i=1}^{k-1} \max(x'_i - y''_i, 0) + \max(x'_k - y''_k, 0) + \max(x'_{k+1} - y_{k+1}, 0)$$

En posant

$a_1 = x'_k$, $b_1 = y''_k$

$a_2 = x'_{k+1}$, $b_2 = y_{k+1}$

on applique le lemme pour n=2 aux trois suites (a_1, a_2), (b_1, b_2) et $(a'_1, a'_2) = (a_1, a_2)$ alors la suite (b'_1, b'_2) est définie par

$b'_1 = \min\{b_1, b_2\} = \min(y''_k, y_{k+1})$

$b'_2 = \max(b_1, b_2) = \max(y''_k, y_{k+1}) = \max_{1 \leq i \leq k+1}(y_i) = y'_{k+1}$

d'où

$$XY \geq \sum_{i=1}^{k-1} \max(x'_i - y''_i, 0) + \max(x'_k - b'_1, 0) + \max(x'_{k+1} - y'_{k+1}, 0)$$

On applique le lemme pour n=k. Car les ensembles

$\{y''_1, y''_2, \ldots, y''_{k-1}, b'_1\} = \{y_1, y_2, \ldots, y_{k+1}\} - \{y'_{k+1}\}$

$\{y'_1, y'_2, \ldots, y'_k\} = \{y_1, y_2, \ldots, y_{k+1}\} - \{y'_{k+1}\}$

coïncident, et que la suite $(y'_1, y'_2, \ldots, y'_k)$ est triée, on a:

$$XY \geq \sum_{i=1}^{k} \max(x'_i - y'_i, 0) + \max(x'_{k+1} - y'_{k+1}, 0) = \sum_{i=1}^{k+1} \max(x'_i - y'_i, 0)$$

Q.E.D.

Théorème 3.4

Etant donnée une séquence partielle S, s'il existe deux tâches $i,j \in N\text{-}S$ telles que $p_i \geq p_j$, $\Phi(i,E(S)) \leq \Phi(j,E(S))$ et $d_i \leq d_j$ alors $T(\Sigma(S,i)) \leq T(\Sigma(S,j))$.

E(S) représente la date de fin de la dernière tâche de la séquence S (0 si la séquence S est vide) ; E(S) est notée Δ dans la démonstration. $\Sigma(S,j)$ est une séquence constituée de S, suivie de j, complétée par une sous-séquence optimale.

Démonstration

Considérons un ordonnancement $\Sigma(S,j)$.

Construisons un autre ordonnancement O' en échangeant les positions des tâches i et j, et en gardant les emplacements des autres tâches (voir les notations sur la figure 3.1). Puisque l'on a $\Phi(i,\Delta)\leq\Phi(j,\Delta)$, et $p_i\geq p_j$, alors O' est réalisable.

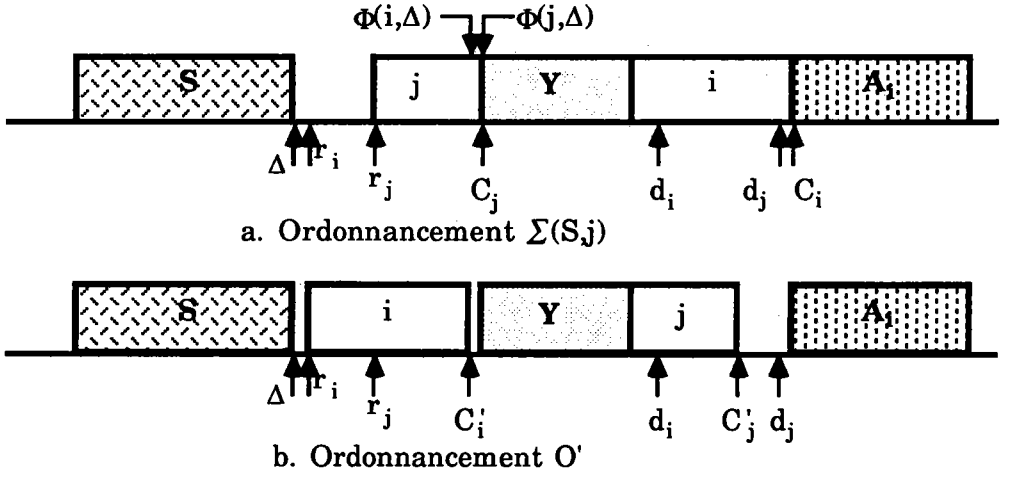


Fig. 3.1 Illustration du théorème 3.4

Comme les emplacements des autres tâches ne sont pas modifiés, on a:
$\forall k\in S\cup Y\cup A_i$, $T'_k=T_k$
où Y est l'ensemble de tâches entre i et j dans $\Sigma(S,j)$.
On en déduit que:

$$T(\Sigma(S,j))-T'=T_i+T_j-T'_i-T'_j$$
$$=\max(C_j-d_j,0)+\max(C_i-d_i,0)-\max(C'_i-d_i,0)-\max(C'_j-d_j,0)$$

Comme on a encore:
$C'_i\leq C_j$, $C'_i<C'_j\leq C_i$

On peut appliquer le lemme 3.2 avec $(x_1,x_2)=(C_j,C_i)$, $(y_1,y_2)=(d_j,d_i)$, $(x'_1,x'_2)=(C'_i,C'_j)$ et $(y'_1,y'_2)=(d_i,d_j)$ d'où on déduit $T'\leq T(\Sigma(S,j))$
Par définition de $\Sigma(S,i)$, on a: $T(\Sigma(S,i))\leq T'$
d'où: $T(\Sigma(S,i))\leq T(\Sigma(S,j))$.

Q.E.D.

3.4.2. ALGORITHMES DE RESOLUTION

En ce qui concerne les méthodes de résolution, nous proposons des méthodes approchées (section 3.4.3.3) et une méthode optimale du type "Procédure par Séparation et Evaluation" ou PSE (section 3.4.3.2) qui nécessite le calcul de minorants de bonne qualité que nous présentons dans la section 3.4.3.1.

3.4.2.1. Minorants de la somme des retards

Dans une procédure par séparation et évaluation qui minimise un critère donné, on supprime tout sous-ensemble de solutions dont le minorant est plus grand que la valeur du critère d'une solution réalisable déjà trouvée. La méthode est d'autant plus performante qu'on dispose au départ d'une bonne solution réalisable, que les minorants sont calculés le plus rapidement possible, et qu'ils sont les plus proches possibles du minimum du sous-ensemble.

On trouve dans la littérature des articles construisant des PSE pour le problème $n/1/r_i=0/\sum T_i$ (par exemple [Fisher 1976]) et même pour le problème $n/1/r_i=0/\sum w_i T_i$ (voir par exemple [Potts & Van Wassenhove 1985]), mais nous n'avons rien trouvé lorsque les dates d'arrivée des tâches sont différentes, d'où la nécessité de construire une nouvelle PSE qui puisse tourner en un temps raisonnable sur des exemples les plus gros possibles (une vingtaine de tâches actuellement).

Nous présentons maintenant la calcul d'un minorant calculé en $O(n^2)$ ou en O(nlogn) si on utilise une structure de données adéquate. Dans ce calcul, nous avons besoin de la règle SRPT (Shortest Remaining Processing Time) que nous présentons d'abord.

Algorithme SRPT (Shortest Remaining Processing Time)

On ordonnance à chaque itération une tâche dont la durée opératoire restante est la plus petite. Son exécution peut être interrompue lorsqu'une tâche, ayant une durée opératoire strictement plus petite que la durée restante de la tâche en cours, arrive. Dans l'ordonnancement obtenu, la machine n'est jamais laissée inoccupée lorsqu'il existe une tâche arrivée et non terminée (ordonnancement sans délai où on accepte d'interrompre les tâches).

Théorème 3.5

Etant donné un problème P(r,p,d) ([6]), soient:

i) $(d'_1, d'_2, \ldots, d'_n)$ la suite triée par valeurs non décroissantes correspondant à la suite $(d_1, d_2, \ldots, d_n)$;

ii) Ω le problème relaxé d'ordonnancement obtenu à partir du problème P en supposant que les tâches sont interruptibles;

iii) O_Ω un ordonnancement du problème Ω, non nécessairement réalisable pour P, obtenu en appliquant la règle SRPT.

Pour tout O, ordonnancement réalisable de P, on a:

$$\sum_{i=1}^{n} \max(C[i,O_\Omega]-d'_i,0) \leq T(P,O)$$

où $C[i,\sigma]$ est la date de fin de la tâche se terminant à la $i^{ième}$ position dans l'ordonnancement σ, $T(\Pi,\sigma)$ est la somme des retards de l'ordonnancement σ (réalisable pour le problème Π) calculée à partir des données du problème Π.

Démonstration

La démonstration consiste à appliquer le lemme 3.2. Il suffit de démontrer que
$\forall i,\ C[i,O_\Omega] \leq C[i,O]$
où $C[i,\sigma]$ est la date de fin de la tâche se terminant à la $i^{ième}$ position dans la séquence σ.

On peut constater que O est réalisable (pas forcément semi-actif) pour Ω.

Il suffit de démontrer que pour tout ordonnancement O" réalisable (pas forcément semi-actif) pour Ω, nous avons:

$$\forall i,\ C[i,O_\Omega] \leq C[i,O"] \qquad (3.10)$$

La démonstration utilise le même procédé que celui employé dans [Schrage 1968] pour démontrer l'optimalité de la règle de priorité SRPT pour la minimisation du nombre moyen de tâches dans un système de file d'attente.

Dans cette démonstration, nous notons:

$$a(i,t,S) = \begin{cases} 1 \text{ si la tâche i est exécutée à l'instant t dans l'ordonnancement } S \\ 0 \text{ sinon} \end{cases}$$

---

[6] P(r,p,d) désigne le problème dont les paramètres sont donnés par les vecteurs $r=(r_i)_{i=1,\ldots,n}$, $p=(p_i)_{i=1,\ldots,n}$ et $d=(d_i)_{i=1,\ldots,n}$.

$p(i,t,S)$ la durée opératoire restante de la tâche i à l'instant t dans l'ordonnancement S.
$\theta(t,S)$ l'ensemble des tâches non terminées et disponibles à être exécutées à l'instant t dans l'ordonnancement S.

Nous avons alors ([Schrage 1968])

$$\forall\Delta,\ \forall S,\ C(i,S)=\min\left\{x/\int_{\Delta}^{x} a(i,t,S)dt=p(i,\Delta,S)\right\}$$

$\forall\Delta,\ \forall S,\ \forall i\neq j,\ a(i,\Delta,S)+a(j,\Delta,S)\leq 1$
$\forall\Delta,\ \forall S,\ \theta(\Delta,S)=\{i/p(i,\Delta,S)>0 \text{ et } r_i\leq\Delta\}$

Un ordonnancement S vérifie la règle SRPT si et seulement si S vérifie la propriété suivante:
$\forall t,\ a(i,t,S)=1 \Rightarrow i\in\theta(t,S)$ et $p(i,t,S)=\min_{j\in\theta(t,S)}(p(j,t,S))$

(C'est-à-dire qu'une tâche en cours de traitement est obligatoirement une des tâches disponibles dont la durée opératoire restante est la plus petite.)

Démontrons maintenant la relation (3.10).

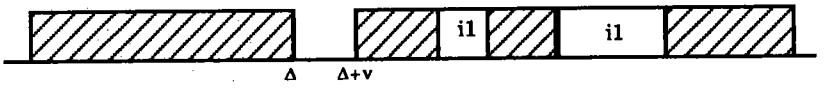
Si O" ne respecte pas la règle SRPT, il existe certainement un intervalle de temps $(\Delta,\Delta+v)$ avec $v>0$ et une tâche $i1\in\theta(\Delta,O")$ tels que:
$\forall i\in\theta(\Delta,O"),\ i\neq i1,\ p(i1,\Delta,O")<p(i,\Delta,O")$ et $\forall t,\ \Delta\leq t\leq\Delta+v,\ a(i1,t,O")=0$

Deux cas sont alors à envisager.

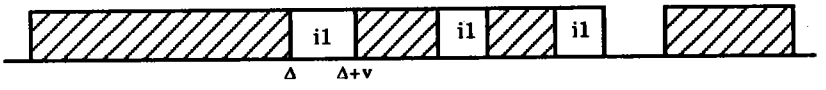
Cas 1° $\forall i\in\theta(\Delta,O"),\ \forall t,\ \Delta\leq t\leq\Delta+v,\ a(i,t,O")=0$

Dans ce cas, la machine est laissée inoccupée pendant l'intervalle $(\Delta,\Delta+v)$.

Construisons un autre ordonnancement $O^{(1)}$ en exécutant une partie de i1 sur l'intervalle $(\Delta,\Delta+v)$, et en avançant les autres parties de i1, si elles existent (voir la figure 3.2). Alors, les dates d'achèvement des autres tâches ne sont pas changées, et celle de i1 est diminuée. C'est-à-dire:
$C(i1,O^{(1)})\leq C(i1,O")$ et $\forall i,\ i\neq i1,\ C(i,O^{(1)})=C(i,O")$
où $C(i,\sigma)$ est la date de fin de la tâche i dans l'ordonnancement σ.



a. Ordonnancement O"



b. Ordonnancement $O^{(1)}$

Fig. 3.2. Cas 1°

Examinons à présent la modification des $C[i,O^{(1)}]$ par rapport aux C[i,O"]. Deux cas sont possibles.

1°a) La position de la tâche i1 (et des autres tâches) est la même dans O" et dans $O^{(1)}$ (la fin de i1 dans $O^{(1)}$ ne précède pas la fin d'une autre tâche qui se terminait avant i1 dans O"). Dans ce cas, on a:
$\forall i,\ C[i,O^{(1)}] \leq C[i,O"]$

1°b) La position de la tâche i1 est antérieure dans $O^{(1)}$ par rapport à O": la fin de i1 précède la fin d'une suite (triée) de tâches $k_1, k_2, \ldots, k_l$ ($l \geq 1$) (qui la précédait dans O") on a alors

$$C(i1,O^{(1)}) \leq C(k_1, O^{(1)}) = C(k_1,O")$$
$$C(k_1,O^{(1)}) = C(k_1,O") \leq C(k_2,O")$$
$$\vdots$$
$$C(k_l,O^{(1)}) = C(k_l,O") \leq C(i1,O")$$

On en déduit immédiatement que
$\forall i,\ C[i,O^{(1)}] \leq C[i,O"]$

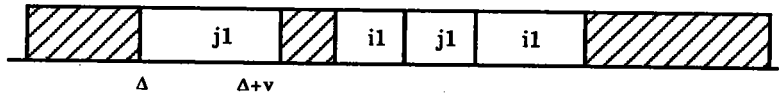
Cas 2° $\exists j1 \in \theta(\Delta,O"),\ \forall t,\ \Delta \leq t \leq \Delta+v,\ a(j1,t,O")=1$

Construisons un autre ordonnancement $O^{(1)}$ dans lequel i1 est exécutée aux instants suivants

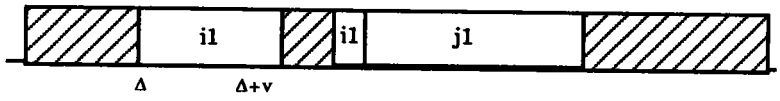
$$\left\{ y / a(i1,y,O)+a(j1,y,O)=1 \text{ et } y \leq \min\left\{ x / \int_{\Delta}^{x} (a(i1,t,O)+a(j1,t,O))dt = p(i1,\Delta,O) \right\} \right\}$$

et où j1 reprend les derniers emplacements libérés par i1. C'est-à-dire que la tâche i1 est exécutée aux instants qui étaient affectés, dans O", soit à la tâche

i1, soit à la tâche j1, jusqu'à la fin de i1, la tâche j1 est ensuite exécutée aux instants restants (voir la figure 3.3).
On a alors $\forall w$, $w \neq i1$, $w \neq j1$, $C(w,O^{(1)})=C(w,O'')$
$C(i1,O^{(1)}) \leq \min(C(i1,O''),C(j1,O''))$
$C(j1,O^{(1)})=\max(C(i1,O''),C(j1,O''))$



a. Ordonnancement O''



b. Ordonnancement $O^{(1)}$

Fig. 3.3 Cas 2°

Une démonstration analogue à celle du 1° (analysant les possibilité des fins de tâches, l'avance de la fin de i1 et l'utilisation éventuelle de la fin de i1 pour j1) permet d'obtenir à nouveau:
$\forall i$, $C[i,O^{(1)}] \leq C[i,O'']$

Dans tous les cas, on peut construire un ordonnancement $O^{(1)}$ tel que
$\forall i$, $C[i,O^{(1)}] \leq C[i,O'']$

Nous considérons ensuite $O^{(1)}$. Si $O^{(1)}$ ne vérifie pas SRPT, alors nous construisons un autre ordonnancement $O^{(2)}$ à partir de $O^{(1)}$, par le même procédé que le procédé utilisé pour passer de O'' à $O^{(1)}$. Cette démarche est poursuivie jusqu'à obtenir un ordonnancement $O^{(n)}$ vérifiant SRPT.

Par construction, nous avons
$C[i,O^{(n)}] \leq \ldots \leq C[i,O^{(2)}] \leq C[i,O^{(1)}] \leq C[i,O'']$
et pour tout ordonnancement O_Ω du problème Ω obtenu par la règle SRPT, on a: $\forall i$, $C[i,O^{(n)}]=C[i,O_\Omega]$ [7]

Q.E.D.

---

[7]: La règle SRPT peut laisser une part de choix aléatoire en cas d'égalité mais la suite des dates de fin de tâches est toujours la même.

Remarque

Le tri de $(d_1, d_2, ..., d_n)$ a une complexité en O(nlogn). L'application de la règle SRPT demande une prise de décision à chaque fin de tâches et à chaque arrivée de tâches, on choisit alors entre au plus n tâches candidates. Le calcul de ce minorant est bien en $O(n^2)$. Si on met les tâches candidates sous la forme d'un arbre binaire, le calcul de ce minorant se réduit à O(nlogn).

Si nous voulons obtenir plus rapidement un minorant , il est possible d'utiliser le minorant présenté sous la forme du théorème suivant.

Théorème 3.6

Etant donné un problème P(r,p,d) où les tâches sont numérotées dans l'ordre de leur durée opératoire (la suite $(p_1, p_2, ..., p_n)$ est une suite par valeurs non décroissantes), soient:

i) $(d'_1, d'_2, ..., d'_n)$ la suite triée correspondant à la suite $(d_1, d_2, ..., d_n)$;

ii) $(r'_1, r'_2, ..., r'_n)$ la suite non décroissante définie par $\forall i,\ r'_i=\min_{j\geq i}(r_j)$;

iii) O' l'ordonnancement, non nécessairement réalisable, du problème P obtenu en relaxant les r_i par les r'_i, et en ordonnançant les tâches dans l'ordre de leur numéro (C'_i est alors la date d'achèvement de la tâche i).

Pour tout ordonnancement O de P, on a la relation suivante:

$$\sum_{i=1}^{n}\max(C'_i-d'_i,0)\leq T(P,O)$$

Remarque

Il est évident par définition que la suite $(r'_1, r'_2, ..., r'_n)$ est non décroissante puisqu'il s'agit d'une valeur minimale définie sur une suite d'ensembles inclus les uns dans les autres quand i décroît. Le calcul de ce minorant nécessite un temps de calcul en O(n) en utilisant une structure de données adéquate (par exemple, on garde toujours la liste des délais des tâches non ordonnancées dans l'ordre non décroissant comme dans la PSE que nous avons construite et que nous présentons plus tard).

Démonstration

Nous démontrons ce théorème en montrant que le minorant est toujours inférieur ou égal à celui calculé à partir du théorème 3.5.

Pour cela, nous construisons le problème P", relaxation du problème P' lorsque l'on accepte que les tâches soient interrompues. Comme $\forall j$ et k tels que $j \leq k$ on a $r'_j \leq r'_k$ et $p'_j \leq p'_k$, l'ordonnancement O" obtenu en appliquant SRPT au problème P" est identique à l'ordonnancement O' si en cas d'égalité on choisit toujours la tâche de plus petit numéro. Nous avons donc $\forall i$: C[i,O"]=C[i,O'].

Nous construisons ensuite le problème $P^{(3)}$, relaxation du problème P lorsque l'on accepte que les tâches soient interrompues. Soit $O^{(3)}$ l'ordonnancement obtenu en appliquant SRPT au problème $P^{(3)}$.

Lors de la démonstration du théorème 3.5, nous avons montré que la tâche se terminant au rang i dans un ordonnancement construit selon la règle SRPT se termine au moins aussi tôt que la tâche se terminant au rang i dans un autre ordonnancement réalisable quelconque. Comme $O^{(3)}$ est réalisable pour P" et que O' vérifie SRPT pour P", nous pouvons appliquer ici ce résultat:

$\forall i$, $C[i,O'] \leq C[i,O^{(3)}]$

Ces dernières conditions jointes aux caractéristiques des paramètres du problèmes P' permettent d'appliquer le lemme 3.2 et d'obtenir:

$$\sum_{i=1}^{n} \max(C[i,O']-d'_i,0) \leq \sum_{i=1}^{n} \max(C[i,O^{(3)}]-d'_i,0)$$

où le membre de droite de l'inégalité est le minorant calculé dans le théorème 3.5.

Q.E.D.

Remarque

La construction de P' à partir de P demande un calcul des r'_i (en O(n) [8]) et une construction de l'ordonnancement O' (en O(n)). Le tri des délais peut être évité en adoptant une structure de données adéquate. La complexité de l'algorithme de calcul de ce minorant est donc bien en O(n). Bien que le temps de calcul soit plus court, ce minorant est moins proche de l'optimum que le précédent. Les expériences confirment bien que la PSE utilisant ce minorant est moins efficace que celle utilisant le minorant proposé dans le théorème 3.5. Comme le nombre de tâches dans les exemples est relativement petit, le gain sur la rapidité n'est pas suffisant pour compenser la perte de qualité du

8: On calcule tout d'abord, $r'_n=r_n$, et ensuite pour tout i inférieur à n, $r'_i=\min(r'_{i+1}, r_i)$.

minorant. Ce minorant pourrait devenir intéressant pour les premières séparations de problèmes de plus grande taille (traités sur des ordinateurs plus puissants, par exemple utilisant des processeurs en parallèle).

3.4.2.2. Procédure par Séparation et Evaluation

Dans ce paragraphe, nous présentons une procédure par séparation et évaluation en utilisant les résultats théoriques introduits ci-dessus. Les théorèmes 3.2 (ou le corollaire 3.1) et 3.4 peuvent être utilisés pour éliminer des sous-ensembles de solutions qui ne peuvent pas conduire à une solution optimale ou qui conduisent à une solution donnant une somme des retards au moins aussi grande que celle pouvant être obtenue dans un autre sous-ensemble.

Les théorèmes 3.5 ou 3.6 peuvent être intégrés dans cette procédure pour calculer un minorant pour chaque sous-ensemble.

Les méthodes approchées du paragraphe suivant servent à obtenir une "bonne" solution réalisable initiale. Après expérimentation, nous nous sommes limités aux meilleures heuristiques. Elles pourraient aussi servir à déterminer un majorant de chaque sous-ensemble. Nous avons abandonné cette possibilité car elle augmentait le coût associé à chaque séparation de manière non négligeable sans avoir d'effet important sur les éliminations possibles.

L'algorithme est composé de trois phases: initialisation, séparation et terminaison.

L'initialisation comporte la détermination des valeurs initiales des variables, la construction d'une séquence initiale et le calcul d'une borne supérieure.

La séparation est une procédure itérative. Elle développe un arbre de recherche, dans lequel un nœud représente une séquence partielle des tâches ordonnancées. Un descendant d'un nœud est alors une séquence partielle obtenue en ordonnançant une tâche non ordonnancée immédiatement derrière la séquence partielle correspondant à ce nœud.

A chaque itération, la procédure
(1) identifie le nœud courant (celui avec la plus petite valeur de minorant ([9]), en cas d'égalité en choisissant celui ayant le maximum de tâches ordonnancées dans la séquence partielle),
(2) génère les descendants du nœud courant,
(3) élimine les nœuds dominés,
(4) calcule les bornes inférieures pour les nœuds non éliminés et met à jour la liste des nœuds actifs. (Un nœud qui n'est pas courant est dominé si sa borne inférieure n'est pas strictement inférieure à la somme des retards de la meilleure solution réalisable trouvée.)

La phase de terminaison consiste à identifier la séquence optimale et à calculer la valeur du critère.

Nous conservons en permanence la meilleure solution réalisable trouvée S, dont la somme des retards est désignée par UB, et une liste des nœuds non explorés $\mathcal{L}$, qui est triée dans l'ordre des bornes inférieures non décroissantes et en cas d'égalité, dans l'ordre des nombres décroissants de tâches ordonnancées. Chaque nœud correspond à un ordonnancement partiel. Les tâches non ordonnancées doivent être ordonnancées derrière cet ordonnancement partiel.

Un nœud o est représenté par les six paramètres suivants:
B(o): ordonnancement partiel des tâches ordonnancées;
A(o): ensemble des tâches non ordonnancées;
D(o): liste des délais des tâches non ordonnancées dans l'ordre non décroissant;
t(o): date d'achèvement de la dernière tâche ordonnancée (t(o)=E(B(o));
T(o): somme des retards des tâches ordonnancées;
LB(o): borne inférieure.

Connaissant B(o), les autres paramètres peuvent être parfaitement déterminés. Mais si nous ne conservions pas ces paramètres il serait nécessaire de restituer ces informations chaque fois que l'on en a besoin.

---

[9] Cela signifie que nous avons programmé une P.S.E.P. selon la terminologie de Roy (1970) ou une méthode du type "branch and bound with jumptracking" selon la terminologie anglo-saxonne.

Le processus de cet algorithme est décrit ci-dessous.

Initialisation

Initialement, $\mathcal{L}$ contient un seul nœud ω où $B(\omega)$ est initialisé à Ø. $A(\omega)$ contient l'ensemble des indices de toutes les tâches. $D(\omega)$ est la liste des délais de toutes les tâches dans l'ordre des valeurs non décroissantes. $t(\omega)$, $T(\omega)$ et $BI(\omega)$ sont initialisés à 0. S est initialisé à la meilleure solution fournie par IPRTT et SDPR (voir section 3.4.2.3.), UB est initialisé à la somme des retards correspondant à S.

Séparation

L'algorithme cherche toujours à développer un nœud h (la tête de la liste $\mathcal{L}$). Pour chaque tâche de A(h), ou bien des tests nous amènent à rejeter la séquence partielle B(h)|i (où S|i est la séquence partielle obtenue en ordonnançant immédiatement la tâche i derrière la séquence donnée S), ou bien on la construit et on la met dans la liste $\mathcal{L}$, ou encore on construit un ordonnancement complet (le meilleur des ordonnancements B(h)|i|j|k et B(h)|i|k|j) et on utilise ce nouvel ordonnancement complet pour tronquer la liste $\mathcal{L}$. Ces trois possibilités sont contenues dans le schéma suivant:

Eventuelle création d'un nœud B(h)|i:

Etape 1: ***Test 1*** ≡ *"élimination de B(h)|i lorsqu'il ne peut pas conduire à un ordonnancement actif"*
S'il existe une autre tâche $j \in A(h)$ telle que $\Phi(j,t(h)) \leq R(i,t(h))$ **alors aller à l'étape 8, finsi.**

Etape 2: ***Test 2*** ≡ *"élimination de B(h)|i quand il ne peut pas conduire à un ordonnancement T-actif"*
Si $card(B(h)) \geq 1$ et $R(x,\phi_x) \geq R(i,\phi_x)$ et $PRTT(x,\phi_x) > PRTT(i,\phi_x)$ (où x est la dernière tâche ordonnancée dans B(h)) **alors aller à l'étape 8, finsi.**

Etape 3: ***Test 3*** ≡ *"élimination de B(h)|i lorsqu'il est dominé par un autre séquence partielle B(h)|j (théorème 3.4)"*
S'il existe une autre tâche $j \in A(h)$ telle que $p_j \geq p_i$, $\Phi(j,t(h)) \leq \Phi(i,t(h))$ et $\max(\Phi(j,t(h)),d_j) < \max(\Phi(i,t(h)),d_i)$ **alors aller à l'étape 8, finsi.**

Etape 4 *"Création d'un nouveau nœud o ≡ B(h) | i"*

B(o):=B(h) | i;
t(o):=Φ(i,t(h));
A(o):=A(h)-{i};
D(o):=D(h)~{d_i} [10];
T(o):=T(h)+max(t(o)-d_i,0);

Etape 5 *"Obtention d'un nouvel ordonnancement réalisable complet et conséquences"*

Si card(A(o))≤2 **alors**
Terminer l'ordonnancement partiel par la séquence σ des tâches de A(o) placées à partir de l'instant t(o) en utilisant l'algorithme IPRTT qui sera présenté à la section suivante. Cette séquence partielle est optimale d'après le théorème 3.2.

Si T(o)+T(σ)<UB **alors** *"nous avons obtenu un ordonnancement strictement meilleur que l'ordonnancement précédent S"*, T(o)+T(σ) est alors affecté à UB, S devient B(o) suivi de σ et on élimine de la liste $\mathcal{L}$ tout nœud o' tel que LB(o')≥UB, **finsi,**

aller à l'étape 8, finsi.

Etape 6 *"Calcul d'une borne inférieure pour o (≡B(h) | i) et conséquences"*

LB(o):=T(o)+MINOR(t(o),A(o),D(o)), où MINOR(t(o),A(o),D(o)) est une fonction décrite ci-dessous qui calcule une borne inférieure pour des paramètres t(o), A(o) et D(o).

Test 4 ≡ *"élimination de o≡B(h) | i lorsqu'il ne peut pas conduire à un ordonnancement dont la somme des retards sera strictement inférieure que UB"*

si LB(o)≥UB **alors aller à l'étape 8, finsi.**

Test 5 ≡ *"élimination de o≡B(h) | i lorsqu'il est dominé par un autre nœud"*

S'il existe un nœud o' dans la liste $\mathcal{L}$ tel que: LB(o')≤LB(o) et A(o')=A(o) et t(o')≤t(o), **alors aller à l'étape 8, finsi.**

---

[10] E~{x} signifie "éliminer l'élément {x} de la suite E".

Etape 7 *"Insertion du noeud o≡B(h)|i à son rang dans la liste $\mathcal{L}$ et conséquences"*
Insertion du noeud o dans la liste triée $\mathcal{L}$.
Elimination de tout nœud o' dans $\mathcal{L}$ tel que: LB(o')≥LB(o), A(o')=A(o) et t(o)≥t(o').

Etape 8 *"Fin de traitement de création de B(h)|i"*

Fonction MINOR(t,A,D)
"t est un instant, A est un ensemble de tâches et D est la liste triée dans l'ordre non décroissant des valeurs des délais des tâches de A"

Initialisation:
MINOR:=0; t:=max(t,$\min_{i\in A}(r_i)$);
j:=1; $\forall i\in A$, $\pi_i:=p_i$;
Tantque j≤card(A) **faire**
J:={i / i∈A and $r_i\leq t$ and $\pi_i>0$ };
Choisir une tâche k∈J telle que $\pi_k=\min_{i\in J}(\pi_i)$;
Calculer l'instant suivant t' (t'>t) où de nouvelles tâches arrivent.
Si $t'\geq t+\pi_k$ **alors** *"la tâche k est terminée"*
$t:=t+\pi_k$; $\pi_k:=0$;
MINOR:=MINOR+max(t-D_j,0); j:=j+1;
sinon *"seulement une partie de la tâche k est effectuée"*
$\pi_k:=\pi_k-(t'-t)$; t:=t';
finsi
fintantque.

(La fonction MINOR calcule le minorant présenté par le théorème 3.5 pour la fin de la séquence. Il est évident que le calcul de cet minorant est polynomial).

Terminaison

La PSE se termine lorsque la liste $\mathcal{L}$ devient vide. Une solution optimale est fournie par S et la somme minimale des retards est alors UB.

3.4.2.3. Nouveaux algorithmes approchés

Baker (1974) a proposé des générateurs d'ordonnancements actifs et sans délai. Nous avons déjà présenté le schéma d'algorithme pour construire des

ordonnancements sans délai à partir d'une hiérarchie de règles et nous avons déjà défini la notion d'ordonnancement actif. Nous rappelons ci-dessous le fonctionnement du générateur d'ordonnancements actifs tel que l'a proposé Baker.

Générateur d'ordonnancements actifs (noté ACT)

1° Initialisation de l'ensemble W à l'ensemble des indices de toutes les tâches et du temps t à 0;
2° **Tantque** $W \neq \emptyset$ **faire**

2.1° Calculer la plus petite date de fin possible $G1:=\min_{i \in W}(\Phi(i,t))$;
2.2° Sélectionner une tâche $j \in W$ telle que: $R(j,t)<G1$, en cas d'ex æquo j est choisi selon une hiérarchie de règles.
2.3° Ordonnancer la tâche j: $t_j:=R(j,t)$;
2.4° Eliminer la tâche ordonnancée: $W:=W-\{j\}$;
2.5° Faire progresser le temps: $t:=\Phi(j,t)$;

fintantque

Combinaison des règles classiques avec les générateurs d'ordonnancements ou les schémas d'algorithmes

Dans tous les cas, il s'agit de choisir une ou deux hiérarchies de règles.

On construit un ensemble d'algorithmes en prenant comme première règle des hiérarchies la nouvelle règle PRTT. Pour la comparer avec d'autres règles de priorité classiques, on construit un autre ensemble d'algorithmes en prenant comme première règle des hiérarchies les règles SPT et SLK qui nous paraissent les plus intéressantes.

Pour les ordonnancements sans délai, on utilise SPT comme règle secondaire si la première règle de la hiérarchie est PRTT ou SLK, et EDD si la première règle utilisée est SPT. On résout tous les cas d'ex æquo en terminant par la tâche de plus petit indice si deux tâches ont exactement les mêmes caractéristiques.

Pour les ordonnancements actifs, on distingue deux types d'ordonnancements:
– ACT1: La deuxième règle utilisée est R(.) (date de commencement au plus tôt de la tâche), la troisième règle est SPT.

– ACT2: La deuxième règle utilisée est $\Phi(.)$ (date d'achèvement au plus tôt), la troisième règle est LPT.

Pour simplifier, on code le nom des heuristiques en concaténant deux sigles: le premier correspond au type de l'ordonnancement (SD pour Sans Délai ou ACT pour ACTif) le deuxième correspond à la première règle utilisée. SDPR, par exemple, désigne l'algorithme construit en utilisant le schéma d'algorithme sans délai et en prenant comme première règle de la hiérarchie PRTT (les autres règles étant imposées par les choix faits antérieurement). Il faut noter qu'il n'est pas possible de construire un algorithme ACT2SPT.

Autres schémas d'algorithmes pour PRTT

Ils sont construits à partir des schémas d'algorithmes par insertion ou par choix alternatifs (voir section 3.2.).

Algorithme IPRTT

C'est un algorithme par insertion.

La première hiérarchie de règles est composée de la règle PRTT, suivie de $\Phi(.)$ (date de commencement au plus tôt de la tâche), suivie de la règle R(.).

La deuxième hiérarchie de règles est composée de la règle R(.), suivie de la règle PRTT, suivie de la règle SPT.

Algorithme ACPRTT1

C'est un algorithme de type choix alternatif.

La première hiérarchie de règles est composée de la règle PRTT suivie de la règle R(.) (date de commencement au plus tôt de la tâche), suivie de SPT.

La seconde hiérarchie de règles peut donner un ensemble vide de tâches. Elle est composée d'une première règle de type sélectif: la tâche β doit pouvoir se terminer complètement avant α, suivie de la règle $\Phi(.)$ (date d'achèvement au plus tôt), suivie de la règle SPT.

La condition à vérifier est ici confondue avec la première règle sélective de la deuxième hiérarchie: s'il existe une tâche β sélectionnée, on ordonnance β, sinon on ordonnance α.

Algorithme ACPRTT2

C'est un algorithme de type choix alternatif.

La première hiérarchie de règles est composée de la règle PRTT suivie de la règle R(.) (date de commencement au plus tôt de la tâche), suivie de SPT.

La seconde hiérarchie de règles peut donner un ensemble vide de tâches. Elle est composée d'une première règle de type sélectif: la tâche β doit pouvoir se terminer complètement avant α, suivie de la règle R(.) (date d'achèvement au plus tôt), suivie de la règle SPT.

La condition à vérifier est ici confondue avec la première règle sélective de la deuxième hiérarchie: s'il existe une tâche β sélectionnée, on ordonnance β, sinon on ordonnance α.

Algorithme ACPRTT3

C'est un algorithme de type choix alternatif.

La première hiérarchie de règles est composée de la règle PRTT suivie de la règle R(.) (date de commencement au plus tôt de la tâche), suivie de SPT.

La seconde hiérarchie de règles peut donner un ensemble vide de tâches. Elle est composée d'une première règle de type sélectif: la tâche β doit pouvoir se terminer complètement avant α, suivie de la règle Φ(.) (date d'achèvement au plus tôt), suivie de la règle LPT.

La condition à vérifier est ici confondue avec la première règle sélective de la deuxième hiérarchie: s'il existe une tâche β sélectionnée, on ordonnance β, sinon on ordonnance α.

D'après le déroulement des algorithmes, on a les propriétés suivantes:

1° Les quatre derniers algorithmes construisent un ordonnancement T-actif.

2° Les quatre derniers algorithmes sont polynomiaux et leur complexité est en $O(n^2)$. En effet, les algorithmes décrits sont des algorithmes gloutons: à chaque itération, ils placent définitivement au moins une tâche. Le nombre d'itérations nécessaires pour aboutir à un ordonnancement complet est au plus

aussi grand que le nombre n de tâches, et à chaque itération il explore au plus n fois les tâches non encore placées. La complexité des algorithmes est de l'ordre du nombre de tâches au carré. Ce sont donc des algorithmes polynomiaux en $O(n^2)$.

En ce qui concerne l'algorithme par insertion, on peut démontrer une propriété intéressante.

Propriété 3.1

Aucune des tâches insérées par la phase d'insertion de IPRTT n'est en retard.

Démonstration

La démonstration est triviale s'il n'y a aucune tâche à insérer au cours de la phase d'insertion. Nous supposons donc qu'il en existe au moins une. La phase d'insertion de l'algorithme génère alors un sous-ordonnancement Ω comme celui illustré dans la figure 3.4.

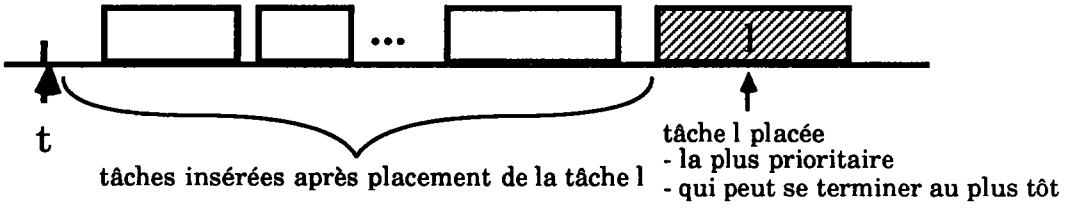


Fig. 3.4. Sous-ordonnancement Ω

Pour toute tâche i de Ω ($i \neq l$), i a une "priorité face aux délais" plus faible que l:

$$\max(t,r_i)+\max[\max(t,r_i)+p_i,d_i] > \max(t,r_l)+\max[\max(t,r_l)+p_l,d_l] \quad (3.11)$$

Par ailleurs $t_i+p_i \leq t_l$ et $\max(t,r_l)=t_l$ [11] entraînent:

$$\max(t,r_i) \leq t_i < t_l = \max(t,r_l)$$

et en combinant avec (3.11) on obtient:

$$d_i > \max[\max(t,r_l)+p_l,d_l] > t_i+p_i=C_i.$$

la tâche i n'est donc pas en retard.

Q.E.D.

[11]: En effet, si Ω est non vide, l doit arriver strictement après t et toute tâche i de Ω est insérable devant la tâche l.

Pour évaluer l'erreur commise dans le pire des cas par IPRTT (dans le cas des durées égales), nous avons le théorème suivant dont la démonstration figure dans l'annexe B.

Théorème 3.7

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D [12], si O est un ordonnancement issu de l'algorithme IPRTT et dont la somme des retards par rapport aux délais est égale à T, et si T^* désigne la somme des retards d'une solution minimale, on a:
$\Delta T=T-T^* \leq \frac{(2n+1)^2}{16} D$

Nous n'avons pas démontré que c'est une borne supérieure, mais seulement un majorant. En effet, nous n'avons pas trouvé d'exemple pour lequel ce majorant est atteint. Le meilleur exemple que l'on a trouvé permet d'atteindre asymptotiquement $\left[\frac{(2n-1)^2}{24}\right] D$, où [a] désigne la partie entière d'un nombre réel a. Nous présentons cet exemple ci-dessous.

Enoncé:

$n=3k$;
$\forall i \leq k$, $r_i=i(2D-\varepsilon)-D$, $d_i=2kD$
$\forall i$, $k<i\leq n$, $r_i=0$, $d_i=(2k+1)D$

Ordonnancement donné par IPRTT

1, 2, ..., k, k+1, k+2, ..., n
$\forall i \leq k$, $C_i=i(2D-\varepsilon)$, $T_i=0$;
$C_{k+1}=(2k+1)D-k\varepsilon$, $T_{k+1}=0$;
$\forall i$, $k+1<i\leq n$; $C_i=(k+i)D-k\varepsilon$, $T_i=(i-k-1)D-k\varepsilon$
$T= \sum_{i=k+2}^{n}[(i-k-1)D-k\varepsilon]=\frac{[n-(k+1)](n-k)}{2} D-[n-(k+1)]k\varepsilon$

Un ordonnancement optimal

k+1, 1, k+2, 2, ...,
$\forall i \leq k$; $C^*_i=2iD$, $T^*_i=0$;
$\forall i$, $k<i\leq 2k$, $C^*_i=[2(i-k)-1]D$, $T^*_i=0$;

12: On suppose que D ne divise pas le PGCD (Plus Grand Commun Diviseur) des dates d'arrivée r_i, car dans ce cas le problème est facilement résolu par un algorithme polynomial.

$\forall i>2k;\ C^*_i=iD,\ T^*_i=[i-(2k+1)]D$

$$T^*=\sum_{i=2k+1}^{n}[i-2k-1]D=\frac{(n-2k)(n-2k-1)}{2}D$$

$$T-T^*=\frac{2kn-3k^2-k}{2}D-[n-(k+1)]k\varepsilon=\frac{n^2-n}{6}D-[n-(k+1)]k\varepsilon=\left[\frac{(2n-1)^2}{24}\right]D-[n-(k+1)]k\varepsilon$$

$$\rightarrow\left[\frac{(2n-1)^2}{24}\right]D$$

3.4.3. RESULTATS EXPERIMENTAUX

3.4.3.1. Résultats des comparaisons des heuristiques

Génération des énoncés

Nous avons réalisé deux séries d'expériences. Dans la première série, nous avons pris n=20 afin de pouvoir toujours obtenir un ordonnancement optimal. Dans la deuxième série, nous avons pris n=200. Dans ce cas il est impossible d'obtenir un ordonnancement optimal.

Considérant que les performances relatives des algorithmes approchés peuvent être liées à la nature des jeux de données (grande ou petite dispersion des dates d'arrivée ou des marges), chaque série comprend 12 échantillons de 20 exemples. Un échantillon est défini par l'intervalle de dates d'arrivée BR et par l'intervalle des marges BSLK. Les dates d'arrivée sont uniformément générées entre 0 et BR et les marges absolues (pour chaque tache i, la marge absolue est $d_i-p_i-r_i$) sont uniformément générées entre 0 et BSLK.

Pour la première série d'expérience (n=20), nous avons généré les durées opératoires des tâches uniformément entre 1 et 100. Et pour la deuxième série, nous les avons générées entre 1 et 10. De cette manière, les sommes moyennes des durées opératoires sont comparables pour ces deux séries. On peut donc espérer des résultats qui se ressemblent entre les deux séries pour les mêmes paramètres BR et BSLK. Toutefois dans la première série, les durées opératoires des tâches sont plus dispersées que celles de la deuxième série.

Résultats

Les résultats sont fournis par des tables.

Table 3.1: Codage des algorithmes

	PRTT	SPT	SLK
ND	NDPR	NDSPT	NDSLK
ACT1	ACT1PR	ACTSPT	ACT1SLK
ACT2	ACT2PR	ACTSPT	ACT1SLK
I	IPRTT		
AC	ACPRTT1 ACPRTT2 ACPRTT3		

La table 3.1 résume les notations utilisées pour les différents algorithmes.

Table 3.2 (n=200)

BR / BSLK	0	500	1000	1500
50	100%	100%	100%	100%
250	100%	100%	95%	100%
500	100%	100%	100%	100%

Table 3.2 (n=20)

BR / BSLK	0	500	1000	1500
50	80%	65%	65%	75%
250	65%	25%	80%	85%
500	40%	40%	75%	85%

La table 3.2 indique les pourcentages de fois où un algorithme utilisant PRTT fournit le meilleur résultat pour chaque échantillon (ce meilleur résultat est la meilleure valeur du critère fournie par les heuristiques pour la série avec n=200 et est la valeur optimale du critère pour la série avec n=20).

On peut observer que dans le cas où n=200, il est raisonnable d'utiliser les heuristiques de la famille PRTT et que dans le cas n=20 pour BR=500 et BSLK=250 les algorithmes utilisant PRTT ne fournissent l'optimum qu'une fois sur quatre seulement.

Table 3.3 (n=200)

BR / BSLK	0	500	1000	1500
50	tout PRTT	NDPR + IPRTT	NDPR + ACT1PR	NDPR
250	tout PRTT	NDPR	NDPR	IPRTT
500	tout PRTT	NDPR	IPRTT	IPRTT

Table 3.3 (n=20)

BR / BSLK	0	500	1000	1500
50	PRTT + SPT	NDPR + NDSPT	NDPR + NDSPT	tout PRTT sauf NDPR
250	tout PRTT	NDPR	NDPR	IPRTT
500	tout PRTT	NDPR	NDPR	ACTPR

La table 3.3 indique les codes des meilleurs algorithmes pour chaque échantillon. Nous constatons que pour le cas BR=0, tous les algorithmes utilisant PRTT sont meilleurs, et que IPRTT est meilleur quand BR devient grand. Dans presque tous les cas, il existe au moins un algorithme utilisant PRTT qui est un des meilleurs algorithmes.

Table 3.4 (n=200)

	$\Sigma T_i/n$	"$\Sigma T_i/n$"	per1	per2
PRTT	99.20	0.05	56.4%	79.2%
SPT+SLK	111.10	7.81	43.0%	60.4%
RND	168.45	5.41	0%	0%
"OPT"	98.13	0	71.25%	100%

Table 3.4 (n=20)

	$\Sigma T_i/n$	"$\Sigma T_i/n$"	per1	per2
PRTT	90.47	0.74	63.21%	76.6%
SPT+SLK	108.02	12.00	32.33%	39.2%
RND	166.67	20.25	1.87%	2.3%
OPT	87.52	0	82.5%	100%

La table 3.4 montre les performances des valeurs moyennes du critère fournies pour les différentes familles d'algorithmes par rapport à la valeur moyenne des solutions optimales (dans le cas où n=200, la solution dite "optimale" est la meilleure solution fournie par tous les algorithmes).

Dans la première colonne figurent les valeurs moyennes obtenues par l'ensemble des algorithmes de chaque famille pour le critère somme des retards pour l'ensemble des 240 exemples.

En voulant obtenir des informations sur les performances relatives des algorithmes par rapport à l'optimum (ou à défaut par rapport à la meilleure solution obtenue) pour chaque exemple de l'échantillon, nous avons rencontré une difficulté: comme calculer une performance relative lorsque la solution optimale pour la somme des retards est nulle? Ceci nous a conduit à séparer les échantillons en deux sous-échantillons: les exemples pour lesquels on n'a pas obtenu une valeur nulle pour la somme des retards dans le meilleur des cas et ceux pour lesquels au moins un des algorithmes a conduit à une solution

-(optimale) où la somme des retards est nulle. Pour n=200, 69 exemples sur 240 conduisent à une somme des retards nulles, 42 exemples sur 240 lorsque n=20.

La seconde colonne de la table 3.4 contient la moyenne de la somme des retards obtenue pour chaque famille d'algorithmes pour les seuls exemples où on a trouvé une solution où la somme des retards est nulle.

Les troisième et quatrième colonnes de la table 3.4 présentent les pourcentages de fois où la valeur moyenne du critère fournie par une famille d'algorithmes est inférieure à 110% de la meilleure valeur du critère (erreur relative d'au plus 10%). Les pourcentages ne sont calculés que pour les exemples où la somme des retards n'est pas nulle, 100% correspondant aux 240 exemples, alors que la dernière colonne fournit les pourcentages corrigés en ne considérant que les exemples où la somme des retards n'est pas nulle, 100% correspond à 171 exemples pour n=200 et à 198 pour n=20.

En examinant cette table, nous pouvons constater que les algorithmes proposés améliorent globalement d'environ 10% les solutions trouvées par d'autres familles d'algorithmes et que ces algorithmes donne une valeur du critère très proche de l'optimum. La supériorité des algorithmes utilisant PRTT est encore plus nette en valeur relative lorsque la somme des retards peut être nulle (pour n=200, nous constatons que la moyenne des algorithmes utilisant le hasard est meilleure que la moyenne des algorithmes utilisant SLACK et SPT. Ceci est dû aux ordonnancements actifs qui sont particulièrement mauvais sur ces exemples).

Table 3.5 (n=200)

	$\Sigma T_i/n$	"$\Sigma T_i/n$"	per1	per2
PRTT	98.1	0	71.25%	100%
SPT+SLK	98.5	0.0015	67.5%	94.7%
RND	164.7	4.65	0%	0%
"OPT"	98.1	0	71.25%	100%

Table 3.5 (n=20)

	$\Sigma T_i/n$	"$\Sigma T_i/n$"	per1	per2
PRTT	88.05	0.124	81.25%	98.5%
SPT+SLK	91.03	0.324	54.58%	66.2%
RND	153.87	9.75	3.75%	4.5%
OPT	87.52	0	82.5%	100%

La table 3.5 montre les performances moyennes en retenant, pour chaque famille d'algorithmes, la meilleure solution trouvée dans la famille par rapport à la meilleure solution générale trouvée par tous les algorithmes optimaux ou approchés.

Dans la première colonne figurent les valeurs moyennes du critère.

La seconde colonne de la table 3.5 contient les valeurs moyennes pour les cas où la meilleure valeur obtenue est nulle (et donc optimale).

Les troisième et quatrième colonnes de la table 3.5 montrent les pourcentages du nombre de fois où la valeur du critère fournie par la meilleure solution d'une famille d'algorithmes est inférieure de 110% à la meilleure valeur obtenue par tous les algorithmes.

Comme pour le tableau précédent, la troisième colonne élimine les exemples où la meilleure somme des retards est nulle tout en donnant un pourcentage calculé sur l'ensemble de l'échantillon alors que la quatrième colonne fournit un pourcentage corrigé en ne considérant que les exemples où la somme des retards n'est pas nulle.

En examinant cette table, nous observons que la différence entre la meilleure solution trouvée par les algorithmes utilisant PRTT et la meilleure de celle trouvée par les autres algorithmes considérés est positive, mais relativement peu importante. Par contre, la meilleure solution trouvée par les algorithmes proposés utilisant PRTT est très proche de l'optimum.

Table 3.6

	PRTT	SLK+SPT	RND	"OPT"
tt(n=20)	289.41	158.04	221.62	49609.4
tm(n=20)	41.35	31.61	110.81	49609.4
tt(n=200)	703.31	363.52	168.30	"1241.2"
tm(n=200)	100.62	72.70	84.15	"1241.2"

La table 3.6 montre les temps de calcul pour les différents ensembles d'algorithmes où tt représente la moyenne des temps totaux de calcul d'une famille d'algorithmes pour un jeu de données et où tm représente la moyenne des temps moyens de calcul d'une famille d'algorithmes pour un jeu de données. Nous observons que les algorithmes proposés consomment légèrement plus de temps que les algorithmes utilisant des règles de priorité classiques. Cette augmentation peut être considérée comme négligeable par rapport à l'amélioration de la valeur du critère. Un algorithme optimal consomme énormément plus de temps, il ne peut être utilisé que pour des petites valeurs de n et dans ce cas, le gain de la solution optimale par rapport à la meilleure solution approchée est relativement peu important.

3.4.3.2. Expérimentations complémentaires de la Procédure par Séparation et Evaluation

Cet algorithme a été programmé sur SUN 3/110 en utilisant le langage C. Le test comporte 120 exemples de 20 tâchesb chacun. Chaque exemple a été généré selon trois distributions uniformes pour les paramètres r_i, p_i et d_i-(r_i+p_i). La valeur de p_i varie entre 1 à 10, celle de r_i entre 0 et $\alpha \cdot \Sigma p_i$ et celle de d_i-(r_i+p_i) entre 0 et $\beta \cdot \Sigma p_i$. Quatre valeurs ont été choisies pour α et trois valeurs pour β ce qui donne 12 échantillons de 10 exemples numériques.

Les tables 3.7, 3.8, 3.9 et 3.10 fournissent respectivement des statistiques sur le nombre de séparations faites, le nombre de sous-ensembles générés, le nombre maximal de sous-ensembles stockés dans la liste $\mathcal{L}$ en mémoire, et la durée d'exécution en centièmes de secondes. Dans chaque table, pour tout échantillon, figurent trois valeurs. La première colonne contient la valeur minimale, la seconde la valeur maximale et la troisième la valeur moyenne de l'information présentée dans la table.

On peut constater sur ces expériences que la difficulté du problème croît puis décroît avec α (c'est-à-dire avec la dispersion des dates d'arrivée des tâches). Quand α est petit, la dispersion des dates d'arrivée n'est pas grande, le problème devient équivalent à un problème avec des dates d'arrivée identiques après l'ordonnancement de quelques tâches. Quand α est grand, la dispersion des dates d'arrivée est grande, le problème est relativement facile parceque l'ensemble de solutions est petite. Cependant, il est difficile de tirer des conclusions en ce qui concerne la difficulté du problème en fonction de β (dispersion des marges). Le cas le plus difficile en ce qui concerne toutes les mesures correspond à α=0.5 et β=0.50.

Table 3.7. Statistiques sur le nombre de séparations faites

	β								
α	0.05			0.25			0.50		
0.0	1	31	18.1	35	134	66.5	39	229	107
0.5	8	162	47.9	24	1242	478	79	4060	1338
1.0	1	56	28.8	12	261	79.4	0	242	72.7
1.5	11	49	21.1	0	54	23.1	0	38	6.8

Table 3.8. Statistiques sur le nombre de sous-ensembles générés

	β								
α	0.05			0.25			0.50		
0.0	1	31	18.1	35	134	70.2	39	237	111
0.5	8	167	53.6	24	2198	761	97	4104	1408
1.0	1	56	28.8	12	291	87.5	1	242	76.4
1.5	11	49	21.5	1	54	24.7	1	40	7.8

Table 3.9. Statistiques sur le nombre maximal de sous-ensembles stockés

	β								
α	0.05			0.25			0.50		
0.0	1	13	5.6	11	36	22.0	9	81	30.0
0.5	3	39	17.1	5	1007	349.0	21	1447	398.0
1.0	1	14	7.4	3	52	21.9	1	19	6.5
1.5	2	13	4.9	1	9	5.2	1	15	3.5

Table 3.10. Statistiques sur la durée d'exécution (en centièmes de secondes)

	β								
α	0.05			0.25			0.50		
0.0	17	32	25	35	88	52	40	163	80
0.5	55	560	150	483	6981	2062	160	21142	4645
1.0	18	152	79	50	438	156	12	567	122
1.5	42	95	56	15	113	59	12	118	29

3.5. MINIMISATION DE LA SOMME DES TEMPS DE PRESENCE (FLOW-TIMES)

Ce problème est équivalent aux problèmes $n/1/r_i/\Sigma C_i$, $n/1/r_i/\Sigma L_i$ et au problème de minimisation des en-cours si les en-cours sont exprimés en nombre moyen de tâches dans l'atelier (Conway et al. 1967]). Pour les entreprises, il y a donc grand intérêt à minimiser ΣF_i.

Ce problème a été démontré NP-difficile ([Rinnooy Kan 1976]). Il faut donc utiliser des méthodes approchées pour des problèmes de grande taille.

Dans ce travail, nous démontrons une condition nécessaire et suffisante d'optimalité locale et, à partir de cette condition, nous définissons un ensemble dominant de solutions qui contient toutes les solutions optimales et proposons des algorithmes efficaces qui construisent un ordonnancement contenu dans cet ensemble.

Dessouky & Deogun (1981) ont démontré des propriétés de dominance. Elles sont obligatoirement vérifiées dans notre ensemble dominant. Par contre, elles n'impliquent pas la condition nécessaire et suffisante présentée ici.

3.5.1. CONDITION NECESSAIRE ET SUFFISANTE D'OPTIMALITE LOCALE

On peut obtenir une condition nécessaire et suffisante grâce à une nouvelle mesure de priorité de chaque tâche que nous appelons PRTF [13].

Définition 3.5

Nous définissons $PRTF(i,\Delta)$ comme une fonction de la tâche i à l'instant Δ par $PRTF(i,\Delta)=2R(i,\Delta)+p_i=R(i,\Delta)+\Phi(i,\Delta)$

Cette fonction est la somme de 2 fois la date d'exécution au plus tôt et de la durée ou encore la somme de la date d'exécution au plus tôt et de la date de fin au plus tôt. C'est une fonction continue linéaire par morceaux (voir la figure 3.5).

13: Sigle de l'expression anglaise "Priority Rule for Total Flow-time"

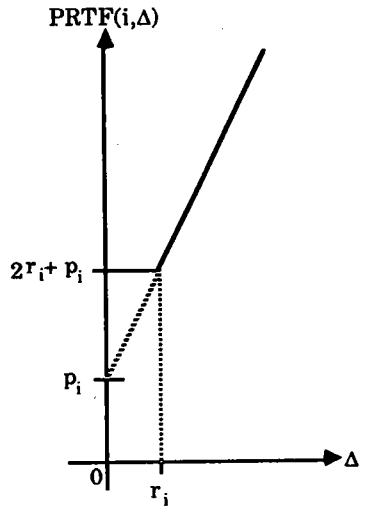


Fig. 3.5. Fonction PRTF

La condition nécessaire et suffisante est obtenue grâce à cette fonction.

Théorème 3.8

Nous considérons, pour le critère somme des temps de présence, le problème partiel d'ordonnancement de deux tâches i et j sur une machine disponible à partir de l'instant Δ. La condition nécessaire et suffisante d'optimalité locale pour ordonnancer i avant j est:

$PRTF(i,\Delta) \leq PRTF(j,\Delta)$

C'est-à-dire: $PRTF(i,\Delta) \leq PRTF(j,\Delta) \Leftrightarrow F_{ij} \leq F_{ji}$, où F_{ij} est la somme des temps de présence des tâches i et j.

Démonstration

D'après la définition et les contraintes du problème, on a:

$$\begin{aligned} F_{ij} &= R(i,\Delta)+p_i-r_i+\max[R(i,\Delta)+p_i,R(j,\Delta)]+p_j-r_j \\ &= R(i,\Delta)+\max[R(i,\Delta)+p_i,R(j,\Delta)]+p_i+p_j-(r_i+r_j) \\ &= R(i,\Delta)+\max[R(i,\Delta)+p_i,R(j,\Delta)]+P_{ij}-R_{ij} \\ &= \max[2R(i,\Delta)+p_i,R(i,\Delta)+R(j,\Delta)]+P_{ij}-R_{ij} \\ &= \max[PRTF(i,\Delta),Z_{ij}]+P_{ij}-R_{ij} \end{aligned}$$

où

$Z_{ij}(\Delta)=R(i,\Delta)+R(j,\Delta)$

$P_{ij}=p_i+p_j$

$R_{ij}=r_i+r_j$

De manière symétrique, on obtiendrait:
$F_{ji}=\max[PRTF(j,\Delta),Z_{ji}]+P_{ji}-R_{ji}=\max[PRTF(j,\Delta),Z_{ij}]+P_{ij}-R_{ij}$

La condition suffisante ($PRTF(i,\Delta)\leq PRTF(j,\Delta) \Rightarrow F_{ij}\leq F_{ji}$) est immédiate. Démontrons maintenant la condition nécessaire.

Nous distinguons trois cas.

Cas 1° $r_j\geq\Phi(i,\Delta)$ *(i peut être terminé avant l'arrivée de j)*

Dans ce cas, nous avons
$R(j,\Delta)\geq\Phi(i,\Delta)$
alors $2R(j,\Delta)+p_j\geq 2R(i,\Delta)+2p_i+p_j>2R(i,\Delta)+p_i$
$\Rightarrow PRTF(i,\Delta)\leq PRTF(j,\Delta)$

Cas 2° $r_j<\Phi(i,\Delta)$ et $r_i<\Phi(j,\Delta)$ *(i et j demandent la machine sur un intervalle commun)*

Dans ce cas, on a:
$F_{ij}=2\Phi(i,\Delta)+p_j-(r_i+r_j) = PRTF(i,\Delta)+P_{ij}-R_{ij}$
$F_{ji}=2\Phi(j,\Delta)+p_i-(r_i+r_j) = PRTF(j,\Delta)+P_{ij}-R_{ij}$

$F_{ij}\leq F_{ji} \Rightarrow PRTF(i,\Delta)\leq PRTF(j,\Delta)$

Cas 3° $r_i\geq\Phi(j,\Delta)$ *(j peut être terminé avant l'arrivée de i)*

Dans ce cas, nous avons
$F_{ij}=p_i+r_i+p_i+p_j-r_j$
$F_{ji}=2\Phi(j,\Delta)+p_i-(r_i+r_j)$

alors $F_{ij}>2\Phi(j,\Delta)+2p_i+p_j-(r_i+r_j)$
$>2\Phi(j,\Delta)+p_i-(r_i+r_j)$
$=F_{ji}$.

Ce cas est impossible lorsque l'on suppose $F_{ij}\leq F_{ji}$.

Q.E.D.

Cette condition nécessaire et suffisante nous permet de définir un ensemble dominant pour le critère ΣF_i.

Définition 3.6

Considérons un ordonnancement actif O pour un problème du type $n/1/r_i/\sum F_i$. Etant données i et j deux tâches consécutives (j suivant i), si pour tout couple (i,j) de tâches consécutives on a:

$R(i,\phi_i)<R(j,\phi_i)$

ou $PRTF(i,\phi_i)\leq PRTF(j,\phi_i)$

alors on dit que **l'ordonnancement O est F-actif.**

Le théorème 3.8 nous permet d'obtenir le corollaire suivant.

Corollaire 3.2

Dans le cas du problème à une machine pour minimiser la somme des temps de présence, tous les ordonnancements optimaux sont F-actifs.

Démonstration

Supposons qu'il existe un ordonnancement optimal O qui n'est pas F-actif, alors, il existe au moins une paire de tâches consécutives i et j (j suivant i) telle que:

$R(i,\phi_i)\geq R(j,\phi_i)$

et $PRTF(i,\phi_i)>PRTF(j,\phi_i)$

alors $F=\sum_{l=1}^{n} F_l = \sum_{l\in B_i} F_l + \sum_{l\in A_j} F_l + F_{ij}$

Construisons un autre ordonnancement O' en mettant les tâches de A_j et de B_i dans la même position que dans O, et en ordonnançant i et j dans l'ordre inverse. Cet ordonnancement est réalisable parce que $R(i,\phi_i)\geq R(j,\phi_i)$.

D'après le théorème 3.8, on a $F_{ji}<F_{ij}$.

Alors

$$F'=\sum_{l=1}^{n} F'_l = \sum_{l\in B'_j} F'_l + \sum_{l\in A'_i} F'_l + F_{ji} = \sum_{l\in B_i} F'_l + \sum_{l\in A_j} F'_l + F_{ji} < \sum_{l\in B_i} F_l + \sum_{l\in A_j} F_l + F_{ij} = F$$

Ceci contredit l'hypothèse que O est un ordonnancement optimal.

Q.E.D.

Remarques

1° La fonction PRTF(i,Δ) est en fait une combinaison des règles FIFO (First In First Out) et SPT.

2° Si toutes les tâches arrivent simultanément, c'est-à-dire $\forall i$, $r_i=0$, la condition nécessaire et suffisante devient:
$p_i \leq p_j$
C'est bien la règle SPT, qui donne une solution optimale dans ce cas.

3° Si $\forall i$, $p_i=D$, D étant une constante, PRTF(i,Δ) est équivalent à FIFO, ce qui fournit une solution optimale dans ce cas particulier.

4° Si $\forall i$, $d_i=-\infty$, alors $n/1/r_i/\Sigma T_i$ est équivalent à $n/1/r_i/\Sigma F_i$. PRTT(i,Δ) (voir la section 3.4) est équivalent à PRTF(i,Δ) (mais PRTT(i,Δ) n'est pas une condition strictement nécessaire dans le cas général).

Etude analytique de la fonction PRTF

Si l'on compare la priorité de deux tâches i et j, r_i étant supposé inférieur ou égal à r_j, on peut distinguer deux cas possibles. Les figures 3.6 et 3.7 correspondent respectivement aux cas où $p_i>p_j$ et $p_i \leq p_j$.

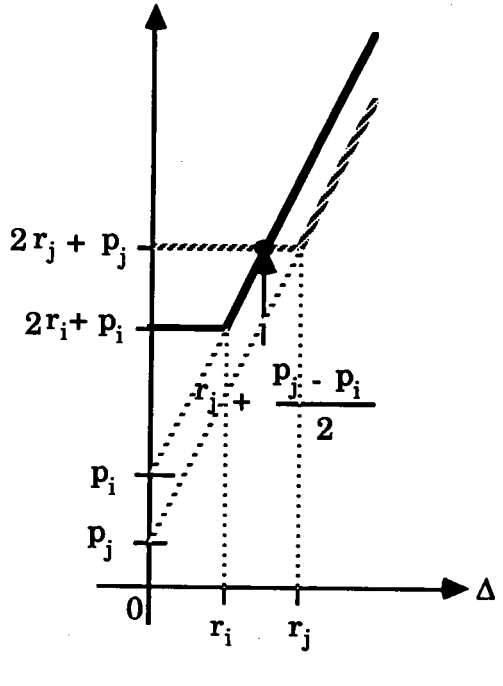


Fig. 3.6 Cas $p_i>p_j$

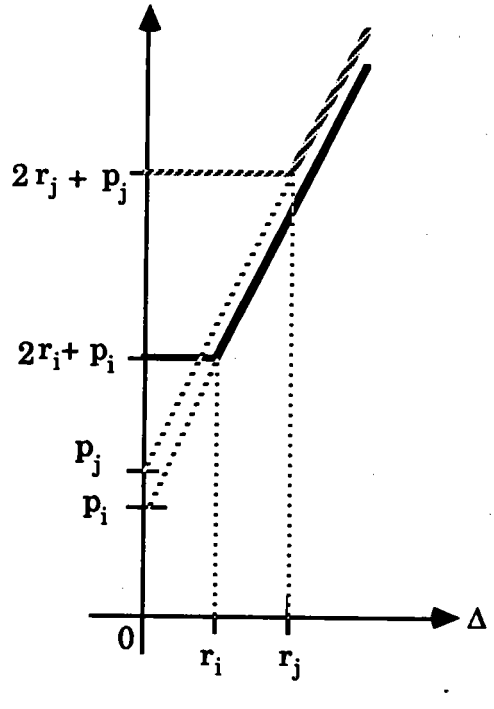


Fig. 3.7 Cas $p_i \leq p_j$

Dans le cas $p_i > p_j$ (Fig. 3.6), la tâche i est prioritaire jusqu'à $r_j + \frac{p_j - p_i}{2}$. Pendant cette période, PRTF est équivalent à la règle de priorité FIFO. A partir de cet instant, j demeure prioritaire, PRTF est donc équivalent à la règle de priorité SPT.

Dans le cas $p_i \leq p_j$ (Fig. 3.7), la tâche i reste toujours prioritaire. En tenant compte que FIFO est dans ce cas équivalent à SPT, on peut aussi considérer qu'il existe une période où PRTF est équivalent à la règle de priorité FIFO, et qu'il existe une autre période où PRTF est équivalent à SPT.

Dans les deux cas, on peut considérer que PRTF est une bascule automatique entre les règles de priorité FIFO et SPT.

3.5.2. ALGORITHMES APPROCHES

Selon le corollaire 3.2, nous pouvons ne prendre en compte que les ordonnancements F-actifs. Nous proposons deux nouvelles heuristiques polynomiales, APRTF et PRTF, qui construisent un ordonnancement appartenant à cet ensemble.

Dans la littérature, les chercheurs ont proposé des algorithmes approchés. Dessouky & Deogun (1981) ont testé ECT (Earliest Completion Time) et EST (Earliest Starting Time, identique à "SPT" considéré dans [Chandra 1979]). D'après Dessouky & Deogun, ECT fournit une bonne solution approchée au problème. Selon Chandra, EST est un bon algorithme approché pour le problème de minimisation de la somme des temps de présence. Nous présentons maintenant les algorithmes ECT et EST et nous proposons deux nouveaux algorithmes approchés appelés APRTF et PRTF.

Algorithme ECT (Earliest Completion Time)

C'est un algorithme du type "Règle Pure".

La hiérarchie de règles est la suivante:
Règle n° 1: la plus petite date de fin (la plus petite valeur de $\Phi(i,\Delta)$)
Règle n° 2: la plus petite date de début (la plus petite valeur de $R(i,\Delta)$)

Algorithme EST

C'est un algorithme du type "Sans Délai"

La hiérarchie de règles se réduit à:
Règle n° 1: la plus petite durée opératoire.

Algorithme APRTF

C'est un algorithme de type "Choix Alternatif"

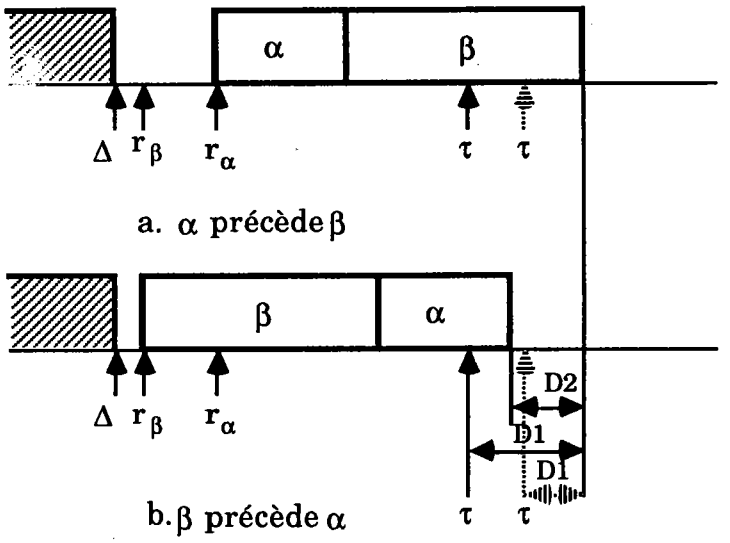
La première hiérarchie de règles est:
Règle n° 1: la plus petite valeur de PRTT;
Règle n° 2: la plus petite date d'exécution au plus tôt.

La deuxième hiérarchie de règles est:
Règle n° 1: la plus petite date d'exécution au plus tôt;
Règle n° 2: la plus petite durée opératoire.

La condition de choix entre les tâches α et β est la suivante:
En notant μ le nombre de tâches non ordonnancées autres que α et β, et τ la plus petite date d'arrivée de ces tâches, la condition pour ordonnancer α est donnée par l'équation: $F_{\beta\alpha}-F_{\alpha\beta}<\mu\cdot\min[R(\alpha,\Delta)-R(\beta,\Delta),\Phi(\beta,\Phi(\alpha,\Delta))-\tau]$.

Commentaire

La première hiérarchie de règles permet de choisir pour α la tâche la plus prioritaire. La deuxième hiérarchie de règles consiste à sélectionner pour β une tâche dont l'exécution ne laisse pas la machine inactive donc ne retarde pas l'exécution des autres tâches. La condition permet de comparer le gain et la perte en ordonnançant α, donc de prévoir une éventuelle conséquence sur les tâches qui vont suivre. Si on ordonnance une tâche plus prioritaire α, alors qu'il existe une tâche β qui pourrait être exécutée avant α, on laissera la machine inoccupée plus longtemps au lieu d'ordonnancer β en priorité. La durée de l'inactivité de la machine est de $R(\alpha,\Delta)-R(\beta,\Delta)$. Chacune des tâches restantes est retardée au plus de $\Delta t=\min[R(\alpha,\Delta)-R(\beta,\Delta),\Phi(\beta,\Phi(\alpha,\Delta))-\tau]$ (voir la figure 3.8). On perd au total au plus $\mu\cdot\Delta t$ en ordonnançant α avant β. Alors que l'on gagne $F_{\beta\alpha}-F_{\alpha\beta}$ si on ordonnance α avant β. Si le gain est plus grand que la perte maximale, on a intérêt à exécuter α avant β, sinon, on aurait intérêt à exécuter β avant α.



$D2=R(\alpha,\Delta)-R(\beta,\Delta)$
$D1=\Phi(\beta,\Phi(\alpha,\Delta))-\tau$

Fig. 3.8. Comparaison des perte et gain pour les ordres αβ et βα

Algorithme PRTF

C'est un algorithme du type "Règle Pure"

La hiérarchie de règles est:
Règle n° 1: la plus petite valeur de PRTF
Règle n° 2: la plus petite date d'exécution au plus tôt.

Ces algorithmes proposés sont des algorithmes polynomiaux et construisent un ordonnancement F-actif.

3.5.3. COMPARAISON DE THEOREMES ET DE MINORANTS

Dans cette section, nous comparons des théorèmes et des minorants. Ceci a pour objectif d'éliminer les théorèmes redondants et des minorants faibles dans la PSE que nous allons construire, pour en diminuer le temps de calcul. Dans la littérature, nous pouvons trouver beaucoup de théorèmes de dominance, conditions suffisantes d'optimalité, et de minorants. Nous allons montrer que certains théorèmes de dominance décrivent un cas particulier d'un autre théorème (ils sont donc redondants), que des conditions d'optimalité

sont contenues dans une propriété très connue des chercheurs opérationnels, et que des minorants sont moins efficaces qu'un autre. Par conséquent, nous pouvons n'utiliser que les théorèmes qui ne sont pas redondants, la condition suffisante d'optimalité la plus performante, ainsi que le minorant le plus efficace (sous réserve qu'il ne soit pas significativement plus coûteux).

3.5.3.1. Comparaison de théorèmes de dominance

Dans [Dessouky & Deogun 1981], les auteurs ont démontré le théorème suivant:

Théorème 3.9 ([Dessouky & Deogun 1981])
On considère un problème du type $n/1/r_i/\Sigma F_i$. N étant l'ensemble des tâches et S une séquence partielle de tâches de N, si une tâche $y \in N\text{-}S$ vérifie $p_y \leq p_i$ pour toute tâche $i \in N\text{-}S$, et si une autre tâche $x \in N\text{-}S$ vérifie $R(x,E(S)) \geq R(y,E(S))$, alors $\Sigma(S,y)$ domine $\Sigma(S,x)$.

La propriété suivante montre que tout ordonnancement F-actif vérifie "presque" ce théorème ([14]).

Propriété 3.2
Dans un ordonnancement F-actif, pour toute tâche x, il n'y a aucune tâche $y \in A_x$ telle que:
$p_y < p_i$, $\forall i \in A_x \cup \{x\} - \{y\}$ et $R(x,\phi_x) \geq R(y,\phi_x)$

Démonstration

Supposons qu'il existe un ordonnancement F-actif O dans lequel il y a au moins une tâche x telle qu'il existe une tâche $y \in A_x$ (l'ensemble de tâches suivant la tâche x) telle que:
$p_y < p_i$, $\forall i \in A_x \cup \{x\} - \{y\}$ et $R(x,\phi_x) \geq R(y,\phi_x)$

Considérons la tâche w qui précède immédiatement la tâche y. Comme O est F-actif, on a au moins une des relations suivantes:

$R(w,\phi_w) < R(y,\phi_w)$

ou $2R(w,\phi_w) + p_w \leq 2R(y,\phi_w) + p_y$

---

[14] La différence réside dans le fait que l'on a $p_y < p_i$ au lieu de $p_y \leq p_i$. Ceci permet donc à Dessouky et Deogun d'éliminer certains ordonnancements F-actifs parce que dominés par d'autres ordonnancements.

Nous considérons deux cas.

1° $R(w,\phi_w)<R(y,\phi_w)$

Dans ce cas, on a:
$r_y>R(w,\phi_w)$ (3.12)

On a deux sous-cas à envisager:

1.1° $w=x$

Dans ce cas $R(w,\phi_w)=R(x,\phi_x)$

1.2° $w\neq x$

Dans ce cas on a $w\in A_x$, ce qui implique
$R(w,\phi_w)\geq\Phi(x,\phi_x)=R(x,\phi_x)+p_x>R(x,\phi_x)$

Dans les deux sous-cas, on a obtenu: $R(w,\phi_w)\geq R(x,\phi_x)$.
Avec (3.12) on obtient alors $r_y>R(x,\phi_x)$
qui implique: $R(x,\phi_x)<R(y,\phi_x)$

Ceci contredit l'hypothèse
$R(x,\phi_x)\geq R(y,\phi_x)$

2° $R(x,\phi_w)\geq R(y,\phi_w)$ et $2R(x,\phi_w)+p_w\leq 2R(y,\phi_w)+p_y$

Dans ce cas, on a la relation suivante
$p_w\leq p_y$

Ceci contredit l'hypothèse:
$p_y<p_i$, $\forall i\in A\cup\{x\}-\{y\}$

Dans tous les cas, il n'y a aucune tâche x vérifiant les conditions annoncées dans la propriété 3.2.

Q.E.D.

Cette propriété montre qu'un ordonnancement F-actif vérifie le théorème 3.9 sauf dans le cas où il existe une tâche $z\in N-(S\cup\{y\})$ telle que $p_y=p_z$.

Dans [Dessouky & Deogun 1981], les auteurs ont également démontré le théorème suivant.

Théorème 3.10 ([Dessouky & Deogun 1981])
On considère un problème du type $n/1/r_i/\Sigma F_i$, N étant l'ensemble des tâches et S une séquence partielle de tâches de N. Soit y une tâche de N-S telle que $\Phi(y,E(S))\leq\Phi(i,E(S))$ pour toute tâche $i\in$N-S, alors un ordonnancement $\Sigma(S,x)$ est strictement dominé si $r_x\geq\Phi(y,E(S))$.

Par ailleurs, dans [Bianco & Ricciardelli 1982], les auteurs ont démontré le théorème suivant.

Théorème 3.11 ([Bianco & Ricciardelli 1982])
On considère un problème du type $n/1/r_i/\Sigma w_iF_i$, N étant l'ensemble des tâches et S une séquence partielle de tâches de N. Soient x et y deux tâches de N-S, si $r_x\geq\Phi(y,E(S))$, alors $\Sigma(S,y)$ domine $\Sigma(S,x)$.

Or Conway et al. (1967) et Baker (1974) ont déjà montré que l'ensemble des ordonnancements actifs est dominant pour les critères réguliers (voir aussi le paragraphe 1.2.5.). Comme "la minimisation de la somme (pondérée) des temps de présence" est un critère régulier, il suffit donc de considérer l'ensemble des ordonnancements actifs. Dans le cas d'une machine, ceci revient à considérer seulement les ordonnancements tels que:
$\forall$i, $\forall$j, tel que la tâche j suit la tâche i dans l'ordonnancement, on a $r_i<\Phi(j,\phi_i)$.

Nous avons immédiatement la propriété suivante.

Propriété 3.3
Les théorèmes 3.10 et 3.11 ne sont en fait qu'une façon particulière d'exprimer que l'ensemble des ordonnancements actifs est dominant pour un critère régulier.

Le théorème suivant a été démontré dans [Dessouky & Deogun 1981].

Théorème 3.12 ([Dessouky & Deogun 1981])
On considère un problème du type $n/1/r_i/\Sigma F_i$, N étant l'ensemble des tâches et S une séquence partielle de tâches de N. Soient i et j deux tâches de N-S, si $p_i\geq p_j$ et $\Phi(i,E(S))\leq\Phi(j,E(S))$, alors $\Sigma(S,i)$ domine $\Sigma(S,j)$.

Nous allons montrer que ce théorème est un cas particulier du théorème suivant obtenu à partir d'un théorème démontré pour le problème $n/1/r_i/\Sigma w_iF_i$ par Bianco et Ricciardelli ([1982]) en le simplifiant avec des poids w_i égaux à 1.

Théorème 3.13

On considère un problème du type $n/1/r_i/\Sigma F_i$, N étant l'ensemble des tâches et S une séquence partielle de tâches de N. Soient i et j deux tâches de N-S, si $\Phi(i,E(S))\leq\Phi(j,E(S))$ and $\Phi(i,E(S))-\Phi(j,E(S))\leq(p_i-p_j)[card(N-S)-1]$, alors $\Sigma(S,i)$ domine $\Sigma(S,j)$.

Propriété 3.4

Le théorème 3.13 est plus général que le théorème 3.12.

En effet, si $p_i\geq p_j$, $\Phi(i,E(S))\leq\Phi(j,E(S))$, alors $\Phi(i,E(S))-\Phi(j,E(S))\leq 0$ et $(p_i-p_j)[card(N-S)-1]\geq 0$, ce qui implique $\Phi(i,E(S))-\Phi(j,E(S))\leq(p_i-p_j)[card(N-S)-1]$. A partir du théorème 3.13, $\Sigma(S,i)$ domine $\Sigma(S,j)$.

En conclusion, parmi les théorèmes cités ci-dessus, seuls les théorèmes 3.8, 3.13 et la dominance de l'ensemble des ordonnancements F-actifs ne sont pas redondants. (3.9 n'est que partiellement redondant, mais nous ne l'utiliseront pas pour des raisons de coût). Les autres théorèmes peuvent ne pas être utilisés dans une PSE.

3.5.3.2. Comparaison de conditions suffisantes d'optimalité

Pour présenter les conditions suffisantes d'optimalité proposées par Dessouky & Deogun (1981) et par Deogun (1983), nous avons besoin de rappeler un autre algorithme approché nommé SPT dans [Deogun 1983] et destiné au problème à tâches non interruptibles. Les problèmes à tâches interruptibles peuvent être résolus optimalement par la règle de priorité SRPT présentée dans la section 3.4. L'optimalité de ce dernier algorithme a été démontrée par Schrage (1968) et par Smith (1978).

Algorithme SPT (Shortest Processing Time)

C'est un algorithme du type "Règle pure".

La hiérarchie de règles est la suivante:
Règle n° 1: la plus petite durée opératoire
Règle n° 2: la plus petite date de début (la plus petite valeur de $R(i,\Delta)$)

Tous les chercheurs concernés utilisent la propriété suivante:

Etant donné un problème d'ordonnancement P où l'on minimise un critère et où les tâches sont non interruptibles, on construit un problème relaxé P' à partir de P en autorisant l'interruption des tâches. Toute solution optimale du problème P' fournit alors un minorant pour le problème P. De plus s'il existe un ordonnancement optimal de P' où aucune interruption n'a lieu, alors cet ordonnancement est aussi optimal pour P.

Dans [Dessouky & Deogun 1981], les auteurs ont démontré le théorème suivant:

Si pour un problème P du type $n/1/r_i/\Sigma F_i$, on peut construire un ordonnancement vérifiant ECT dans lequel la première tâche de toute sous-séquence contiguë de tâches a une date d'arrivée au plus aussi grande que celle de toutes les tâches qui suivent, alors cette ordonnancement est optimal pour P.

En fait, le théorème de Dessouky & Deogun et la propriété citée ci-dessus sont équivalents.

La démonstration de cette équivalence repose sur le lemme suivant qui servira aussi dans d'autres démonstrations.

Lemme 3.3

Etant donné un problème P du type $n/1/r_i/\Sigma F_i$, on construit le problème relaxé P' où l'interruption des tâches est autorisée. S'il existe un ordonnancement O pour P vérifiant ECT dans lequel la première tâche de toute sous-séquence contiguë a une date d'arrivée au plus aussi grande que celle des tâches qui suivent, alors il existe un ordonnancement O' pour P' vérifiant SRPT et dans lequel aucune interruption n'a lieu, les tâches étant exécutées dans le même ordre dans O et dans O'.

Démonstration

Démontrons d'abord que la première tâche dans O est aussi la première tâche dans O', et que l'exécution de cette tâche dans O' n'est pas interrompue. Dans ce but, considérons cette tâche indicée par i. Comme O vérifie ECT on a $\Phi(i,\Delta)\leq\Phi(j,\Delta)$ pour toute tâche j suivant i dans O.
Comme dans O la première tâche de toute sous-séquence contiguë a une date d'arrivée au plus aussi grande que celle des tâches qui suivent on a $R(i,\Delta)\leq R(j,\Delta)$ pour toute tâche j suivant i dans O.

On a donc pour tout j: $\Phi(i,\Delta)\leq\Phi(j,\Delta)$ et $R(i,\Delta)\leq R(j,\Delta)$. Ceci implique que toute tâche k telle que $R(k,\Delta)=R(i,\Delta)$ a une durée supérieure à celle de i, aussi ne peut-elle pas interrompre i dans O'. De même toute tâche k telle que $R(k,\Delta)>R(i,\Delta)$ aura une durée supérieure à la durée restante de i quand k arrivera dans l'atelier comme illustre la figure 3.9.

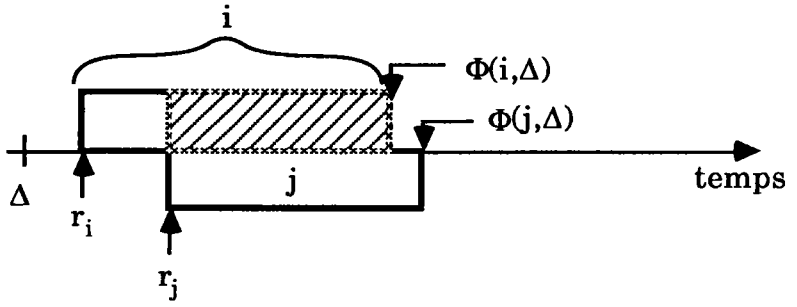


Fig. 3.9. Exécution de la tâche i non interrompue

Si on considère les deux problèmes d'ordonnancement P_1 et P'_1 obtenus à partir des problèmes P et P' en ôtant la tâche i et en démarrant à l'instant Δ et si on définit les ordonnancements O_1 et O'_1 de la même façon que O et O', il est évident que O_1 et O'_1 fournissent la fin des ordonnancements O et O' (par construction) et on obtient comme ci-dessus que leur première tâche est identique et qu'elle n'est pas interrompue dans O'_1. On peut ainsi construire une suite de problèmes et d'ordonnancements dont les tâches coïncident dans O, O', O_k et O'_k. Pour k=n-1, le lemme est démontré.

Q.E.D.

Dans [Deogun 1983], l'auteur a démontré le théorème suivant:

Si pour un problème P du type $n/1/r_i/\Sigma F_i$, nous pouvons construire un ordonnancement vérifiant SPT dans lequel la première tâche de toute sous-séquence contiguë a une date d'arrivée au plus aussi grande que celle des tâches qui suivent, alors cet ordonnancement est optimal pour P.

Une solution optimale vérifiant le théorème de Deogun vérifie aussi le théorème de Dessouky & Deogun. Car tout ordonnancement O vérifiant SPT et dans lequel la date d'arrivée de la première tâche de toute sous-séquence contiguë est au plus aussi grande que celle des tâches qui suivent vérifie aussi ECT.

Par contre, il peut ne pas exister d'ordonnancement vérifiant le théorème de Deogun alors qu'il existe des ordonnancements vérifiant celui de Dessouky & Deogun comme le montre le contre-exemple suivant:
n=2, r_1=0, p_1=8, r_2=3, p_2=6 où ECT est constitué de 1 suivi de 2 (sans interruption) et donne la valeur optimale de 19 pour la somme des temps de présence, alors que SPT est constituée de 2 suivie de 1 n'est pas optimale (somme des temps de présence=23).

En conclusion de cette comparaison de diverses conditions suffisantes d'optimalité, les plus puissantes sont équivalentes et consistent à tester s'il existe un ordonnancement vérifiant SRPT pour le problème relaxé (l'interruption des tâches est permise) dans lequel aucune interrution n'a lieu. Comme nous utilisons aussi la règle SRPT pour le calcul du minorant de la PSE (voir le paragraphe 3.5.4), le test d'optimalité peut être effectué en même temps que le calcul du minorant.

3.5.3.3. Comparaison de minorants

Passons maintenant aux comparaisons entre les minorants proposés.

Dans [Dessouky & Deogun 1981], les auteurs ont proposé un minorant noté M_D calculé de la manière suivante:
Pour un problème donné P du type $n/1/r_i/\Sigma F_i$, on construit un ordonnancement O vérifiant ECT. On construit un autre problème P', et un ordonnancement O' pour P' en conservant l'ordre des tâches de l'ordonnancement O, mais en modifiant les dates d'arrivée des tâches de P' de la manière suivante [15]

$C'_{[0]}=0$
Pour** i **de** 1 **à** n **faire
$m_{[i]}=\min_{j\geq i}(r_{[j]})$
$r'_{[i]}=\min(r_{[i]},\max(C'_{[i-1]},m_{[i]}))$
$C'_{[i]}=\max(r'_{[i]},C'_{[i-1]})+p_{[i-1]}$
finpour

La valeur $M_D=\sum_{i=1}^{n}(C'_{[i]}-r_{[i]})$ *est un minorant du problème de départ P.*

---

[15] où $C'_{[i]}$ est la date de fin de la tâche en $i^{ième}$ position dans O', par convention $C'_{[0]}=0$)

Propriété 3.5

Le minorant M_D proposé par Dessouky et Deogun est toujours inférieur ou égal au minorant M_I calculé en utilisant SRPT pour le problème relaxé qui autorise l'interruption des tâches.

Démonstration:

Pour un problème donné P, construisons un ordonnancement O vérifiant ECT pour P. Construisons un problème $P^{(1)}$ et un ordonnancement $O^{(1)}$ pour $P^{(1)}$ de la manière suivante ($C[0,O^{(1)}]=0$, $s[i,\sigma]$ est la tâche se terminant à la $i^{ième}$ position dans l'ordonnancement σ):

i) les tâches sont exécutées dans le même ordre dans O et dans $O^{(1)}$;

ii) $\forall i,\ p_i^{(1)}=p_i$

iii) $\forall i,\ m_i=\min_{j\geq i}(r_{[j,O]})$

iv) $\forall i,\ r_{s[i,O]}^{(1)}=\min[r_{s[i,O]},\ \max(C[i-1,O^{(1)}],m_i)],\ C[i,O^{(1)}]=\max(r_{s[i,O]}^{(1)},C[i-1,O^{(1)}])$

Construisons un problème $P^{(2)}$ identique à P sauf en ce que les tâches sont interruptibles et un autre problème $P^{(3)}$ identique à $P^{(2)}$ sauf en ce que $\forall i,\ r_i^{(3)}=r_i^{(1)}\leq r_i=r_i^{(2)}$

En notant C_Π^* la somme des dates de fin des tâches d'un ordonnancement optimal du problème Π, nous avons $C_{P(3)}^*\leq C_{P(2)}^*$

Dans [Dessouky & Deogun 1981], les auteurs ont démontré que $O^{(1)}$ vérifie ECT et que dans $O^{(1)}$ la première tâche de toute sous-séquence contiguë a une date d'arrivée au plus aussi grande que celle des tâches qui suivent. D'après le lemme 3.3, on a alors, $C_{P(1)}^*=C_{P(3)}^*\leq C_{P(2)}^*$

Ce qui implique:

$$M_D=C_{P(1)}^*-\sum_{i=1}^{n}r_i\leq C_{P(2)}^*-\sum_{i=1}^{n}r_i=M_I$$

Q.E.D.

Dans ([Deogun 1983]) l'auteur utilise un autre minorant calculé de la manière suivante:

Pour un problème donné P du type $n/1/r_i/\Sigma F_i$, on construit un autre problème P' avec
$\forall i$, $r'_i=0$, $p'_i=p_i$
On construit un ordonnancement O_G vérifiant SPT pour P', alors $M_G=\sum_{i=1}^{n}[C_i(O_G)-r_i]$ est un minorant du problème P.

On constate immédiatement en examinant les modes de calcul de M_G et de M_D que:
$M_G \leq M_D$

Ces comparaisons nous conduisent à n'utiliser que le minorant calculé en appliquant la règle SRPT pour le problème relaxé.

3.5.4. UN ALGORITHME EXACT DE TYPE PSE

Outre les théorèmes 3.8 et 3.13, nous citons et utilisons les théorèmes de dominance suivants pour éliminer davantage de sous-ensembles de solutions.

Théorème 3.14 ([Deogun 1983])

On considère un problème du type $n/1/r_i/\Sigma F_i$. Etant donné un ordonnancement σ composé d'une séquence partielle S suivie d'une séquence de tâches de N-S obtenue en suivant la règle SPT, notons I_k la somme des temps libres sur la machine devant la tâche en $k^{ième}$ position, $I_k=I_{k-1}+max(r_{[k]}-C_{[k-1]},0)$ avec $I_0=0$, $C_{[0]}=0$. S'il existe une tâche $i \in N-S$ en position h ($h>card(S)$), telle que $\Phi(i,E(S))-E(S) \geq I_{h-1}-I_{card(S)}$, alors $\Sigma(S,i)$ domine $\Sigma(S,j)$ pour toute tâche $j \in N-S$ telle que $p_i \leq p_j$ et $R(i,E(S)) \leq R(j,E(S))$.

Par ailleurs, en simplifiant un théorème proposé dans ([Bianco & Ricciardelli 1982]) pour le problème $n/1/r_i/\Sigma w_i F_i$ en prenant tous les w_i égaux à 1, on obtient le théorème suivant:

Théorème 3.15

On considère un problème du type $n/1/r_i/\Sigma F_i$, N étant l'ensemble des tâches et S une séquence partielle de tâches de N. Soient i et j deux tâches de N-S, si $\Phi(i,E(S)) \geq \Phi(j,E(S))$ et $p_i-p_j \geq [\Phi(i,E(S))-\Phi(j,E(S))] \cdot card(N-S)$, alors $\Sigma(S,i)$ domine $\Sigma(S,j)$.

Dans la PSE, les théorèmes de dominance 3.8, 3.13, 3.14, 3.15 sont utiles pour éliminer des sous-ensembles de solutions.

Les algorithmes PRTF et APRTF peuvent être utilisés pour déterminer une bonne solution initiale.

Le minorant peut être calculé à partir de la règle de priorité SRPT en considérant le problème relaxé comportant des tâches interruptibles.

Le test d'optimalité peut être fait pendant le calcul du minorant.

Nous décrivons maintenant en détail la PSE.

Comme pour la minimisation de la somme des retards, l'algorithme est composé de trois phases: initialisation, séparation et terminaison. Les phases d'initialisation et de terminaison sont similaires à celle de la PSE pour la somme des retards. Nous expliquons uniquement la partie "séparation".

Création éventuelle d'un nœud B(h)|i:

Etape 1: ***Test 1** "éliminer B(h)|i lorsqu'il ne peut pas conduire à un ordonnancement actif"*
S'il existe une autre tâche $j \in A(h)$ telle que $\Phi(j,t(h)) \leq R(i,t(h))$ **alors aller à l'étape 10, finsi.**

Etape 2: ***Test 2** "éliminer B(h)|i lorsqu'il ne peut pas conduire à un ordonnancement F-actif"*
Si $card(B(h)) \geq 1$ et $R(x,\phi_x) \geq R(i,\phi_x)$ et $PRTF(x,\phi_x) > PRTF(i,\phi_x)$, où x est la dernière tâche ordonnancée dans B(h) **alors aller à l'étape 10, finsi.**

Etape 3: ***Test 3** "éliminer B(h)|i lorsqu'il est dominé par une autre séquence B(h)|j d'après le théorème 3.13"*
S'il existe une autre tâche $j \in A(h)$ telle que $\Phi(j,t(h)) \leq \Phi(i,t(h))$ et $\Phi(j,t(h)) - \Phi(i,t(h)) \leq (p_j - p_i) \cdot [card(A(h))-1]$ **alors aller à l'étape 10, finsi.**

Etape 4: ***Test 4** "éliminer B(h)|i lorsqu'il est dominé par une autre séquence B(h)|j d'après le théorème 3.14"*
S étant un ordonnancement composé de B(h) suivi des tâches de A(h) dans l'ordre de leurs durées opératoires, **s'il** existe une autre tâche $j \in A(h)$ telle que j occupe $k^{ième}$ position dans S, et que

$\Phi(j,t(h))-t(h) \geq I_{k-1}-I_{card(B(h))}$ et $p_j \leq p_i$ et $R(j,t(h)) \leq R(i,t(h))$ **alors aller à l'étape 10, finsi.**

Etape 5: ***Test 5*** *"éliminer B(h)|i lorsqu'il est dominé par une autre séquence B(h)|j d'après le théorème 3.15"*
S'il existe une autre tâche $j \in A(h)$ telle que $\Phi(j,t(h)) \geq \Phi(i,t(h))$ et $p_j-p_i \geq [\Phi(j,t(h))-\Phi(i,t(h))] \cdot card(A(h))$ **alors aller à l'étape 10, finsi.**

Etape 6 *"Créer un nouveau nœud o correspondant à l'ordonnancement partiel B(h)|i"*
B(o):=B(h)|i;
t(o):=Φ(i,t(h));
A(o):=A(h)-{i};
F(o):=F(h)+t(o)-r_i;

Etape 7 *"Obtenir un ordonnancement réalisable complet et en tirer les conséquences"*
Si card(A(o))≤2 **alors**
Construire l'ordonnancement σ des tâches de A(o) à partir de l'instant t(o) en appliquant l'algorithme PRTF. Cet ordonnancement local est optimal d'après le théorème 3.8.
Si F(o)+F(σ)<UB **alors** *"nous avons obtenu un ordonnancement complet strictement meilleur que l'ordonnancement antérieur réalisable S"*, UB est réinitialisé à F(o)+F(σ), S devient B(o) suivi de σ.
Eliminer tout nœud o' de L telle que LB(o')≥UB, **finsi,**
aller à l'étape 10, finsi.

Etape 8 *"Calculer un minorant pour o et en tirer les conséquences"*
LB(o) := F(o) + MINOR(t(o), A(o));

> MINOR(t(o),A(o)) est une fonction similaire à la fonction décrite dans la PSE pour la somme des retards. Il suffit d'y remplacer l'instruction MINOR:=MINOR+max(t-D_j,0) par l'instruction MINOR:=MINOR+t-r_k.

Test 6 *"éliminer o lorsqu'il ne peut pas conduire à un ordonnancement dont la somme des temps de présence est strictement inférieure à UB"*:
Si LB(o)≥UB **alors aller à l'étape 10, finsi.**

Test 7 *"éliminer o lorsqu'il est dominé par un autre nœud o' déjà généré"*
S'il existe un nœud o' dans $\mathcal{L}$ tel que: $LB(o')\leq LB(o)$ et $A(o')=A(o)$ et $t(o')\leq t(o)$, **alors aller à l'étape 10, finsi.**

Etape 9 *"Insérer le nœud o à son rang dans la liste $\mathcal{L}$ et en tirer les conséquences"*
Insérer le nœud o dans la liste triée $\mathcal{L}$.
Eliminer tout nœud o' dominé par o dans la liste $\mathcal{L}$, c'est-à-dire tout nœud tel que $LB(o')\geq LB(o)$, $A(o')=A(o)$ et $t(o)\geq t(o')$.

Etape 10 *"Terminer la traitement de la création du nœud B(h)|i"*

3.5.5. RESULTATS NUMERIQUES

Dans cette section, nous fournissons des résultats numériques afin d'évaluer les performances des algorithmes.

Les algorithmes heuristiques et la PSE ont été programmé sur un SUN 3/110. Nous avons généré au hasard 7 échantillons d'exemples avec respectivement 20, 30, 40, 50, 60, 70, 80 tâches. Les exemples sont générés de la même manière que dans [Hariri & Potts 1983] pour le problème $n/1/r_i/\Sigma w_iF_i$ en prenant ici tous les w_i égaux à 1. Les durées opératoires des tâches ont été générées entre 1 et 100. Les dates d'arrivées des tâches ont été générées entre 0 et $50,5 \cdot n \cdot \alpha$. 10 valeurs de α (0,2, 0,4, 0,6, 0,8, 1,0, 1,25, 1,50, 1,75, 2,0, 3,0) nous ont permis de générer 10 exemples pour chaque échantillon.

Table 3.11. Résultats de la PSE

n	20	30	40	50	60	70	80
NNC	19.2	52.1	153.7	193.7	1358.6	2099.3	3585.3
NNG	19.7	57.5	188.9	265.7	1542.5	2658.3	4235.5
CPUT	0.18	0.76	4.93	11.93	83.08	304.44	274.40*

NNC: nombre de nœuds considérés
NNG: nombre de nœuds générés
CPUT: temps de CPU en secondes
*: calculé à partir des exemples résolus en moins de 36 minutes. Seuls cas, où le compteur du temps de CPU donne une valeur correcte (pas de débordement du compteur).

La table 3.11 résume les résultats concernant la PSE. Dans cette table, on peut voir que cet algorithme est capable de résoudre des problèmes contenant jusqu'à 80 tâches. Seulement un exemple avec 80 tâches n'a pas pu être résolu en moins de 36 minutes.

Table 3.12. Valeur moyenne du critère $\frac{1}{n}\sum F_i$ pour chaque échantillon

n	20	30	40	50	60	70	80
APRTF	199.760	217.633	253.295	340.962	356.570	364.367	397.551
PRTF	202.140	220.440	255.745	348.794	369.203	376.660	411.139
EST	200.315	218.100	253.532	341.134	356.868	364.786	397.741
ECT	220.975	243.920	289.665	385.876	420.875	424.991	464.089
UPRTF	199.760	217.380	252.372	340.854	356.397	364.216	397.501
UET	200.315	217.870	253.532	141.134	356.868	364.786	397.741
BAB	199.610	215.847	251.192	339.332	355.768	362.653	396.406

UPRTF: un "macro" algorithme consistant à toujours retenir la meilleure solution fournie par APRTF et PRTF

UET: un "macro" algorithme consistant à toujours retenir la meilleure solution EST et ECT

La table 3.12 montre les valeurs moyennes du critère obtenues par différents algorithmes. A partir de cette table, on voit qu'en moyenne, la meilleure solution trouvée par APRTF et PRTF (i.e. la solution trouvée par UPRTF) est très proche de l'optimum. La déviation relative ne dépasse jamais 0,71%. On voit aussi que pour chaque échantillon la solution trouvée par UPRTF est toujours meilleure que la meilleure solution trouvée par ECT et EST (i.e. la solution trouvée par UET).

Table 3.13. Temps moyen de calcul (en secondes CPU)

n	20	30	40	50	60	70	80
APRTF	0.0128	0.0240	0.0368	0.0608	0.0816	0.1024	0.1264
PRTF	0.0080	0.0176	0.0224	0.0272	0.0480	0.0656	0.0800
EST	0.0080	0.0112	0.0144	0.0352	0.0432	0.0432	0.0640
ECT	0.0016	0.0208	0.0224	0.0352	0.0544	0.0720	0.0928
UPRTF	0.0208	0.0416	0.0592	0.0880	0.1296	0.1680	0.2064
UET	0.0096	0.0320	0.0368	0.0704	0.0976	0.1152	0.1568
BAB	0.18	0.76	4.93	11.93	83.08	304.44	274.40

La table 3.13 montre le temps moyen de calcul sur un SUN 3/110 consommé par les différents algorithmes pour chaque échantillon. On peut remarquer que l'algorithme APRTF consomme un peu plus de temps, mais comme on peut le voir dans table 3.12 et dans la suite, il est meilleur que les autres.

Si nous comparons les heuristiques proposées avec respectivement UET et la PSE, nous obtenons les tables 3.14 et 3.15. Dans ces tables, on voit que APRTF est toujours au moins aussi bon que UET. Bien que PRTF ne soit pas toujours meilleur que UET, il est toujours au moins aussi bon que ECT. Pour certains exemples, il est même meilleur que APRTF. Par conséquent, utiliser UPRTF ou tout simplement APRTF semble une bonne solution pour trouver une solution très proche de l'optimum. A partir de la table 3.15, nous constatons que le nombre de fois où UPRTF donne l'optimum décroît avec le nombre de tâches. A partir des tables 3.14 et 3.15, nous pouvons conclure que l'efficacité de la PSE provient aussi de la bonne performance des heuristiques.

Table 3.14. B : SB

n	20	30	40	50	60	70	80
APRTF	10 : 4	10 : 6	10 : 3	10 : 6	10 : 4	10 : 3	10 : 5
PRTF	5 : 3	6 : 5	4 : 3	3 : 3	2 : 2	2 : 2	2 : 2
UPRTF	10 : 4	10 : 7	10 : 4	10 : 7	10 : 4	10 : 4	10 : 7

Chaque case de la table contient les informations B : SB où
B: nombre d'exemples (sur 10) pour lesquels un algorithme est au moins aussi bon que UET
SB: nombre d'exemples (sur 10) pour lesquels un algorithme est strictement meilleur que UET

Table 3.15. O : SO

n	20	30	40	50	60	70	80
APRTF	9 : 4	5 : 2	4 : 2	2 : 0	1 : 0	0 : 0	1 : 0
PRTF	5 : 3	4 : 1	2 : 1	0 : 0	0 : 0	0 : 0	0 : 0
UPRTF	9 : 4	6 : 3	4 : 2	2 : 0	1 : 0	0 : 0	1 : 0

Chaque case de la table contient les informations O : SO où
O: nombre d'exemples pour lesquels un algorithme donne l'optimum
SO: nombre d'exemples pour lesquels un algorithme donne l'optimum alors que UET ne le donne pas

Bien que, dans les expériences faites, UPRTF soit toujours meilleur que UET, on peut construire des contre-exemples comportant un petit nombre de tâches.

Un contre-exemple montrant que EST peut donner une solution strictement meilleure que celle donnée par UPRTF est donnée ci-dessous.

Jeu de données du premier contre-exemple

Tâche	r	p
1	2	13
2	0	20
3	20	1

La solution donnée par UPRTF est 1, 3, 2 ce qui donne une somme des temps de présence de 55, alors que la solution donnée par EST est 2, 3, 1, ce qui donne une somme des temps de présence de 53.

Un contre-exemple montrant que ECT peut donner une solution strictement meilleure que celle donnée par UPRTF est le suivant.

Jeu de données du second contre-exemple

Tâche	r	p
1	0	200
2	10	181
3	191	10

La solution donnée par UPRTF est 1, 3, 2 dont la somme des temps de présence correspondante est 600, alors que la solution donnée par ECT est 2, 3, 1 dont la somme des temps de présence correspondante est 592.

On peut penser que ces cas de figure où EST ou ECT sont strictement meilleurs que APRTF ou PRTF sur quelques tâches sont beaucoup plus rares que les cas contraires, ce qui fait que pour des exemples comportant un grand nombre de tâches APRTF et PRTF sont meilleurs que EST et ECT.

3.6. CONCLUSION GENERALE DU CHAPITRE

Dans ce chapitre, nous avons considéré des problèmes d'ordonnancement à une machine. Nous avons obtenu des résultats théoriques que nous avons utilisés pour développer des heuristiques.

En particulier, nous avons démontré différentes conditions suffisantes ou nécessaires et suffisantes d'optimalité locale pour des problèmes NP-difficiles. Cela nous a permis de définir des ensembles dominants pour différents critères et de construire des algorithmes approchés performants que nous avons utilisés dans la mémoire artificielle pour les problèmes d'ordonnancement à une machine. Ces conditions peuvent aussi être utilisées comme règles de priorité dans des cas plus généraux. En particulier, ces règles nous ont permis d'enrichir les règles de priorité pour les problèmes d'ordonnancement de type Job-shop.

Les résultats théoriques obtenus sont d'autant plus intéressants pour le critère minimisation de la somme des retards avec des dates d'arrivée des tâches différentes que la littérature est pauvre pour ce type de problème.

Chapitre 4

EVALUATION DE LA MEMOIRE ARTIFICIELLE

4.1. INTRODUCTION

Dans ce chapitre, nous allons montrer des résultats numériques obtenus dans le cadre de la mémoire artificielle pour l'évaluer dans le cas monocritère ; en effet il n'est pas possible de l'évaluer dans le cas d'une utilisation multicritère interactive faisant intervenir un décideur. Cette évaluation est faite en comparant les résultats obtenus en appliquant l'heuristique proposée par la mémoire artificielle et ceux obtenus en appliquant systématiquement la "meilleure" heuristique (appelée aussi stratégie habituelle).

Ces résultats numériques recouvrent les trois applications retenues. Dans la section 4.2, nous présentons les résultats obtenus pour les problèmes d'ordonnancement de type Job-shop. La section 4.3 est dédiée aux problèmes d'ordonnancement à une machine et la section 4.4 est consacrée aux problème de container loading. Dans toutes ces applications, nous présentons respectivement les résultats relatifs à la construction et à l'utilisation. Ces résultats comportent deux aspects: le temps de calcul (mesuré sur un SUN 3/110 sauf indication contraire) et les résultats des valeurs des mesures. Pour la construction, nous présentons les résultats et les coûts des phases d'expérimentation, d'élimination, de recouvrement par des classes et de partitionnement en régions.

Dans tout ce chapitre, les résultats sont présentés sous forme de tableaux. Nous utilisons des notations suivantes:

ν: nombre de classes;
M: moyenne du nombre d'occurrences d'énoncé dans une classe;
E: écart-type du nombre d'occurrences d'énoncé dans une classe;
%q: pourcentage d'occurrences d'énoncé dans la classe de modalité q;
ψ: nombre de régions obtenues;
ϕ: nombre total de points flous;
ϕq: nombre de points flous dans la classe de modalité q;
NG: nombre de fois où la mémoire artificielle donne une valeur du critère strictement meilleure que la stratégie habituelle;
NP: nombre de fois où la mémoire artificielle donne une valeur du critère strictement moins bonne que la stratégie habituelle;
NI: nombre de fois où la mémoire artificielle demande un rafraîchissement;
NH: nombre de fois où la mémoire artificielle propose la stratégie habituelle;

VH: valeur moyenne du critère fournie par la stratégie habituelle;
VM: valeur moyenne du critère fournie par la mémoire artificielle;
TH: temps total de calcul en appliquant systématiquement la stratégie habituelle;
TM: temps total de calcul en utilisant la mémoire artificielle;
TRI: nombre de fois où l'heuristique donne la meilleure valeur au critère, critère principal de tri des heuristiques dans les tableaux et pour l'utilisation de la mémoire;
MC: moyenne du critère pour chaque heuristique sur l'ensemble des occurrences d'énoncé.

Pour le partitionnement des classes en régions, nous avons choisi s0=3 pour l'épaisseur des régions, une taille minimale s1=5 pour les régions .

4.2. PROBLEMES D'ORDONNANCEMENT DE JOB-SHOP

Rappelons que l'on considère un atelier donné avec un nombre de machines fixé ici égal à 13 et que l'on gérère a priori un catalogue de produits dans lequel on choisit des produits de manière aléatoire pour créer les occurrences d'énoncé.

Pour ces problèmes, nous avons généré six grandes familles comportant chacune 100 occurrences d'énoncé pour la construction de la mémoire artificielle. Les familles diffèrent par le nombre de produits choisis dans la catalogue, les fourchettes de génération des dates d'arrivée et celles des dates de fin souhaitées. Les nombres de produits sont compris entre 30 et 60. Cela nous a donné des exemples de 60 à 160 opérations. Il nous a fallu 6 heures pour appliquer 27 heuristiques à ces 600 exemples et obtenir la valeur correspondante des critères.

Pour ces problèmes, nous choisissons les critères "minimisation de la somme des retards" et "minimisation de la somme des temps de présence" qui nous intéressent particulièrement. Les paragraphes 4.3.1 et 4.3.2 présentent respectivement des résultats concernant ces deux critères.

Pour évaluer la mémoire artificielle, nous avons généré 600 nouvelles occurrences d'énoncé avec les mêmes paramètres de génération que ceux utilisés pour la génération des occurrences d'énoncé pendant la construction

de la mémoire artificielle (mais les suites de nombres aléatoires générés ne sont pas les mêmes).

4.2.1. MINIMISATION DE LA SOMME DES RETARDS

Dans ce paragraphe, nous présentons des résultats obtenus pour le critère "minimisation de la somme des retards". Minimiser la somme des retards est un critère équivalent à minimiser la moyenne des retards par travail ; en outre, si l'on multiplie par une constante toutes les données d'une occurrence d'énoncé, le problème d'ordonnancement obtenu a toujours le même sous-ensemble de solutions optimales, mais la valeur correspondante du critère est multipliée par la constante ; on aurait donc des occurrences d'énoncé qui auraient les mêmes caractérisriques, mais pour lesquelles la valeur du critère, au lieu d'être identique, serait multipliée par la constante. Pour éviter cette anomalie, nous avons en fait utilisé dans la mémoire artificielle le critère équivalent pour toute occurrence d'énoncé: "minimisation de la somme des retards divisée par le nombre de travaux et par la somme totale de toutes les durées opératoires" C'est ce critère qui est présenté dans les tableaux ci-dessous, multiplié par 1000 en raison de sa valeur très petite. Avec ce critère, nous avons essayé tous les types de normalisation possibles.

1° Normalisation du type "Valeur initiale" et nombre de modalités q égal à 5

Les résultats intermédiaires sont résumés dans la table 4.1, où les heuristiques présentées sont celles qui ne sont pas dominées par les autres. L'ordre de présentation des heuristiques est l'ordre défini dans le chapitre 2, c'est l'ordre décroissant de la colonne TRI. L'heuristique EDD, placée en premier est donc la stratégie habituelle pour le critère "minimisation de la somme des retards". Il est assez surprenant que des heuristiques (MPRTF, PRTF, SR) ne tenant pas compte des délais des produits puissent fournir de meilleures valeurs que celles qui en tiennent compte.

En outre, on a vérifié que seulement 10 points sont flous pour toutes les heuristiques non dominées. Le temps utilisé pour calculer la matrice générale des distances (avec la métrique diagonale) et pour effectuer le recouvrement en classes et le partitionnement en régions est d'environ une heure. Le temps de calcul des grandeurs caractéristiques des régions (centre de gravité, métrique diagonale propre à la région, rayons maximal et minimal) est d'environ 10 minutes.

Table 4.1.

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
EDD	2	323	183	84.3	5	77	11	132	4.60
MPRTF	2	326	200	87.6	4	55	5	113	4.53
OPNDD	2	324	169	82.0	5	80	16	98	4.85
PRTF	2	329	177	84.2	4	85	12	92	4.80
MOD	2	324	158	80.3	6	82	18	70	5.04
SR	2	331	171	83.7	5	80	15	70	5.08
S_1	3	220	183	77.7	5	130	25	39	5.35
S_2	4	166	180	75.8	3	131	31	36	5.56
ECT	3	233	165	72.7	7	113	27	31	6.13

On peut remarquer qu'avec la normalisation "Valeur initiale", le nombre de points flous pour une heuristique n'est pas très important. Cela signifie que l'hypothèse "deux occurrences d'énoncé proches ont des valeurs du critère proches pour la même heuristique" est souvent vérifiée (hypothèse de continuité de la valeur du critère pour une heuristique donnée).

Les résultats finaux de la phase d'utilisation de cette version de la mémoire artificielle sont résumés dans la table 4.2.

Table 4.2.

NG	NP	NI	NH	VH	VM	TSH	TMA
37	50	7	503	4.685	4.692	25'	35'

On constate que pour 503 exemples sur 600, c'est-à-dire pour presque 84% des nouvelles occurrences d'énoncé générées, la mémoire propose la stratégie habituelle comme meilleure stratégie possible. Ceci n'est pas étonnant si l'on remarque dans le tableau 4.1 que 84,3% des occurrences d'énoncé de la mémoire se trouve dans la classe de meilleure modalité pour l'heuristique habituelle et si l'on se souvient que la mémoire recherche en priorité un rattachement à une région de meilleure modalité en commençant par l'heuristique habituelle (première dans le classement des heuristiques).

On voit, pour cette version, que le pourcentage de points flous n'est pas très important, mais la mémoire artificielle est moins bonne en moyenne que la stratégie habituelle (mais l'écart sur la moyenne est négligeable si on le compare avec les

écarts séparant les moyennes des heuristiques) aussi bien en temps de calcul qu'en valeur du critère.

2° Normalisation du type "Translation sans réduction", et nombre de modalités q égal à 5

Table 4.3.

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
EDD	3	208	256	94.8	23	3	2	132	4.60
MPRTF	2	308	273	96.8	21	9	0	113	4.53
OPNDD	2	309	242	92	34	8	5	98	4.85
PRTF	2	341	253	94.5	33	3	1	92	4.80
MOD	3	211	240	91.2	26	2	9	70	5.04
SR	2	317	243	93.3	34	2	6	70	5.08
S_1	2	316	204	86.5	38	7	13	39	5.35
S_2	3	217	209	84.7	59	3	18	36	5.56
ECT	3	218	179	77.0	41	60	31	31	6.13

Il n'y a aucun point toujours flou pour toutes ces heuristiques. La matrice des distances a déjà été calculée pour la version ci-dessus, le temps nécessaire pour le partitionnement en classes et le recouvrement en région est d'une dizaine de minutes. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ 5 minutes.

Les résultats finaux sont résumés dans la table 4.4.

Table 4.4.

NG	NP	NI	NH	VH	VM	TSH	TMA
0	2	3	595	4.685	4.686	25'	35'

L'expérience montre que la mémoire artificielle construite avec la normalisation "translation sans réduction" n'est pas meilleure pour le critère "minimisation de la somme des retards" qu'une bonne heuristique testée par simulation, mais ici encore, l'écart sur la valeur moyenne est négligeable.

3° Normalisation du type "Translation avec réduction" et nombre de modalités q égal à 5

Table 4.5.

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
EDD	5	135	99	54.0	9	397	132	132	4.60
MPRTF	5	136	75	44.2	10	458	141	113	4.53
OPNDD	5	135	75	46.3	8	477	184	98	4.85
PRTF	5	134	41	34.0	11	531	163	92	4.80
MOD	5	134	40	34.8	6	538	168	70	5.04
SR	5	135	32	30.3	13	532	139	70	5.08
S_1	5	136	29	30.7	7	545	147	39	5.35
S_2	5	133	42	27.5	10	525	152	36	5.56
ECT	5	129	102	14.7	9	369	81	31	6.13

Il y a 136 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques est environ 15 minutes.

On remarque qu'avec la normalisation "Translation avec réduction", le nombre de points flous est beaucoup plus important qu'avec les autres types de normalisation. Cela signifie que l'hypothèse "pour deux occurrences d'énoncé proches, c'est la même heuristique qui fournit la meilleure valeur du critère" est moins souvent vérifiée.

Les résultats finaux sont résumés dans la table 4.6.

Table 4.6.

NG	NP	NI	NH	VH	VM	TSH	TMA
22	26	48	500	4.685	4.678	25'	35'

On peut remarquer que bien que la mémoire construite avec la normalisation "translation avec réduction" soit plus souvent perdante que gagnante, elle fournit en moyenne une meilleure valeur au critère (comme précédemment, l'écart n'est pas très important).

4.2.2. MINIMISATION DE LA SOMME DES TEMPS DE PRESENCE

Comme pour la minimisation de la somme des retards, nous avons été amenés à utiliser le critère équivalent "minimisation de la somme des temps de présence divisée par le nombre de travaux et par la somme totale de toutes les durée opératoire".

Pour la normalisation "translation sans réduction", toutes les occurrences d'énoncé se trouvent dans la meilleure classe pour l'heuristique PRTF, la mémoire artificielle ne sert plus à rien, puisque pour toute nouvelle occurrence d'énoncé, la mémoire va proposer PRTF. C'est pourquoi nous présentons uniquement les normalisations "valeur initiale" et "translation avec réduction".

1° Normalisation du type "Valeur initiale" et nombre de modalités q égal à 5

Table 4.7.

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
PRTF	4	117	114	26.3	8	114	30	295	43.137
MPRTF	4	176	105	23.3	9	159	36	189	43.843
SR	4	183	97	19.7	9	121	44	55	45.400
ECT	4	180	108	15.3	11	176	41	47	45.723

Il y a 28 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ 10 minutes.

Les résultats finaux sont résumés dans la table 4.8.

Table 4.8.

NG	NP	NI	NH	VH	VM	TSH	TMA
3	11	4	582	43.378	43.476	25'	35'

La mémoire construite avec la normalisation "valeur initiale" donne de moins bons résultats que ceux obtenus en appliquant systématiquement la stratégie habituelle.

2° Normalisation du type "Translation avec réduction" et nombre de modalités q égal à 5

Table 4.9.

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
PRTF	5	128	148	69.8	3	227	50	295	43.137
MPRTF	5	132	89	51.3	10	428	151	189	43.843
SR	5	130	85	21.3	12	396	124	55	45.400
ECT	5	129	110	16.3	5	338	93	47	45.723

Il y a 70 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ 5 minutes.

Les résultats finaux sont résumés dans la table 4.10.

Table 4.10.

NG	NP	NI	NH	VH	VM	TSH	TMA
3	2	13	582	43.378	43.372	25'	35'

On constate à nouveau pour ce type de normalisation qu'il y a plus de flous mais que la mémoire construite avec la normalisation "translation avec réduction" est en moyenne légèrement meilleure que la stratégie habituelle.

4.3. RESULTATS POUR LES PROBLEMES D'ORDONNANCEMENT A UNE MACHINE

Pendant la construction de la mémoire artificielle pour ce problème, nous avons généré six grandes familles comportant 100 occurrences d'énoncé. Le nombre de tâches pour un exemple varie entre 200 et 400. L'expérience sur ces exemples a duré environ dix heures pour l'ensemble de 27 règles. Après l'expérience, pour le critère "minimisation de la somme des temps de présence", l'algorithme APRTF domine les autres, sauf pour un exemple où l'algorithme EST domine APRTF. On peut ainsi conclure que la mémoire artificielle n'a pas d'intérêt pour ce critère parce que le nouvel algorithme que nous avons proposé domine quasiment tous les algorithmes proposés antérieurement. Une bonne simulation peut remplacer complètement la mémoire artificielle. Pour le critère "minimisation du retard maximal", l'algorithme sans délai utilisant la règle EDD domine les autres. Il n'y a donc

pas d'intérêt non plus d'appliquer la mémoire artificielle. Pour le critère "minimisation de la somme des retards", les nouveaux algorithmes proposés au chapitre 3 (SDPRTT et IPRTT) dominent globalement tous les autres (sauf pour 20 occurrences d'énoncé), ce sont donc les seuls que la mémoire artificielle conserve.

Lorsque l'on utilise la normalisation "translation sans réduction", pour la stratégie habituelle (SDPRTT), toutes les occurrences d'énoncé se trouvent dans la meilleure classe. Ça signifie que même dans le cas où cette heuristique ne fournit pas la meilleure valeur, la différence entre la meilleure valeur et la valeur qu'elle fournit est très petite. Dans cette circonstance, toute nouvelle occurrence d'énoncé sera attachée à la meilleure classe de la stratégie habituelle. La mémoire artificielle perd son utilité.

1° Normalisation du type "Valeur initiale", et nombre de modalités q égal à 5

Table 4.11

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
NDPRTT	5	146	51	27.3	10	40	1	562	59.563
IPRTT	5	144	44	23.5	93	0	4	34	61.062

Il n'y a 21 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ 1 minute.

Les résultats finaux sont résumés dans la table 4.12.

Table 4.12.

NG	NP	NI	NH	VH	VM	TSH	TMA
0	0	30	570	83.831	83.831	35'	45'

La mémoire construite avec la normalisation "valeur initiale" est sans intérêt pour le critère.

2° Normalisation du type "Translation avec réduction", et nombre de modalités q égal à 2

Les résultats intermédiaires sont résumés dans la table 4.13.

Table 4.13

h	ν	M	E	%q	ψ	ϕ	ϕq	TRI	MC
NDPRTT	2	300	266	94.3	2	41	7	562	59.563
IPRTT	2	300	262	6,36	5	42	34	34	61.062

Il y a 41 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques est d'environ 30 secondes.

Les résultats finaux sont résumés dans la table 4.14.

Table 4.14.

NG	NP	NI	NH	VH	VM	TSH	TMA
0	0	2	598	83.831	83.831	35'	45'

Avec la normalisation "translation avec réduction", la classe d'occurrences d'énoncé pour lesquelles l'heuristique IPRTT fournit une meilleure valeur au critère est tellement petite qu'une nouvelle occurrence d'énoncé est toujours attachée à la stratégie habituelle. La mémoire artificielle n'est pas plus performante que la stratégie habituelle.

4.4. RESULTATS POUR LES PROBLEMES DE BIN-PACKING

Pour ces problèmes, nous avons généré 600 occurrences d'énoncé provenant de 14 familles pour construire la mémoire. Elles diffèrent par le nombre d'objets à placer, par les pourcentages des objets gros, moyens et petits et par les pourcentages des objets plats, cubiques, allongés et plats/allongés, et encore par le fait que des objets se répètent plusieurs fois dans une occurrence d'énoncé ou encore que l'on retrouve souvent ou non une dimension comme hauteur autorisée d'objets différents.

A partir d'une liste d'objets à placer, nous avons généré une ou plusieurs occurrences d'énoncé en générant un ou plusieurs emballages. Le volume de l'emballage est généré entre 110% et 130% du volume total des objets. Mais il est très rare que les différentes heuristiques réussissent à placer tous les objets dans l'emballage, aussi le critère utilisé ici, pourcentage en volume des objets placés (par rapport au volume total des objets) est compris entre 38 (quand on rejette accidentellement un gros objet) et 100, avec une moyenne comprise entre 70 et 75 pour l'ensemble des heuristiques.

Pour ce problème, avec le produit C.I.A.P. réalisé dans notre équipe, on a besoin d'environ 12 heures pour l'application d'une heuristique à 600 exemples sur un IBM PC AT et de 9 heures sur un SUN 3/60. Le nombre d'objets à mettre dans les emballages varie entre 15 et 30.

Pour ce problème, aucun algorithme n'est dominant. Les algorithmes sont très complémentaires.

D'après les expériences effectuées pour les autres applications, nous nous sommes aperçu que la normalisation "translation avec réduction" semble plus intéressante que les autres. Nous nous sommes donc limité à cette normalisation et nous ne testons que sur 98 exemples appartenant à 14 familles (car l'application des heuristiques est assez coûteuse). Par contre, nous avons essayé deux types de caractérisation: caractéristiques brutes, caractéristiques transformées.

Table 4.15

h	ν	M	E	%1	ψ	ϕ	ϕ1	TRI	MC
6	3	214	121	64,2	6	281	90	232	74.36
5	3	215	120	64,2	5	289	102	230	74.31
2	3	214	121	64,3	6	281	91	228	74.33
1	3	216	118	63,8	4	299	103	226	74.26
51	3	219	67	52,8	13	393	157	164	72.88
21	3	216	76	54,5	12	390	160	163	73.12
11	3	221	64	51,8	13	400	162	163	72.82
61	3	218	77	54,5	12	390	160	162	73.13
52	3	218	54	48,7	10	442	175	135	72.35
62	3	217	62	50,5	11	450	190	133	72.49
22	3	218	61	50,5	10	449	185	132	72.48
12	3	219	54	48,6	11	439	175	132	72.30
41	3	220	54	36,8	20	401	172	110	70.34
4	3	214	80	28,0	13	359	141	105	69.74
42	3	319	70	33,5	14	407	137	99	69.86
3	3	218	65	32,2	15	390	151	92	70.18
31	3	218	58	35,0	17	420	183	89	70.25
32	3	217	68	33,0	12	409	167	83	69.78

1° Caractéristiques brutes

Avec ces caractéristiques, le temps de calcul des distances générales est d'environ 1 heure et quart. Les résultats intermédiaires sont résumés dans la table 1.15. Il y a 47 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ 2 heures et 15 minutes.

Les résultats finaux sont résumés dans la table 4.16.

Pour ce problème, on voit que la mémoire artificielle est légèrement meilleure que la stratégie habituelle (car ici on cherche à maximiser le critère).

Table 4.16.

NG	NP	NI	NH	VH	VM	TSH	TMA
7	2	0	82	79.08	79.27	2h00	2h05

Table 4.17

h	ν	M	E	%1	ψ	ϕ	ϕ1	TRI	MC
6	3	214	121	64,2	4	304	98	232	74.36
5	3	215	120	64,2	8	317	126	230	74.31
2	3	214	121	64,3	4	302	97	228	74.33
1	3	216	118	63,8	6	319	121	226	74.26
51	3	219	67	52,8	9	410	159	164	72.88
21	3	216	76	54,5	8	381	136	163	73.12
11	3	221	64	51,8	13	409	161	163	72.82
61	3	218	77	54,5	8	381	136	162	73.13
52	3	218	54	48,7	13	432	175	135	72.35
62	3	217	62	50,5	12	409	156	133	72.49
22	3	218	61	50,5	12	409	156	132	72.48
12	3	219	54	48,6	15	415	161	132	72.30
41	3	220	54	36,8	14	456	172	110	70.34
4	3	214	80	28,0	10	377	144	105	69.74
42	3	319	70	33,5	12	418	168	99	69.86
3	3	218	65	32,2	13	405	160	92	70.18
31	3	218	58	35,0	18	426	173	89	70.25
32	3	217	68	33,0	11	399	162	83	69.78

2° Caractéristiques transformées

Il y a 34 points toujours flous pour toutes ces heuristiques. Le temps utilisé pour calculer les grandeurs caractéristiques des régions est d'environ une heure.

Les résultats finaux sont résumés dans la table 4.18.

Table 4.18.

NG	NP	NI	NH	VH	VM	TSH	TMA
1	1	0	95	79.08	79.12	2h00	2h05

Avec la caractérisation considérée, la mémoire est à peine meilleure que la stratégie habituelle.

4.5. CONCLUSION

Dans ce chapitre, nous avons présenté des résultats numériques pour évaluer l'approche mémoire artificielle. Nous constatons que l'approche mémoire artificielle ne fournit pas une efficacité aussi bonne que nous le souhaitions. Mais dans l'ensemble, elle fournit en moyenne des résultats équivalents à une bonne heuristique choisie par simulation pour tous les problèmes considérés. Elle fournit même une moyenne légèrement meilleure lorsque la mémoire est construite avec la normalisation "translation avec réduction". Par contre, comme elle est à peu près aussi bonne pour chaque critère donné, elle constitue un bon outil pour une approche multicritère où le décideur peut changer dynamiquement de critères au cours du temps (le problème d'ordonnancement devient dynamique si à l'instant Δ, on considère les travaux restant à faire). Mais malheureusement, il paraît difficile d'évaluer la mémoire dans ce contexte.

CONCLUSION

Ce travail comprend deux grandes parties: d'une part des applications du concept de mémoire artificielle à plusieurs problèmes d'ordonnancement et des améliorations de l'approche mémoire artificielle, d'autre part des développements théoriques pour certains problèmes d'ordonnancement particuliers à une machine.

En ce qui concerne la mémoire artificielle, l'idée a été proposée dans notre équipe en 1983. Son utilisation stricte sur nos exemples de taille relativement importante ne nous a pas donné des résultats satisfaisants. Cela nous a conduit à concevoir des adaptations. En effet la transcription intégrale de la méthode proposée par Bonneau & Proth (1985b) nous a conduit à une mémoire artificielle contenant un pourcentage important de points flous, mais qui surtout fonctionnait à un coût prohibitif en mémoire et en temps (car l'utilisation de la mémoire artificielle demandait que l'on conserve toutes les occurrences d'énoncé avec leurs caractéristiques, l'utilisation demandant non seulement la position des centres de gravité des régions, mais aussi la connaissance des points situés dans les régions pour des affectations utilisant une méthode de plus proches voisins).

Nous avons proposé un recouvrement en classes au lieu du partitionnement initial. Ceci nous a permis au moins pour les deux premiers types de normalisations de faire chuter considérablement le nombre de points flous.

Par ailleurs, notre méthode d'utilisation de la mémoire artificielle se limitant à la notion d'attachement fort ou faible des occurrences d'énoncé aux régions et à la grandeur des modalités correspondantes, nous avons pu obtenir un coût d'utilisation du même ordre de grandeur que l'application d'une heuristique.

Nous avons également cherché à évaluer cette approche dans le cas monocritère. L'évaluation utilisait deux critères d'évaluation: le coût de construction et d'utilisation et la qualité des solutions obtenues.

Apparemment, cette approche ne se révèle pas aussi efficace que l'on pouvait espérer au niveau de la qualité des solutions obtenues. Ceci est peut être dû à un choix maladroit ou incomplet des caractéristiques des occurrences d'énoncé ou à la non convexité des régions construites.

Mais dans l'ensemble, elle fournit en moyenne des résultats équivalents à une bonne heuristique choisie par simulation pour tous les problèmes considérés. Elle fournit même une moyenne légèrement meilleure lorsque la mémoire est construite avec la normalisation "translation avec réduction". Par contre, comme elle est à peu près aussi bonne pour chaque critère donné, elle constitue un bon outil pour une approche multicritère où le décideur peut changer dynamiquement de critères au cours du temps.

En ce qui concerne les études théoriques du chapitre 3, les résultats obtenus sont très satisfaisants. Nous avons démontré des conditions suffisantes ou nécessaires et suffisantes d'optimalité locale qui peuvent aussi être considérées comme des règles de priorité. Ceci nous a permis de définir des ensembles dominants de solutions. L'ensemble F-actif dominant contient même toutes les solutions optimales pour la minimisation de la somme des temps de présence. Les nouvelles heuristiques construites à partir des définitions des ensembles dominants donnent des résultats très proches de l'optimum pour les petits exemples où l'on est capable de calculer l'optimum et dominent très souvent les autres heuristiques pour les exemples de grande taille.

Pour la minimisation de la somme des retards, nous avons démontré un théorème complémentaire de dominance, Nous avons en outre été amenés à démontrer des théorèmes (non évidents) pour construire des minorants. Pour la minimisation de la somme des temps de présence, nous avons comparé des théorèmes de dominance, des conditions d'optimalité et des minorants, proposés dans la littérature ou démontrés par nous même de manière à éliminer ceux qui sont dominés. Ces résultats joints aux nouvelles heuristiques efficaces proposées nous ont permis de construire des algorithmes exacts. Ces algorithmes exacts permettent de résoudre des problèmes de plus grande taille que ceux résolus par les algorithmes proposés dans la littérature (pour la minimisation de la somme des retards, la taille des problèmes résolus n'est pas très grande, mais il n'existait pas d'algorithme publié ni exact ni approché lorsque les dates d'arrivée des travaux ne sont pas identiques; pour la minimisation de la somme des temps de présence, notre algorithme peut résoudre des problèmes avec 80 tâches, 30 tâches de plus que les algorithmes existants dans la littérature: pour un problème NP-difficile, cette amélioration n'est pas négligeable).

En ce qui concerne les perspectives, il semble intéressant d'effectuer des recherches plus approfondies dans les directions suivantes:

1° Définir des concepts plus appropriés que ceux de régions afin de réduire l'effet de la non convexité des régions. Les nouveaux objets à définir ne seraient pas nécessairement convexes, mais on exigerait au minimum que leur centre de gravité soit à l'intérieur de leurs frontières.

2° Construire de nouvelles mémoires artificielles avec d'autres types de caractérisation, afin de mieux mettre en évidence la ressemblance entre les occurrences d'énoncé (cela demanderait une analyse plus fine des données et des résultats obtenus pour les différents critères par les différentes heuristiques).

3° Etendre les résultats théoriques pour les problèmes d'ordonnancement à une machine aux problèmes d'ordonnancement comportant des machines en parallèle. Nous avons déjà généralisé des règles de priorité aux problèmes d'ordonnancement de type job-shop et les résultats obtenus sont assez encourageants.

4° Les problèmes de minimisation de la somme pondérée des temps de présence ont été abondamment étudiés dans la littérature, mais nous n'avons pas trouvé d'article traitant les problèmes de minimisation de la somme pondérée des retards lorsque les dates d'arrivées des tâches sont différentes. Il serait intéressant de travailler dans ce sens.

Il y a donc de nombreuses extensions possibles à ce travail, aussi bien pour l'amélioration de la mémoire artificielle que pour le développement de nouveaux algorithmes de résolution de problèmes d'ordonnancement.

REFERENCES

ABDUL-RAZAQ T.S., POTTS C.N. (1988), "Dynamic programming state-space relaxation for single-machine scheduling", *J. Opl. Res. Soc.*, 39, 141-152

ACHUGBUE J.O., F.Y. CHIN (1982), "Scheduling the open shop to minimize mean flow time", *SIAM J. Comput.*, 11, 709-720

ADAMS J., BALAS E., ZAWACK D. (1988), "The shifting bottleneck procedure for job shop scheduling", *Man. Sci.*, 34, 391-401

BAGCHI U., CHANG Y.L., SULLIVAN R.S. (1987), "Minimizing absolute and square deviation of completion times with different earliness and tardiness penalities and a common due date", *Nav. Res. Logist.*, 34, 739-751

BAGCHI U., AHMADI R.H. (1987), "An improved lower bound for minimizing weighted completion times with deadlines", *Opns. Res.*, 35, 311-313

BAGGA P.C., CHAKRAVARTI N.K. (1968), "Optimal m-stage production schedule", *J. Can. Opns. Res. Soc.*, 6, 71-78

BAKER B.S., COFFMAN JR E.G., RIVEST R.L. (1980), "Orthogonal packing in two dimensions", *SIAM. J. Comput.*, 9, 846-855

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

BAKER K.R., BERTRAND J.W.M. (1982), "A dynamic priority rule for scheduling against due-dates", *J. of Opns. Man.*, 3, 37-42

BAKER K.R., NUTTLE H.L.M. (1980), "Sequencing independent jobs with a single resource", *Nav. Res. Logist. Quart.*, 27, 499-510

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

BALZEWICZ J., CELLARY W., SLOWINSKI R., WEGLARZ J. (1986), *Scheduling under resource constraints — deterministic models*, J.C. BALTZER, Basel, Switzerland

BARKER J.R., McMAHON G.B. (1985), "Scheduling the general job-shop", *Man. Sci.*, 31, 594-598

BECHLER E., PROTH J.-M., VOYIATZIS K. (1984), "Artificial memory in production management", Rapport de Recherche, INRIA, Rocquencourt, France, n° 336, Septembre

BEL G., BENSANA E., DUBOIS D. (1986), "Un système d'ordonnancement prévisionnel d'atelier utilisant des connaissances théoriques et pratiques", *Proceeding de Sixièmes Journées Internationales "Les Systèmes Experts et leurs Applications"*, Avignon, 757-770

BELLMAN R. (1954), "The theory of dynamic programming", *Bull. Amer. Math. Soc.*, 60, 503-515

BENSANA E., BEL G., DUBOIS D. (1988), "OPAL: a knowledge-based system for industrial job-shop scheduling", in: I. BURHAN TURKSEN (ed.) *NATO ASI Series, Vol. F49, Computer Integrated Manufacturing*, Germany

BERKEY J.O., WANG P.Y. (1987), "Two-dimensional finite bin-packing algorithms", *J. Oper. Res. Soc.*, 38, 423-429

BIANCO L., RICCIARDELLI S. (1982), "Scheduling of a single machine to minimize total weighted completion time subject to release dates", *Nav. Res. Logist. Quart.*, 29, 151-167

BISCHOFF E.E., DOWSLAND W.B. (1982), "An application of the micro to product design and distribution", *J. Opns. Res. Soc.*, 33, 271-280

BISCHOFF E.E., MARRIOTT M.D. (1987), "A comparative evaluation for container loading", EURO 9 and TIMS, Paris

BONNEAU F., PROTH J.-M. (1985a), "Application de règles de gestion à un système de fabrication: classification des objectifs atteints en vue de leur utilisation", Rapport de Recherche, INRIA, Rocquencourt, France, n° 372, Mars

BONNEAU F., PROTH J.-M. (1985b), "Analyse discriminante: méthode du type plus proches voisins utilisant un prétraitement des données", Rapport de Recherche, INRIA, Rocquencourt, France, n° 440, Septembre

BRATLEY P., FLORIAN M., P. ROBILLARD (1973), "On scheduling with earliest start and due dates with application to computing bounds for the (n/m/G/Fmax) problem", *Nav. Res. Logist. Quart.*, 20, 57-67

CARLIER J. (1978), "Ordonnancement à contraintes disjonctives", *RAIRO*, 12, 333-351

CARLIER J. (1982), "The one machine sequencing problem", *Europ. J. Oper. Res.*, 11, 42-47

CARLIER J. (1984), *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité*, Thèse d'état, Unversité Pierre et Marie Curie (Paris 6), 28 mai

CARLIER J., PINSON E. (1989), "An algorithm for solving the job-shop problem", *Man. Sci.*, 35, 164-176

CHAND S., SCHNEEBERGER H. (1986), "A note on the single machine scheduling problem with minimum weighted completion time and maximum allowable tardiness", *Nav. Res. Logist. Quart.*, 33, 551-557

CHANDRA R. (1979), "On n/1/$\overline{F}$ dynamic deterministic systems", *Nav. Res. Logist. Quart.*, 26, 537-544

CHRETIENNE P. (1983), *Les réseaux de Petri temporisés*, Thèse d'état, Université Pierre et Marie Curie (Paris 6), juin

CHU C., PORTMANN M.C. (1989), "Minimisation de la somme des retards pour les problèmes d'ordonnancement à une machine", Raport de recherche, INRIA, France, n° 1023

CHU.C., PORTMANN M.C. (1990 a), "Some new efficient methods to solve the n/1/r_i/ΣT_i scheduling problem", *Europ. J. Oper. Res.*, à apparaître

CHU.C., PORTMANN M.C. (1990 b), "Single machine scheduling problems with release dates", Raport de recherche, INRIA, France, n° 1232

CHU.C., PORTMANN M.C. (1990 c), "Single machine scheduling problems with release dates", *Proceeding of WG-MPS*, Compiègne, 21-25 juin

COFFMAN JR E.G., FLATTO L., GAREY M.R., WEBER R.R. (1987), "Minimizing expected makespan on uniform processor systems", *Adv. Appl. Prob.*, 19, 177-201

COFFMAN JR E.G., GAREY M.R., JOHNSON D.S., TARJAN R.E. (1980), "Performance bounds for level-oriented two-dimensional packing algorithms", *SIAM. J. Comput.*, 9, 808-826

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

COOK S.A. (1971), "The complexity of theorem-proving procedures", *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, Association for Computing Machinery, New York, 151-158

DEOGUN J.S. (1983), "On scheduling with ready times to minimize mean flow time", *The Comp. J.*, 26, 320-328

DESSOUKY M.I., DEOGUN J.S. (1981), "Sequencing jobs with unequal ready times to minimize mean flow time", *SIAM J. Comput.*, 10, 192-202

DESSOUKY M.I., MARGENTHALER C.R. (1972), "The one-machine sequencing problem with early starts and due dates", *AIIE Transactions*, 4, 214-222

DOWSLAND W.B. (1987), "Computational consideration in packing algorithm development", EURO 9 and TIMS, Paris

DU J., LEUNG J.Y.-T. (1990), "Minimizing total tardiness on one machine is NP-hard", *Math. Oper. Res.*, to appear

DUDEK R.A., TEUTON JR O.F. (1965), "Development of m-stage decision rule for scheduling n jobs through m-machines", *Opns. Res.*, 13, 471-497

EASTMAN W.L., EVEN S., ISAACS I.M. (1964), "Bounds for the optimal scheduling of n jobs on m processors", *Man. Sci.*, 11, 268-279

EMMONS H. (1969), "One-machine sequencing to minimize certain functions of job tardiness", *Opns. Res.*, 17, 701-715

EMMONS H. (1987), "Scheduling to a common due date on parallel uniform processors", *Nav. Res. Logist.*, 34, 803-810

ERSCHLER J. (1976), *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*, thèse d'état, Université Paul Sabatier, 23 novembre

ERSCHLER J., FONTAN G., MERCE C. (1985), "Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine", *RAIRO*, 19, 15-26

ERSCHLER J., FONTAN G., MERCE C., ROUBELLAT F. (1983), "A new dominance concept on a single machine with ready times and due dates", *Opns. Res.*, 31, 114-127

FISHER M.L. (1976), "A dual algorithm for the one-machine scheduling problem", *Math. Prog.*, 11, 229-251

FISHER M.L., KRIEGER A.M. (1984), "Analysis of a linearization heuristic for single machine scheduling to maximize profit", *Math. Prog.*, 28, 218-225

FISHER M.L., LAGEWEG B.J., LENSTRA J.K., RINNOOY KAN A.H.G. (1983), "Surrogate duality relaxation for the job shop scheduling", *Discr. Appl. Math.*, 5, 65-75

FOX M.S. (1983), *Constraint-directed search: a case study of job-shop scheduling*, Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA, December, Technical Report CMU-R1-TR-83-22

FOX M.S., SMITH S.F. (1984), "ISIS: A knowledge based system for factory scheduling", *Expert System Journal*, 1, 25-49

GAREY M.R., JOHNSON D.S. (1978), *Computers and intractability: a guide to theory of NP-completeness*, W.H. Freeman, San Francisco

GAREY M.R., JOHNSON D.S. (1981), "Approximation algorithms for bin packing propblems", in: G. AUSIELLO, M. LUCERTINI (eds.), *Analysis and Design of Algorithms in Combinatorial Optimization*, Springer

GEORGE J.A., ROBINSON D.F. (1980), "A heuristic for packing boxes into a container", *Comput. and Opns. Res.*, 7, 147-156

GERES JR W.S. (1966), "Heuristics in job shop scheduling", *Man. Sci.*, 13, 167-190

GLOVER F. (1975), "Surrogate constraint duality in mathematical programming", *Opns. Res.*, 23, 434-451

GONZALEZ T., SANHI S. (1978), "Flowshop and jobshop schedules: complexity and approximation", *Opns. Res.*, 26, 36-67

GRABOWSKI J., NOWICKI E., ZDRZALKA S. (1986), "A block approach for single-machine scheduling with release dates and due dates", *Europ. J. Oper. Res.*, 26, 278-285

GRABOWSKI J., SKUBALSKA E., SMUTNICKI C. (1983), "On flow shop scheduling with release and due dates to minimize maximum lateness", *J. Opl. Res. Soc.*, 34, 615-620

GRAHAM E.L., LAWLER E.L., LENSTRA J.K., RINNOOY KAN A.H.G. (1979), "Optimization and approximation in deterministic sequencing and scheduling: a survey", *Ann. of Discr. Math.*, 5, 287-326

GUPTA J.N.D., REDDI, S.S. (1978) "Improved dominance conditions for the three-machine flowshop scheduling problem", *Opns. Res.*, 26, 201-203

GUPTA S.K., KYPARISIS J. (1987), "Single machine scheduling research", *OMEGA Int. J. Man. Sci.*, 15, 207-227

HALL N.G., RHEE W.T. (1986), "Average and worst-case analysis of heuristics for the maximum tardiness problem", *Europ. J. Oper. Res.*, 26, 272-277

HAN C.P., KNOTT K., EGBELU P.J. (1989), "A heuristic approach to the three-dimensional cargo-loading problem", *Int. J. Prod. Res.*, 27, 757-774

HARIRI A.M.A., POTTS C.N. (1983), "An algorithm for single machine sequencing with release dates to minimize total weighted completion time", *Discr. Appl. Math.*, 5, 99-109

HEFETZ N., ADIRI I. (1982), "An efficient algorithm for the two-machines unit-time job-shop schedule-length problem", *Math. of Oper. Res.*, 7, 334-360

HILLION H. (1989), *Modélisation et analyse des systèmes de production discrets par les réseaux de Petri temporisés*, Thèse de doctorat, 23 janvier

JACKSON J.R. (1955), "Scheduling a production line to minimize maximum tardiness", Research Report 43, Management Science Research Project, University of California, Los Angeles

JACKSON J.R. (1956), "An extension of Johnson's results on job-lot scheduling", *Nav. Res. Logist. Quart.*, 3, 201-203

JOHNSON S.M. (1954), "Optimal two- and three-stage production schedules with setup times included", *Nav. Res. Logist. Quart.*, 1, 61-68

JONES C.H. (1973), "An economic evaluation of job shop dispatching rules", *Man. Sci.*, 20, 293-307

KÄMPKE T. (1987), "On the optimality of static priority policies in stochastic scheduling on parallel machines", *J. Appl. Prob.*, 24, 430-448

KANET J.J. (1986), "Expert systems in production scheduling", *Europ. J. Oper. Res.*, 29, 51-59

KEMPF K.G. (1985), "Manufacturing and artificial intelligence", *Robotics*, 1, 13-25

KUSIAK A. (1987), "Artificial intelligence and operations research in flexible manufacturing systems", *INFOR*, vol. 25, n° 1, 2-12

KUSIAK A., CHEN M. (1988), "Expert systems for planning and scheduling manufacturing systems", *Europ. J. Oper. Res.*, 34, 113-130

LAGEWEG B.J., LENSTRA J.K., RINNOOY KAN A.H.G. (1976), "Minimizing maximum lateness on one machine: computational experience and some applications", *Statist. Neer.*, 30, 25-41

LAGEWEG B.J., LENSTRA J.K., RINNOOY KAN A.H.G. (1977), "Job-shop scheduling by implicit enumeration", *Man. Sci.*, 441-450

LAGEWEG B.J., LENSTRA J.K., RINNOOY KAN A.H.G. (1978), "A general bounding scheme for the permutation flow-shop problem", *Opns. Res.*, 26, 53-67

LAKSHMINARAYAN S., LAKSHMANAN R., PAPINEAU R.L., ROCHETTE R. (1978), "Optimal single-machine scheduling with earliness and tardiness penalities", *Opns. Res.*, 26, 1079-1082

LARSON R.E., DESSOUKY M.I., DEVOR R.E. (1985), "A forward-backward procedure for the single machine problem to minimize maximum lateness", *IIE Transactions*, 17, 252-260

LAWLER E.L. (1964), "On scheduling problems with deferral costs", *Man. Sci.*, 11, 280-288

LAWLER E.L. (1973), "Optimal sequencing of a single machine subject to precedence constraints", *Man. Sci.*, 19, 544-546

LAWLER E.L. (1977), "A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness", *Ann. of Discr. Math.*, 4, 331-342

LAWLER E.L. (1978), "Sequencing jobs to minimize total weighted completion time subject to precedence constraints", *Ann. of Discr. Math.*, 2, 75-90

LAWLER E.L. (1982), "A fully polynomial approximation scheme for the total tardiness problem", *Opns. Res. Letters*, 1, 207-208

LAWLER E.L., LABETOULLE J. (1978), "On preemptive scheduling of unrelated parallel processors by linear programming", *J. of Ass. for Comput. Mach.*, 25, 612-619

LAWLER E.L., LENSTRA J.K., RINNOOY KAN A.H.G., SHMOYS D.B. (1989), "Sequencing and scheduling: algorithms and complexity", Report BS-R8909 Center for Mathematics and Computer Science, Amsterdam, The Netherlands

LENSTRA J.K., RINNOOY KAN A.H.G., BRUCKER P. (1977), "Complexity of machine scheduling problems", *Ann. of Discr. Math.*, 1, 343-362

LE PAPE C. (1985), "SOJA: a daily working scheduling system, SOJA's system and inference engine", *Proceeding of the 5th Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Warwick, 195-211

LE PAPE C., SAUVE B. (1985), "SOJA: un système d'ordonnancement journalier d'atelier", *Proceeding de Cinquièmes Journées Internationales "Les Systèmes Experts et leurs Applications"*, Avignon, 849-867

LUH P.B., HOITOMT D.J., MAX E., RATTIPATI K.R. (1988), "Parallel machine scheduling using lagrangian relaxation", *IEEE*, 244-248

McMAHON G. (1969), "Optimal production schedules for flow shops", *Canad. Opns. Res. Soc. J.*, 7, 141-151

McMAHON G., FLORIAN M. (1975), "On scheduling with ready times and due dates to minimize maximum lateness", *Opns. Res.*, 23, 475-482

MOHAN R., RACHAMADUGU V. (1987), "A note on the weighted tardiness problem", *Opns. Res.*, 35, 450-452

MOLET H. (1989), *La nouvelle gestion de production*, HERMES

MOORE J.M. (1968), "Sequencing n jobs on one machine to minimize the number of tardy jobs", *Man. Sci.*, 15, 102-109

MUTH J.F., THOMPSON G.L. (1963), *Industrial scheduling*, Prentice Hall, Englewood Cliffs, NJ

OW P.S., MORTON T.E. (1989), "The single machine early/tardy problem", *Man. Sci.*, 35, 177-191

PANWALKER S.S., ISKANDER W. (1977), "A survey of scheduling rules", *Opns. Res.*, 25, 45-61

PIERREVAL H., RALAMBONDRAINY H. (1988), "Generation of knowledge about the control of a flow shop using data-analysis oriented learning techniques and simulation", Rapport de Recherche, INRIA, Rocquencourt, France, n° 897, septembre

PIERREVAL H., RALAMBONDRAINY H. (1989), "Generation of knowledge about the control of a flow shop using simulation and a learning algorithm", in: F. KIMURA, A. ROLSTADAS (eds.) *Computer Application in Production and Engineering*, North-Holland

PORTMANN M.-C. (1987), *Méthodes de décomposition spatiale et temporelle en ordonnancement de la production*, Thèse d'état, Université de Nancy 1, 18 septembre

POSNER M.E. (1986), "A sequencing problem with release dates and clustered jobs", *Man. Sci.*, 32, 731-738

POTTS C.N. (1980), "Analysis of heuristics for sequencing jobs on one machine with release dates and delivery times", *Opns. Res.*, 28, 1436-1441

POTTS C.N., VAN WASSENHOVE L.N. (1982), "A decomposition algorithm for the single machine total tardiness problem", *Opns. Res. Letters*, 1, 177-181

POTTS C.N., VAN WASSENHOVE L.N. (1983), "An algorithm for single machine sequencing with deadlines to minimize total weighted completion time", *Europ. J. Oper. Res.*, 12, 379-387

POTTS C.N., VAN WASSENHOVE L.N. (1985), "A branch and bound algorithm for the total weighted tardiness problem", *Opns. Res.*, 33, 363-377

POTTS C.N., VAN WASSENHOVE L.N. (1987), "Dynamic programming and decomposition approaches for the single machine total tardiness problem", *Europ. J. Oper. Res.*, 32, 405-414

PROTH J.-M., QUINQUETON J., RALAMBONDRAINY H., VOYIATZIS K. (1983a), "Utilisation de l'intelligence artificielle dans un problème d'ordonnancement", *Proceeding du Congrès Automatique: Productique et Robotique Intelligente, AFCET*, Besançon, 15-17 novembre, 53-61

PROTH J.-M., QUINQUETON J., RALAMBONDRAINY H., VOYIATZIS K. (1983b), "Problème d'ordonnancement: utilisation de l'intelligence artificielle", *Le Nouvel Automatisme*, novembre-décembre, 59-62

RINNOOY KAN A.H.G. (1976), *Machine sequencing problems: classification, complexity and computation*, Nijhoff, The Hague

RINNOOY KAN A.H.G., LAGEWEG B.J., LENSTRA J.K. (1975), "Minimizing total costs in one-machine scheduling", *Opns. Res.*, 23, 908-927

RODE M., ROSENBERG O. (1987), "An analysis of heuristic trim-loss algorithms", *Eng. Costs and Prod. Econ.*, 12, 71-78

ROY B. (1970), *Algèbre moderne et théorie des graphes*, DUNOD

ROY B., SUSSMAN B. (1964), "Les problèmes d'ordonnancement avec contraintes disjonctives", Notes DS n° 9 bis, SEMA, Paris

SAHNI S.K. (1976), "Algorithms for scheduling independent tasks", *J. of Ass. for Comput. Mach.*, 23, 116-127

SCHRAGE L. (1968), "A proof of the optimality of the shortest remaining processing time discipline", *Opns. Res.*, 16, 687-690

SCHRAGE L., BAKER K.R. (1978), "Dynamic programming solution of sequencing problems with precedence constraints", *Opns. Res.*, 26, 445-449

SEN T.T., AUSTIN L.M., GHANDFOROUSH P. (1983), "An algorithm for the single-machine sequencing problem to minimize total tardiness", *IIE Transactions*, 15, 363-366

SHWIMER J. (1972), "On the n-job, one-machine, sequence-independent scheduling problem with tardiness penalities: a branch-bound solution", *Man. Sci.*, 18, B301-B313

SIDNEY J.B. (1975), "Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs", *Opns. Res.*, 23, 283-298

SMITH D.R. (1978), "A new proof of the optimality of the shortest remaining processing time discipline", *Opns. Res.*, 26, 197-199

SMITH R.D., DUDEK R.A. (1967), "A general algorithm for solution of the n-job m-machine sequencing problem of the flow shop", *Opns. Res.*, 15, 71-82

SMITH W.E. (1956), "Various optimizers for single-stage production", *Nav. Res. Logist. Quart.*, 3, 59-66

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

STEFFEN M.S. (1986), "A survey of artificial intelligence-based scheduling systems", *Proceeding of Fall Industrial Engineering Conference*, Dec. 7-10, Boston, MA, 395-405

SUNDARARAGHAVAN P.S., AHMED M.U. (1984), "Minimizing the sum of absolute lateness in single machine and multimachine scheduling", *Nav. Res. Logist. Quart.*, 31, 325-333

SZWARC W. (1971), "Elimination methods in the mxn sequencing problem", *Nav. Res. Logist. Quart.*, 18, 295-305

SZWARC W. (1973), "Optimal elimination methods in the mxn flow-shop scheduling problem", *Opns. Res.*, 21, 1250-1259

SZWARC W. (1978), "Dominance conditions for the three-machine flow-shop problem", *Opns. Res.*, 26, 203-206

SZWARC W., POSNER M.E., LIU J.J. (1988), "The single machine problem with a quadratic cost function of completion times", *Man. Sci.*, 34, 1480-1488

TOVEY C.A. (1986), "Rescheduling to minimize makespan on a changing number of identical processors", *Nav. Res. Logist. Quart.*, 33, 717-724

TOWNSEND W. (1977), "Sequencing n jobs on m machines to minimise maximum tardiness: a branch-and-bound solution", *Man. Sci.*, 23, 1016-1019

VEPSALAINEN A.P.J., MORTON T.E. (1987), "Priority rules for job shops with weighted tardiness costs", *Man. Sci.*, 33, 1035-1047

VOYIATZIS K. (1987), *Utilisation de l'intelligence artificielle pour les problèmes d'ordonnancement*, thèse de doctorat ingénieur, Université Paris Dauphine

WANG P.Y. (1982), "Two algorithms for constrained two-dimensional cutting stock problems", *Opns. Res.*, 31, 573-586

WILKERSON L.J., IRWIN J.D. (1971), "An improved algorithm for scheduling independent tasks", *AIIE Transactions*, 3, 239-245

YAMAMOTO M. (1977), "An approximate solution of machine scheduling problems by decomposition method", *Int. J. Prod. Res.*, 15, 599-608

ZDRZALKA S., GRABOWSKI J. (1989), "An algorithm for single machine sequencing with release dates to minimize maximum cost", *Disc. Appl. Math.*, 23, 73-89

Annexe A

ETUDE ANALYTIQUE DE LA FONCTION PRTT

La fonction PRTT est donnée par la formule:
$PRTT(i,\Delta)=R(i,\Delta) + \max[\Phi(i,\Delta), d_i]$

En étudiant les différents cas possibles et en supposant que l'on travaille sur les délais corrigés ($d_i \geq r_i+p_i$), on obtient immédiatement:

$$PRTT(i,\Delta)=\begin{cases} r_i+d_i \text{ si } \Delta \leq r_i \\ \Delta+d_i \text{ si } r_i \leq \Delta \leq d_i-p_i \\ 2\Delta+p_i \text{ si } d_i-p_i \leq \Delta \end{cases}$$

C'est une fonction non décroissante, continue et linéaire par morceaux (voir Fig. A.1). Elle est constante tant que la tâche i n'est pas prête à être exécutée. Elle croît à la même vitesse que Δ tant que la tâche planifiée au plus tôt n'est pas en retard, et son ordonnée à l'origine est d_i. Aussi, si on compare sa priorité avec celle d'une autre tâche disponible et non en retard en commençant à l'instant Δ, c'est la tâche de plus petit délai corrigé qui est la plus prioritaire. Elle croît à une vitesse double de Δ lorsque la tâche planifiée au plus tôt est en retard et son ordonnée à l'origine est p_i. Aussi, si on compare sa priorité avec celle d'une autre tâche qui serait aussi en retard en commençant en Δ, c'est la tâche de plus petite durée qui est la plus prioritaire.

On peut considérer que cette règle de priorité est une règle souple qui, lorsque l'on compare deux tâches à placer consécutivement sur une machine, passe de la règle FIFO (First In First Out) quand une des deux tâches n'est pas encore sur le point d'arriver, à la règle EDD (Earliest Due Date) quand aucune des deux tâches n'est en retard, puis à la règle SPT (Shortest Processing Time) lorsque les deux tâches sont en retard. Les cas transitoires (par exemple, l'une en retard et pas l'autre) sont traités également puisque $PRTT(i,\Delta) \leq PRTT(j,\Delta) \Rightarrow T_{i,j}(\Delta) \leq T_{j,i}(\Delta)$. Les instants où la priorité change peuvent être obtenus analytiquement dans chacun des cas possibles comme le montrent les figures A.2, A.3 et A.4.

Toutes les figures ont été réalisées en supposant arbitrairement que la tâche d'indice i est la tâche ayant le plus petit délai ($d_i \leq d_j$).

Les figures de A.2 correspondent au cas où $r_i+d_i \leq r_j+d_j$ (ou encore $d_j-d_i \geq r_i-r_j$, c'est-à-dire que l'écart entre les délais est supérieur à l'écart entre les dates d'arrivées en inversant les tâches et quelle que soit la tâche qui arrive la première). Dans ce cas, la tâche de plus petit délai est toujours prioritaire si

c'est également la tâche de plus petite durée ($p_i \leq p_j$) ; sinon la tâche de plus petit délai est la plus prioritaire jusqu'à l'instant $d_j - p_i$ (à partir duquel les deux tâches seront en retard), puis celle de plus petite durée devient la plus prioritaire.

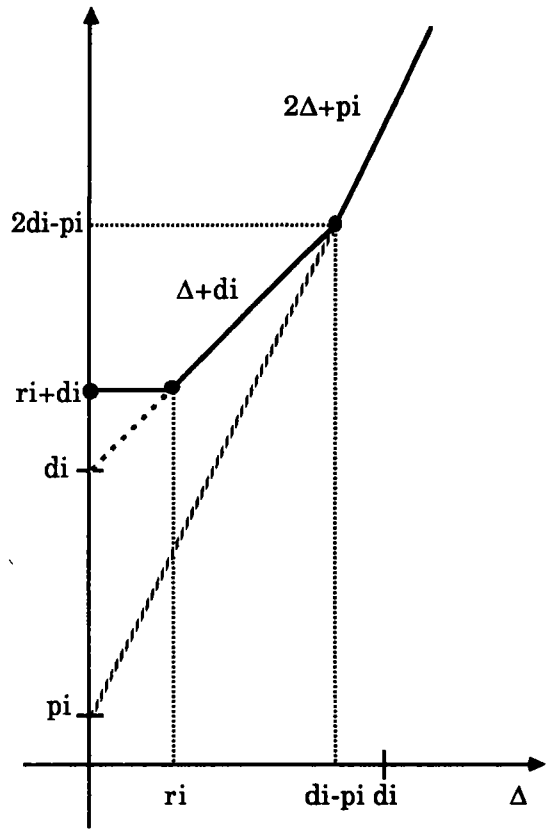


Fig. A.1

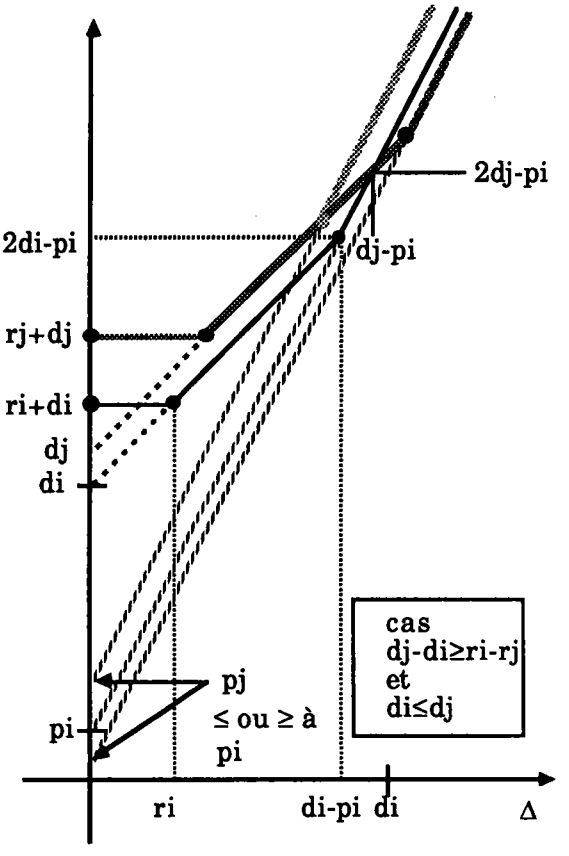


Fig. A.2

Les figures A.3 et A.4 correspondent au cas où $r_i+d_i \geq r_j+d_j$ (ou encore $d_j-d_i \leq r_i-r_j$, c'est-à-dire que l'écart entre les délais est inférieur à l'écart entre les dates d'arrivée en inversant les tâches). Dans ces deux cas, c'est obligatoirement la tâche de plus grand délai qui arrive la première.

La figure A.4 correspond au sous cas où $r_i+d_i \leq 2d_j-p_j$. Alors, la tâche qui arrive le plus tôt est la plus prioritaire jusqu'à l'instant $r_i+d_i-d_j$ (c'est-à-dire la date d'arrivée de celle de plus petit délai diminuée de l'écart entre les délais). Puis la tâche de plus petit délai devient la plus prioritaire et le reste si c'est aussi la tâche de plus courte durée. Sinon, c'est l'autre tâche de plus courte durée qui redevient prioritaire à partir de l'instant d_j-p_i comme dans le cas de la figure A.2.

La figure A.4 correspond au sous cas où $r_i+d_i \geq 2d_j-p_j$. Alors, la tâche qui arrive le plus tôt est la plus prioritaire jusqu'à l'instant $\frac{r_i+d_i-p_j}{2}$. Puis c'est la tâche de plus petit délai qui a également dans ce cas la plus petite durée qui devient la plus prioritaire et qui le reste.

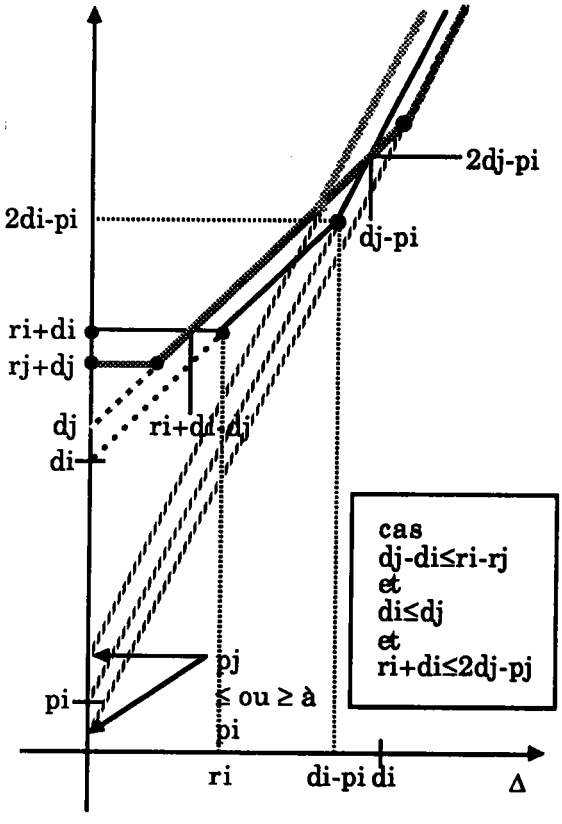


Fig. A.3

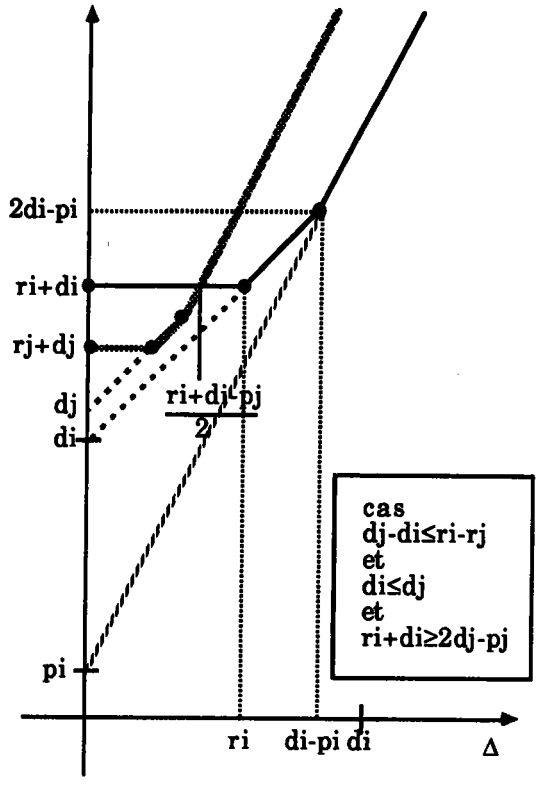


Fig. A.4

Cette étude analytique montre bien que si l'on s'intéresse à l'évolution de la priorité de deux tâches consécutives au cours du temps avec pour objectif de minimiser la somme des retards, alors la fonction PRTT donne d'abord la priorité à la tâche qui arrive la première (FIFO), puis donne la priorité à celle de plus petit délai (EDD) et enfin, lorsque les deux tâches sont en retard, elle donne la priorité à celle de plus petite durée (SPT).

Annexe B

DEMONSTRATION DU THEOREME 3.7

La démonstration se déroule en quatre parties.

Dans la première partie, on démontre des lemmes qui serviront dans la démonstration du théorème; dans la seconde partie, on analyse des caractéristiques de l'ordonnancement obtenu avec l'heuristique IPRTT; dans la troisième pertie, on cherche une fonction de majoration par rapport à des paramètres; dans la quatrième partie, on maximise cette fonction en choisissant des paramètres.

Première Partie:

Pour simplifier les notations, on note dans la suite de l'article le "délai corrigé" de la tâche i par d_i et on ne travaille plus qu'avec des "délais corrigés".

Lemme B.1:

Considérons un problème P et un problème P' identique à P, sauf en ce qui concerne les dates d'arrivées des tâches qui sont inférieures:
$\forall i,\ r'_i \leq r_i$
alors on a:
$T(P',O^*_{P'}) \leq T(P,O^*_P)$

Démonstration:

Tout ordonnancement O réalisable pour P est également réalisable, il fournit la même somme des retards pour P et pour P'. C'est-à-dire $T(P',O) \leq T(P,O)$, en particulier $T(P',O^*_P)=T(P,O^*_P)$.

Par ailleurs, on a: $T(P',O^*_{P'}) \leq T(P',O^*_P)$.

Ce qui implique $T(P',O^*_{P'}) \leq T(P,O^*_P)$.

Q.E.D.

Lemme B.2:

Considérons un problème P et un problème P' identique à P, sauf en ce qui concerne les délais qui sont supérieurs:
$\forall i,\ d'_i \geq d_i$
alors on a: $T(P',O^*_{P'}) \leq T(P,O^*_P)$

Démonstration:

Les problèmes P et P' ont le même ensemble de solutions réalisables. Tout ordonnancement réalisable O a une somme des retards pour P' inférieure ou égale à celle obtenue pour P ($T(P',O) \leq T(P,O)$), car si l'on augmente le délai d'une tâche, son retard ne peut que diminuer. On en déduit que: $T(P,O_P^*) \geq T(P',O_P^*) \geq T(P',O_{P'}^*)$.

Q.E.D.

Lemme B.3:

Etant donné un problème P et un ordonnancement réalisable O pour P, nous considérons le problème P' obtenu en conservant les dates d'arrivée et les durées des tâches de P et en modifiant les délais des tâches de la manière suivante:

$\forall i,\ i \in E,\ d'_i = d_i$ (B.1)

$\forall i,\ i \notin E,\ d'_i \leq d_i$ (B.2)

où $E=\{i/C_i<d_i\}$: c'est l'ensemble des tâches strictement en avance dans O.

Alors on a:

$T(P,O)-T(P,O_P^*) \leq T(P',O)-T(P',O_{P'}^*)$

Démonstration:

$$T(P,O)-T(P',O)=\sum_{i=1}^{n}\max(t_i+p_i-d_i,0)-\sum_{i=1}^{n}\max(t_i+p_i-d'_i,0)$$

$$=\sum_{i\notin E}[t_i+p_i-d_i-(t_i+p_i-d'_i)]+\sum_{i\in E}0$$

$$=\sum_{i\notin E}(d'_i-d_i)+\sum_{i\in E}(d'_i-d_i)$$

$$=\sum_{i=1}^{n}(d'_i-d_i) \qquad \text{(B.3)}$$

Par ailleurs, on a:

$T(P',O_P^*) \geq T(P',O_{P'}^*)$ (B.4)

$$T(P',O^*_P)-T(P,O^*_P)=\sum_{i=1}^{n}\max(t^*_i+p_i-d'_i,0)-\sum_{i=1}^{n}\max(t^*_i+p_i-d_i,0)$$

$$=\sum_{i=1}^{n}[\max(t^*_i+p_i,d'_i)-d'_i]-\sum_{i=1}^{n}[\max(t^*_i+p_i,d_i)-d_i]$$

$$=\sum_{i=1}^{n}[\max(t^*_i+p_i,d'_i)-\max(t^*_i+p_i,d_i)]+\sum_{i=1}^{n}(d_i-d'_i) \qquad \text{(B.5)}$$

et (B.1) et (B.2) $\Rightarrow d'_i \le d_i \Rightarrow \max(t^*_i+p_i,d'_i) \le \max(t^*_i+p_i,d_i)$ (B.6)

(B.3), (B.5) et (B.6) $\Rightarrow T(P',O^*_P)-T(P,O^*_P) \le \sum_{i=1}^{n}(d_i-d'_i)=T(P',O)-T(P,O)$ (B.7)

En combinant (B.4) et (B.7), on a:

$T(P',O^*_{P'})-T(P,O^*_P) \le T(P',O)-T(P,O) \Rightarrow T(P,O)-T(P,O^*_P) \le T(P',O)-T(P',O^*_{P'})$

Q.E.D.

Corollaire B.1:

Etant donné un problème P et un ordonnancement réalisable O pour P, considérons le problème P' obtenu en conservant les dates d'arrivée et les durées des tâches de P, et en modifiant les délais des tâches de la manière suivante:

$\forall i,\ i \in E,\quad d'_i \ge d_i$

$\forall i,\ i \notin E,\quad d'_i \le d_i$

où $E=\{i/C_i<d_i\}$ l'ensemble des tâches strictement en avance dans O, alors on a:

$T(P,O)-T(P,O^*_P) \le T(P',O)-T(P',O^*_{P'})$

Démonstration:

Construisons un problème P" en conservant les dates d'arrivée et les durées des tâches du problème P et en modifiant les délais de la manière suivante:

$\forall i,\ i \in E,\quad d''_i=d_i$

$\forall i,\ i \notin E,\quad d''_i=d'_i \le d_i$

Alors, d'après le lemme B.3, on a:

$T(P,O)-T(P,O^*_P) \le T(P'',O)-T(P'',O^*_{P''})$ (B.8)

Par ailleurs, on a: $T(P'',O)=T(P',O)$ (B.9)

et, d'après le lemme B.2, on a encore:
$T(P',O^*_{P'})\leq T(P'',O^*_{P''})$ (B.10)

En combinant (B.8), (B.9) et (B.10), on a donc:
$T(P,O)-T(P,O^*_P)\leq T(P',O)-T(P',O^*_{P'})$

Q.E.D.

Deuxième Partie:

Si dans O il y a au moins un intervalle de temps inutilisé de longueur supérieure ou égale à D, on peut toujours décomposer O en sous-ordonnancements indépendants les uns des autres, chaque sous-ordonnancement ne contenant que des intervalles de longueur strictement inférieure à D. En effet, O étant actif, toutes les tâches placées derrière un intervalle de longueur supérieure ou égale à D ne sont pas encore arrivées au début de cet intervalle. Le fait que la machine aurait pu être libérée plus tôt en ordonnançant différemment ce qui précède l'intervalle n'a donc aucune influence sur les décisions ultérieures.

On ne s'intéresse donc qu'au cas où O ne laisse la machine inutilisée que pendant des intervalles de temps de longueur strictement inférieure à D, sachant que lorsqu'il y a plusieurs sous-ordonnancements indépendants, l'erreur totale est la somme des erreurs commises pour chacun d'entre eux.

On simplifie la description en choisissant l'origine de l'axe du temps de telle sorte que:

$$\min_{i=1}^{n}(r_i)=0$$

et on suppose que les tâches sont renumérotées dans l'ordre de leur position dans O.

O est constitué de φ ($\varphi\geq 1$) sous-séquences contiguës, telles que devant chaque sous-séquence k il y a un intervalle de temps inutilisé de longueur Δ_k avec

$$\begin{cases} 0<\Delta_k<D & \forall k>1 \\ 0\leq\Delta_k<D & \text{pour } k=1 \end{cases}$$

On a aussi:

$$t_i=(i-1)D+\sum_{k=1}^{\sigma_i}\Delta_k$$

où σ_i est le numéro de la sous-séquence à laquelle appartient la tâche i.

Considérons maintenant les caractéristiques de l'ordonnancement construit à l'aide de notre heuristique.

Les itérations successives ne placent qu'une seule tâche lorsque la phase d'insertion n'a pas assez de place pour insérer des tâches "moins prioritaires". On décide de regrouper dans un ensemble noté δ les tâches obtenues avec des itérations successives qui ne placent qu'une seule tâche.

Par ailleurs, on décide de noter par δ' l'ensemble des tâches insérées par une phase d'insertion (voir la figure B.1).

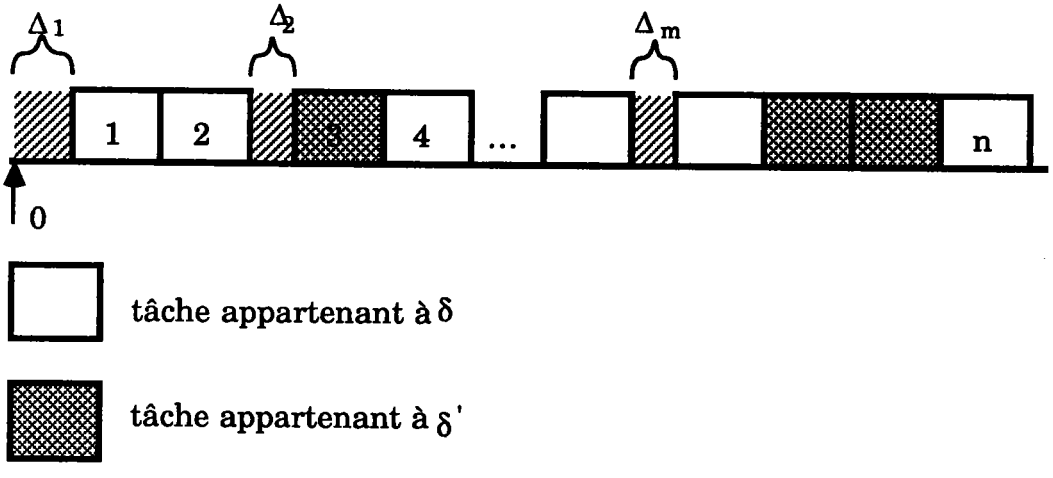


Fig. B.1. Ordonnancement obtenu avec IPRTT

Montrons que si une tâche j suit une tâche i qui est strictement en avance, alors j ne serait pas en retard en terminant à l'instant de fin de la tâche i. Pour cela, on considère les cas où la tâche i est dans δ puis dans δ':

$\forall i,j$, $i\in\delta$ et $j>i$, on a $PRTT(i,t_i)\leq PRTT(j,t_i)$, ce qui implique selon les deux cas possibles a) et b):

$$\begin{cases} a) & t_i \geq r_j \Rightarrow \max(C_i,d_j) \geq \max(C_i,d_i) \\ b) & t_i < r_j \Rightarrow d_j > C_i \end{cases}$$

et $\forall i,j$, $i \in \delta'$ et $j>i$, la règle d'insertion implique selon les deux cas possibles a) et b):

$$\begin{cases} a) & t_i \geq r_j \Rightarrow d_j \geq C_i \\ b) & t_i < r_j \Rightarrow d_j > C_i \end{cases}$$

on a donc:

$\forall i,j$, $j>i$, si $d_i > C_i$ alors $d_j \geq C_i$ (B.11)

Nous cherchons maintenant une formule minorant le délai de toute tâche numérotée i.

Soit h_k la première tâche de la sous-séquence k, alors:

$$\begin{cases} \forall i,\ \sigma_{h_{\sigma_i}} = \sigma_i,\ h_{\sigma_i+1} > i \geq h_{\sigma_i} \geq \sigma_i \\ \forall i,\ 1 \leq i \leq n-1,\ \sigma_i \leq \sigma_{i+1} \leq \sigma_i + 1 \end{cases} \quad \text{(B.12)}$$

Pour tout i, on a encore (car on travaille sur les délais corrigés: $d_i \geq r_i + D$)

$$\begin{cases} \text{si } i = h_{\sigma_i},\ r_i = t_i = (i-1)D + \sum_{k=1}^{\sigma_i} \Delta_k \Rightarrow d_i \geq r_i + D = h_{\sigma_i} D + \sum_{k=1}^{\sigma_i} \Delta_k \\ \text{si } i > h_{\sigma_i}, \begin{cases} \text{si } r_i \geq r_{h_{\sigma_i}} \Rightarrow d_i \geq r_i + D \geq r_{h_{\sigma_i}} + D = h_{\sigma_i} D + \sum_{k=1}^{\sigma_i} \Delta_k \\ \text{si } r_i < r_{h_{\sigma_i}},\ d_i > d_{h_{\sigma_i}} \geq h_{\sigma_i} D + \sum_{k=1}^{\sigma_i} \Delta_k \quad (1) \end{cases} \end{cases}$$

1: Dans ce cas, on a:

$\text{PRIOR}(h_{\sigma_i}, t_{h_{\sigma_i}} - \Delta_{\sigma_i}) < \text{PRIOR}(i, t_{h_{\sigma_i}} - \Delta_{\sigma_i})$

$\Rightarrow r_{h_{\sigma_i}} + d_{h_{\sigma_i}} < \max(r_i, t_{h_{\sigma_i}} - \Delta_{\sigma_i}) + \max[\max(r_i, t_{h_{\sigma_i}} - \Delta_{\sigma_i}) + D, d_i]$

et par ailleurs, $\max(r_i, t_{h_{\sigma_i}} - \Delta_{\sigma_i}) < r_{h_{\sigma_i}}$

Il y a obligatoirement: $d_i > d_{h_{\sigma_i}}$

c'est-à-dire une formule de minoration du délai de la i^{ème} tâche:

$$\forall i,\quad d_i \geq h_{\sigma_i} D + \sum_{k=1}^{\sigma_i} \Delta_k \qquad \text{(B.13)}$$

Troisième Partie:

Si B désigne la somme de tous les intervalles, c'est-à-dire: $B=\sum_{k=1}^{\varphi} \Delta_k$, et si q est le quotient de la division entière de B par D, nous notons λ la valeur de q augmentée de 1, on a:

$(\lambda-1)D \leq B < \lambda D$

où λ est un entier strictement positif. Il est évident que $\lambda \leq \varphi$, car sinon, il y aurait au moins un intervalle de temps de longueur supérieure ou égale à D.

Nous cherchons une majoration de l'erreur commise en fonction de λ.

Soit P' le problème obtenu à partir du problème P en conservant les dates d'arrivée et les durées des tâches mais en modifiant les délais de la manière suivante:

$$\forall i,\quad d'_i = d_i + \min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k \qquad (d'_i \geq d_i) \qquad \text{(B.14)}$$

alors le lemme 2 nous permet de déduire que:

$$T(P', O^*_{P'}) \leq T^* \qquad \text{(B.15)}$$

On construit un nouvel ordonnancement O' (Fig. B.2.b) en décalant vers la droite chaque tâche i de O de la durée $\min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k$. Ceci conduit à mettre λ intervalles de longueur D devant les λ premières tâches, les tâches restantes étant alors toutes consécutives.

On a pour tout i:

$T_i(P',O') = \max(C'_i - d'_i, 0) = \max[(\min(i,\lambda)+i)D - d'_i, 0]$

$$=\max\{[\min(i,\lambda)+i]D-[\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k+d_i],0\}$$

$$=\max(iD+\sum_{k=1}^{\sigma_i}\Delta_k-d_i,0)$$

$$=T_i$$

et on a donc:

$$T(P',O')=T \tag{B.16}$$

avec (B.15) et (B.16) on a donc:

$$T-T^*\leq T(P',O')-T(P',O^*_{P'}) \tag{B.17}$$

Il suffit maintenant de majorer $T(P',O')-T(P',O^*_{P'})$.

Soit $E'=\{i/C'_i<d'_i\}\cup\{0\}=\{0,e_1,e_2,\ldots,e_\omega\}$, où e_τ est le numéro de la τ-ième tâche strictement en avance par rapport à son délai dans O' pour le problème P'.

Soit $\tau_i=\text{card}\{l/0<l\leq i \text{ et } l\in E'\}$ le nombre de tâches strictement en avance situées avant la tâche i+1 dans O'. Alors e_{τ_i} est la tâche strictement en avance la plus proche de i et on a:

$$\begin{cases}\forall i,\ e_{\tau_i+1}>i\geq e_{\tau_i}\geq\tau_i,\ \tau_{e_{\tau_i}}=\tau_i\\ \forall i,\ 1\leq i\leq n-1,\ \tau_i\leq\tau_{i+1}\leq\tau_i+1\end{cases} \tag{B.18}$$

Par convention, on pose: $e_0=0$.

On a:

$$\forall i,\quad i\in E' \Rightarrow d'_i>C'_i$$

$$\Rightarrow d_i+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k>[\min(i,\lambda)+i]D$$

$$\Rightarrow d_i>iD+\sum_{k=1}^{\sigma_i}\Delta_k=C_i$$

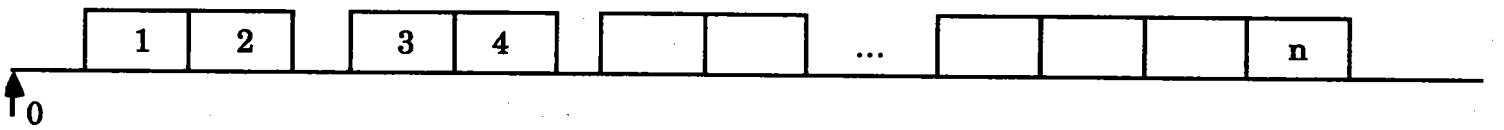


Fig. B.2.a. Ordonnancement O

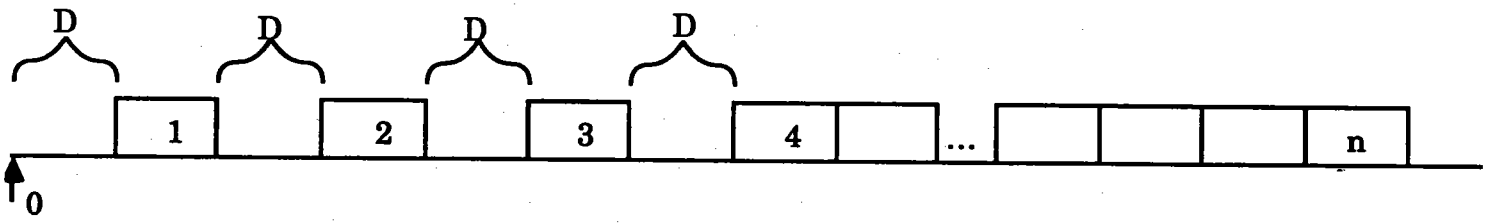


Fig. B.2.b. Ordonnancement O'

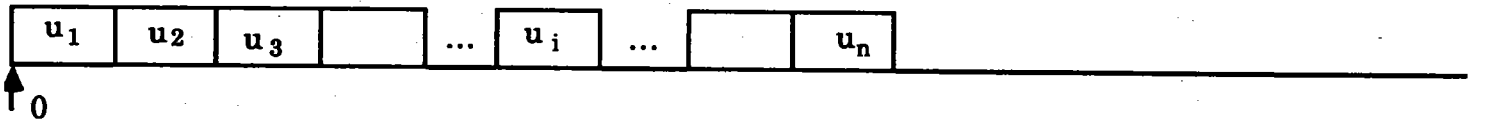


Fig. B.2.c. Ordonnancement O''^{m} avec $d_{u_1} \leq d_{u2} \leq \ldots \leq d_{u_n}$

En particulier:

$$\forall i, \quad e_{\tau_i} \in E' \Rightarrow d_{e_{\tau_i}} > C_{e_{\tau_i}} \tag{B.19}$$

On déduit de (B.18), (B.19) et (B.11):

$$\forall i,\ d_i \geq c_{e_{\tau_i}} = e_{\tau_i} D + \sum_{k=1}^{\sigma_{e_{\tau_i}}} \Delta_k$$

formule que l'on utilise pour minorer (B.14):

$$\begin{aligned} \forall i,\ d'_i &= d_i + \min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k \\ &\geq e_{\tau_i} D + \sum_{k=1}^{\sigma_{e_{\tau_i}}} \Delta_k + \min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k \\ &= e_{\tau_i} D + \min(i,\lambda)D - \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k \end{aligned} \tag{B.20}$$

Par ailleurs (B.13) et (B.14) impliquent:

$$\begin{aligned} d'_i &= d_i + \min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k \\ &\geq h_{\sigma_i} D + \sum_{k=1}^{\sigma_i} \Delta_k + \min(i,\lambda)D - \sum_{k=1}^{\sigma_i} \Delta_k \\ &= [h_{\sigma_i} + \min(i,\lambda)]D \end{aligned} \tag{B.21}$$

Avec (B.20) et (B.21) on a donc:

$$d'_i \geq \max\left(h_{\sigma_i} D, e_{\tau_i} D - \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k\right) + \min(i,\lambda)D$$

Nous cherchons maintenant à simplifier cette minoration en distinguant deux cas:

$$1)\ \sigma_{e_{\tau_i}} = \sigma_i \Rightarrow e_{\tau_i} \geq h_{\sigma_i} \text{ et } \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k = 0$$

$$\Rightarrow d'_i \geq [e_{\tau_i} + \min(i,\lambda)]D$$

$$2)\ \sigma_{e_{\tau_i}}<\sigma_i \Rightarrow e_{\tau_i}<h_{\sigma_i} \text{ et } \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k>0$$

$$\Rightarrow e_{\tau_i}D- \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k<h_{\sigma_i}D$$

$$\Rightarrow d'_i\geq[h_{\sigma_i}+\min(i,\lambda)]D$$

Dans tous les cas, on a:

$$\forall i,\ d'_i\geq[\max(h_{\sigma_i},e_{\tau_i})+\min(i,\lambda)]D \qquad \text{(B.22)}$$

On construit alors un problème P" à partir du problème P' en conservant les dates d'arrivée et les durées des tâches et en modifiant les délais de la manière suivante:

$\forall i,\ i\in E',\quad d''_i=+\infty>d'_i$

$\forall i,\ i\notin E',\quad d''_i=[\max(h_{\sigma_i},e_{\tau_i})+\min(i,\lambda)]D \quad (\leq d'_i \text{ d'après (B.22)})$

Alors en utilisant le corollaire B.1 et (B.17), on obtient:

$$T-T^*\leq T(P',O')-T(P',O^*_{P'})\leq T(P'',O')-T(P'',O^*_{P''}) \qquad \text{(B.23)}$$

On construit un problème $P^{(3)}$ à partir de P" en conservant les durées et les délais des tâches, mais on diminue les dates d'arrivée des tâches à 0, d'après le lemme B.1, on a:

$$T(P'',O^*_{P''})\leq T(P^{(3)};O^*_{P^{(3)}}) \qquad \text{(B.24)}$$

avec (B.23) et (B.24) on a donc:

$$T-T^*\leq T(P'',O')-T(P'',O^*_{P''})\leq T(P'',O')-T(P^{(3)};O^*_{P^{(3)}}) \qquad \text{(B.25)}$$

Pour $P^{(3)}$, on peut facilement construire un ordonnancement optimal $O^*_{P^{(3)}}$ (voir la figure B.2.c). Il suffit pour cela de mettre les tâches dans l'ordre EDD (Earliest Due Date). On place donc les tâches n'appartenant pas à E' dans l'ordre de leur numérotation, les unes derrière les autres, sans laisser la machine inoccupée, et en plaçant les tâches de E' immédiatement derrière dans n'importe quel ordre puisqu'elles ne sont jamais en retard.

En fait, cet ordonnancement vérifie EDD, parce que:
$\forall i,j,\ i\notin E',\ j\notin E',\ j>i$:

$\min(j,\lambda)\geq\min(i,\lambda)$ (B.26)

et (B.12) et (B.18) $\Rightarrow e_{\tau_j}\geq e_{\tau_i}$ et $h_{\sigma_j}\geq h_{\sigma_i}$

$\Rightarrow \max(h_{\sigma_j},e_{\tau_j})\geq\max(h_{\sigma_i},e_{\tau_i})$ (B.27)

Avec (B.26) et (B.27), on a:
$\forall i,j,\ i\notin E',\ j\notin E',\ j>i$,
$\max(h_{\sigma_j},e_{\tau_j})+\min(j,\lambda)\geq\max(h_{\sigma_i},e_{\tau_i})+\min(i,\lambda) \Rightarrow d''_j\geq d''_i$

Pour finir on majore $T(P'',O')-T(P^{(3)};O^*_{P^{(3)}})$:

$\forall i,\ C'_i=[i+\min(\lambda,i)]D$

$$T_i(P'',O')=\begin{cases}0 \text{ si } i\in E'\\ C'_i-d''_i=[i-\max(h_{\sigma_i},e_{\tau_i})]D \text{ si } i\notin E'\end{cases}$$

et $\forall i,\ i\notin E'\ C^{(3)*}_i=[i-\tau_i]D$

On a donc:

$$\forall i,\ T(P^{(3)};O^*_{P^{(3)}})=\begin{cases}0 \text{ si } i\in E'\\ \max(C^{(3)*}_i-d''_i,0)=\max[i-\tau_i-\min(\lambda,i)-\max(h_{\sigma_i},e_{\tau_i}),0]D \text{ si } i\notin E'\end{cases}$$

on a:
$\forall i,\ i\notin E'$,

$$\begin{aligned}T_i(P'',O')-T_i(P^{(3)};O^*_{P^{(3)}})&=\{i-\max(h_{\sigma_i},e_{\tau_i})-\max[i-\tau_i,\min(i,\lambda)-\max(h_{\sigma_i},e_{\tau_i}),0]\}D\\ &=-\max[-\tau_i-\min(i,\lambda),\max(h_{\sigma_i},e_{\tau_i})-i]D\\ &=\min[\tau_i+\min(i,\lambda),i-\max(h_{\sigma_i},e_{\tau_i})]D\\ &=\min[\tau_i+\min(i,\lambda),-\max(h_{\sigma_i}-i,e_{\tau_i}-i)]D\\ &=\min[\tau_i+\min(i,\lambda),\min(i-h_{\sigma_i},i-e_{\tau_i})]D\\ &=\min[\min(i+\tau_i,\lambda+\tau_i),\min(i-h_{\sigma_i},i-e_{\tau_i})]D\\ &=\min(i+\tau_i,\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D\\ &=\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D \ ^{(2)}\end{aligned}$$ (B.28)

2: En fait, on a toujours $\tau_i\geq 0$, $i-h_{\sigma_i}\leq i$ et $i-e_{\tau_i}\leq i$, on a donc: $\min(i-h_{\sigma_i},i-e_{\tau_i})\leq i+\tau_i$

et $\forall i,\ i\in E',\ T_i(P'',O')-T_i(P^{(3)};O^{*}_{P(3)})=0=\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D$ [3] (B.29)

De (B.28) et (B.29) on déduit:

$$T(P'',O')-T(P^{(3)};O^{*}_{P(3)})=\sum_{i=1}^{n}[T_i(P'',O')-T_i(P'',O''^{m})]$$

$$=\sum_{i=1}^{n}\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D \qquad (B.30)$$

Il reste donc à chercher la borne supérieure de $\sum_{i=1}^{n}\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D$.

Quatrième Partie:

On doit résoudre le problème de maximisation suivant:

$$\text{Max}\sum_{i=1}^{n}\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})$$

sous les contraintes:

$$\begin{cases} h_{\sigma_1}=\sigma_1=1,\ \sigma_n=m\geq\lambda,\ \tau_n=\omega \\ \forall i,\ \sigma_{h_{\sigma_i}}=\sigma_i,\ \tau_{e_{\tau_i}}=\tau_i \\ \forall i,\ h_{\sigma_i+1}>i\geq h_{\sigma_i}\geq\sigma_i,\ e_{\tau_i+1}>i\geq e_{\tau_i}\geq\tau_i \\ \forall i,\ 1\leq i\leq n-1,\ \sigma_i\leq\sigma_{i+1}\leq\sigma_i+1,\ \tau_i\leq\tau_{i+1}\leq\tau_i+1 \end{cases}$$

Ce problème a une solution unique qui est:

$$\begin{cases} \forall i,\ i\leq\lambda,\ \sigma_i=h_{\sigma_i}=i \\ \forall i,\ i>\lambda,\ \sigma_i=h_{\sigma_i}=\lambda \\ \forall i,\ i\leq\omega,\ \tau_i=e_{\tau_i}=i \\ \forall i,\ i>\omega,\ \tau_i=e_{\tau_i}=\omega \end{cases}$$

---

[3]: Pour tout $i\in E'$, $e_{\tau_i}=i$, $\lambda+\tau_i\geq\lambda>0$ et $i\geq h_{\sigma_i}$, on a donc: $\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D=0$

Avec (B.25) et (B.30) on a donc:

$$T\text{-}T^* \le \sum_{i=1}^{n} \min(\lambda+\tau_i, i-h_{\sigma_i}, i-e_{\tau_i})D$$

$$\le \sum_{i=1}^{\max(\lambda,\omega)} 0 + \sum_{i=\max(\lambda,\omega)+1}^{n} \min(\lambda+\omega, i-\lambda, i-\omega)D$$

$$= \sum_{i=\max(\lambda,\omega)+1}^{n} \min(\lambda+\omega, i-\lambda, i-\omega)D$$

$$= \sum_{i=\max(\lambda,\omega)+1}^{n} \min[\lambda+\omega, i-\max(\lambda,\omega)]D$$

$$= \sum_{i=1}^{n-\max(\lambda,\omega)} \min(\lambda+\omega, i)D$$

$$= \sum_{i=1}^{n-\max(\lambda,\omega)} \min[\max(\lambda,\omega)+\min(\lambda,\omega), i]D$$

Pour faciliter le calcul, on pose:
$n1=\min(\lambda,\omega)$ et $n2=\max(\lambda,\omega)$

Alors on a:
$1 \le n1 \le n2 \le n$

et $T\text{-}T^* \le \sum_{i=1}^{n-n2} \min(n1+n2, i)D$

Et on doit chercher la valeur maximale de la fonction

$$f(n1,n2) = \sum_{i=1}^{n-n2} \min(n1+n2, i)$$

en examinant les deux cas suivants.

A) $n-n2 \le n1+n2$

Alors:

$$f(n1,n2) = \sum_{i=1}^{n-n2} i = \frac{(n-n2+1)(n-n2)}{2}$$

On a en outre: $n-2n2 \le n1 \le n2 \Rightarrow n2 \ge \frac{n}{3}$,

$$\text{d'où } f(n1,n2) \leq \frac{(n-\frac{n}{3}+1)(n-\frac{n}{3})}{2}$$
$$= \frac{2n^2+3n}{9}$$
$$= \frac{32n^2+48n}{9 \cdot 16}$$
$$= \frac{(4n+6) \cdot 8n}{9 \cdot 16}$$
$$= \frac{[(6n+3)-(2n-3)][(6n+3)+(2n-3)]}{9 \cdot 16}$$
$$= \frac{(6n+3)^2-(2n-3)^2}{9 \cdot 16}$$
$$\leq \frac{(6n+3)^2}{9 \cdot 16}$$
$$= \frac{(2n+1)^2}{16}$$

B) n-n2>n1+n2

Alors:

$$f(n1,n2) = \sum_{i=1}^{n1+n2} i + \sum_{i=n1+n2+1}^{n-n2} (n1+n2)$$
$$= \frac{(n1+n2)(n1+n2+1)}{2} + [n-(n1+2n2)](n1+n2)$$
$$= \frac{(n1+n2)(2n+1-n1-3n2)}{2}$$
$$= \frac{(4n1+4n2)[2(2n+1)-2n1-6n2]}{16}$$
$$= \frac{[(2n+1+n1-n2)-(2n+1-3n1-5n2)][(2n+1+n1-n2)+(2n+1-3n1-5n2)]}{16}$$
$$= \frac{[2n+1+n1-n2]^2 - [2n+1-3n1-5n2]^2}{16}$$
$$\leq \frac{[2n+1+n1-n2]^2}{16}$$
$$\leq \frac{(2n+1)^2}{16}$$

Dans tous les cas, on a:

$$T-T^* \leq f(n1,n2)D \leq \frac{(2n+1)^2}{16}D$$

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

Résumé de la thèse

L'objectif de la thèse a été de construire une structure générale de mémoire artificielle et de l'évaluer sur de nombreux problèmes d'ordonnancement. En cherchant des algorithmes approchés efficaces pour la mémoire artificielle relative à ces problèmes, il a été trouvé des résultats analytiques nouveaux sur les problèmes à une machine que nous présentons au chapitre 3.

Le chapitre 1, essentiellement bibliographique, est une introduction générale aux problèmes d'ordonnancement et à leur résolution.

La première partie du chapitre 2 est consacrée aux concepts et aux outils qui permettent de construire une structure générale de mémoire artificielle.

La deuxième partie du chapitre 2 montre comment construire les modules spécifiques propres aux problèmes considérés dans cette thèse:
- ordonnancement d'atelier de type job-shop pour différents critères,
- ordonnancement d'atelier à une machine avec dates d'arrivée des tâches non identiques pour différents critères,
- ordonnancement du placement de boîtes parallalélépipédiques dans un emballage parallalélépipédique (problème de "container loading").

Le chapitre 3 est consacré aux problèmes à une machine, et en particulier aux résultats analytiques nouveaux trouvés (problèmes $n/1/r_i \geq 0/\sum F_i$ et $n/1/r_i \geq 0/\sum T_i$ essentiellement).

Le chapitre 4 est consacré à une première évaluation du concept de mémoire artificielle grâce à de nombreuses expériences réalisées pour les trois applications.

Mots clés

Algorithmes Exacts et Approchés - Ordonnancement d'Atelier - Séquencement - Propriétés de Dominance - "Container Loading" - Analyse des Données - Mémoire Artificielle

I S B N - 2 - 7261 - 0653 - 6