| | |
|---|---|
| **Titre:** Title: | Problèmes d'ordonnancement sur une machine |
| **Auteur:** Author: | Louis-Philippe Bigras |
| **Date:** | 2006 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Bigras, L.-P. (2006). Problèmes d'ordonnancement sur une machine [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7748/ |

UNIVERSITÉ DE MONTRÉAL

# PROBLÈMES D'ORDONNANCEMENT SUR UNE MACHINE

LOUIS-PHILIPPE BIGRAS
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR (Ph.D.)
(MATHÉMATIQUES DE L'INGÉNIEUR)
MAI 2006

*Your file Votre référence*
*ISBN: 978-0-494-17975-8*
*Our file Notre référence*
*ISBN: 978-0-494-17975-8*

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

PROBLÈMES D'ORDONNANCEMENT SUR UNE MACHINE

présentée par : BIGRAS Louis-Philippe
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. SOUMIS François, Ph.D., président
M. SAVARD Gilles, Ph.D., membre et directeur de recherche
M. GAMACHE Michel, Ph.D., membre et codirecteur de recherche
M. BAPTISTE Pierre, Doctorat, membre
M. VILLENEUVE Daniel, Ph.D.., membre externe

*À mes parents...*

# Remerciements

Je tiens à d'abord remercier Michel Gamache qui est celui qui m'a fait découvert la recherche opérationnelle. Je remercie également Gilles Savard, qui m'a fourni un cadre de travail idéal lors de ces quatre dernières années. Merci à tous mes collègues, dont Éric Rancourt et Anne Mercier, pour les échanges constructifs sur nos problèmes respectifs, Francois Lessard pour son aide dans l'utilisation de la librairie Netgen et tous les joueurs de tarot du Gerad pour leur amour du jeu.

Enfin, un merci spécial à toute ma famille (en particulier ma grand-mère) pour leur encouragement durant toutes ces années d'étude.

# Résumé

Dans le problème générique d'ordonnancement sur une machine, on s'intéresse à la planification d'activités, appelées *pièces*, sur un processeur, appelé *machine*, ne pouvant traiter qu'une activité à la fois. Il s'agit d'un problème d'optimisation *NP-difficile* dans le cas général, pour lequel le développement de bornes serrées constitue un défi. Dans cette thèse, on étudie quelques variantes du problème d'ordonnancement sur une machine et on propose pour celles-ci des approches de résolution exactes basées sur des techniques classiques de programmation en nombres entiers.

Il existe plusieurs façons de concevoir le problème d'ordonnancement sur une machine mais on a choisi de l'aborder avec la perspective du problème de stable de poids maximum. En effet, de nombreuses variantes du problème (avec fenêtres de temps, avec temps de réglage dépendants de la séquence, avec objectifs non-réguliers, etc.) peuvent être modélisées comme des problèmes de stable de poids maximum. De plus, les formulations de programmation en nombres entiers sous cette forme offrent souvent de bonnes bornes de programmation linéaire pouvant être exploitées efficacement à l'intérieur d'une procédure d'énumération implicite.

Pour le cas avec temps de réglage indépendants de la séquence, on obtient avec l'approche de stable une formulation classique du problème, dite de type temps indexé. Cette formulation est reconnue pour offrir des relaxations linéaires très serrées. En dépit de cela, peu d'approches de résolution exactes ont jusqu'à maintenant exploité la formulation de type temps indexé car il peut être très difficile de calculer sa relaxation linéaire en raison de sa grande densité. On présente dans ce document une reformulation du modèle de type temps indexé, obtenue avec le principe de décomposition de Dantzig-Wolfe (1960) et un principe de décomposition temporelle, dont la relaxation

linéaire est nettement plus facile à calculer. En imbriquant cette nouvelle borne de programmation linéaire dans une procédure d'énumération implicite, on obtient un algorithme très efficace pour le problème de minimisation de la somme pondérée des retards sur une machine. Avec cette approche, nous sommes en mesure de résoudre de nombreux exemplaires ouverts de *OR-LIBRARY*.

Pour le cas avec temps de réglage dépendants de la séquence, on obtient avec l'approche de stable une formulation du problème dite de commis-voyageur dépendant du temps. Cette formulation permet de modéliser de nombreux problèmes d'ordonnancement sur une machine avec temps de réglage dépendants de la séquence (dont un de ses célèbres cas particuliers, le problème de commis-voyageur). Cependant, en plus de compter un grand nombre de variables et de contraintes, cette formulation offre parfois des relaxations linéaires trop lâches pour être exploitées efficacement à l'intérieur d'un algorithme d'évaluation et de séparation progressive. Afin d'obtenir de meilleures bornes, on propose diverses reformulations du modèle de commis-voyageur dépendant du temps, en ayant recours au principe de décomposition de Dantzig-Wolfe et aux techniques de coupes de programmation en nombres entiers. En se basant sur ces formulations, on développe des approches de résolution efficaces pour deux problèmes d'ordonnancement difficiles (minimisation de la somme des temps de complétion et minimisation de la somme des retards dans un environnement avec temps de réglage dépendants de la séquence). On doit noter qu'avec l'approche de résolution proposée pour le problème de minimisation des retards, on résoud pour la première fois plusieurs exemplaires proposés par Rubin et Ragatz (1995).

On considère finalement un problème d'horaires très pratique retrouvé au sein des Forces Armées Canadiennes. Ce problème consiste à affecter simultanément du personnel et des véhicules à des missions et peut être vu comme un problème d'ordonnancement sur machines parallèles avec contraintes de ressources humaines. En modélisant ce problème sous la forme d'un stable, on obtient une formulation de programmation en nombres entiers qui permet d'accomoder la plupart des contraintes

humaines et matérielles recensées et qui est assez facile à résoudre pour des scénarios de taille vraisemblable.

# Abstract

The generic single machine scheduling problem involves sequencing of activities, called *jobs*, on a single processor, called *machine*, able to handle only one activity at anytime. In the general case, this is a *NP-hard* optimization problem for which the development of tight bounds is challenging. In this thesis, we study different versions of this scheduling problem and we present some exact solution approaches based on classical integer programming techniques.

The single machine scheduling problem can be approached with different points of view but we choose to address it as a node packing problem. In fact, several versions of the problem (with time-windows, with sequence-dependent setup times, with non-regular objectives, etc.) can be modeled as a node packing problem. Moreover, node packing integer programming formulations often offer good linear programming relaxations that can be efficiently exploited inside a branch and bound scheme.

For the sequence-independent setup times case, we obtain with the node packing approach a classical model, coined as the time-indexed formulation in the literature. This formulation is known to give a very good linear programming relaxation. Despite the quality of its LP relaxation, the use of such formulation inside a branch-and-bound scheme is not common. Indeed, even for small instances, solving the LP relaxation can be difficult due to the large number of variables and constraints required. To alleviate this difficulty, we propose a reformulation of the time-indexed model, using the Dantzig-Wolfe (1960) and a temporal decomposition scheme, for which it is much easier to compute the linear relaxation. This new bounding scheme is implemented inside an implicit enumeration algorithm. When using this algorithm, several open instances proposed by Potts and Van Wassenhove (1985) for the total weighted tardiness problem are solved to optimality.

For the sequence-dependent setup times case, we obtain with the node packing approach a model coined as the time-dependent traveling salesman formulation. This formulation can be used to model several single machine scheduling problem with sequence-dependent setup time. This formulation has however a main drawback : its LP relaxation sometimes gives a poor lower bound. In order to obtain stronger bounds, we propose several reformulations of the time-dependent traveling salesman model, using the Dantzig-Wolfe decomposition principle and some cutting planes of the integer programming theory. Based on these new formulations, we develop efficient solution approaches for two difficult single machine scheduling problems (minimizing total completion time and minimizing total tardiness in a sequence-dependent setup times environment). With these new bounds, several open instances proposed by Rubin and Ragatz (1995) for the total tardiness problem are solved to optimality.

We finally consider a practical scheduling problem found in the Canadian Forces. This problem consists essentially in assigning simultaneously crews and vehicles to a set of missions and can be viewed as a parallel machines scheduling problem with labor constraints. When using the node packing approach, we obtain an integer programming formulation that captures the main aspects of the problem and that is quite easy to solve for real-life instances.

# Table des matières

# Liste des tableaux

# Liste des figures

# Liste des annexes

# Liste des algorithmes

# Introduction

Les problèmes d'ordonnancement sur une machine consistent à organiser le traitement de $n$ pièces sur cette dernière sous divers modes d'opération. Il existe toute une panoplie de mesures (minimiser l'heure de complétion de la journée de travail, minimiser les retards, minimiser une fonction de poids) et de contraintes supplémentaires (fenêtre de temps sur l'heure à laquelle on peut traiter une pièce, heure de livraison, préséance entre pièces, avec ou sans interruption du traitement, temps de réglage dépendant de la séquence) qui permettent de définir divers problèmes d'ordonnancement sur une machine dont les difficultés sont variables. Par exemple, le problème de minimisation de la somme pondérée des temps de complétion sur une machine se résoud en temps polynomial. Si on considère le même objectif, mais si on ajoute des contraintes sur les dates de début de traitement des pièces, on obtient un problème *NP-difficile* au sens fort.

Une légère modification peut donc suffire à donner (ou à détruire) pour un problème d'ordonnancement une structure exploitable algorithmiquement. Pour cette raison, la littérature des problèmes d'ordonnancement regorge d'algorithmes spécialisés pour des versions de problème bien précises.

On peut cependant construire des algorithmes plus génériques pour ces problèmes en ayant recours aux méthodes de la programmation en nombres entiers. Puisqu'il existe de nombreuses façons de concevoir le problème d'ordonnancement sur une machine (par exemple comme un problème de programmation disjonctive, Balas 1985 ; comme un problème de stable de poids maximum, Waterer *et al.* 2002 ; comme un problème de commis voyageur dépendant du temps, Picard et Queyranne 1978), il existe de nombreuses formulations possibles. Cette thèse est surtout consacrée aux formulations

construites autour du problème de stable de poids maximum et au cours des prochains chapitres, on montre comment ces formulations peuvent être utilisées pour résoudre divers problèmes d'ordonnancement sur une machine.

Dans le chapitre 1, on définit plus formellement le problème général d'ordonnancement sur une machine. On présente d'abord la notation de Graham *et al.* (1979) avec les objectifs et les contraintes supplémentaires classiques pour ensuite formuler divers problèmes d'ordonnancement sur une machine et caractériser leur complexité. Des équivalences entre certains problèmes d'ordonnancement sur une machine sont aussi données.

Le chapitre 2 porte sur le problème de stable de poids maximum et ses applications. On présente d'abord certains résultats concernant le problème général de stable de poids maximum. On montre ensuite comment le problème d'ordonnancement sur machines parallèles et le problème de commis-voyageur dépendant du temps peuvent être vus comme des problèmes de stable de poids maximum. En étudiant ces deux problèmes sous cette perspective, on développe des formulations de programmation en nombres entiers autour desquelles seront construites les approches de résolution présentées aux chapitres 3, 4 et 5.

Le chapitre 3 est consacré aux problèmes d'ordonnancement sur une machine où les temps de traitement pour les pièces sont indépendants de la séquence. Dans ce chapitre, on présente divers modèles de programmation 0-1 basés sur l'analogie avec le problème du stable et on développe une approche de résolution très efficace conçue autour de ces modèles pour le problème de minimisation des retards non-négatifs qui est compétitive avec les meilleures approches exactes existantes.

Le chapitre 4 est consacré à des problèmes d'ordonnancement sur une machine moins classiques où les temps de réglage des pièces sont maintenant dépendants de la séquence (c.-à-d. que la pièce $j$ nécessite un traitement de durée $s_{ij} + p_j$ où $i$ est la

pièce effectuée juste avant $j$ et $s_{ij}$ est le temps de réglage). On montre dans ce chapitre comment cette famille de problèmes partage des similitudes avec le problème du commis voyageur dépendant du temps. En exploitant ces liens, on présente diverses formulations de programmation en nombres entiers pouvant modéliser certains problèmes d'ordonnancement sur une machine avec temps de réglage dépendants de la séquence, comme minimiser les retards non-négatifs dans un environnement où les temps de réglage sont dépendants de la séquence, pour lesquels il existe encore très peu d'approches exactes.

Le chapitre 5 porte sur un problème de planification retrouvé au sein des Forces Armées Canadiennes. Le problème peut être vu comme un problème d'ordonnancement sur machines parallèles identiques avec des contraintes de ressources humaines. Dans ce chapitre, on propose une formulation basée encore une fois sur les modèles de stable de poids maximum qui intègre la plupart des contraintes humaines et matérielles que l'on a recensées et avec laquelle on arrive à obtenir de bonnes solutions en des temps raisonnables.

# CHAPITRE 1 : GÉNÉRALITÉS SUR LES PROBLÈMES D'ORDONNANCEMENT DE PRODUCTION

Les problèmes d'ordonnancement de production peuvent être vus comme des problèmes d'ordonnancement d'activités avec contraintes de ressources où l'ensemble des activités à planifier est un ensemble de pièces, qui nécessite comme ressources des machines pour des durées de traitement données. La littérature concernant les problèmes d'ordonnancement de production est très riche et les variantes de problèmes sont nombreuses. Graham *et al.* (1979) ont proposé une classification de ces problèmes selon trois champs $\alpha|\beta|\gamma$, où $\alpha$ indique l'environnement de machines considéré, $\beta$ précise la façon dont les pièces doivent être traitées, tandis que $\gamma$ caractérise la mesure de performance à minimiser.

## 1.1 Notations

Dans la notation de Graham *et al.* (1979), l'ensemble des pièces est représenté par $N = \{1, \ldots, n\}$ ($n$ étant le nombre de pièces) et l'ensemble des machines est représenté par $M = \{1, \ldots, m\}$ ($m$ étant le nombre de machines).

Pour chacune des pièces $j \in N$, on peut être amené à spécifier :

- le nombre d'opérations nécessaires pour traiter la pièce $j$ ;
- les temps de traitement requis pour $j$ sur les machines $i = 1, \ldots, m$, notés $p_{ij}$ (s'il y a une seule machine, le temps de traitement est simplement noté $p_j$) ;
- le moment $r_j$ à partir duquel la pièce $j$ peut être traitée ;
- le moment $d_j$ avant lequel on aimerait que la pièce $j$ soit complétée ;
- un poids $w_j$ indiquant l'importance relative de la pièce $j$ ;

- une fonction $f_j$ donnant le coût $f_j(t)$ encouru si la pièce $j$ est complétée au moment $t$. Si la fonction $f_j$ est non décroissante, on dit qu'elle est régulière.

Par défaut, on suppose que le nombre d'opérations par pièce est de 1, que $r_j$, $d_j$, $w_j$ sont entiers et que $r_j = 0$, $d_j = \infty$ et $w_j = 1$.

### 1.1.1 Environnement des machines

Sous la notation de Graham, le premier champ ($\alpha$) peut contenir sept types de caractères, soit $\emptyset$, $P$, $Q$, $R$, $F$, $J$, $O$.

Le caractère blanc ($\emptyset$) indique qu'il n'y a qu'une seule machine dans le problème.

Le caractère $P$ signifie que l'on est en présence d'un environnement à machines parallèles identiques, c.-à-d. que plusieurs machines sont disponibles pour traiter les pièces et que le temps de traitement pour traiter une pièce est le même sur toutes les machines.

Le caractère $Q$ signifie qu'il s'agit d'un environnement de machines parallèles uniformes, c.-à-d. que plusieurs machines sont disponibles pour traiter les pièces sauf qu'on suppose que certaines machines sont plus rapides que d'autres, et ce pour tous les traitements.

Le caractère $R$ signifie qu'il s'agit d'un problème d'ordonnancement sur machines parallèles sans relation, c.-à-d. que les durées de traitement peuvent varier d'une machine à l'autre.

Le caractère $F$ signifie qu'il s'agit d'un environnement d'atelier monogamme (en anglais, *flow shop*), c.-à-d. un environnement où chaque pièce doit subir plusieurs opérations sur $m$ machines différentes dans une séquence prédéfinie qui est la même pour toutes les pièces.

Le caractère $J$ signifie qu'il s'agit d'un environnement d'atelier multigamme (en anglais, *job shop*), c.-à-d. un environnement semblable à l'atelier monogamme mais où la séquence d'opérations à respecter peut être différente d'une pièce à l'autre.

Finalement, le caractère $O$ signifie qu'il s'agit d'un environnement d'atelier ouvert (en anglais, *open shop)*, semblable à l'environnement d'atelier monogamme, à la seule différence que les opérations pour chaque pièce peuvent maintenant être exécutées dans n'importe quel ordre.

Tous ces caractères peuvent être suivis d'un nombre, qui indique le nombre de machines considéré dans le problème.

### 1.1.2 Contraintes supplémentaires sur les pièces

Dans le deuxième champ ($\beta$), on retrouve habituellement les indications suivantes : *prec*, $r_j$, $d_j$, $p_j = 1$, $p_j = p$ et *preemp*.

La présence du mot *prec* dans le champ $\beta$ indique qu'il y a des contraintes de préséance entre les pièces.

La présence de $r_j$ signifie que les pièces sont soumises à des contraintes de date de départ tandis que la présence de $d_j$ signifie que les pièces sont soumises à des contraintes dures de date de livraison.

La présence de $p_j = p$ indique que la durée de traitement est identique pour toutes les pièces alors que $p_j = 1$ indique que l'on considère le cas particulier de $p_j = p$ avec $p$ unitaire.

Finalement, la présence de *preemp* indique qu'on permet la préemption dans le problème, c'est-à-dire qu'on tolère que le traitement d'une même pièce soit réparti sur plusieurs intervalles de temps non contigus.

La notation originale de Graham *et al.* (1979) ne prévoit pas le cas où les temps de réglage peuvent dépendre de la séquence. Pour cette raison, nous proposons d'étendre le champ $\beta$ et d'utiliser le mot clé $s_{ij}$ afin de spécifier que les temps de réglage sont dépendants de la séquence ($s_{ij}$ dénotant le temps de réglage associé au fait de traiter la pièce $j$ juste après $i$). Dans la littérature, on utilise parfois le mot-clé *seq-dep* afin de signifier que les temps de réglage sont dépendants de la séquence.

### 1.1.3 Mesure de performance

Dans le troisième champ ($\gamma$), on spécifie l'objectif que l'on souhaite optimiser. Pour un horaire donné, on peut calculer pour chaque pièce $j \in N$ :

- son moment de complétion $C_j$ ;
- son délai de livraison $L_j = C_j - d_j$ ;
- son retard $T_j = \max\{0, C_j - d_j\}$ ;
- sa pénalité unitaire $U_i = 1$ si $C_j > d_j, U_j = 0$ autrement.

La plupart des objectifs classiques sont exprimés en fonction de ces quantités. En général, ces objectifs consistent soit à minimiser la somme (ou la somme pondérée) de ces quantités, comme par exemple $\min \sum_{j \in N} w_j C_j$, ou à minimiser le maximum mesuré pour une quantité donnée sur toutes les pièces, comme par exemple $\min \max_{j \in N} C_j$. Dans la notation de Graham *et al.* (1979), on suppose toujours que l'on cherche à minimiser une fonction objectif faisant intervenir les quantités mesurées sur toutes les pièces. Pour cette raison, on note souvent l'objectif $\min \sum_{j \in N} w_j C_j$ comme $\sum w_j C_j$ tandis que $\min \max_{j \in N} C_j$ est simplement noté $C_{max}$.

On note que certains de ces objectifs sont équivalents, au sens où un horaire qui est optimal pour l'un l'est également pour l'autre. Par exemple, l'objectif $\sum w_j L_j$ est équivalent à $\sum w_j C_j$ car ils sont identiques à une constante près $\left(\sum w_j (C_j - d_j) = \sum w_j C_j - \sum w_j d_j, \sum w_j d_j \text{ étant la constante }\right)$. De même, un horaire optimal pour $L_{max}$ l'est également pour $T_{max}$ et pour $U_{max}$ (mais un horaire optimal pour $T_{max}$ ou pour $U_{max}$ ne l'est pas nécessairement pour $L_{max}$).

## 1.2 Complexité des problèmes d'ordonnancement de production

Avec cette notation, on peut formuler de nombreux problèmes d'ordonnancement de production. Certains de ces problèmes sont faciles (au sens où on connaît un algorithme polynomial pour les résoudre) mais la plupart sont difficiles (dans la mesure où la mise au point d'un algorithme polynomial pour un de ceux-ci impliquerait que $P = NP$, voir Garey et Johnson, 1979). Dans cette section, on discute de la complexité théorique et pratique de ces problèmes.

### 1.2.1 Problèmes d'ordonnancement sur une machine

Il est difficile d'imaginer plus simple comme problème d'ordonnancement de production que le cas où il n'y a qu'une seule machine. Cette facilité apparente est parfois réelle : il existe en effet des cas pour lesquels on connaît des algorithmes polynomiaux.

Par exemple, pour résoudre $1|\cdot|L_{max}$, Jackson (1955) a montré qu'il suffit de séquencer les pièces sans attente dans l'ordre non-décroissant des dates de livraison (ordre $EDD$, *pour earliest due date*). De même, pour $1|\cdot|w_jC_j$, Smith (1956) a montré qu'il suffit de séquencer les pièces sans attente dans l'ordre non-croissant des $\frac{w_j}{p_j}$. Pour $1|\cdot|\sum U_j$ consistant à minimiser le nombre de pièces en retard, Moore (1968) a proposé un algorithme constructif $O(n \log n)$ consistant essentiellement à ajouter les pièces une à une à un horaire partiel en respectant l'ordre $EDD$. Lorsque la dernière pièce ajoutée est en retard, l'algorithme identifie la pièce planifiée à temps dans l'horaire partiel dont la durée de traitement est maximum et la déclare en retard.

Comme l'indique tableau le 1.1, il existe cependant de nombreux cas qui sont difficiles.

En analysant ce tableau, on réalise que :

- Les objectifs $\sum T_j$, $\sum w_j T_j$, $\sum w_j U_j$ sont des objectifs difficiles. Même les problèmes sans contraintes supplémentaires sont *NP-difficiles* lorsqu'ils font intervenir ces objectifs.
- Les contraintes supplémentaires de $r_j$ et de *prec* sont complicantes, dans la mesure où un problème facile (comme $1|\cdot|\sum C_j$) devient *NP-difficile* lorsqu'on introduit l'une ou l'autre des contraintes supplémentaires.
- Tolérer la préemption (*preemp*) peut parfois faciliter les choses. Le problème $1|r_j|\sum C_j$ devient par exemple facile dans le cas où on permet la préemption. Cependant si $r_j = 0, j \in N$ et l'objectif est régulier, permettre la préemption ne modifie pas la complexité du problème. En effet dans ce cas, Conway (1967) a montré qu'une solution optimale pour le problème sans préemption est également une solution optimale pour le problème avec préemption.
- Considérer que les temps de réglage sont dépendants de la séquence est très difficile. Un problème trivial comme $1|\cdot|C_{max}$ devient *NP-difficile* lorsqu'on ajoute des contraintes de type $s_{ij}$. En fait, ce type de contrainte introduit une composante de commis-voyageur au problème d'ordonnancement sur une machine. D'ailleurs $1|s_{ij}|C_{max}$ peut se résoudre directement comme un problème de commis-voyageur, pour lequel on est aujourd'hui capable de résoudre des problèmes comptant plusieurs centaines de villes, alors que $1|s_{ij}|\sum C_j$ peut se résoudre comme un problème de livreur (*deliveryman problem* en anglais), qui est en fait une version du problème de commis-voyageur où l'on cherche à minimiser la moyenne des temps d'arrivée aux villes, pour lequel on est à ce jour capable de résoudre des exemplaires comptant entre 30 et 60 villes.

### 1.2.2 Problèmes d'ordonnancement sur plusieurs machines

Considérer un environnement à plusieurs machines complique un peu les choses, comme l'indique le tableau 1.2. Trouver un horaire minimisant $\sum w_j C_j$ devient par

Tableau 1.1 – Complexité et problèmes d'ordonnancement sur une machine

| | |
|---|---|
| $1\|\cdot\|\sum w_jC_j$ | polynomial |
| $1\|r_j\|\sum C_j$ | NP-difficile |
| $1\|prec\|\sum C_j$ | NP-difficile |
| $1\|\cdot\|\sum T_j$ | NP-difficile (au sens ordinaire) |
| $1\|\cdot\|\sum w_jT_j$ | NP-difficile |
| $1\|\cdot\|\sum U_j$ | polynomial |
| $1\|\cdot\|\sum w_jU_j$ | NP-difficile |
| $1\|\cdot\|L_{max}$ | polynomial |
| $1\|r_j\|C_{max}$ | polynomial |
| $1\|r_j\|L_{max}$ | NP-difficile |
| $1\|p_j=p\|\sum w_jC_j$ | polynomial |
| $1\|p_j=1\|\sum w_jT_j$ | polynomial |
| $1\|p_j=p,r_j\|\sum w_jT_j$ | ouvert |
| $1\|preemp\|\sum w_jC_j$ | polynomial |
| $1\|preemp,r_j\|\sum C_j$ | polynomial |
| $1\|preemp,r_j\|\sum w_jC_j$ | NP-difficile |
| $1\|s_{ij}\|C_{max}$ | NP-difficile |
| $1\|s_{ij}\|\sum C_j$ | NP-difficile |

exemple *NP-difficile* si on considère un environnement de machines parallèles identiques plutôt qu'un environnement à une seule machine. En pratique, on est cependant capable de résoudre des exemplaires de taille intéressante pour le cas des machines parallèles. Mais considérer un environnement à plusieurs machines et plusieurs opérations semble compliquer énormément les choses. Même en introduisant des simplications, les problèmes d'atelier demeurent souvent *NP-difficiles.* Par exemple le problème d'atelier multigamme demeure *NP-difficile* au sens fort même si l'on fixe le nombre de machines à 2 et la durée des opérations à 1. Cette difficulté théorique se répercute également en pratique. Les problèmes d'ateliers multigammes sont encore reconnus aujourd'hui pour être parmi les plus difficiles de l'optimisation combinatoire.

Tableau 1.2 – Complexité et problèmes d'ordonnancement sur plusieurs machine

| | |
|---|---|
| $R\|\cdot\|\sum C_j$ | polynomial |
| $P\|\cdot\|\sum w_j C_j$ | NP-difficile |
| $F2\|\cdot\|\sum C_j$ | NP-difficile |
| $F2\|\cdot\|C_{max}$ | polynomial |
| $F3\|\cdot\|C_{max}$ | NP-difficile |
| $O2\|\cdot\|C_{max}$ | polynomial |
| $O3\|\cdot\|C_{max}$ | NP-difficile |
| $O2\|\cdot\|\sum C_j$ | NP-difficile |
| $J2\|p_j = 1\|\sum C_j$ | polynomial |
| $J2\|p_j = 1\|\sum w_j C_j$ | NP-difficile |
| $J2\|\cdot\|C_{max}$ | NP-difficile |

## 1.3 Conclusion

La notation de Graham permet non seulement de décrire de façon succincte un problème d'ordonnancement de production, mais permet également de créer des familles de problèmes présentant des similarités. On connaît en effet la complexité de la plupart des problèmes d'ordonnancement de production que l'on peut obtenir avec cette notation, et ce malgré leur très grand nombre (voir le site de Brucker : *http ://www.mathematik.uni-osnabrueck.de/research/OR/class/*). Graham *et al.* (1977) ont même écrit un programme capable d'évaluer la complexité d'un problème d'ordonnancement de production écrit selon leur notation.

Les problèmes d'ordonnancement de production sont cependant très souvent difficiles, même pour des cas où le nombre de machines est fixé à 1. L'existence d'un algorithme polynomial pour un de ces problèmes est donc peu probable. Le recours à des algorithmes exploitant à la fois des techniques de programmation dynamique, de programmation linéaire et d'énumération implicite, comme ceux développés dans les prochains chapitres, semble donc justifié.

# CHAPITRE 2 : LE PROBLÈME DE STABLE DE POIDS MAXIMUM

## 2.1 Introduction

Le problème de recherche d'un stable de poids maximum dans un graphe est l'un des problèmes les plus étudiés de l'optimisation combinatoire. De nombreux problèmes de programmation en nombres entiers possèdent une structure de stable. Le problème de partitionnement d'ensemble constitue un cas classique de problème qui peut être résolu comme un problème de stable de poids maximum. Dans le cadre du problème de commis-voyageur asymétrique, Balas (1989) a proposé une nouvelle famille de coupes (les coupes OddCat) qui ne sont que des coupes de trous liftées du problème de stable. Plus récemment, Waterer *et al.* (2002) ont étudié le problème d'ordonnancement sur une machine sans préemption avec la perspective du stable afin de retrouver certains résultats connus. Les bons solveurs commerciaux de programmation mixte (comme cplex) essaient même en prétraitement d'identifier une structure de stable dans le problème qu'on leur a soumis afin de générer des coupes non triviales. La structure du problème de stable est donc omniprésente en programmation en nombres entiers : les nombreux résultats connus pour celui-ci peuvent ainsi être utiles dans de multiples contextes.

Tous les problèmes abordés dans cette thèse possèdent également de forts liens avec le problème de stable de poids maximum et pour cette raison, on présente dans ce chapitre certains résultats concernant celui-ci. On montre comment en identifiant une structure de stable, on peut dériver des formulations de programmation en nombres entiers fortes (c.-à-d. que celles-ci offrent des relaxations linéaires serrées). En étudiant un problème de machines parallèles et un problème de commis-voyageur (dépendant

du temps) avec la perspective du stable, on arrive ainsi à déduire des formulations fortes pour ces deux problèmes qui sont au coeur des approches de résolution présentées dans ce travail.

## 2.2 Le problème de stable de poids maximum

Soit un graphe $G(\mathcal{N}, \mathcal{A})$ non-orienté où $\mathcal{N}$ est l'ensemble des sommets de $G$ et $\mathcal{A}$ est l'ensemble de ses arêtes. Un sous-ensemble de sommets $S \subseteq \mathcal{N}$ est dit stable si $i, j \in S$ implique que $(i, j) \notin \mathcal{A}$ (autrement dit, si le sous-graphe induit par $S$ ne contient aucune arête). Soit $c_j$ un poids associé au sommet $j \in \mathcal{N}$. Le problème de stable de poids maximum consiste à trouver un ensemble stable $S$ qui maximise $\sum_{j \in S} c_j$. On note que le problème de stable de poids maximum est intimement lié au problème de recherche d'une clique de poids maximum puisque tout stable dans le graphe $G$ est une clique dans le graphe complémentaire $\bar{G}(\mathcal{N}, \bar{\mathcal{A}})$, car par définition $(i, j) \in \bar{\mathcal{A}}$ lorsque $(i, j) \notin \mathcal{A}$.

Le problème de stable de poids maximum se formule comme ce programme en nombres entiers, noté $NP$ (pour *Node Packing*) :

$$Z_{NP} = \max \sum_{j \in \mathcal{N}} c_j x_j \tag{2.1}$$

$$x_i + x_j \leq 1, \quad (i, j) \in \mathcal{A} \tag{2.2}$$

$$x_j \in \{0, 1\}, \quad j \in \mathcal{N} \tag{2.3}$$

Dans le cas où $G$ est un graphe biparti, la relaxation linéaire de $NP$ est entière. Cependant, dans le cas général, la relaxation linéaire de $NP$ peut être fractionnaire et donne une borne trop lâche pour être utilisée directement à l'intérieur d'un algorithme d'évaluation et de séparation progressive. On doit cependant noter que la relaxation

linéaire de $NP$ possède des propriétés intéressantes (voir Nemhauser et Trotter 1974). Les variables qui sont entières dans la relaxation linéaire conservent les mêmes valeurs dans la solution optimale entière et peuvent par conséquent être fixées. De plus, cette relaxation linéaire peut être résolue comme un problème de flot à coût minimal, pour lequel il existe des algorithmes spécialisés très efficaces.

Il existe cependant des formulations de programmation en nombres entiers pour le problème de stable de poids maximum qui donnent de meilleures bornes de programmation linéaire que la formulation $NP$, telles que les formulations faisant intervenir des inégalités de cliques et de trous.

### Inégalités de cliques

Soit $C \subseteq N$, un sous-ensemble de noeuds de $G$ induisant un sous-graphe complet, c.-à-d. que $i, j \in C$ implique $(i, j) \in \mathcal{A}$. On appelle un tel sous-ensemble de noeuds une clique. Alors l'inégalité suivante :

$$\sum_{j \in C} x_j \leq 1 \tag{2.4}$$

est valide pour $NP$. De plus, Padberg (1973) a prouvé que cette inégalité définit une facette pour l'enveloppe convexe définie par les contraintes de $NP$ si $C$ est une clique maximale ($C$ est maximale s'il n'existe pas de sous-ensemble $C' \subset N$ tel que $C'$ est une clique et $C \subset C'$). Cela implique entre autres que le problème de recherche du stable de poids maximum peut être formulée de façon plus forte comme suit :

$$Z_{SP} = \max \sum_{j \in N} c_j x_j \tag{2.5}$$

$$\sum_{j \in C} x_j \leq 1, \quad C \in \mathcal{C} \tag{2.6}$$

$$x_j \in \{0, 1\}, \quad j \in \mathcal{N}. \tag{2.7}$$

Dans la formulation $SP$ (pour *Set Packing*), l'ensemble $\mathcal{C}$ est un ensemble de cliques couvrant toutes les arêtes du graphe. La force de la formulation $SP$ dépend évidemment de la richesse de l'ensemble $\mathcal{C}$. En choisissant $\mathcal{C}$ de telle sorte que cet ensemble contienne toutes les cliques maximales de $G$, on obtient la formulation de type $SP$ la plus forte, tandis que si l'on choisit $\mathcal{C}$ contenant seulement les cliques à 2 (deux noeuds reliés par une arête forment une clique à deux), on retombe sur $NP$, qui est en fait la plus faible des formulations de type $SP$.

Dans le cas où $G$ est un graphe parfait (un graphe est parfait si $w(G) = \gamma(G)$, où $w(G)$ est la taille de la plus grande clique de $G$ et $\gamma(G)$ est le nombre chromatique de $G$, c-.à-d. le nombre de couleurs minimum nécessaires pour colorier $G$), il a été montré que la relaxation linéaire de $SP$, avec $\mathcal{C}$ comprenant toutes les cliques maximales de $G$, est entière. Dans le cas général, les contraintes de cliques ne suffisent pas toujours à décrire l'enveloppe convexe de $NP$. D'autres familles d'inégalités peuvent cependant être utilisées afin de resserrer la relaxation de $SP$, comme les inégalités de trous que l'on présente succinctement ici.

### Inégalités de trous

Soit $\mathcal{H}$ un cycle sans corde (c.-à-d. un cycle dans lequel il n'y a pas d'arête qui relie deux sommets non consécutifs) de longueur $k$ (c.-à-d. composé de $k$ noeuds). Si $\mathcal{H}$ est de longueur impaire $k \geq 5$, alors $\mathcal{H}$ est appelé un trou (voir figure 2.1) et l'inégalité suivante :

$$\sum_{i \in \mathcal{H}} x_i \leq \lfloor \frac{k}{2} \rfloor \tag{2.8}$$

est valide pour l'enveloppe convexe de (NP). Notons que cette inégalité définit rarement une facette.

Il est cependant possible d'augmenter la dimension d'une inégalité en ayant recours à une procédure générique de *lifting* de la programmation en nombres entiers. Soit

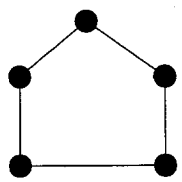

Figure 2.1 – Trou à cinq sommets

$\sum_{j=2}^{n} \pi_j x_j \leq \pi_0$ une inégalité valide pour le polytope $S \subset \{0,1\}^n$. L'idée derrière la procédure de *lifting* consiste à intégrer d'autres variables à une inégalité en trouvant un coefficient $\alpha$ tel que

$$\alpha_1 x_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0$$

soit encore valide pour le polytope $S$. Plus grand $\alpha$ sera choisi, plus forte sera l'inégalité liftée. Wolsey (1976) a montré qu'on peut prendre $\alpha \leq \pi_0 - \max_{x \in S: x_1 = 1} \sum_{j=2}^{n} \pi_j x_j$. En effet, si $x_1 = 0$, alors $\alpha x_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0$ est valide pour $S \cap (x_1 = 0)$ (car on a supposé que $\sum_{j=2}^{n} \pi_j x_j \leq \pi_0$ l'était pour $S$) tandis que pour $x_1 = 1$, $\alpha x_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0$ est également valide pour $S \cap (x_1 = 1)$ par la définition même de $\alpha$.

Pour illustrer cela, on considère l'exemple suivant tiré du livre de Nemhauser et Wolsey (1988) (voir figure 2.2). On a vu qu'on peut formuler le problème de stable de cardinalité maximum (qui est évidemment le cas particulier du problème de stable de poids maximum où tous les poids sont égaux à 1) comme ce programme en nombres entiers $NP$ :

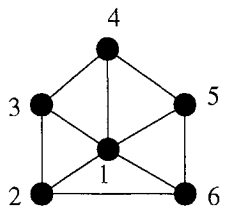


Figure 2.2 – Graphe considéré dans l'exemple

$$
\begin{aligned}
\max\, & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
& x_1 + x_2 \leq 1 \\
& x_1 + x_3 \leq 1 \\
& x_1 + x_4 \leq 1 \\
& x_1 + x_5 \leq 1 \\
& x_1 + x_6 \leq 1 \\
& x_2 + x_3 \leq 1 \\
& x_2 + x_6 \leq 1 \\
& x_3 + x_4 \leq 1 \\
& x_4 + x_5 \leq 1 \\
& x_5 + x_6 \leq 1 \\
& x_j \in \{0,1\}, \quad j = 1, \ldots, 6
\end{aligned}
$$

La solution optimale de la relaxation linéaire de ce programme est

$$x = (x_1, x_2, x_3, x_4, x_5, x_6) = (0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$$

de coût 3. Comme on l'a vu précédemment, il y a moyen de trouver une formulation plus forte en identifiant toutes les cliques maximales du graphe. Par inspection, on identifie les cliques maximales du graphe soit $C_1 = \{1, 2, 3\}$, $C_2 = \{1, 3, 4\}$, $C_3 = \{1, 4, 5\}$, $C_4 = \{1, 5, 6\}$ et $C_5 = \{1, 2, 6\}$. Avec l'ensemble de cliques $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$, on peut donc maintenant construire le modèle de type $SP$,

qui est plus fort que $NP$ :

$$\begin{aligned}
\max\, & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
& x_1 + x_2 + x_3 \leq 1 \\
& x_1 + x_3 + x_4 \leq 1 \\
& x_1 + x_4 + x_5 \leq 1 \\
& x_1 + x_5 + x_6 \leq 1 \\
& x_1 + x_2 + x_6 \leq 1 \\
& x_j \in \{0,1\}, \quad j = 1, \ldots, 6.
\end{aligned}$$

En effet, si on résoud la relaxation linéaire de ce modèle, on obtient comme solution $x = (0, 0.5, 0.5, 0.5, 0.5, 0.5)$ de coût 2.5. L'utilisation des contraintes de cliques a donc permis de resserer le saut d'intégrité, sans toutefois le faire complètement disparaître.

On remarque cependant que $x = (0, 0.5, 0.5, 0.5, 0.5, 0.5)$ viole la contrainte de trous associé à $\mathcal{H} = \{2, 3, 4, 5, 6\}$ (soit $x_2 + x_3 + x_4 + x_5 + x_6 = 2.5 \nleq 2$). Si on ajoute cette inégalité valide, on obtient maintenant comme solution de la relaxation linéaire $x = (0.2, 0.4, 0.4, 0.4, 0.4, 0.4)$ de coût 2.2. On voit que la solution est encore fractionnaire même après avoir ajouté cette inégalité de trous. On peut cependant diminuer le saut d'intégrité en liftant l'inégalité de trous, c.-à-d. en tentant de trouver un $\alpha$ tel que

$$\alpha x_1 + (x_2 + x_3 + x_4 + x_5 + x_6) \leq 2$$

soit valide. En posant $x_1 = 1$, on se rend compte que la plus grande valeur que peut prendre $x_2 + x_3 + x_4 + x_5 + x_6$ tout en respectant les contraintes de cliques est 0, donc $\alpha = 2$. En ajoutant la contrainte de trou liftée, on fait disparaître complètement le saut d'intégrité et on trouve la solution optimale au problème soit $x = (0, 1, 0, 1, 0, 0)$, qui n'est pas unique.

On voit donc à travers cet exemple que les inégalités de cliques, les inégalités de trous et les techniques de *lifting* peuvent être utiles afin de résoudre le problème de stable de poids maximum. L'utilité de ces techniques dépasse d'ailleurs largement le

contexte du problème de stable de poids maximum, comme le démontre par exemple l'article de Hoffman et Padberg (1993) sur le problème de fabrication de rotations en transport aérien, dans lequel on retrouve des techniques efficaces pour séparer les contraintes de cliques et de trous.

### 2.2.1 Un cas facile : les graphes d'intervalles

Le problème de stable de poids maximum est *NP-difficile* dans le cas général. Il existe certaines familles de graphes pour lequel celui-ci est cependant facile, dont la famille des graphes d'intervalles. On présente dans cette section comment ce problème peut être résolu pour cette famille de graphes.

Soit $I$ un ensemble d'intervalles et $c_i$ le poids de l'intervalle $i \in I$. Si on associe à chaque intervalle de $I$ un sommet et que l'on place une arête entre deux sommets si ceux-ci correspondent à des intervalles qui se chevauchent, alors on obtient un graphe d'intervalles, qui est en fait un graphe parfait. Le problème de recherche de stable de poids maximum dans ce graphe peut donc se résoudre en temps polynomial. On a en effet vu que la relaxation linéaire du modèle $SP$ construite avec toutes les cliques maximales est entière pour les graphes parfaits. Or, pour les graphes d'intervalles, identifier toutes les cliques maximales est un problème facile et il y en a au plus $|I|$ (voir Gavril, 1972). C'est ainsi qu'en résolvant la relaxation linéaire de $SP$ construit avec cet ensemble de cliques, on résoud le problème de stable de poids maximum pour les graphes d'intervalles.

Il existe une façon plus simple de concevoir le problème de stable de poids maximum dans un graphe d'intervalles. On peut en effet voir celui-ci comme un problème de plus court chemin dans un réseau construit comme suit. Pour commencer, on suppose que l'on a disposé tous les intervalles sur l'axe des réels. Pour chaque début ou fin d'un

intervalle, on associe un noeud et il existe un arc de coût nul entre deux noeuds qui sont situés un à la suite de l'autre. Pour tous les intervalles $i \in I$, on associe finalement un arc de coût $-c_i$ allant du début de l'intervalle jusqu'à la fin de l'intervalle. Tout chemin dans ce réseau correspond à un stable dans le graphe d'intervalle et tout stable dans le graphe d'intervalles peut être transposé comme un chemin dans ce réseau. En résolvant un problème de plus court chemin sur ce réseau (acyclique par construction), on parvient donc à résoudre le problème de stable de poids maximum. Comme il y a dans ce réseau au plus $3|I| - 1$ arcs et que le problème de plus court chemin sur un graphe acyclique peut se résoudre en $O(m)$ où $m$ est le nombre d'arcs dans le réseau considéré, on peut donc résoudre le problème de stable de poids maximum dans un graphe d'intervalles en $O(|I|)$ opérations. La figure 2.3 illustre cette transformation sur un exemple à quatre intervalles.

On voit donc que le problème de stable de poids maximum dans un graphe d'intervalles peut se ramener à un problème de plus court chemin sur graphe acyclique. Ce résultat n'est guère surprenant puisque la matrice des contraintes de la formulation $SP$ obtenue en prenant toutes les contraintes de cliques maximales est en fait une matrice de réseau (voir Nemhauser et Wolsey, 1988).

### 2.2.2 Un cas un peu plus difficile : le graphe d'intervalles de tâches

#### Présentation du problème

On s'intéresse dans cette section au problème de stable de poids maximum dans une famille de graphes appelée les graphes d'intervalles de tâches. Les applications de ce problème sont nombreuses que ce soit dans le domaine de l'ordonnancement de production, de la biologie moléculaire ou en télécommunications (voir Erlebach et Spieksma, 2002).

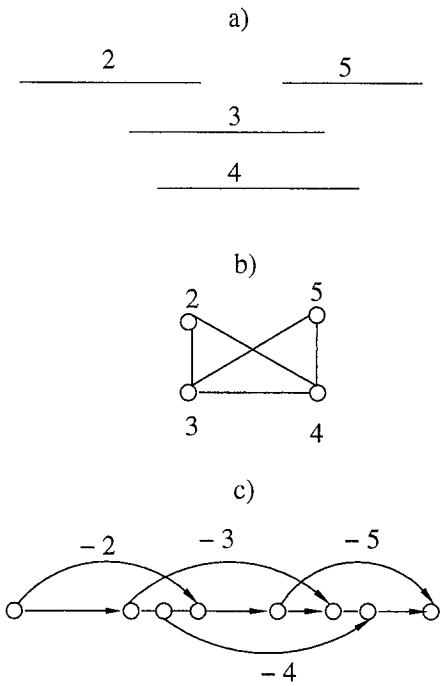


Figure 2.3 – a) Intervalles considérés b) Graphe d'intervalles c) Réseau

Dans un graphe d'intervalles de tâches, chaque sommet est associé à une tâche, qui est en fait un intervalle de temps au cours duquel on traite au complet une pièce. Il existe une arête entre deux tâches si leur intervalles de traitement associés se chevauchent (arêtes d'incompatibilité temporelle) ou si elles correspondent au traitement de la même pièce (arêtes d'incompatibilité de pièce). Ce graphe peut être vu comme l'union d'un graphe d'intervalles (composé des arêtes d'incompatibilité temporelle) et d'un graphe composé de cliques disjointes (composé des arêtes d'incompatibilité de pièces). Trouver un stable de poids maximal (ou de cardinalité maximale) dans ce graphe est un problème *NP-difficile* pour lequel ont surtout été proposés des algorithmes d'approximation (Spiekma, 1999 ; Spiekma et Erleback, 2001). On remarque cependant qu'il existe certains cas particuliers faciles. Dans le cas où il y a une seule tâche par pièce, cela revient au problème de recherche de stable dans un graphe d'intervalles tandis que dans le cas où toutes les tâches ont la même durée, le problème peut se formuler comme un problème de flot dans un réseau. Dans le cas général, on peut formuler le problème de stable de poids maximum dans le graphe d'intervalles de tâches avec ce programme en nombres entiers, inspiré de la formulation $SP$ du problème général du stable :

$$Z_{JISP} = \max \sum_{i \in I} c_i x_i \tag{2.9}$$

$$\sum_{i \in C} x_i \leq 1, \quad C \in \mathcal{C}^1 \tag{2.10}$$

$$\sum_{i \in C} x_i \leq 1, \quad C \in \mathcal{C}^2 \tag{2.11}$$

$$x_i \in \{0,1\}, \quad i \in I \tag{2.12}$$

Dans la formulation $JISP$ (pour *Job Interval Scheduling Problem*), $\mathcal{C}^1$ est l'ensemble des cliques maximales du graphe d'incompatibilité de pièces et $\mathcal{C}^2$ est l'ensemble des cliques maximales du graphe d'incompatibilité temporelle. Cette formulation est équivalente à celle proposée par Spieksma (2001) pour le problème de stable de cardinalité maximum pour laquelle celui-ci a prouvé que la valeur de sa relaxation linéaire était plus petite ou égale à deux fois la valeur de la solution optimale.

### Une application à un problème d'ordonnancement

On considère le problème d'ordonnancement suivant. Soit $m$ machines (identiques ou non) et $n$ pièces à traiter. Chaque pièce doit être traitée entièrement à l'intérieur d'une fenêtre de temps, possède une durée de traitement (qui peut dépendre de la machine utilisée, dans le cas où les machines ne sont pas identiques) et un poids donné. Le problème d'ordonnancement considéré ici consiste à trouver l'ordre de traitement des pièces sans préemption maximisant la somme des poids des pièces traitées. Ce problème, en apparence très simple, capture la difficulté de bien des problèmes d'ordonnancement sur une machine ou sur machines parallèles et peut être résolu comme un problème de stable dans un graphe d'intervalles de tâches si les fenêtres de temps et les durées de traitement sont entières. L'idée consiste d'abord à projeter l'horizon de planification de chaque machine sur différentes parties de l'axe des réels puis à énumérer tous les intervalles de traitement possibles pour les pièces qui débutent à des moments entiers et qui respectent les contraintes de fenêtre de temps. Si on construit le graphe d'intervalles de tâches à partir de ces intervalles, alors tout stable dans ce graphe correspond à un horaire légal et résoudre le problème du stable de poids maximum revient à résoudre le problème d'ordonnancement.

Considérons l'exemple suivant à trois pièces et deux machines. Les trois pièces possèdent respectivement une durée de traitement de $p_1 = 1, p_2 = 2, p_3 = 3$ sur les deux machines et des poids unitaires. La pièce 1 ne peut être traitée que sur la machine 1. Finalement chaque pièce doit être traitée à l'intérieur de la fenêtre $[0, 3]$.

Afin de transformer ce problème à deux machines en un problème à une machine, on a vu qu'il suffisait de projeter l'horizon de planification des machines sur différentes parties de l'axe des réels. Pour cette raison, on associe ainsi la partie $[0, 3]$ à la machine 1 et la partie $[3, 6]$ à la machine 2. La figure 2.4 illustre les intervalles de traitement possibles pour chacune des pièces.

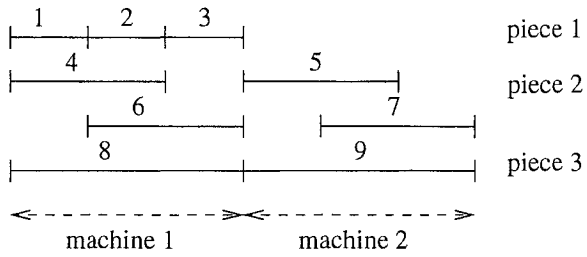


Figure 2.4 – Intervalles de l'exemple

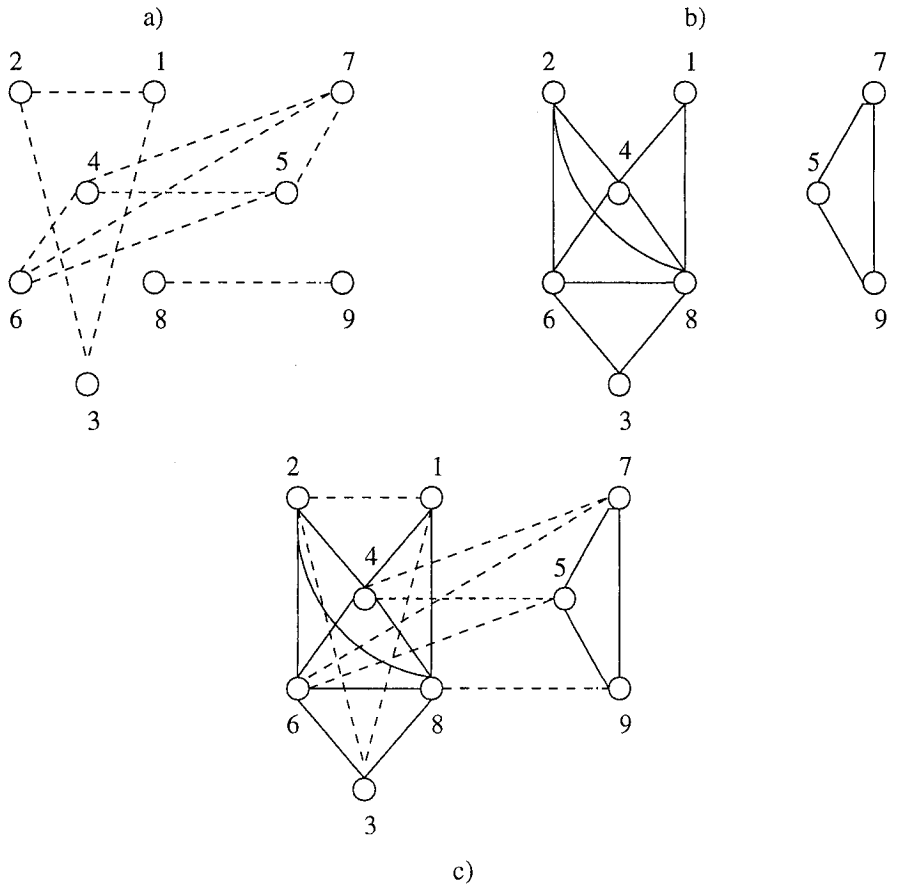


Figure 2.5 – a) Graphe d'incompatibilité de pièces b) Graphe d'incompatibilité temporelle c) Graphe d'intervalles de tâches

Ce problème peut se formuler comme ce programme en nombres entiers :

$$\begin{aligned}
\max\, & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \\
& x_1 + x_2 + x_3 \leq 1 \\
& x_4 + x_5 + x_6 + x_7 \leq 1 \\
& x_8 + x_9 \leq 1 \\
& x_1 + x_4 + x_8 \leq 1 \\
& x_2 + x_4 + x_6 + x_8 \leq 1 \\
& x_3 + x_6 + x_8 \leq 1 \\
& x_5 + x_7 + x_9 \leq 1 \\
& x_j \in \{0, 1\}, \quad j = 1, \ldots, 9.
\end{aligned}$$

L'objectif consiste à maximiser le nombre d'intervalles choisis (tous les poids des pièces sont unitaires). Les trois premières contraintes correspondent aux contraintes de cliques du graphe obtenu en ne considérant que les arêtes d'incompatibilité de pièces (voir figure 2.5 a) tandis que les quatre contraintes suivantes sont les contraintes de cliques maximales du graphe obtenu en ne considérant que les arêtes d'incompatibilité temporelle (voir figure 2.5 b). On note qu'on pourrait resserer cette formulation en ajoutant certaines contraintes de cliques maximales issues directement du graphe d'intervalles de tâches (voir figure 2.5 c) comme :

$$x_1 + x_2 + x_4 + x_8 \leq 1$$

ou

$$x_2 + x_3 + x_6 + x_8 \leq 1.$$

### 2.2.3 Un autre cas difficile : le problème de commis voyageur

Soit un ensemble de noeuds $\mathcal{N} = 1, \ldots, n$ et un ensemble d'arcs $\mathcal{A}$ où est associé à chaque arc $(i, j) \in \mathcal{A}$ un coût $c_{ij}$. Le problème du commis voyageur consiste à trouver

un tour de coût minimum qui débute et se termine au noeud 1. Ce problème est l'un des problèmes *NP-difficiles* qui a été le plus étudié (voir le livre de Lawler *et al.*, 1985). Dans cette section, on formule ce problème comme un problème de stable et on en déduit une formulation de programmation en nombres entiers très proche de celle proposée par Hadley (1964) et revisitée par Houck *et al.* (1980).

L'idée derrière la transformation consiste à remarquer qu'une tournée à travers $n$ noeuds est en fait un chemin comptant exactement $n$ arcs. Le problème du commis-voyageur consiste donc à trouver quel arc doit apparaître en position $t$ et ce pour les positions $t = 1, \ldots, n$ de façon à minimiser la somme des coûts tout en s'assurant d'avoir une tournée légale. Soit $(i,j)^t$ la décision consistant à placer l'arc $(i,j)$ à la position $t$. La décision $(i,j)^t$ est dite incompatible avec la décision $(k,l)^s$ si elles ne peuvent jamais faire partie toutes les deux d'une solution réalisable, c'est-à-dire lorsque :

- $j = l$ ou $i = k$ (incompatibilité de noeud) ;
- $s = t + 1, k \neq j$ ou $t = s + 1, i \neq l$ (incompatibilité de flot).

Le premier cas traduit simplement le fait qu'un même noeud ne peut pas être visité deux fois, tandis que le deuxième cas permet de faire appliquer les contraintes de flot pour les positions adjacentes de la tournée. On remarque que ce graphe peut être vu comme l'union de deux graphes, le premier graphe contenant les arêtes d'incompatibilité de noeud et le second contenant les incompatibilités de flot. Si on associe à chaque décision $(i,j)^t$ un sommet de poids $M - c_{ij}$ et si la constante $M$ est choisie suffisamment grande, le stable de poids maximum dans ce graphe aura une cardinalité de $n$ et représentera une tournée de coût minimum.

Cela implique entre autres que l'on peut formuler le problème de commis-voyageur

comme un problème de stable de poids maximum :

$$\max \sum_{(i,j)\in\mathcal{A}} \sum_{t=1}^{n} (M - c_{ij}) x_{ij}^t \tag{2.13}$$

$$\sum_{(i,j)^t\in C} x_{ij}^t \leq 1, \quad C \in \mathcal{C}^1 \tag{2.14}$$

$$\sum_{(i,j)^t\in C} x_{ij}^t \leq 1, \quad C \in \mathcal{C}^2 \tag{2.15}$$

$$x_{(i,j)}^t \in \{0,1\}, \quad (i,j) \in \mathcal{A}, t = 1,\ldots,n \tag{2.16}$$

où $\mathcal{C}^1$ et $\mathcal{C}^2$ sont respectivement les ensembles de cliques maximales dans les graphe d'incompatibilité de noeud et d'incompatibilité de flot. Comme les contraintes de cliques associées à $\mathcal{C}^1$ sont :

$$\sum_{t=1}^{n} \sum_{(i,j)\in\mathcal{A}} x_{ij}^t \leq 1, \quad j = 1,\ldots,n,$$

et que celles associées à $\mathcal{C}^2$ sont :

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^t + \sum_{(k,l)\in\mathcal{A}:k\neq j} x_{kl}^{t+1} \leq 1, \quad t = 1,\ldots,(n-1), \quad j = 1,\ldots,n,$$

cette formulation compte $O(n^3)$ variables et $O(n^2)$ contraintes.

En remplaçant les contraintes associées à $C_1$ par des égalités, on obtient de nouveau la formulation proposée par Hadley (1964) pour le problème de commis-voyageur, que Picard et Queyranne (1978) ont par la suite utilisée en vue de résoudre le problème de commis-voyageur dépendant du temps.

En raison de sa taille, cette formulation n'est pas vraiment intéressante pour le problème de commis-voyageur classique. Pour un problème de 100 noeuds par exemple, que l'on résoud de façon routinière avec d'autres formulations, il peut être très difficile de seulement calculer sa relaxation linéaire (dans ce cas, il y a en effet 1 000 000

de variables et 10000 contraintes). Par contre, pour la version dépendante du temps (c.-à-d. que le coût d'un arc dépend de sa position dans la tournée), il s'agit encore d'une des meilleures formulations disponibles (voir Vander Wiel et Sahinidis, 1995).

## 2.3 Conclusion

On a vu dans ce chapitre que l'on peut parfois construire de bonnes formulations de programmation en nombres entiers pour un problème d'ordonnancement sur machines parallèles et pour le problème de commis-voyageur en identifiant une structure de stable. Ces formulations de stable sont au coeur des approches développées au cours des chapitres suivants pour des problèmes d'ordonnancement sur une machine classiques (chapitre 3), d'autres moins classiques dans lesquels les temps de réglage dépendent de la séquence (chapitre 4) et même pour un problème très pratique d'ordonnancement sur machines parallèles avec contraintes de ressources humaines retrouvé dans les Forces Armées Canadiennes (chapitre 5).

# CHAPITRE 3 : ORDONNANCEMENT SUR UNE MACHINE I

## 3.1 Introduction

On a vu au chapitre précédent comment modéliser un problème d'ordonnancement sur machines parallèles comme un problème de stable de poids maximum dans un graphe d'intervalles de tâches, pour lequel on connaît une formulation de programmation en nombres entiers assez forte. Si on applique cette approche au problème d'ordonnancement sur une machine, on réobtient une formulation classique du problème dite de type temps indexé. Cette formulation est reconnue pour être très forte, dans la mesure où les sauts d'intégrité associés à celle-ci sont petits. Pourtant, peu d'approches de résolution ont été développées à partir de cette formulation en raison d'un irritant majeur : calculer sa relaxation linéaire peut être très difficile, en raison du grand nombre de variables et de contraintes retrouvées au sein de celles-ci. On présente dans ce chapitre comment on peut déduire plus facilement, à partir des formulations de type temps indexé, de très bonnes bornes pour les problèmes d'ordonnancement sur une machine, en ayant recours au principe de décomposition de Dantzig-Wolfe et à un principe de décomposition temporelle. On montre également comment, en imbriquant cette borne dans un algorithme d'énumération implicite, on peut obtenir une approche de résolution efficace pour un problème difficile d'ordonnancement sur une machine. De nombreux exemplaires ouverts du problème $1|\cdot|w_jT_j$ retrouvés dans la banque de problèmes *OR-LIBRARY* ont pu être résolus grâce à cette approche.

Ce chapitre est divisé comme suit. À la section 3.2, nous présentons diverses formulations de programmation en nombres entiers pour le problème d'ordonnancement sur une machine. À la section 3.3, nous montrons une nouvelle méthode pour obtenir plus efficacement des bornes inférieures pour ces formulations. À la section 3.3,

nous introduisons un nouvel algorithme exact d'énumération implicite qui a recours à ce mécanisme de calcul de bornes et à des règles de dominance pour le problème $1|\cdot|\sum w_j T_j$ et des résultats numériques sont présentés. À la section 3.4, nous présentons des pistes de recherche et une conclusion.

## 3.2 Formulation de type temps indexé

### 3.2.1 Formulation comme un problème de stable

Soit $H = \{1, \ldots, T\}$ l'horizon de planification considéré contenant $T$ périodes de durée unitaire. Soit $(j,t)$ la décision consistant à commencer le traitement de la pièce $j$ à la période $t$, débutant au moment $t-1$ et se terminant au moment $t$, et $c_{jt}$ le poids associé à cette décision. Si on associe un sommet à chaque décision, et que l'on place une arête entre deux décisions si celles-ci sont incompatibles, c'est-à-dire si elles concernent la même pièce (incompatibilité de pièce) ou se déroulent en même temps (incompatibilité temporelle), on obtient un graphe d'intervalles de tâches. On peut trouver un stable de poids maximum dans ce graphe en résolvant le programme en nombres entiers :

$$z_{NP(E)} = \max \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \tag{3.1}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} \leq 1, \quad j = 1, \ldots, n \tag{3.2}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, \quad t = 1, \ldots, T \tag{3.3}$$

$$x_{jt} \in \{0,1\}, \quad j = 1, \ldots, n, \quad t = 1, \ldots, T - p_j + 1. \tag{3.4}$$

Les contraintes (3.2) représentent les contraintes de cliques maximales du graphe d'in-

compatibilité de pièce alors que les contraintes (3.3) représentent les contraintes de cliques maximales du graphe d'incompatibilité temporelle (et possiblement quelques contraintes associées à des cliques dominées). Le nombre de contraintes (3.3), tout comme le nombre de variables est de l'ordre de la longueur de l'horizon de planification ($O(T)$ contraintes et $O(nT)$ variables). On peut donc au mieux construire avec cette formulation un algorithme pseudopolynomial. Cette formulation est donc surtout intéressante pour les problèmes "difficiles" d'ordonnancement sur une machine, c.-à-d. ceux pour lesquels on ne connaît pas d'algorithmes polynomiaux ou pseudopolynomiaux.

On peut par exemple modéliser le problème $1|\cdot|\sum T_j$, en posant $c_{jt} = M - \max\{0, t + p_j - 1 - d_j\}$ où $M$ est une constante suffisamment grande pour forcer la prise en compte de toutes les pièces. De même, on peut modéliser le problème $1|r_j|\sum C_j$ en posant $c_{jt} = M - (t + p_j - 1)$ et en posant $x_{jt} = 0$ si $t \leq r_j$.

Cette formulation est similaire à une formulation classique du problème d'ordonnancement sur une machine, que Dyer et Wolsey (1988) ont appelé $E$ :

$$z_E = \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \tag{3.5}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \quad j = 1, \ldots, n \tag{3.6}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, \quad t = 1, \ldots, T \tag{3.7}$$

$$x_{jt} \in \{0, 1\}, \quad j = 1, \ldots, n, \quad t = 1, \ldots, T - p_j + 1. \tag{3.8}$$

Elle est peut-être plus appropriée que $NP(E)$ pour traiter le cas où toutes les pièces doivent être considérées. La formulation $NP(E)$ met cependant en évidence une structure de stable du problème que l'on peut exploiter afin de déduire diverses coupes.

Les formulations de type temps indexé ont été étudiées par plusieurs chercheurs : Dyer et Wolsey (1988) ; Sousa et Wolsey (1992) ; Schultz (1994) ; van den Akker *et al.* (1995), (1999), (2000) ; Waterer *et al.* (2002). Leurs efforts ont surtout mené à la caractérisation de la structure polyédrale des formulations de type temps indexé et plus récemment au développement d'algorithmes d'approximation avec garantie de performance basés sur la relaxation linéaire de ces formulations (Phillips *et al.*, 1998 ; Goemans, 1997 ; Hall *et al.*, 1997).

Sousa et Wolsey (1992) ont dérivé pour les formulations de type temps indexé des inégalités valides avec membre de droite égal à 1 et 2 en se servant des résultats sur les inégalités valides de couverture pour le problème de sac à dos avec contraintes de borne supérieure généralisée. van den Akker *et al.* (1999) ont complètement caractérisé les inégalités valides avec des membres de droite égaux à 1 et 2 et proposé des procédures de séparation pour celles-ci. Waterer *et al.* (2002) ont été les premiers à mettre en lumière les relations entre les formulations de type temps indexé et le problème de recherche du stable de poids maximal. Ils ont ainsi démontré que les inégalités valides à coefficients entiers et membres de droite égaux à 1 ou 2 correspondent soit à des inégalités de cliques ou soit à des inégalités de trous maximalement liftées.

Considérons cet exemple à trois pièces avec $p_1 = 4$, $p_2 = 4$ et $p_3 = 3$ pour illustrer les résultats de Waterer *et al.* (2002).

La solution $x_{1,1} = x_{3,1} = x_{3,5} = x_{2,6} = x_{1,8} = x_{2,10} = 0.5$ respecte toutes les contraintes de la relaxation linéaire de $NP(E)$ (ou de $E$). Si on considère le sous-graphe d'incompatibilité induit par ces variables (où les arêtes en pointillés représentent les incompatibilités temporelles et les autres les incompatibilités de pièces), on peut déduire des inégalités valides pour $NP(E)$ qui sont violées par cette solution. Par exemple, elle viole la contrainte de clique

$$x_{1,8} + x_{2,6} + x_{2,10} \leq 1$$

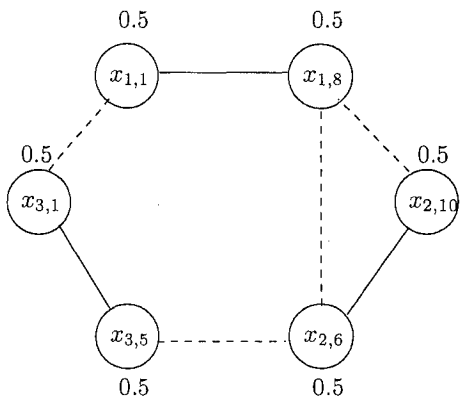


Figure 3.1 – Sous-graphe d'incompatibilité

et la contrainte de trou

$$x_{1,1} + x_{3,1} + x_{3,5} + x_{2,6} + x_{1,8} \leq 2$$

induite par les relations d'incompatibilité des variables.

Sousa et Wolsey (1992) et van den Akker *et al.* (1995) ont conçu à partir de certains de ces résultats des algorithmes d'énumération et d'évaluation progressive avec génération dynamique d'inégalités valides (*branch and cut*). Les auteurs ont réussi à résoudre des problèmes comptant jusqu'à 30 pièces, avec $p_{max} = \max_{j \in N} p_j = 10$. Les algorithmes basés directement sur le modèle $E$ fonctionnent donc bien pour les problèmes comptant un nombre modéré de pièces et avec de petits $p_{max}$. Mais pour des problèmes où l'horizon de planification est plus long (lorsque le nombre de pièces est plus élevé ou lorsque $p_{max}$ est plus grand), résoudre la relaxation linéaire de $E$ devient difficile.

### 3.2.2 Reformulation de flot

Résoudre la relaxation linéaire de $E$ et $NP(E)$ est parfois difficile car ces formulations, en plus de compter beaucoup de variables et de contraintes, sont très denses. Par exemple, si $p_{max} = 100$, certaines variables contribuent à 101 contraintes.

Il est cependant possible de remplacer les contraintes de capacité de la machine de $NP(E)$ et $E$ par des contraintes moins denses. En effet, si on relaxe les contraintes (3.6), on remarque que le problème résultant se réduit à un problème de stable dans un graphe d'intervalles, qui peut se résoudre comme un problème de plus court chemin dans un réseau (voir chapitre 2 pour une description du réseau).

Cela implique entre autres que l'on peut réécrire $E$ comme suit :

$$z_F = \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \tag{3.9}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \quad j = 1, \ldots, n \tag{3.10}$$

$$\sum_{j=1}^{n} x_{j1} + e_1 = 1, \tag{3.11}$$

$$\sum_{j=1}^{n} x_{j(t-p_j)} - \sum_{j=1}^{n} x_{jt} + e_{t-1} - e_t = 0, \quad t = 2, \ldots, T-1 \tag{3.12}$$

$$\sum_{j=1}^{n} x_{j(T-p_j+1)} + e_{T+1} = 1, \tag{3.13}$$

$$x_{jt} \in \{0,1\}, \quad j = 1, \ldots, n, \quad t = 1, \ldots, T - p_j + 1. \tag{3.14}$$

que l'on note $F$, où les contraintes de cliques maximales du problème de stable dans un graphe d'intervalles ont été remplacées par des contraintes de flot équivalentes. Ce problème compte à peu près le même nombre de variables que $E$ et le même nombre

de contraintes. La matrice des contraintes est cependant beaucoup moins dense : peu importe la durée de traitement d'une pièce $j$, les variables $x_{jt}$ contribueront à au plus 2 contraintes parmi (3.10)-(3.13).

### 3.2.3 Réécriture de Dantzig-Wolfe

van den Akker *et al.* (2000) ont proposé d'utiliser le principe de décomposition de Dantzig-Wolfe afin d'obtenir une formulation moins dense.

Si on note $\Omega$ l'ensemble des chemins légaux dans le réseau défini par les contraintes de flot du modèle $F$ et $x_{jt}^p$ une constante dénotant le niveau de flot du chemin $p$ sur l'arc associé à la décision $(j,t)$ et que l'on associe à chaque chemin la variable $\theta_p$, valant 1 si le chemin appartient à la solution et 0 autrement, on peut formuler le modèle suivant noté $DW(F)$ :

$$z_{DW(F)} = \min \sum_{p\in\Omega} (\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^p)\theta_p \tag{3.15}$$

$$\sum_{p\in\Omega} (\sum_{t=1}^{T-p_j+1} x_{jt}^p)\theta_p = 1, \quad j = 1,\ldots,n \tag{3.16}$$

$$\sum_{p\in\Omega} \theta_p = 1, \tag{3.17}$$

$$\theta_p \in \{0,1\}, \quad p \in \Omega \tag{3.18}$$

qui est la réécriture de Dantzig-Wolfe de $F$ (donc de $E$) comptant beaucoup plus de variables, mais beaucoup moins de contraintes. En fait, on a remplacé la description compacte de la capacité de la machine (à l'aide des facettes définies par les contraintes de flot) par une description explicite (à l'aide de tous les points extrêmes).

Comme van den Akker *et al.* (2000) l'ont souligné, il peut être plus facile de résoudre

la relaxation linéaire de $DW(F)$ que celle de $F$ (ou $E$) par génération de colonnes. On peut en effet identifier pour un vecteur donné de variables duales, noté $\pi$, la colonne de plus petit coût réduit en résolvant ce problème auxiliaire (appelé sous-problème ou oracle) :

$$z_{SP} = \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt} \tag{3.19}$$

$$\sum_{j=1}^{n} x_{j1} + e_1 = 1, \tag{3.20}$$

$$\sum_{j=1}^{n} x_{j(t-p_j)} - \sum_{j=1}^{n} x_{jt} + e_{t-1} - e_t = 0, \quad t = 2, \ldots, T-1 \tag{3.21}$$

$$\sum_{j=1}^{n} x_{j(T-p_j+1)} + e_{T+1} = 1, \tag{3.22}$$

$$x_{jt} \in \{0,1\}, \quad j = 1, \ldots, n, \quad t = 1, \ldots, T - p_j + 1 \tag{3.23}$$

qui est en fait un problème de plus court chemin dans le réseau décrit au chapitre 2 pour résoudre le problème de stable de poids maximum dans un graphe d'intervalles.

Avec cette reformulation de Dantzig-Wolfe, van den Akker *et al.* (2000) ont été en mesure de trouver la relaxation linéaire pour des exemplaires d'assez grande taille ($p_{max} = 100$) pour lesquels ils n'avaient même pas été pas en mesure de calculer directement la relaxation linéaire de $E$. Résoudre $DW(F)$ au lieu de $E$ semble donc être l'une des solutions aux difficultés engendrées par de grandes valeurs de $p_{max}$. Par contre, pour des exemplaires comptant plus de pièces, le processus devient beaucoup moins efficace. Pour certains exemplaires de plus de 100 pièces, l'algorithme de génération de colonnes de van den Akker *et al.* (2000) n'a pas convergé en plus d'une heure vers la relaxation linéaire exacte. Afin de contrer les phénomènes de convergence en longue queue, les auteurs ont donc proposé d'utiliser la borne lagrangienne donnée par le sous-problème (Lasdon, 1970; Vanderbeck et Wolsey, 1996) afin de stopper prématurément le processus de génération de colonnes avec une solution $\epsilon$-optimale

pour la relaxation linéaire.

On présente dans ce chapitre deux autres méthodes pour faciliter l'obtention de bornes de qualité pour les problèmes d'ordonnancement sur une machine. Les deux méthodes sont basées sur la désagrégation des sous-problèmes, qui est reconnue pour accélérer la génération de colonnes. La première méthode garantit les mêmes bornes que $DW(F)$ et permet d'accélérer légèrement la convergence de la génération de colonnes. La deuxième méthode ne garantit pas une aussi bonne borne que $DW(F)$ mais accélère significativement le processus de génération de colonnes.

## 3.3 Décomposition temporelle

Une façon naturelle de séparer un problème ayant une structure temporelle consiste à diviser l'horizon de planification en $K$ sous-horizons. Une application heuristique très courante de ce mécanisme consiste à résoudre les $K$ sous-problèmes ainsi obtenus l'un à la suite de l'autre, sans retour vers l'arrière. Dans cette section, on propose plutôt une implantation exacte de cette décomposition temporelle pour le problème d'ordonnancement sur une machine, en ayant recours à un problème maître de Dantzig-Wolfe pour superviser les décisions prises localement.

Afin de simplifier l'exposé, on propose une écriture pour les modèles de type temps indexé inspirée du concept de tâche présenté au chapitre précédent. Une tâche peut être vue comme un intervalle de temps continu au cours duquel on effectue en partie ou en totalité le traitement d'une ou plusieurs pièces. On note respectivement $l(v)$ et $u(v)$ la période de début et la période de fin pour la tâche $v$ et on associe à $v$ les constantes $a_v^i$ indiquant la proportion du traitement de la pièce $i$ que l'on y effectue. On remarque que dans le cas où on ne permet pas la préemption, $a_v^i$ ne peut valoir que 1 ou 0. On note finalement $c_v$ le coût associé à l'exécution de $v$.

Si on associe à chacune des actions $(j,t)$ une tâche $v$ avec $l(v) = t, u(v) = t + p_j - 1$, $a_v^i = 1$ pour $i = j$ et $a_v^i = 0$ autrement et $c_v = c_{jt}$, on obtient un ensemble de tâches $V$, à partir duquel on peut formuler ce modèle :

$$z_{G_1} = \min \sum_{v \in V} c_v x_v \tag{3.24}$$

$$\sum_{v \in V} a_v^j x_v = 1, \quad j = 1, \ldots, n \tag{3.25}$$

$$\sum_{v \in V : t \in [l(v), u(v)]} x_v \leq 1, \quad t = 1, \ldots, T \tag{3.26}$$

$$x_v \in \{0, 1\}, \quad v \in V \tag{3.27}$$

où $x_v$ vaut 1 si la tâche $v$ est exécutée et 0 sinon. Les contraintes (3.25) assurent que chaque pièce est entièrement traitée tandis que les contraintes (3.26) permettent de respecter la contrainte de capacité de la machine (soit les contraintes de cliques du graphe d'incompatibilité temporelle).

On note ce modèle $G_1$. Le 1 en indice indique qu'il y a une seule structure de bloc angulaire. On présente à la prochaine section une méthode pour dériver les modèles $G_K$, comptant $K$ structures de bloc angulaires.

### 3.3.1 Une décomposition temporelle exacte

On considère maintenant que l'horizon de planification $H$ est décomposé en $K$ sous-horizons de planification contigus, où

$$H_1 = 1, \ldots, T_1, \; H_2 = T_1, \ldots, T_2 \, , \quad \ldots, \; H_K = T_{K-1}, \ldots, T_K.$$

La période de départ de l'horizon $H_k$ est notée $T_{k-1}$ et la période de fin est notée $T_k$.

On dit que l'horizon $H_k$ contient la tâche $v$ si elle se déroule entièrement à l'intérieur de celui-ci, c.-à-d. si $l(v) \geq T_{k-1}$ et $u(t) \leq T_k$. Une tâche se déroulant au cours d'au moins deux sous-horizons de planification est dite compliquante. L'ensemble des tâches appartenant à l'horizon $H_k$ est noté $V_k$ tandis que l'ensemble des tâches compliquantes est noté $V_c$ avec $V_c = V \setminus \bigcup_{k=1}^{K} V_k$. Une période $t$ pour laquelle il existe au moins une tâche $v \in V_c$ telle que $t \in [l(v), u(v)]$ est dite compliquante et l'ensemble de ces périodes compliquantes est noté $H_c$. Finalement, on note $V(t)$ l'ensemble des tâches se déroulant au cours de la période $t$ ($V(t) = \{v \in V \mid t \in [l(v), u(v)]\}$).

En se basant sur ces définitions, on peut reformuler le modèle $G_1$ en faisant apparaître une structure de bloc angulaire (à $K$ blocs) :

$$z_{G_K} = \min \sum_{k=1}^{K} \sum_{v \in V_k} c_v x_v + \sum_{v \in V_c} c_v x_v \tag{3.28}$$

$$\sum_{k=1}^{K} \sum_{v \in V_k} a_v^j x_v + \sum_{v \in V_c} a_v^j x_v = 1, \quad j = 1, \ldots, n \tag{3.29}$$

$$\sum_{v \in V_k(t)} x_v \leq 1, \quad t \in H_k, \quad k = 1, \ldots, K \tag{3.30}$$

$$\sum_{v \in V(t)} x_v \leq 1, \quad t \in H_c \tag{3.31}$$

$$x_v \in \{0, 1\}, \quad v \in V \tag{3.32}$$

Pour cette formulation, les contraintes de capacité de la machine (3.30) sont locales pour chacun des sous-horizons de planification $H_k$ alors que les contraintes de partitionnement pour le traitement des pièces (3.29) et les contraintes de capacité de la machine pour les périodes compliquantes (3.31) demeurent globales.

Si on décompose ce modèle avec le principe de décomposition de Dantzig-Wolfe, en transférant les contraintes locales (3.30) à l'intérieur de $K$ sous-problèmes de plus

court chemin, construits comme celui utilisé pour $DW(F)$, on obtient ce modèle de génération de colonnes :

$$z_{DW(G_K)} = \min \sum_{k=1}^{K} \sum_{p \in \Omega_k} (\sum_{v \in V_k} c_v x_v^p) \theta_p^k + \sum_{v \in V_c} c_v x_v \tag{3.33}$$

$$\sum_{k=1}^{K} \sum_{p \in \Omega_k} \left( \sum_{v \in V_k} a_v^j x_v^p \right) \theta_p^k + \sum_{v \in V_c} a_v^j x_v = 1, \quad j = 1, \ldots, n \tag{3.34}$$

$$\sum_{k=1}^{K} \sum_{p \in \Omega_k} \left( \sum_{v \in V_k(t)} x_v^p \right) \theta_p^k + \sum_{v \in V_c(t)} x_v \leq 1, \quad t \in H_c \tag{3.35}$$

$$\sum_{p \in \Omega_k} \theta_p^k \leq 1, \quad k = 1, \ldots, K \tag{3.36}$$

$$\theta_p^k \in \{0, 1\}, \quad p \in \Omega_k, \quad k = 1, \ldots, K \tag{3.37}$$

$$x_v \in \{0, 1\}, \quad v \in V_c, \quad k = 1, \ldots, K. \tag{3.38}$$

Les contraintes (3.34) assurent que chaque pièce est traitée une et une seule fois. Les contraintes (3.35) modélisent la contrainte de capacité de la machine pendant les périodes compliquantes. Comme le nombre de périodes compliquantes est habituellement moins élevé que le nombre total de périodes, on obtient après cette décomposition de Dantzig-Wolfe un modèle comptant un peu plus de contraintes que $DW(F)$, mais comptant $K$ sous-problèmes de plus petites tailles, au lieu d'un seul de plus grande taille comme pour $DW(F)$. Il y a donc moyen de désagréger le sous-problème initial en autant de sous-problèmes que l'on souhaite en remontant au niveau du problème maître certains contraintes locales du sous-problème pour ainsi obtenir les modèles $G_K$.

On a testé cette approche sur des problèmes de $1|\cdot|\sum w_j T_j$ comportant 40, 50 et 100 pièces. En utilisant les contraintes (3.35) pour empêcher la violation de la capacité de la machine pendant les périodes compliquantes, les résultats sont décevants. Le temps sauvé à générer des colonnes ne compense pas l'augmentation du temps de résolution

du problème maître. Par contre, en remplaçant ces contraintes par des contraintes de flot semblables à celles utilisées dans le modèle $F$ qui sont équivalentes à (3.35), mais beaucoup moins denses, on arrive à diminuer le temps passé à résoudre la relaxation linéaire. On a observé, au cours d'expériences préliminaires, des temps de calcul un peu plus faibles grâce à cette décomposition mais rien de vraiment spectaculaire.

### 3.3.2 Une décomposition temporelle relaxée

On propose ici une version relaxée du mécanisme de décomposition temporelle présenté précédemment qui permet d'améliorer considérablement les temps de calcul.

On a vu que le principal irritant de la décomposition temporelle exacte réside dans l'ajout de contraintes au problème maître pour modéliser la capacité de la machine pendant les périodes compliquantes. On remarque cependant que s'il n'y a pas de tâches qui chevauchent deux horizons de planification (donc si $V_c = \emptyset$), alors ces contraintes ne sont plus nécessaires.

On considère toujours la partition de $H$ en $K$ sous-horizons de planification. Si la tâche $v$ associée à la pièce $j \in N$ est compliquante, cela implique que la tâche $v$ se déroule au moins au cours de $m$ sous-horizons (où $m > 1$), c.-à-d. au cours des horizons $H_k, \ldots, H_{k+m-1}$. L'idée consiste à remplacer $v$ par les $m$ tâches $v_i$ où

$$l(v_i) = \max\{l(v), T_{i-1}\}, u(v_i) = \min\{u(v), T_i\}, a^j_{v_i} = \frac{u(v_i) - l(v_i) + 1}{p_j}, c_{v_i} = a^j_{v_i} c_v,$$

pour $i = k, \ldots, k+m-1$ et à tolérer le fait que les $k$ tâches $v_k$ ne soient pas effectuées consécutivement. En découpant ainsi toutes les tâches $v \in V_c$ chevauchant les deux horizons de planification, on arrive ainsi à séparer complètement les contraintes de

capacité de la machine pour obtenir le modèle suivant :

$$z_{G_K^r} = \min \sum_{k=1}^{K} \sum_{v \in V_k} c_v x_v \tag{3.39}$$

$$\sum_{k=1}^{K} \sum_{v \in V_k} a_v^j x_v = 1, \quad j = 1, \ldots, n \tag{3.40}$$

$$\sum_{v \in V_k : t \in [l(v), u(v)]} x_v \leq 1, \quad t \in H_k, \quad k = 1, \ldots, K \tag{3.41}$$

$$x_v \in \{0, 1\}, \quad v \in V \tag{3.42}$$

ou cette version décomposée selon le principe de Dantzig-Wolfe :

$$z_{DW(G_K^r)} = \min \sum_{k=1}^{K} \sum_{p \in \Omega_k} (\sum_{v \in V_k} c_v x_v^p) \theta_p^k \tag{3.43}$$

$$\sum_{k=1}^{K} \sum_{p \in \Omega_k} (\sum_{v \in V_k} a_v^j x_v^p) \theta_p^k = 1, \quad j = 1, \ldots, n \tag{3.44}$$

$$\sum_{p \in \Omega_k} \theta_p^k = 1, \quad k = 1, \ldots, K \tag{3.45}$$

$$\theta_p^k \in \{0, 1\}, \quad p \in \Omega_k, \quad k = 1, \ldots, K \tag{3.46}$$

qui est une relaxation de $DW(G_K)$ mais qui peut être plus facile à résoudre.

### 3.3.3 Résultats numériques

On a appliqué cette approche en vue de trouver des bornes inférieures de programmation linéaire pour des exemplaires de $1|\cdot|w_j T_j$ comptant 100 pièces avec $p_{max} = 100$. Ces exemplaires proviennent de Potts et Van Wassenhove (1985) et ont été construits selon la méthode suivante. Les durées de traitement $p_j$ et les poids $w_j$ pour les pièces

$j = 1, \ldots, n$ sont générés aléatoirement respectivement selon une distribution uniforme $[1, 100]$ et $[1, 10]$. Les dates de livraison sont générées selon une distribution $[T(1 - TF - RDD/2), T(1 - TF + RDD/2)]$, où $TF$ (pour *tardiness factor*) et $RDD$ (pour *relative range of due dates*) sont des paramètres pouvant prendre comme valeur $\{0.2, 0.4, 0.6, 0.8, 1\}$ et $T$ est la longueur de l'horizon de planification. En ajustant ces paramètres, on bâtit des classes de problèmes de difficultés différentes. Pour chaque paire de paramètres, Potts et Van Wassenhove (1985) ont généré cinq problèmes. Dans cette banque de problèmes, il y a donc 125 exemplaires, tous disponibles sur *OR-LIBRARY* (http ://people.brunel.ac.uk/ mastjjb/jeb/info.html).

On a testé l'approche sur le premier problème de chaque classe. Pour ces 25 problèmes, on a résolu la relaxation linéaire de la formulation décomposée pour cinq partitions différentes de l'horizon de planification, soit $K = 1, 2, 4, 6, 8, 10$. Une limite de 100 000 itérations de génération de colonnes a été imposée. Les périodes $T_K$ nécessaires pour la définition de la partition de l'horizon

$$H_1 = T_0, \ldots, T_1, \; H_2 = T_1, \ldots, T_2, \; \ldots, \; H_K = T_{K-1}, \ldots, T_K,$$

ont été générées comme suit :

$$\begin{aligned} &T_0 = 1 \\ &T_k = T_{k-1} + \lfloor \tfrac{T}{K} \rfloor, k = 2, \ldots, K-1 \\ &T_K = T. \end{aligned}$$

Finalement, on doit noter qu'une procédure de prétraitement, qui sera expliquée dans la prochaine section, a été appliquée afin d'éliminer certains arcs dominés.

L'algorithme de génération de colonnes a été mis en place à l'aide de NetGen6.1, qui est une collection de librairies qui facilitent le développement d'applications pour GENCOL-CPLEX 9.1, un générateur de colonnes développé pour résoudre des problèmes de transport et de fabrication d'horaires de personnel (Villeneuve, 1999). Tous

les tests ont été exécutés sur un PC PENTIUM 4 doté d'un micro-processeur de 3.4 Ghz.

Tableau 3.1 – Temps CPU (*sec*) obtenus avec différentes valeurs pour $K$

| *Id* | $K$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 6 | 8 | 10 |
| 1 | 46.5 | 16.9 | 6.5 | 4.0 | 3.6 | 2.9 |
| 6 | 315.7 | 83.2 | 34.0 | 15.9 | 12.7 | 9.6 |
| 11 | 922.8 | 212.0 | 45.2 | 26.0 | 19.7 | 14.8 |
| 16 | **28572.5** | 1469.7 | 309.7 | 95.3 | 63.4 | 36.9 |
| 21 | **12917.6** | **25038.1** | 167.6 | 63.5 | 39.2 | 24.9 |
| 26 | 1.7 | 1.4 | 0.7 | 0.4 | 0.3 | 0.2 |
| 31 | 124.2 | 37.7 | 20.6 | 10.5 | 9.0 | 7.0 |
| 36 | 1040.9 | 584.4 | 64.5 | 37.6 | 28.9 | 18.2 |
| 41 | **22326.5** | 1209.6 | 142.3 | 67.3 | 37.9 | 31.1 |
| 46 | **17353.3** | **4523.7** | 143.4 | 70.6 | 35.2 | 28.2 |
| 51 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |
| 56 | 92.2 | 43.4 | 15.9 | 9.4 | 6.7 | 5.3 |
| 61 | **26201.2** | 557.4 | 132.8 | 46.3 | 34.9 | 32.9 |
| 66 | **32920.1** | 1425.2 | 151.6 | 62.6 | 34.6 | 27.4 |
| 71 | **28814.5** | 600.1 | 143.8 | 53.7 | 39.5 | 27.8 |
| 76 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 |
| 81 | 33.9 | 14.9 | 5.9 | 3.2 | 2.9 | 2.2 |
| 86 | 1591.7 | 333.6 | 85.4 | 36.6 | 24.2 | 16.8 |
| 91 | **27097.2** | 6166.1 | 127.6 | 70.3 | 43.0 | 37.3 |
| 96 | **25149.2** | 1889.2 | 166.7 | 86.9 | 51.4 | 47.0 |
| 101 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 |
| 106 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 |
| 111 | 2695.3 | 462.5 | 101.0 | 54.2 | 40.8 | 27.9 |
| 116 | 3848.2 | 414.5 | 111.9 | 57.3 | 43.4 | 35.2 |
| 121 | **26806.1** | 1138.1 | 142.2 | 79.1 | 53.0 | 40.7 |

Les résultats sont présentés aux tableaux 3.1 et 3.2. Le tableau 3.1 présente le temps *cpu* requis pour trouver la relaxation linéaire des divers exemplaires lorsque $K$ sous-problèmes ont été utilisés. Le tableau 3.2 présente les sauts d'intégrité obtenus, définis comme

$$\frac{\bar{z} - \underline{z}}{\bar{z} + 0.000001} * 100,$$

où $\bar{z}$ est une borne supérieure connue du problème et 0.000001 est un petit nombre utilisé pour éviter les divisions par 0 pour les exemplaires sans retard. Le saut d'intégrité représente une mesure de la qualité de la relaxation linéaire.

Tableau 3.2 – Sauts obtenus (%) avec différentes valeurs pour $K$

| *id* | $K$ | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 6 | 8 | 10 |
| 1 | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 | 0.68 |
| 6 | 0.00 | 0.00 | 0.07 | 0.01 | 0.07 | 0.20 |
| 11 | 0.07 | 0.14 | 0.14 | 0.14 | 0.17 | 0.16 |
| 16 | **-0.02** | 0.06 | 0.06 | 0.08 | 0.06 | 0.08 |
| 21 | **-0.10** | **0.001** | 0.01 | 0.01 | 0.01 | 0.01 |
| 26 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 31 | 1.07 | 1.07 | 1.09 | 1.16 | 1.20 | 1.21 |
| 36 | 0.03 | 0.11 | 0.13 | 0.16 | 0.19 | 0.21 |
| 41 | **0.06** | 0.10 | 0.10 | 0.13 | 0.11 | 0.14 |
| 46 | **-0.15** | **-0.09** | 0.02 | 0.03 | 0.03 | 0.05 |
| 51 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 56 | 0.19 | 0.19 | 0.19 | 0.19 | 0.19 | 0.75 |
| 61 | **0.47** | 0.47 | 0.47 | 0.51 | 0.53 | 0.53 |
| 66 | **0.16** | 0.21 | 0.24 | 0.27 | 0.27 | 0.31 |
| 71 | **-0.05** | 0.01 | 0.02 | 0.03 | 0.04 | 0.04 |
| 76 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 81 | 18.12 | 18.12 | 18.12 | 18.13 | 18.95 | 18.14 |
| 86 | 0.69 | 0.70 | 0.75 | 0.89 | 0.90 | 0.86 |
| 91 | **0.22** | 0.24 | 0.25 | 0.27 | 0.33 | 0.35 |
| 96 | **0.003** | 0.05 | 0.07 | 0.08 | 0.09 | 0.10 |
| 101 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 106 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 111 | 0.23 | 0.24 | 0.26 | 0.34 | 0.35 | 0.41 |
| 116 | 0.09 | 0.09 | 0.11 | 0.13 | 0.15 | 0.17 |
| 121 | **0.04** | 0.07 | 0.09 | 0.11 | 0.11 | 0.13 |

Pour certains exemplaires et certaines partitions, il n'a pas été possible de trouver la relaxation linéaire exacte à l'intérieur des 100 000 itérations. Dans ces cas, les résultats sont indiqués en caractères gras.

Les résultats numériques montrent clairement l'efficacité de la décomposition temporelle. Pour tous les exemplaires, le temps de calcul est considérablement réduit lorsqu'on désagrège le sous-problème. Par exemple, pour les exemplaires 16, 21, 46 et 71, 100 000 itérations de génération de colonnes (ce qui représente plus de 2 heures de calcul dans les quatre cas) n'ont pas été suffisantes pour abaisser l'objectif sous la valeur de la meilleure borne supérieure connue pour $K = 1$, tandis que des bornes

inférieures de très bonne qualité (saut d'intégrité $< 0.05\%$) ont été trouvées en moins d'une minute en ayant recours à une désagrégation aggressive ($K = 8, 10$).

Comme on pouvait s'y s'attendre, la qualité des bornes diminue lorsqu'on augmente le nombre de sous-problèmes utilisés. Cependant, les résultats du tableau indiquent que la détérioration de la borne n'est pas dramatique (typiquement $< 0.2\%$).

Dans la prochaine section, on montre comment l'approche testée ici pour le calcul de bornes peut être implantée dans un algorithme d'énumération implicite afin de résoudre optimalement le problème $1|\cdot|\sum w_j T_j$.

## 3.4 Un algorithme d'énumération implicite pour $1|\cdot|\sum w_j T_j$

Le problème $1|\cdot|\sum w_j T_j$, consistant à trouver la séquence de pièces minimisant la somme pondérée des retards est *NP-difficile* au sens fort. Des approches de résolution basées sur la programmation dynamique (Lawler, 1979 ; Schrage et Baker, 1978 ; Gelinas et Soumis, 1996) ou basées sur l'énumération implicite (Rinnooy Kan *et al.*, 1975 ; Shwimer, 1972 ; Potts et Van Wassenhove, 1985 ; Babu *et al.*, 2004) ont été proposées afin de résoudre ce problème. Avec ces algorithmes, on est capable de résoudre au mieux des exemplaires comptant environ cinquante pièces.

Au-dessus de ce nombre, l'algorithme d'énumération implicite développé par Potts et Van Wassenhove (1985), considéré actuellement comme le meilleur, qui utilise une borne faible mais facilement calculable, génère des arbres de recherche de trop grande taille (Congram *et al.*, 2002). L'utilisation de meilleures bornes, se calculant en temps pseudo-polynomial, semble donc justifiable. Les résultats récents obtenus par Babu *et al.* (2004) semblent le confirmer. En utilisant une borne lagrangienne dérivée de $G_1$, ceux-ci ont en effet été capables de résoudre des problèmes de 40 et 50 pièces qui n'avaient encore jamais été résolus.

On peut modéliser le problème $1|\cdot|\sum w_jT_j$ à l'aide de $G_1$ en posant le coût $c_{jt}$ associé à la décision $(j,t)$ à $\max\{w_j(t+p_j-d_j-1),0\}$. De plus, comme il existe toujours une solution optimale à ce problème qui est sans attente, on peut poser $T=\sum_{j\in N}p_j$ et remplacer dans le modèle compact $G_1$ les inégalités (3.26) par des égalités. En appliquant le principe de décomposition de Dantzig-Wolfe et le principe de décomposition temporelle, on peut obtenir des formulations $DW(G_K^r)$ valides pour borner $1|\cdot|\sum w_jT_j$, dont les relaxations linéaires peuvent être plus faciles à évaluer. La solution de la relaxation linéaire de $DW(G_1)$ est susceptible de violer plusieurs contraintes d'intégrité. Encore pire, celle de $DW(G_K^r)$ est susceptible de violer en plus des contraintes de contiguïté (c.-à-d. qu'il peut y avoir de la préemption dans la solution de la relaxation linéaire). Il est donc nécessaire d'intégrer ces modèles à un algorithme d'énumération implicite afin de faire imposer ces contraintes violées. On présente dans cette section un mécanisme de branchement qui peut exploiter la borne donnée par n'importe quel modèle que l'on peut dériver de $G_K$ que ce soit avec le principe de Dantzig-Wolfe ou encore avec le principe de décomposition temporelle. Pour développer cet algorithme, on s'est inspiré d'une règle de branchement classique développée pour le problème $1|\cdot|\sum w_jT_j$, qui consiste à construire les séquences à rebours (c.-à-d. qu'on débute en fixant la pièce qui est la dernière dans la séquence puis on poursuit avec l'avant-dernière, et ainsi de suite). À chacun des noeuds de l'arbre, on calcule la relaxation linéaire exacte de l'un des modèles de type temps indexé adapté à $1|\cdot|\sum w_jT_j$ afin d'obtenir en premier lieu une borne inférieure sur l'objectif du problème qui indique le potentiel d'un noeud et en deuxième lieu une solution approchée (mais peut-être non réalisable) au problème qui peut aider à déterminer sur quelle variable il faut brancher.

### 3.4.1 Règle de branchement

La règle de branchement consiste à choisir la variable $x_f$ associée à la tâche se terminant le plus tard et qui n'a pas encore été fixée. Si $x_f$ ne viole pas une contrainte de contiguïté, l'algorithme impose sur une branche que $x_f=1$ et sur l'autre que $x_f=0$. Autrement, si $x_f$ viole une contrainte de contiguité $x_f=x_g$, où $f$ et $g$ sont deux

tâches que l'on a obtenu en découpant une tâche compliquante, alors l'algorithme impose sur une branche que $x_f = x_g = 1$ et sur l'autre que $x_f = x_g = 0$. Si plusieurs pièces peuvent être fixées en dernière position, on accorde la priorité aux pièces qui sont à l'heure. Si plusieurs pièces peuvent être fixées en dernière position à l'heure, on choisit la pièce dont la date de livraison est la plus tardive (soit la dernière pièce dans l'ordre $EDD$).

Si on utilise un modèle décomposé avec le principe de Dantzig-Wolfe, la décision $x_f = 0$ peut être appliquée en enlevant dans l'un des sous-problèmes l'arc associé à la tâche $f$ tandis que la décision $x_f = 1$ peut être appliquée en enlevant tous les arcs associés aux tâches $t \in T$ tels que $t$ se déroule pendant la tâche $f$ ou tels que $t$ revient à traiter la même pièce que $f$. Les décisions $x_f = x_g = 0$ et $x_f = x_g = 1$ s'appliquent de façon semblable.

On remarque qu'avec cette règle de branchement, il peut arriver de brancher sur des variables qui sont déjà entières. Ce type de branchement, peu commun en programmation en nombres entiers, permet ici d'utiliser les puissantes règles de dominance développées pour le problème $1|\cdot|\sum w_j T_j$.

### 3.4.2 Quelques règles de dominance

La plupart des algorithmes efficaces pour résoudre le problème $1|\cdot|\sum w_j T_j$ ont recours à des règles de dominance, afin de réduire le domaine des solutions à explorer. Dans cette section, on montre comment certaines de ces règles peuvent être intégrées à l'algorithme d'énumération implicite.

Les règles suivantes, communément appelées règles d'Emmons (même si elles ont en fait été déduites par Rinnooy Kan *et al.*, 1975 à partir des travaux d'Emmons sur $1|\cdot|\sum T_j$) constituent les trois règles de dominance classiques pour $1|\cdot|\sum w_j T_j$. Soit $N$ l'ensemble des pièces, $B_j$ l'ensemble des pièces devant être traitées avant la pièce $j$ et $A_j$ l'ensemble des pièces devant être traitées après $j$.

Règles d'Emmons :

*Il y a une séquence optimale dans laquelle la pièce $i$ est traitée avant la pièce $j$, si l'une des trois conditions ci-dessous est respectée :*

**(R1)** $p_i \leq p_j, w_i \geq w_j, d_i \leq \max\{d_j, \sum_{h \in B_j} p_h + p_j\}$

**(R2)** $w_i \geq w_j, d_i \leq d_j, d_j \geq \sum_{h \in N \setminus A_i} p_h - p_j$

**(R3)** $d_j \geq \sum_{h \in N \setminus A_i} p_h.$

Ces trois règles peuvent donc être utiles pour construire le graphe des préséances entre les pièces. Pour construire ce graphe, il suffit d'identifier successivement les relations de préséance entre différentes paires de pièces. Afin d'obtenir un graphe acyclique (pour une paire de pièces, une règle en contredit parfois une autre), il est important lors de la construction de ne considérer que les paires de pièces pour lesquelles aucune relation de préséance n'a encore été trouvée.

Ce graphe de préséance entre les pièces permet entre autres de dériver des moments de complétion le plus tôt ($C_j^E = \sum_{h \in B_j} p_h + p_j$) et des moments de complétion le plus tard $C_j^L = \sum_{h \in N \setminus A_j} p_h$) pour les pièces $j \in N$ (Abdul-Razaq *et al.*, 1990). On peut se servir de ces moments de complétion afin de se débarasser de certains arcs, en prétraitement ou à un noeud de l'arbre d'énumération. Ces règles peuvent même être utilisées afin de fixer certaines variables. Le lemme d'Elmaghraby (1968), corrolaire de la règle **(R3)** qui stipule qu'il existe toujours une séquence optimale dans laquelle la pièce $j$ est planifiée en dernier si $d_j \geq \sum_{h \in N} p_h$, peut être invoqué afin d'ignorer la branche $x_f = 0$ dans l'algorithme de branchement si $x_f$ consiste à planifier la pièce $j$ sans retard à la dernière position.

D'autres règles de dominance, basées sur des résultats classiques de la théorie de l'ordonnancement, peuvent aussi être développées. Si toutes les pièces peuvent être traitées sans retard, on sait dans ce cas que la règle $EDD$ (Jackson, 1955) est optimale

tandis que si toutes les pièces ne peuvent jamais être traitées sans retard, on sait que c'est la règle de Smith (Smith, 1956) qui est optimale. Une solution optimale pour $1|\cdot|\sum w_jT_j$ dans laquelle toutes les pièces sont sans retard peut donc être générée avec la règle $EDD$. Par contre, une solution optimale pour $1|\cdot|\sum w_jT_j$ dans laquelle toutes les pièces sont planifiées avec retard ne peut pas nécessairement être générée avec la règle de Smith, comme le démontre ce petit exemple à deux pièces :

$$1 : w_1 = 2, p_1 = 5, d_1 = 0$$
$$2 : w_2 = 1, p_2 = 2, d_2 = 6.$$

Il existe deux solutions pour cet exemplaire : $\sigma_1 = 1|2$ ou $\sigma_2 = 2|1$. Le coût de $\sigma_1$ est

$$c(\sigma_1) = 2\max\{0, 5-0\} + \max\{0, 7-6\} = 11$$

tandis que le coût de $\sigma_2$ est

$$c(\sigma_2) = \max\{0, 2-6\} + 2\max\{0, 7-0\} = 14.$$

On remarque que la séquence $\sigma_1$ est la solution optimale unique. Or, dans cette séquence les deux pièces sont en retard, mais elles ne sont pas dans l'ordre de Smith ($\frac{w_1}{p_1}) = 0.4 \not\geq \frac{w_2}{p_2} = 0.5$). La règle de Smith est seulement valide dans le cas où toutes les pièces ne peuvent jamais être planifiées à l'heure, c.-à-d. lorsque $d_j < p_j, \forall j \in N$. On insiste sur ce fait car certains auteurs (voir remarque 3 de Rachamadugu, 1987 et théorème 2.4 de McNaughton, 1959) ont soutenu le contraire.

Pour $1|\cdot|\sum w_jT_j$, la règle $EDD$ implique que les pièces traitées entièrement durant l'intervalle $[0, d_{min}]$, où $d_{min}$ est la date de livraison la plus petite du problème, peuvent être séquencées selon leur date de livraison (puisque toutes les pièces entièrement traitées dans cet intervalle sont nécessairement à l'heure). De même, on a que toutes les pièces traitées entièrement durant l'intervalle $[d_{max}, T]$, où $d_{max}$ est la plus grande date de livraison du problème, peuvent être séquencées dans l'ordre de Smith (puisque toutes les pièces traitées dans cet intervalle sont nécessairement en retard).

En se basant sur ces résultats, on peut formuler ces quatre règles de dominance. Soit $edd(j)$ la position de la pièce $j$ dans l'ordre $EDD$ et $smith(j)$ la position de la pièce $j$ dans l'ordre de Smith.

Nouvelles règles :

**(R4)** Si $C_j \leq d_{min}$ alors $C_j \leq \sum_{h \in N : edd(h) \leq edd(j)} p_h$ ;

**(R5)** Si $C_j \leq d_{min}$ alors $C_j \geq d_{min} - p_{max} + 1 - \sum_{h \in N : edd(h) > edd(j)} p_h$ ;

**(R6)** Si $C_j > d_{max}$ alors $C_j \leq d_{max} + p_{max} - 1 + \sum_{h \in N : smith(h) \leq smith(j)} p_h$ ;

**(R7)** Si $C_j > d_{max}$ alors $C_j \geq T - \sum_{h \in N : smith(h) > smith(j)} p_h$.

*Preuve* : Si $C_j \leq d_{min}$, alors toutes les pièces traitées avant $j$ sont nécessairement sans retard et peuvent toujours être exécutées dans l'ordre $EDD$.

(R4) Si $C_j \leq d_{min}$ et $C_j > \sum_{h \in N : edd(h) \leq edd(j)} p_h$, cela implique qu'il y a une pièce $h \in N$ telle que $edd(h) > edd(j)$ qui est traitée avant $j$, violant ainsi l'ordre $EDD$.

(R5) Si $C_j \leq d_{min}$ et $C_j < d_{min} - p_{max} + 1 - \sum_{h \in N : edd(h) > edd(j)} p_h$, alors toutes les pièces $h \in N$ telles que $edd(h) > edd(j)$ et au moins une pièce $h$ telle que $edd(h) < edd(j)$ peuvent être traitées sans retard avant $j$, violant ainsi l'ordre $EDD$.

Si $C_j > d_{max}$, alors toutes les pièces traitées après $j$ sont nécessairement en retard et peuvent être exécutées dans l'ordre de Smith.

(R6) Si $C_j > d_{max}$ et $C_j > d_{max} + p_{max} - 1 + \sum_{h \in N : smith(h) \leq smith(j)} p_h$, alors toutes les pièces $h \in N$ telles que $smith(h) \leq smith(j)$ et au moins une pièce $h$ telle que $smith(h) > smith(j)$ peuvent être traitées avant $j$, violant ainsi l'ordre de Smith.

(R7) Si $C_j > d_{max}$ et $C_j < T - \sum_{h \in N : smith(h) > smith(j)} p_h$, cela implique qu'il y a une pièce $h \in N$ telle que $smith(h) < smith(j)$ qui est traitée avant $j$ (parce que la

longueur de l'horizon est $T$), violant ainsi la règle de Smith. □

On doit souligner que Babu *et al.* (2004) ont proposé des règles de dominance semblables.

Il est malheureusement impossible d'utiliser simultanément ces nouvelles règles avec les règles d'Emmons. En fait, puisqu'elles utilisent des arguments différents, il est parfois possible d'obtenir des résultats contradictoires. Cependant, Babu *et al.* (2004) ont proposé cette version plus faible des règles d'Emmons.

**Règles de Babu :**

*Il y a une séquence optimale dans laquelle la pièce $i$ est traitée avant la pièce $j$, si l'une des trois conditions ci-dessous est respectée :*

**(R8)** $p_i \leq p_j, edd(i) \leq edd(j), smith(i) \leq smith(j)$ ;

**(R9)** $w_i \geq w_j, edd(i) \leq edd(j), d_j \geq \sum_{h \in N \setminus A_i} p_h - p_j$ ;

**(R10)** $d_j \geq \sum_{h \in N \setminus A_i} p_h, edd(i) \leq edd(j)$

En utilisant cette version des règles d'Emmons, les règles (R4), (R5), (R6), (R7) (R8), (R9), (R10) ne se contredisent plus.

D'autres règles de dominance, basées sur la notion d'échange, ont été proposées dans la littérature. La règle $API$ par exemple (pour *adjacent pairwise interchange*) considère la sous-séquence $(i, j)$ et teste si la sous-séquence $(j, i)$ est moins coûteuse. Si oui, la séquence $(i, j)$ est dominée. Cette notion d'échange peut aussi être appliquée à deux pièces $i$ et $j$ non adjacentes si $p_j \leq p_i$ et $C_i < C_j$. L'argument d'échange peut également être généralisé pour tenir compte de plus de deux pièces adjacentes. En fait, s'il est possible de réarranger une séquence partielle $\sigma_1$ en une séquence $\sigma_2$ moins coûteuse, alors $\sigma_1$ est dominée.

On peut finalement développer des règles de dominance basées sur les coûts réduits

des arcs.

Soit $UB$ une borne supérieure pour le problème et $LB(v)$ une borne inférieure pour le problème lorsqu'on force la tâche $v$ à être planifiée (c.-à-d. la variable associée à cette tâche $x_v$ est fixée à 1). Si $LB(v) \geq UB$ alors on peut fixer $x_v = 0$ puisqu'il est impossible d'améliorer la borne supérieure en fixant cette variable à 1.

Pour obtenir une borne inférieure pour le problème, on peut se servir des techniques de relaxation lagrangienne. Si on relaxe les contraintes (3.25) de la formulation $F$ en utilisant le vecteur de multiplicateurs de Lagrange $\pi$, on obtient la relaxation de $F$,

$$z_{Lag}(\pi) = \min \sum_{v \in V} (c_v - \sum_{v \in V} a_v^j \pi_j) x_v + \sum_{j \in N} \pi_j \tag{3.47}$$

$$\sum_{v \in V : t \in [l(v), u(v)]} x_v \leq 1, \quad t = 1, \ldots, T \tag{3.48}$$

$$x_v \in \{0, 1\}, \quad v \in V \tag{3.49}$$

qui peut se résoudre comme un problème de plus court chemin dans un réseau identique à celui utilisé comme sous-problème pour $DW(F)$ car $\sum_{j \in N} \pi_j$ est une constante pour un vecteur $\pi$ donné. On rappelle que dans ce réseau, il y a un arc pour chaque tâche $v$ allant du noeud associé à la période $l(v)$ au noeud associé à $u(v) + 1$.

On peut se servir de cette relaxation lagrangienne afin d'obtenir les bornes $LB(v)$ qui vont permettre d'éliminer des arcs du sous-problème.

Soit $\lambda_t$ la distance du noeud source au noeud associé à la période $t$ dans le réseau du puits et $\mu_t$ la distance du noeud puits au noeud associé à $t$. Alors

$$LB(v) = \lambda_{l(v)} + c'(v) + \mu_{u(v)+1} + \sum_{j \in N} \pi_j$$

où $c'_v = (c_v - \sum_{j \in N} a_v^j \pi_j)$, est une borne inférieure valide au problème lorsque $x_v = 1$

par la définition même de $\lambda$ et de $\mu$.

Donc si

$$LB(v) = \lambda_{l(v)} + c'(v) + \mu_{u(v)+1} + \sum_{j \in N} \pi_j \geq UB,$$

alors on peut enlever l'arc associé à la tâche $v$ dans le sous-problème.

On voit donc que pour appliquer cette règle de dominance, il suffit, si on dispose d'une borne supérieure et d'un vecteur de multiplicateurs, d'effectuer deux passes de l'algorithme de plus court chemin sur graphe acyclique afin de trouver les vecteurs de distance $\lambda$ et $\mu$.

Dans l'algorithme d'énumération implicite développé ici, que l'on a appelé algorithme *CGTWT* (pour *Column Generation for Total Weighted Tardiness*), les règles de dominance (R4), (R5), (R6), (R7) (R8), (R9), et (R10) sont utilisées en prétraitement tandis que les règles (R8), (R9), (R10) et *API* sont invoquées à chaque noeud de l'arbre d'énumération où une variable est fixée à 1 afin d'éliminer certains arcs dominés. De plus, la règle de dominance basée sur les coûts réduits est appliquée à tous les noeuds de l'arbre en utilisant comme borne supérieure la meilleure solution courante et comme vecteur de multiplicateurs le vecteur des variables duales des contraintes (3.25) obtenu pour le problème maître optimal du noeud parent. Le lemme d'Elmaghraby est finalement appliqué à chaque noeud où l'on considère une pièce qui est traitée à temps. Cela est possible puisque la règle de branchement utilisée fait en sorte que les séquences dans lesquelles les pièces à l'heure sont ordonnées selon l'ordre *EDD* sont toujours considérées en premier par l'algorithme. On a finalement utilisé l'argument de $k$-échanges comme suit. Si pour un noeud de l'arbre de branchement, on trouve, en permutant trois éléments de la séquence partielle qui a été fixée, notée $\sigma_1$, une séquence $\sigma_2$ de moindre coût, alors on supprime le noeud.

L'encadré 3.1 décrit plus formellement l'algorithme *CGTWT*.

**Algorithme 3.1** Pseudo-code de l'algorithme *CGTWT*

Soit $\mathcal{L}$ la liste des noeuds, $X_0^i$ la liste des variables fixées à 0 au noeud $n_i$, $X_1^i$ la liste des variables fixées à 1, $\bar{z}$ une borne supérieure pour le problème et $\underline{z}^i$ la borne inférieure pour le noeud $n_i$.

1. **(Initialisation)** $X_0^0 = \emptyset$, $X_1^0 = \emptyset$, $\mathcal{L} = \{n_0\}$, $\bar{z} = \infty$, $\underline{z}^0 = -\infty$.

2. **(Critère d'optimalité)** Si $\mathcal{L} = \emptyset$, $\bar{z}$ est optimale. **STOP**.

3. **(Sélection d'un noeud)** Extraire de $\mathcal{L}$ le noeud $n_i$ qui a été ajouté en dernier.

4. **(Dominance 1)** Si $n_i == n_0$, mettre à jour l'ensemble des variables $X_0^i$ en vertu des règles (R4), (R5), (R6), (R7).

5. **(Dominance 2)** Appliquer les règles de Babu *et al.* (2004) et mettre à jour $X_0^i$.

6. **(Dominance 3)** Appliquer la règle $API$ et mettre à jour $X_0^i$.

7. **(Dominance 4)** Appliquer la règle de dominance basée sur les coûts réduits et mettre à jour $X_0^i$.

8. **(Calcul de la borne)** Résoudre la relaxation linéaire de $DW(G_k^r)$ en tenant compte des ensembles $X_0^i$ et $X_1^i$ et mettre à jour $\underline{z}^i$ avec la valeur optimale de la relaxation linéaire.

9. **(Évaluation de la solution)**
   **a)** Si $\underline{z}^i \geq \bar{z}$, retourner à l'étape 2.
   **b)** Si la solution de la relaxation linéaire de $DW(G_K^r)$ est réalisable et sans préemption, alors poser $\bar{z} = \underline{z}^i$, éliminer tous les noeuds $n_k \in \mathcal{L}$ tel que $\underline{z}^k \geq \bar{z}$ et retourner à l'etape 2.

10. **(Division)** Soit $x_f$ la variable sur laquelle on doit brancher (voir section 3.4.1). Créer les noeuds fils $k$ et $l$. Pour le fils $k$, poser $X_0^k = X_0^i \cup \{x_f\}$, $X_1^k = X_1^i$ et $\underline{z}^k = \underline{z}^i$. Pour le fils $l$, poser $X_0^k = X_0^i$, $X_1^k = X_1^i \cup \{x_f\}$ et $\underline{z}^l = \underline{z}^i$.

11. **(Mise à jour de la liste)**
   **a)** Si $c_f = 0$, ajouter seulement le noeud $l$ à $\mathcal{L}$ (lemme d'Elmaghraby) et retourner à l'étape 2.
   **b)** Si $c_f > 0$, ajouter le fils $k$ à $\mathcal{L}$. Ajouter également le fils $l$ à $\mathcal{L}$ s'il n'est pas dominé en vertu de la règle de 3-échanges. Retourner à l'étape 2.

### 3.4.3 Résultats numériques

L'algorithme de branchement *CGTWT* a été testé sur les mêmes 25 problèmes utilisés à la section 3.3.3. L'arbre de recherche a été exploré en profondeur d'abord et une borne supérieure très forte a été employée pour chaque problème

$$\bar{z} = z_{OR-Lib} + 1$$

où $z_{OR-Lib}$ est la meilleure solution connue pour celui-ci disponible sur *OR-LIBRARY*. De très bonnes approches heuristiques (par exemple, Congram *et al.*, 2002 et Crauwels *et al.*, 1998) peuvent être utilisées pour trouver cette borne supérieure. Aucune de ces solutions n'est prouvée optimale. Le temps de résolution a été limité à douze heures (43200 secondes) et pour chaque problème, la borne inférieure a été obtenue à partir de la relaxation linéaire du modèle $G^r_{10}$.

Le tableau 3.3 présente le nombre de noeuds explorés (*Nb de noeuds*), le temps de résolution *cpu* (*Temps de résolution*), la meilleure solution trouvée par l'algorithme ($z_{best}$) et le saut d'intégrité (*saut*) pour les 25 problèmes. Les solutions qui ont été prouvées optimales sont marquées d'un crochet dans la colonne *Opt*.

Avec l'algorithme *CGTWT*, on a été en mesure de résoudre, en des temps raisonnables, de nombreux problèmes ouverts. Pour les problèmes 66, 116, l'algorithme n'a cependant pas été capable de prouver la solution optimale, mais a au moins été capable de retrouver la solution de *OR-LIBRARY*.

On a finalement testé l'algorithme sur les 125 exemplaires de *OR-LIBRARY* avec une limite de douze heures de temps de résolution pour chaque problème. On rappelle qu'aucun de ces problèmes n'avait été résolu auparavant à l'optimalité (à l'exception des exemplaires où $z_{best} = 0$). Le tableau 3.4 indique le nombre de problèmes qui ont été résolus à l'optimalité pour chaque classe de problèmes (définie par la valeur des paramètre $TF$ et $RDD$). En tout, 117 problèmes sur 125 ont été résolus à l'optimalité. En analysant ce tableau, on remarque que l'algorithme *CGTWT* est moins efficace avec les exemplaires où $TF$ est élevé. Ces classes de problèmes sont d'ailleurs classifiées

Tableau 3.3 – Résultats obtenus avec $G^r_{10}$

| *Id* | *Nb de noeuds* | *Temps de calcul (sec)* | $z_{best}$ | *Opt* | $\underline{z}$ | *saut (%)* |
|---|---|---|---|---|---|---|
| 1 | 13 | 24.11 | 5988 | ✓ | 5947.53 | 0.68 |
| 6 | 46 | 114.96 | 58258 | ✓ | 58139.2 | 0.2 |
| 11 | 154 | 464.53 | 181649 | ✓ | 181362 | 0.16 |
| 16 | 231 | 466.54 | 407703 | ✓ | 407391 | 0.08 |
| 21 | 238 | 726.62 | 898925 | ✓ | 898820 | 0.01 |
| 26 | 2 | 1.24 | 8 | ✓ | 8 | 0 |
| 31 | 113 | 268.46 | 24202 | ✓ | 23909.7 | 1.22 |
| 36 | 232 | 610.27 | 108293 | ✓ | 108070 | 0.21 |
| 41 | 2267 | 4655.2 | 462324 | ✓ | 461667 | 0.14 |
| 46 | 3502 | 15963.04 | 829828 | ✓ | 829530 | 0.04 |
| 51 | 1 | 0.05 | 0 | ✓ | 0 | 0 |
| 56 | 22 | 81.7 | 9046 | ✓ | 8977.73 | 0.76 |
| 61 | 733 | 1891.66 | 86793 | ✓ | 86330.9 | 0.54 |
| 66 | 13965 | 43200 | 243872 | | 243113 | 0.31 |
| 71 | 1464 | 3964.9 | 640816 | ✓ | 640530 | 0.04 |
| 76 | 1 | 0.04 | 0 | ✓ | 0 | 0 |
| 81 | 51 | 136.16 | 1400 | ✓ | 1145.98 | 22.17 |
| 86 | 738 | 2238.16 | 66850 | ✓ | 66274.5 | 0.87 |
| 91 | 17517 | 31790.1 | 248699 | ✓ | 247838 | 0.35 |
| 96 | 1522 | 3255.34 | 495516 | ✓ | 495035 | 0.1 |
| 101 | 1 | 0.09 | 0 | ✓ | 0 | 0 |
| 106 | 1 | 0.1 | 0 | ✓ | 0 | 0 |
| 111 | 630 | 26492.11 | 159123 | ✓ | 158466 | 0.41 |
| 116 | 17920 | 43200 | 370685 | | 369985 | 0.19 |
| 121 | 3417 | 9866.22 | 471214 | ✓ | 470603 | 0.13 |

comme étant difficiles par Potts et Van Wassenhove (1985) et Babu *et al.* (2004).

## 3.5 Conclusion

Dans ce chapitre, on a vu que l'approche de décomposition temporelle relaxée permettait d'accélérer considérablement le calcul de bornes inférieures fortes pour le problème $1|\cdot|\sum w_j T_j$. De plus, on a montré qu'en imbriquant cette stratégie à l'intérieur d'un algorithme d'énumération implicite, il était possible de résoudre à l'optimalité de

Tableau 3.4 – Nombre d'exemplaires résolus pour chaque classe de problèmes

| | | | *TF* | | |
|---|---|---|---|---|---|
| *RDD* | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
| 0.2 | 5 | 5 | 5 | 5 | 5 |
| 0.4 | 5 | 5 | 5 | 5 | 5 |
| 0.6 | 5 | 5 | 5 | 4 | 5 |
| 0.8 | 5 | 5 | 5 | 4 | 5 |
| 1.0 | 5 | 5 | 3 | 4 | 4 |

nombreux exemplaires de $1|\cdot|\sum w_jT_j$ d'assez grande taille pour lesquels, en premier lieu, on n'était même pas en mesure de calculer la relaxation linéaire associée à leur formulation non décomposée.

Même si on a obtenu avec cet algorithme des résultats intéressants, certains problèmes demeurent très difficiles et de nouvelles stratégies devraient être développées afin de les résoudre. Il serait par exemple intéressant d'ajouter dynamiquement des inégalités valides (coupe de flot à l'interface de deux sous-horizons, coupes issues du problème de stable) afin de diminuer le saut d'intégrité ou de raffiner la règle de branchement et les règles de dominance.

On a aussi vu au début du chapitre que les formulations de type temps indexé étaient très versatiles. Il serait donc intéressant d'adapter l'algorithme d'énumération implicite développé ici à d'autres problèmes difficiles d'ordonnancement. Le problème $1|r_j|w_jC_j$, que van den Akker *et al.* (2000) ont déjà étudié, serait à ce titre un candidat intéressant. En intégrant la borne donnée par la relaxation linéaire de $DW(G_k^r)$ à un algorithme d'énumération implicite utilisant une règle de branchement et des règles de dominance spécialisées pour $1|r_j|w_jC_j$ (voir Beloudouah et Potts, 1992), on croit qu'il serait possible d'obtenir une approche compétitive avec les meilleurs algorithmes exacts développés pour ce problème $1|r_j|w_jC_j$, qui sont actuellement capables de résoudre des problèmes d'environ 60 pièces.

# CHAPITRE 4 : ORDONNANCEMENT SUR UNE MACHINE II

## 4.1 Introduction

On a vu au chapitre précédent comment diverses formulations de programmation en nombres entiers issues du problème du stable de poids maximum pouvaient être utiles pour résoudre certains problèmes d'ordonnancement sur une machine sans préemption. Ces formulations supposent cependant que le temps de réglage sont indépendants de la séquence. Or, il existe des contextes où les temps de réglages dépendent de la séquence. Dans l'industrie de l'aluminium (Gagné *et al.*, 2001) ou du plastique (Rubin et Ragatz, 1995), on doit souvent prévoir un temps de réglage $s_{ij}$ si l'on décide de traiter la commande $j$ après la commande $i$ (le temps total de traitement pour $j$ est donc $s_{ij}+p_j$) et l'obtention de bons horaires passe souvent par la bonne gestion de ces temps de réglage. Dans ce chapitre, on s'intéresse ainsi aux problèmes d'ordonnancement sur une machine avec temps de réglage dépendants de la séquence. On montre comment ces problèmes peuvent être vus comme des cas particuliers du commis-voyageur dépendant du temps, variante du célèbre problème de commis-voyageur où le coût de transition entre deux villes dépend du moment de visite. Des formulations de programmation en nombres entiers, semblables à celle proposée au chapitre 2 pour le problème de commis-voyageur, sont aussi présentées et on montre comment il est possible de développer à partir de celles-ci des approches de résolution efficaces pour $1|s_{ij}|\sum C_j$, $1|s_{ij}|\sum T_j$.

## 4.2 Problème de commis-voyageur dépendant du temps

Le problème du commis-voyageur dépendant du temps est une version du problème de commis-voyageur classique où le coût de transition entre le noeud $i$ et le noeud $j$ dépend de la période de visite du noeud $i$, sachant qu'une période est nécessaire pour passer d'une ville à l'autre.

On peut définir ce problème plus formellement comme suit. Étant donné $G(\mathcal{N}; \mathcal{A})$ un graphe orienté où $\mathcal{N} = \{1, \ldots, n\}$. On connaît pour chaque arc $(i, j) \in \mathcal{A}$ le coût $c_{ij}^t$ associé au fait d'emprunter l'arc en position $t$ de la tournée, où $t = 1, \ldots, n$. Le problème de commis-voyageur dépendant du temps consiste à trouver le circuit hamiltonien $\xi = (i_1 = 1, i_2, \ldots, i_n, i_{n+1} = i_1 = 1)$ du graphe $G$ de coût $C_\xi$ minimum avec :

$$c_\xi = c_{i_1 i_2}^1 + c_{i_2 i_3}^2 + \cdots + c_{i_n i_1}^n.$$

Comme les périodes de transition sont unitaires et que les coûts dépendent en fait de la position, il est peut-être plus approprié de parler de problème de commis-voyageur dépendant de la position. Cependant, afin d'être cohérent avec les travaux antérieurs, on conserve l'appelation originale *commis-voyageur dépendant du temps.*

### 4.2.1 Formulation de Fox

Fox (1973, 1980) a introduit le problème de commis-voyageur dépendant du temps et a donné un exemple d'application dans l'industrie de brassage de la bière. Il a formulé

le problème de façon très compacte comme suit :

$$z_{T1} = \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ij}^{t} x_{ij}^{t} \tag{4.1}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} x_{ij}^{t} = n, \tag{4.2}$$

$$\sum_{j=1}^{n} \sum_{t=2}^{n} t x_{ij}^{t} - \sum_{j=1}^{n} \sum_{t=1}^{n} t x_{ji}^{t} = 1, \quad i = 2, \ldots, n \tag{4.3}$$

$$x_{ij}^{t} \in \{0, 1\}, \quad i = 1, \ldots, n, \quad j = 1, \ldots n, \quad t = 1, \ldots, n. \tag{4.4}$$

Dans cette formulation, $x_{ij}^{t}$ vaut 1 si l'arc $(i, j)$ est placé à la position $t$ de la tournée et 0 autrement et on suppose que le noeud de départ de la tournée est le noeud 1. La contrainte (4.2) permet d'assurer que la tournée est composée de $n$ arcs tandis que les contraintes (4.3) permettent à la fois d'assurer que tous les noeuds sont visités une et une seule fois et d'empêcher la formation de sous-tours. Cette formulation très compacte du problème est cependant peu pratique. Même pour des exemplaires de commis voyageur classique comptant 10 noeuds, il semble difficile de résoudre cette formulation par évaluation et séparation progressive (voir Orman et Williams, 2004 et Fox, 1973) en raison de la trop grande faiblesse de sa relaxation linéaire.

Fox (1973) a également proposé une formulation désagrégée de $T1$ beaucoup plus

forte :

$$z_{T2} = \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ij}^{t} x_{ij}^{t} \tag{4.5}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^{t} = 1, \quad t = 1, \ldots, n \tag{4.6}$$

$$\sum_{j=1}^{n} \sum_{t=1}^{n} x_{ij}^{t} = 1, \quad i = 1, \ldots, n \tag{4.7}$$

$$\sum_{i=1}^{n} \sum_{t=1}^{n} x_{ij}^{t} = 1, \quad j = 1, \ldots, n \tag{4.8}$$

$$\sum_{j=1}^{n} \sum_{t=2}^{n} t x_{ij}^{t} - \sum_{j=1}^{n} \sum_{t=1}^{n} t x_{ji}^{t} = 1, \quad i = 2, \ldots, n \tag{4.9}$$

$$x_{ij}^{t} \in \{0, 1\}, \quad i = 1, \ldots, n, \quad j = 1, \ldots n \quad t = 1, \ldots, n. \tag{4.10}$$

La formulation $T2$ a été très peu étudiée. Pour le même problème de commis-voyageur de 10 noeuds mentionné précédent, Orman et Williams (2004) ont été en mesure d'obtenir la solution optimale du problème grâce à cette formulation.

### 4.2.2 Formulations de Picard et Queyranne

Picard et Queyranne (1978) ont proposé deux formulations de programmation en nombres entiers pour le problème de commis-voyageur dépendant du temps qui sont beaucoup plus fortes. La première formulation, comptant $O(n^3)$ variables et $O(n^2)$ contraintes, est très semblable à la formulation de stable proposée au chapitre 2 pour le problème de commis-voyageur. Dans cette formulation, les contraintes de cliques d'incompatibilité de flot ont cependant été remplacées par des contraintes de flot :

$$z_{T3} = \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ij}^{t} x_{ij}^{t} \tag{4.11}$$

$$\sum_{i=1}^{n} \sum_{t=1}^{n} x_{ij}^{t} = 1, \quad j = 1, \ldots, n \tag{4.12}$$

$$\sum_{(1,j)\in\mathcal{A}} x_{1j}^{1} = 1, \tag{4.13}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^{t} - \sum_{(j,i)\in\mathcal{A}} x_{ji}^{t+1} = 0, \quad t = 2, \ldots, n-1, \quad j = 1, \ldots, n \tag{4.14}$$

$$\sum_{(i,1)\in\mathcal{A}} x_{i1}^{n} = 1, \tag{4.15}$$

$$x_{ij}^{t} \in \{0,1\}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, n \quad t = 1, \ldots, n. \tag{4.16}$$

Cette formulation met en évidence le fait qu'on peut voir le problème de commis-voyageur dépendant du temps comme un problème de plus court chemin dans un réseau multiparti avec contraintes compliquantes. Ce réseau se contruit comme suit. À chaque paire noeud-position $(i,t)$, où $i = 1, \ldots, n$ et $t = 1, \ldots, n+1$, est associé un noeud. Le noeud source du réseau est celui associé à la paire noeud-position $(1,1)$ tandis que le noeud puits est celui associé à la paire noeud-position $(1, n+1)$ et à chaque décision $(i,j)^t$ est associé un arc de coût $c_{ij}^{t}$ allant du noeud $(i,t)$ au noeud $(j, t+1)$. Tout chemin dans ce réseau correspond ainsi à une solution réalisable pour les contraintes (4.13)-(4.16) et il n'y a pas de solution réalisable pour (4.12)-(4.16) qui ne soit pas transposable comme un chemin dans ce réseau. La figure 4.1 illustre ce réseau pour un problème de 4 villes.

Si au lieu d'utiliser des variables de flot on utilise plutôt des variables de chemins, on peut déduire la deuxième formulation proposée par Picard et Queyranne (1978),

pour le problème de commis-voyageur dépendant du temps :

$$z_{DW(T3)} = \min \sum_{p\in\Omega}(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{n} c_{ij}^{t} x_{ij}^{tp})\theta_p \tag{4.17}$$

$$\sum_{p\in\Omega}(\sum_{i=1}^{n}\sum_{t=1}^{n} x_{ij}^{tp})\theta_p = 1, \quad j = 1,\ldots,n \tag{4.18}$$

$$\theta_p \in \{0,1\}, \quad p \in \Omega. \tag{4.19}$$

$\Omega$ est l'ensemble des chemins retrouvés au sein du réseau multiparti décrit précédemment et $x_j^{tp}$ est une constante valant 1 si le chemin $p$ emprunte l'arc $(i,j)$ en position $t$. Ce problème compte seulement $n$ contraintes mais un nombre exponentiel de colonnes et peut être vu comme la réécriture de Dantzig-Wolfe de la formulation de flot $T3$.

Malgré son grand nombre de colonnes, il peut être plus facile de résoudre $DW(T3)$ que $T3$. Pour un vecteur $\pi$ de variables duales, on peut en effet trouver la colonne dont le coût réduit est minimum en résolvant le sous-problème :

$$z_{SP(T3)} = \min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{n}(c_{ij}^{t} - \pi_j)x_{ij}^{t} \tag{4.20}$$

$$\sum_{(1,j)\in\mathcal{A}} x_{1j}^{1} = 1, \tag{4.21}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^{t} - \sum_{(j,i)\in\mathcal{A}} x_{ji}^{t+1} = 0, \quad t = 2,\ldots,n-1, \quad j = 1,\ldots,n \tag{4.22}$$

$$\sum_{(i,1)\in\mathcal{A}} x_{i1}^{n} = 1, \tag{4.23}$$

$$x_{ij}^{t} \in \{0,1\}, \quad i = 1,\ldots,n, \quad j = 1,\ldots,n. \tag{4.24}$$

Le sous-problème est un problème de plus court chemin dans le réseau multiparti où le coût d'un chemin donne maintenant le coût réduit de la variable associée à celui-ci pour un vecteur donné de variables duales des contraintes complicantes (4.18). En utilisant cet oracle pour obtenir les colonnes de coûts réduits négatifs au sein de l'algorithme du simplexe révisé, on contourne une partie des difficultés inhérentes au

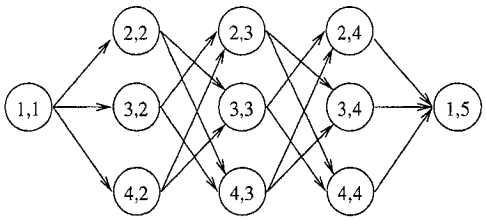


Figure 4.1 – Le réseau multiparti pour un problème à 4 villes

fait d'avoir un grand nombre de variables en ne considérant explicitement qu'un petit sous-ensemble de colonnes.

La formulation de flot de Picard et Queyranne (1978) est à la base de nombreuses approches proposées pour résoudre le problème de commis-voyageur dépendant du temps. Lucena (1990) et Bianco *et al.* (1993) ont utilisé des techniques de relaxation lagrangienne afin de dériver des bornes à partir de cette formulation pour le problème $1|s_{ij}|\sum C_j$, appelé aussi le problème de livreur (*deliveryman problem*), et ont développé des algorithmes d'énumération implicite exploitant ces bornes capables de résoudre des exemplaires comptant environ 35 noeuds. Vander Wiel et Sahinidis (1996) ont appliqué le principe de décomposition de Benders à une reformulation de $T3$ afin d'accélérer l'obtention de bornes inférieures et ont résolu, en imbriquant cette borne dans un algorithme d'évaluation et de séparation progressive, des exemplaires de problème de commis-voyageur dépendant du temps comptant 18 noeuds. La formulation $T3$ a même été utilisée dans le contexte du problème de commis-voyageur classique par Orman et Williams (2004).

La formulation de chemins de Picard et Queyranne (1978) a été beaucoup moins étudiée que la formulation de flot. Cela est peut-être dû au fait qu'il s'agit d'une formulation de génération de colonnes assez difficile à résoudre, en raison du grand nombre d'éléments non nuls retrouvés au sein des colonnes. Pourtant, cette formulation présente certains avantages par rapport à la formulation de flot. En plus d'être souvent plus facile à résoudre, la formulation de chemins peut être renforcée si on élimine de l'ensemble $\Omega$ certains chemins illégaux, avec par exemple les techniques

d'élimination de $k$-cycle (voir Irnich et Villeneuve, 2003), comme nous allons le démontrer à la section 4.3.2.

### 4.2.3 Application aux problèmes d'ordonnancement sur une machine

On peut adapter les formulations de Fox (1973, 1980) ou de Picard et Queyranne (1978) afin d'obtenir des formulations de programmation en nombres entiers pour divers problèmes d'ordonnancement sur une machine.

Soit $N = \{1, \ldots, n\}$ l'ensemble des pièces à traiter où la pièce 1 en est une fictive servant à marquer le début et la fin de la séquence. À chaque pièce devant être traitée est associée une durée de traitement $p_j$, une date de départ $r_j$ et un temps de réglage $s_{ij}$ si la pièce $j$ est traitée après $i$. En introduisant les variables $C_t$ donnant le temps de complétion de la pièce située en position $t$ et en posant :

$$\begin{aligned} & C_1 = 0, \\ & C_t \geq C_{t-1} + \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} (s_{ij} + p_j) x_{ij}^{t-1}, \quad t = 2, \ldots, n \\ & C_t \geq \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} (r_j + s_{ij} + p_j) x_{ij}^{t-1}, \quad t = 2, \ldots, n \end{aligned}$$

on peut générer des horaires respectant les contraintes de dates de départ. En utilisant comme fonction objectif $\sum_{t=1}^{n} C_t$ avec ces contraintes supplémentaires, on peut ainsi modéliser $1|r_j, s_{ij}|\sum C_j$. On note que si $r_j = 0$ pour $j = 1, 2, \ldots, n$, il n'y a pas d'attente dans aucune solution optimale et on n'a pas besoin de ces inégalités pour calculer les temps de complétion pour chaque position. Dans ce cas, on a seulement besoin de minimiser $\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ij}^{t} x_{ij}^{t}$ avec les coûts :

$$c_{ij}^{t} = (n - t + 1)(s_{ij} + p_j).$$

De même, on peut modéliser $1|r_j, s_{ij}|\sum T_j$ en introduisant les variables $T_t$ donnant

le retard associé à la pièce en position $t$ et en posant :

$$T_t \geq C_t - \sum_{i=1}^{n} \sum_{j=1}^{n} d_j x_{ij}^{t-1}, \quad t = 2, \ldots, n$$
$$T_t \geq 0.$$

Pour modéliser $1|r_j, s_{ij}| \sum w_j U_j$ (consistant à minimiser le nombre pondéré de pièces en retard avec des contraintes de date de départ et des temps de réglage dépendants de la séquence), il suffit de remplacer les contraintes imposant la planification de toutes les pièces (pour $T2$, (4.6), pour $T3$, (4.12), pour $DW(T3)$, (4.18)) par :

$$\sum_{i=1}^{n} \sum_{t=1}^{n} x_{ij}^{t} + U_j = 1, \quad j = 1, \ldots, n$$

où $U_j$ est une variable binaire valant 1 si la pièce $j$ est planifiée avec du retard et 0 autrement, et d'ajouter ces contraintes pour tenir compte des dates de livraison :

$$C_t \leq \sum_{i=1}^{n} \sum_{j=1}^{n} d_j x_{ij}^{t-1}, \quad t = 2, \ldots, n.$$

En plus, les formulations sous la forme d'un problème de commis-voyageur dépendant du temps peuvent être pratiques pour modéliser des contraintes moins classiques. Par exemple, des contraintes de date de livraison sur les positions (comme $C_t \leq \bar{d}_t$ qui consiste à imposer la planification d'au moins $t$ pièces avant le moment $\bar{d}_t$) peuvent être prises en compte. Par contre, on doit noter qu'avec ce genre de formulation, il peut être très difficile de modéliser les objectifs $\sum w_j C_j$ et $\sum w_j T_j$, puisqu'on dispose seulement des temps de complétion en fonction des positions, et non en fonction des pièces.

### 4.2.4 Qualité des formulations

On a vu diverses formulations de programmation en nombres entiers pour le problème de commis-voyageur dépendant du temps, de tailles différentes avec lesquelles on peut modéliser de nombreux problèmes d'ordonnancement sur une machine avec temps de

réglage dépendants de la séquence. Mais que valent ces formulations ?

Dans le cadre du problème classique de commis-voyageur (non dépendant du temps), Orman et Williams (2004) ont montré que

$$LP(T1) \leq LP(T2) \leq LP(T3) \leq LP(DFJ)$$

où $LP(T1)$, $LP(T2)$ et $LP(T3)$ sont les valeurs respectives de la relaxation linéaire des formulations $T1$, $T2$ et $T3$ pour le problème de commis-voyageur classique et $LP(DFJ)$ est la valeur de la relaxation linéaire de la formulation de Dantzig Fulkerson et Johnson (1954) :

$$z_{DFJ} = \min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} y_{ij} \tag{4.25}$$

$$\sum_{i=1}^{n} y_{ij} = 1, \quad j = 1, \ldots, n \tag{4.26}$$

$$\sum_{j=1}^{n} y_{ij} = 1, \quad i = 1, \ldots, n \tag{4.27}$$

$$\sum_{(i,j) \in \mathcal{A}: i \in S, j \notin S} y_{ij} \geq 1, \quad S \subset N, |S| \geq 2 \tag{4.28}$$

$$y_{ij} \in \{0, 1\}, \quad (i,j) \in \mathcal{A}. \tag{4.29}$$

De plus, comme $DW(T3)$ est une réécriture de Dantzig-Wolfe de $T3$ et que le sous-problème de Dantzig-Wolfe a la propriété d'intégrité (Geoffrion, 1974), on sait que :

$$LP(T3) = LP(DW(T3)).$$

Les formulations de commis-voyageur dépendant du temps sont donc en théorie moins fortes que la formulation $DFJ$. Le tableau 4.1, contenant les résultats d'une expérience conduite sur de petits exemplaires du problème de commis-voyageur, montre clairement que les modèles faibles en théorie sont également très faibles en pratique.

Tableau 4.1 – Bornes obtenues pour $1|s_{ij}|C_{max}$

| Problème | $T2$ | $T3$ | $MTZ^+$ | $DFJ$ | $OPT$ |
|---|---|---|---|---|---|
| $gr17$ | 1772.8 | 1808.8 | 1684 | 2085 | 2085 |
| $gr21$ | 2499.8 | 2524.3 | 2707 | 2707 | 2707 |
| $gr24$ | 1129.7 | 1136.1 | 1224.5 | 1272 | 1272 |
| $bays29$ | 1841.9 | 1844.6 | 1954 | 2013 | 2020 |
| $bayg29$ | 1490.9 | 1493.3 | 1544 | 1608 | 1610. |

La mauvaise qualité des bornes déduites à partir de la formulation $T2$ (et implicitement $T1$) ne constituent pas vraiment une surprise. En effet, ces formulations contiennent des contraintes avec d'assez grands coefficients (voir la contrainte (4.9)) dont la présence est reconnue pour introduire des grands sauts d'intégrité. Par contre, les mauvais résultats obtenus avec $T3$ constituent une déception. Cette formulation semble à première vue attrayante avec ses contraintes ne comptant que des $1, 0, -1$ comme coefficients. Mais pour tous les exemplaires testés, la borne donnée par la formulation de Picard et Queyranne (1978) est beaucoup moins bonne que celle donnée par la formulation de Dantzig-Fulkerson-Johnson (différence d'au moins 5%). La formulation de Picard et Queyranne est même souvent plus faible que la formulation de Miller, Tucker et Zemlin (1960) renforcée par Desrochers et Laporte (1991), pourtant reconnue pour être d'assez piètre qualité (la formulation $MTZ^+$ ne domine cependant ni $T2$ ni $T3$, comme le montrent les résultats obtenus pour l'exemplaire *gr17*).

Les formulations $T_2$ et $T_3$ semblent également donner des bornes assez faibles pour les problèmes $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$ tel qu'indiqué au tableau 4.2 comparant la valeur des relaxations linéaires et les valeurs des solutions optimales pour les exemplaires considérés lors de l'expérience.

Lorsqu'on utilise des bornes aussi faibles dans un algorithme de branchement, on obtient généralement des arbres de recherche d'assez grande taille. Les résultats d'une expérience préliminaire vont d'ailleurs dans ce sens. Pour un exemplaire de 29 villes d'un problème de commis-voyageur classique, que l'on résoud habituellement en une fraction de secondes avec d'autres approches, on a dû explorer plus de 3000 noeuds

Tableau 4.2 – Bornes obtenues pour $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$

| Type | Problème | $LP(T2)$ | $LP(T3)$ | $OPT$ |
|---|---|---|---|---|
| $1\|s_{ij}\|\sum C_j$ | $gr17$ | 10641.8 | 10897.7 | 12994 |
| | $gr21$ | 19673.1 | 20378.5 | 24345 |
| | $gr24$ | 11372.3 | 11770.5 | 13795 |
| | $bays29$ | 22335.9 | 23163 | 26862 |
| | $bayg29$ | 18678.6 | 19319 | 22230 |
| $1\|s_{ij}\|\sum T_j$ | $prob408$ | 5282.7 | 5342.1 | 5660 |
| | $prob503$ | 3442.7 | 3479.7 | 3497 |
| | $prob507$ | 6891.9 | 7137.9 | 7225 |
| | $prob508$ | 1505.2 | 1642.2 | 1915 |

(avec cplex9.0) en utilisant la borne donnée par $T3$ afin de trouver la solution optimale. Pour cette raison, on a exploré les approches basées sur les techniques de coupes de la programmation en nombres entiers afin de resserrer les formulations du problème de commis-voyageur dépendant du temps.

## 4.3 De nouvelles formulations

### 4.3.1 Ajout de coupes

On peut renforcer les formulations $T1$, $T2$ et $T3$ en récupérant certains résultats connus du problème de commis-voyageur (symétrique) et du problème de stable de poids maximum.

### Problème de commis-voyageur symétrique

En définissant $y_{ij}$ comme :

$$y_{ij} = \sum_{t=1}^{n}(x_{ij}^t + x_{ji}^t), \quad (i,j) \in \mathcal{A}, i < j,$$

on peut réutiliser pour la version dépendant du temps les coupes développées pour le problème de commis-voyageur symétrique, comme les coupes de sous-tour :

$$\sum_{(i,j)\in\mathcal{A}:i,j\in S,i<j} y_{ij} \leq |S| - 1, \quad S \subset N, |S| \geq 2 \tag{4.30}$$

ou les coupes de 2-couplages (*2-matching*) du problème de commis-voyageur symétrique :

$$\sum_{i\in H}\sum_{j\in H} y_{ij} + \sum_{(i,j)\in\hat{E}} y_{ij} \leq |H| + \lfloor\frac{|\hat{E}|}{2}\rfloor, \tag{4.31}$$

où $H$ est un ensemble de noeuds tel que $3 \leq |H| \leq |N|$ et $\hat{E}$ est un ensemble impair d'arêtes ayant au moins une extrémité dans l'ensemble $H$.

### Problème de stable

De plus, on peut réutiliser certaines coupes développées pour le problème de stable de poids maximum, en raison des liens entre ce problème et le problème de commis-voyageur dépendant du temps exposés au chapitre 2.

Pour construire le graphe d'incompatibilité, on s'est servi de ces trois définitions d'incompatibilité (plutôt que seulement des deux présentées au chapitre 2).

La décision $(i,j)^t$ est incompatible avec $(k,l)^s$ si :

- $j = l$ ou $i = k$ (incompatibilité de noeud) ;

- $s = t + 1, k \neq j$ ou $t = s + 1, i \neq l$ (incompatibilité de flot) ;
- $s \neq t + 1, k = j, k \neq 1$ ou $s \neq t - 1, l = i, k \neq 1$ (incompatibilité de noeud et de flot).

Les deux premières relations sont identiques à celles présentées au chapitre 2. La première relation exprime qu'on ne peut pas arriver à un noeud ou partir d'un noeud plus d'une fois. La deuxième relation permet de faire appliquer les contraintes de flot pour les positions adjacentes de la tournée. La troisième relation indique finalement qu'on ne peut pas partir du noeud $k$ en position $t$ si on n'est pas arrivé au noeud $k$ en position $t - 1$ (sauf pour le noeud 1 évidemment). Même si la troisième relation d'incompatibilité est redondante pour le problème en nombres entiers, elle ne l'est pas pour sa relaxation linéaire et on a obtenu en pratique de bien meilleures inégalités en la considérant dans le graphe d'incompatibilité.

Les inégalités suivantes du problème de stable :

$$\sum_{(i,j)^t \in C} x_{ij}^t \leq 1 \tag{4.32}$$

$$\sum_{(i,j)^t \in H} x_{ij}^t \leq \lfloor \frac{|\mathcal{H}|}{2} \rfloor \tag{4.33}$$

où $C$ est une clique et $\mathcal{H}$ un trou dans le graphe d'incompatibitité induit par les décisions $(i,j)^t$ sont ainsi valides pour $T1$, $T2$ et $T3$.

### Algorithme de génération de coupes

À partir de ces résultats, on a conçu trois versions d'un algorithme de génération de coupes afin d'obtenir de meilleures bornes pour le problème de commis-voyageur dépendant du temps. Les trois versions fonctionnent selon la logique de l'algorithme 4.1.

Dans la première version de l'algorithme de génération de coupes, on n'ajoute que

**Algorithme 4.1** Algorithme générique de génération de coupes

1. Résoudre la relaxation linéaire de T3.
2. S'il existe une inégalité valide violée par la solution de la relaxation, alors ajouter celle-ci à T3 et retourner à 1. Sinon STOP.

les contraintes de sous-tour et de 2-couplages du problème de commis-voyageur, dans la deuxième, on n'ajoute que les contraintes de cliques du problème de stable alors que dans la troisième, on ajoute les coupes des trois familles. On n'a jamais ajouté de contraintes de trous car des tests préliminaires ont indiqué que ces coupes n'étaient pas très bonnes.

Pour identifier les contraintes de sous-tour et de 2-couplages violées par la relaxation de $T3$, on s'est servi des algorithmes exacts de séparation implantés dans la librairie Concorde (voir Applegate *et al.*, 1998) basés sur la résolution de problèmes de flot maximum. Pour identifier les contraintes de cliques, on a conçu une procédure de séparation heuristique basée sur l'algorithme *MaxClique* (voir encadré 4.2). En appelant

**Algorithme 4.2** Algorithme *MaxClique*

Soit $L$ une liste de sommets tabous, $v$ le sommet de départ à partir duquel on étend la clique, $N(v)$ les voisins de $v$, et $C$ la clique obtenue en sortie.

1. $C = v$.
2. $N = N(v) \setminus L$
3. **tant que** $N \neq \emptyset$.
4. Soit $w$ le sommet de $N$ de plus grand poids.
5. $C \cup w$.
6. $N = N \cap (N(w) \setminus L)$.
7. **fin tant que**

l'algorithme *MaxClique* à partir de sommets de départ différents et en modifiant la liste taboue des sommets, on peut trouver une clique $C$ du graphe d'incompatibilité des décisions $(i,j)^t$ de grand poids où le poids de $C$ est $\sum_{(i,j)^t \in C} \hat{x}^t_{i,j}$, $\hat{x}^t_{i,j}$ étant la valeur de $x^t_{ij}$ à la relaxation. Si $\sum_{(i,j)^t \in C} \hat{x}^t_{i,j} > 1$, la coupe de clique associée à $C$ est violée par la solution courante et on peut ajouter cette contrainte afin de resserrer la relaxation.

Dans notre implantation originale, la procédure *MaxClique* n'était implantée que

dans sa version gloutonne, sans liste tabou $L$. Les résultats obtenus avec cette approche suggéraient que les coupes de cliques étaient très faibles. Or, en imbriquant cette procédure de séparation gloutonne dans un algorithme de recherche tabou très simple (consistant d'abord à ne rien interdire, puis à interdire le premier sommet $w$ ajouté à la clique, puis les deux premiers ajoutés et ainsi de suite) on a obtenu de bien meilleurs résultats. Cela illustre l'importance des procédures de séparation dans la performance des algorithmes de génération de coupes.

Dans notre algorithme de génération de coupes, on note qu'à chaque itération, une seule clique (séquentiellement liftée) est ajoutée au programme et qu'on a limité à 100 le nombre de cliques générées.

Tableau 4.3 – Bornes obtenues avec l'algorithme de génération de coupes

| Type | Problème | $LP(T3)^{tsp}$ | $LP(T3)_{cliques}$ | $LP(T3)^{tsp}_{clique}$ | $OPT$ |
|---|---|---|---|---|---|
| $1\|s_{ij}\|C_{max}$ | $gr17$ | 2085 | 1998.6 | 2085 | 2085 |
| | $gr21$ | 2707 | 2707 | 2707 | 2707 |
| | $gr24$ | 1272 | 1258.1 | 1272 | 1272 |
| | $bays29$ | 2020 | 1950.1 | 2020 | 2020 |
| | $bayg29$ | 1610 | 1570.5 | 1610 | 1610 |
| $1\|s_{ij}\|\sum C_j$ | $gr17$ | 12073.1 | 12451.4 | 12686.8 | 12994 |
| | $gr21$ | 21785.3 | 22991.2 | 23260.5 | 24345 |
| | $gr24$ | 12751.7 | 13327.2 | 13395 | 13795 |
| | $bays29$ | 24261.4 | 24750 | 25041.5 | 26862 |
| | $bayg29$ | 20316.6 | 20498.1 | 20834.8 | 22230 |
| $1\|s_{ij}\|\sum T_j$ | $prob408$ | 5343.5 | 5357.9 | 5358.2 | 5660 |
| | $prob503$ | 3480.2 | 3480.7 | 3480.7 | 3497 |
| | $prob507$ | 7149.4 | 7150.5 | 7154.2 | 7225 |
| | $prob508$ | 1642.2 | 1649.3 | 1649.2 | 1915 |

On retrouve au tableau 4.3 les valeurs des bornes que l'on a obtenues avec les trois versions de l'algorithme de génération de coupes (notées respectivement $LP(T3)^{tsp}$, $LP(T3)_{cliques}$, $LP(T3)^{tsp}_{cliques}$).

En analysant les résultats du tableau, on remarque que pour le problème $1|s_{ij}|C_{max}$, les coupes les plus fortes sont sans contredit les coupes classiques du problème de commis-

Tableau 4.4 – Comparaison des saut d'intégrité obtenus (en %)

| Type | Problème | $LP(T3)$ | $LP(T3)^{tsp}$ | $LP(T3)_{cliques}$ | $LP(T3)^{tsp}_{cliques}$ |
|---|---|---|---|---|---|
| $1\|s_{ij}\|C_{max}$ | $gr17$ | 13.25 | 0.00 | 4.14 | 0 |
| | $gr21$ | 6.75 | 0.00 | 0.00 | 0.00 |
| | $gr24$ | 10.68 | 0.00 | 1.09 | 0 |
| | $bays29$ | 8.68 | 0.00 | 3.46 | 0 |
| | $bayg29$ | 7.25 | 0.00 | 2.45 | 0 |
| $1\|s_{ij}\|\sum C_j$ | $gr17$ | 18.10 | 7.09 | 4.18 | 2.36 |
| | $gr21$ | 16.29 | 10.51 | 5.56 | 4.45 |
| | $gr24$ | 14.68 | 7.56 | 3.39 | 2.90 |
| | $bays29$ | 13.78 | 9.67 | 7.86 | 6.78 |
| | $bayg29$ | 13.1 | 8.61 | 7.79 | 6.28 |
| $1\|s_{ij}\|\sum T_j$ | $prob408$ | 5.61 | 5.56 | 5.33 | 5.33 |
| | $prob503$ | 0.49 | 0.48 | 0.47 | 0.47 |
| | $prob507$ | 1.21 | 1.05 | 1.03 | 0.98 |
| | $prob508$ | 14.25 | 14.25 | 13.87 | 13.89 |

voyageur. Les cinq exemplaires considérés du problème classique de commis-voyageur ont été résolus en ajoutant simplement les coupes de sous-tour et de 2-couplages alors que seulement un exemplaire a été résolu sans brancher avec les coupes de cliques.

Pour le problème $1|s_{ij}|\sum C_j$, aucun des exemplaires n'a été résolu en ajoutant simplement des coupes, ce qui semble confirmer le fait que $1|s_{ij}|\sum C_j$ soit plus difficile en pratique que $1|s_{ij}|C_{max}$. Les coupes de cliques ont été plus efficaces pour tous les exemplaires que les coupes de sous-tour et de 2-couplages. Dans tous les cas, on a été en mesure de diminuer les sauts d'intégrité d'au moins de moitié en ayant recours aux coupes. Malgré cela, les sauts d'intégrité demeurent assez élevés (notamment pour les exemplaires $bays29$ et $bayg29$ où on a observé des sauts de 6 et 7 %).

Pour le problème $1|s_{ij}|\sum T_j$, les résultats sont très décevants. Pour tous les exemplaires, l'ajout de coupes n'a pas vraiment permis d'obtenir de meilleures bornes. On verra plus loin pourquoi.

### 4.3.2 Décomposition de Benders

Considérons la formulation suivante :

$$z_{T4} = \min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{n} c_{ij}^t x_{ij}^t$$

$$\sum_{i=1}^{n}\sum_{t=1}^{n} x_{ij}^t = 1, \quad j = 1, \ldots, n$$

$$\sum_{(1,j)\in\mathcal{A}} x_{1j}^1 = 1,$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^t - \sum_{(j,i)\in\mathcal{A}} x_{ji}^{t+1} = 0, \quad t = 2, \ldots, n-1, \quad j = 1, \ldots, n$$

$$\sum_{(i,1)\in\mathcal{A}} x_{i1}^n = 1,$$

$$y_{ij} = \sum_{t=1}^{n}(x_{ij}^t + x_{ji}^t), \quad (i,j) \in \mathcal{A}, i < j \tag{4.34}$$

$$y_{ij} = \sum_{p\in\Gamma} y_{ij}^p \lambda_p, \quad (i,j) \in \mathcal{A}, i < j \tag{4.35}$$

$$\sum_{p\in\Gamma} \lambda_p = 1, \tag{4.36}$$

$$\lambda_p \geq 0, \quad p \in \Gamma \tag{4.37}$$

$$x_{ij}^t \in \{0,1\}, \quad i = 1, \ldots, n, \quad j = 1, \ldots n, \quad t = 1, \ldots, n.$$

qui n'est que la formulation $T3$ à laquelle on a ajouté les contraintes (4.34)-(4.37). Dans cette formulation, $y_{ij}$ est une variable qui vaut 1 si l'arête $(i,j)$ est empruntée dans la solution optimale (0 autrement), $\Gamma$ est l'ensemble des solutions du problème de commis-voyageur symétrique (non-dépendant du temps) et $y_{ij}^p$ est une constante qui vaut 1 si le tour $p \in \Gamma$ emprunte l'arête $(i,j)$. Ces contraintes imposent que toute solution du problème de commis-voyageur dépendant du temps soit une combinaison convexe de tours du problème de commis-voyageur non-dépendant du temps. Comme celles-ci ne sont pas nécessairement redondantes pour la relaxation linéaire de $T3$, on peut obtenir en les prenant en compte de meilleures bornes de programmation

linéaire.

Cette formulation est très difficile à résoudre directement, en raison du grand nombre de variables et de colonnes. Par contre, elle possède une structure qui peut être exploitée par le principe de décomposition de Benders. On remarque en effet que si les variables $x$ et $y$ sont fixées, alors le problème se réduit à ce problème de réalisabilité :

$$SPP(\hat{y}) = \min_{\lambda} 0$$

$$\sum_{p \in \Gamma} y_{ij}^p \lambda_p = \hat{y}_{ij}, \quad (i,j) \in \mathcal{A}, i < j \tag{4.38}$$

$$\sum_{p \in \Gamma} \lambda_p = 1, \tag{4.39}$$

$$\lambda_p \geq 0, \quad p \in \Gamma$$

dans lequel on cherche à vérifier s'il est possible d'exprimer la solution $\hat{y}$ comme une combinaison convexe des solutions du problème de commis-voyageur classique. Ce problème peut se résoudre par génération de colonnes (avec un sous-problème de commis-voyageur classique pour identifier les colonnes de plus petit coût réduit). Si le problème est non-réalisable, le dual, qui est dans ce cas toujours réalisable, est donc non-borné et le lemme de Farkas stipule qu'il existe un rayon extrême du polyèdre dual tel que :

$$\sum_{(i,j) \in \mathcal{A}: i<j} \pi_{ij} \hat{y}_{ij} + \alpha > 0$$

où $\pi$ est le vecteur des variables duales associées aux contraintes (4.38) et (4.39). Cela suggère l'algorithme 4.3 de type Benders pour résoudre la relaxation linéaire de $T4$.

**Algorithme 4.3** Algorithme pour résoudre la relaxation linéaire de $T4$.

1. Résoudre la relaxation linéaire de $T3$.
2. Calculer la valeur de $\hat{y}$ à partir du vecteur $x$ obtenu en 1.
3. Résoudre le problème $SPP(\hat{y})$ pour $\hat{y}$ calculé en 2. Si $SPP(\hat{y})$ est réalisable, la solution est optimale, STOP. Sinon identifier un rayon extrême du dual tel que $\sum_{(i,j) \in \mathcal{A}: i<j} \pi_{ij} \hat{y}_{ij} + \alpha > 0$, ajouter la contrainte (de réalisabilité) $\sum_{(i,j) \in \mathcal{A}: i<j} \pi_{ij} y_{ij} + \alpha \leq 0$ à T3 et retourner à l'étape 1.

On a implanté l'algorithme 4.3 afin d'obtenir des bornes inférieures à partir de la formulation $T4$. Dans notre algorithme, on n'a pas résolu le problème $SPP(\hat{y})$, mais plutôt une relaxation de celui-ci :

$$\begin{aligned} SPP'(\hat{y}) &= \min 0 \\ \sum_{p\in\Gamma} y_{ij}^p \lambda_p &\geq \hat{y}_{ij}, \quad (i,j) \in \mathcal{A}, i < j \\ \sum_{p\in\Gamma} \lambda_p &\leq 1, \\ \lambda_p &\geq 0, \quad p \in \Gamma. \end{aligned}$$

Le problème $SPP'(\hat{y})$ est résolu par génération de colonnes et le sous-problème de commis-voyageur non-dépendant du temps utilisé pour générer les colonnes est résolu à l'aide de Concorde.

En résolvant $SPP'(\hat{y})$ plutôt que $SPP(\hat{y})$, on peut seulement générer les coupes de réalisabilité avec des coefficients et des membres de droite positifs. Par contre, il est nettement plus facile de résoudre $SPP'(\hat{y})$. En effet, dans ce cas, on n'est pas obligé de considérer les contraintes avec $\hat{y}_{ij}$ nuls, car à l'optimalité, elles ne sont jamais actives. Leurs variables duales peuvent conséquemment être posées à 0. De plus, comme ces programmes linéaires sont résolus par génération de colonnes, il peut être avantageux de restreindre l'espace de variables duales, comme dans la formulation $SPP'(\hat{y})$, afin d'améliorer la convergence de l'algorithme.

Le tableau 4.5 présente les bornes que l'on a obtenu à partir de la formulation $T4$ pour des exemplaires de $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$.

Avec la formulation $T4$, on parvient à obtenir de meilleures bornes qu'avec $T3^{tsp}$. Ce résultat était cependant prévisible car avec $T4$, on prend implicitement en compte toutes les inégalités valides pour l'enveloppe convexe des solutions du problème de commis-voyageur avec coefficients et membre de droite positifs alors qu'avec $T3^{tsp}$, on ne considère que deux sous-familles de ces inégalités (les inégalités de sous-tour et de 2-couplagles). Même si on obtient avec $T4$ de meilleures bornes qu'avec $T3^{tsp}$, il

Tableau 4.5 – Bornes obtenues avec $T4$

| Type | Problème | $LP(T3)^{tsp}$ | $LP(T4)$ | $OPT$ |
|---|---|---|---|---|
| $1\|s_{ij}\|\sum C_j$ | $gr17$ | 12073.1 | 12076.0 | 12994 |
| | $gr21$ | 21785.3 | 21811.8 | 24345 |
| | $gr24$ | 12751.7 | 12792.0 | 13795 |
| | $bays29$ | 24261.4 | 24278.4 | 26862 |
| | $bayg29$ | 20316.6 | 20329.2 | 22230 |
| $1\|s_{ij}\|\sum T_j$ | $prob408$ | 5343.5 | 5343.5 | 5660 |
| | $prob503$ | 3480.2 | 3480.2 | 3497 |
| | $prob507$ | 7149.4 | 7151.7 | 7225 |
| | $prob508$ | 1642.2 | 1642.2 | 1915 |

ne semble pas vraiment avantageux d'avoir recours à cette formulation. Le processus pour calculer la relaxation linéaire de $T4$ est nettement plus lourd que celui nécessité pour résoudre celle de $T3^{tsp}$ et l'amélioration de la borne n'est pas significative.

### 4.3.3 Élimination de $k$-cycles

La formulation $DW(T3)$ peut être également renforcée en introduisant des coupes dans le sous-problème $SP(T3)$ afin d'éliminer de l'ensemble $\Omega$ des chemins que l'on sait non réalisable. On peut, par exemple, ajouter les contraintes suivantes au sous-problème

$$x_{ij}^t + x_{ji}^{t+1} \leq 1, \quad i = 1, \ldots, n, j = 1, \ldots, n, t = 1, \ldots, n$$

afin d'interdire les chemins contenant des cycles de type $i - j - i$, appelés 2-cycle. Et on peut résoudre le sous-problème $SP(T3)$ avec ces contraintes supplémentaires en utilisant la récursion suivante :

$$\begin{aligned}
&f(\cdot, 1, 1) = 0 \\
&f(i, j, t) = \min_{k \in N \setminus \{i,j\}} \{f(k, i, t-1) + c_{ij}^{t-1} - \pi_j\}, \\
&\text{où} \quad i \in N, \quad j \in N, \quad t = 2, \ldots, n \\
&f(\cdot, 1, t+1) = \min_{k \in N \setminus \{1\}} f(k, 1, t).
\end{aligned}$$

Dans cette récursion $f(i,j,t)$ donne le coût réduit optimal de planifier la pièce $j$ en position $t$ après la pièce $i$. On note qu'on pourrait voir la résolution de cette récursion comme un problème de plus court chemin dans un réseau. Dans le réseau, il y a un noeud pour chaque état $(i,j,t)$ possible et il y a un arc de coût $c_{ij}^{t-1} - \pi_j$ qui relie l'état $(k,i,t-1)$ et $(i,j,t)$ si $k \neq j$ et $k \neq i$, c.-à-d. si le passage d'un état à l'autre n'induit pas de 2-cycles et le plus court chemin de l'état $(\cdot,1,1)$ à l'état $(\cdot,1,t+1)$ est un chemin sans 2-cycle de plus petit coût réduit.

À première vue, cette récursion semble pouvoir se résoudre en $O(n^4)$, puisqu'il y a $(n-1)(n-1)n$ états et $(n-1)$ prolongations à effectuer à chaque état. Houck *et al.* (1980) ont cependant montré que l'état $(i,j,t)$ de coût $f(i,j,t)$ est dominé s'il existe un état $(k,j,t)$ et un état $(l,j,t)$ tel que $f(k,j,t) \leq f(i,j,t)$, $f(l,j,t) \leq f(k,j,t)$ et $k \neq l$. Cette règle de dominance permet d'abaisser la complexité de l'algorithme de résolution à $O(n^3)$.

De même, on pourrait éliminer les $k$-cycles (cycles de longueur $k$) en résolvant $SP(T3)$ comme cette récursion :

$$
\begin{aligned}
&f(\cdot,\cdot,\ldots,\cdot,1,1) = 0 \\
&f(i_1,i_2,\ldots,i_{k-1},i_k,t) = \min_{i_0 \in N \setminus \{i_1,\ldots,i_{k-1},i_k\}} \{f(i_0,i_1,\ldots,i_{k-2},i_{k-1},t-1) + c_{i_{k-1}i_k}^{t-1} - \pi_{i_k}\}, \\
&\text{où } i_0,i_1,i_2,\ldots,i_k \in N, \quad t = 2,\ldots,n \\
&f(\cdot,\cdot,\ldots,\cdot,1,t+1) = \min_{i_1,\ldots,i_{k-1},i_k \in N \setminus \{1\}} \{f(i_1,i_2,\ldots,i_k,1,t)\}.
\end{aligned}
$$

Dans cette récursion, $f(i_1,i_2,\ldots,i_{k-1},i_k,t)$ donne le coût associé au fait d'effectuer la pièce $i_k$ en position $t$ après la séquence de pièces $i_1,i_2,\ldots,i_{k-2},i_{k-1}$. On pourrait encore voir la résolution de cette récursion comme un problème de plus court chemin dans un réseau. Dans le réseau, il y a un noeud pour chaque état possible et il y a un arc de coût $c_{i_{k-1}i_k}^{t-1}$ entre l'état $(i_0,i_1,\ldots,i_{k-2},i_{k-1},t-1)$ et l'état $(i_1,i_2,\ldots,i_{k-1},i_k,t)$ si $i_0 \neq i_1, i_0 \neq i_2,\ldots,i_0 \neq i_{k-1}$, c.-à-d. si le passage d'un état à l'autre n'induit pas de $k$-cycle et le plus court chemin de l'état $(\cdot,\cdot,\ldots,\cdot,1,1)$ à l'état $(\cdot,\cdot,\ldots,\cdot,1,t+1)$ correspond au chemin sans $k$-cycle de plus petit coût réduit.

Résoudre cette récursion semble à première vue très difficile. Irnich et Villeneuve

(2003) ont cependant proposé une règle de dominance très efficace qui permet d'abaisser la complexité de l'algorithme de résolution à $O(n^3k(k-1)!^2)$ (et peut-être même à $O(k!n^3)$, si une conjecture suggérée dans leur article est vraie). Cette règle généralise la règle de Houck *et al.* (1980) et stipule essentiellement qu'un état est dominé si pour toutes ses prolongations potentielles sans $k$-cycle il y a un autre état de coût inférieur ou égal qui permet cette prolongation sans $k$-cycle.

On peut implanter l'élimination de $k$-cycles dans l'algorithme de plus court chemin avec contraintes de ressources de Gencol (le solveur de génération de colonnes utilisé) assez facilement. Avec les librairies Netgen, on a en effet la possibilité de modifier à notre gré la fonction de prolongation utilisée afin de générer les étiquettes dans l'algorithme de plus court chemin et la fonction de filtre appelée par Gencol afin d'éliminer certaines étiquettes lorsque leur nombre pour un noeud est trop important. En réécrivant la fonction de prolongation (de façon à empêcher la création d'une étiquette impliquant un $k$-cycle et à mettre à jour correctement la liste des $k-1$ prédécesseurs) et la fonction de filtre (afin d'éliminer les étiquettes dominées selon la méthode de Irnich et Villeneuve, 2003), on obtient une méthode assez efficace pour résoudre le sous-problème $SP(T3)$ avec élimination de $k$-cycles.

On présente au tableau 4.6 les résultats obtenus en faisant de l'élimination de $k$-cycles pour $k = 2, 3, 4$.

Pour le problème $1|s_{ij}|C_{max}$, les résultats obtenus indiquent qu'en utilisant une élimination de $k$-cycles aggressive ($k \geq 3$), il est possible d'obtenir des meilleures bornes qu'en utilisant seulement des coupes de cliques. On obtient toutefois de moins bonnes bornes qu'avec les coupes de commis-voyageur.

Pour le problème $1|s_{ij}|\sum C_j$, l'élimination de $k$-cycles apparaît cependant comme le meilleur moyen de resserrer la formulation $T3$. Pour le problème *bayg*29, on remarque qu'en éliminant les 4-cycles, on a été en mesure de réduire le saut d'intégrité à 2%, alors qu'en utilisant seulement les coupes de cliques et de commis-voyageur, celui-ci est demeuré à plus de 6%. Trois exemplaires de $1|s_{ij}|\sum C_j$ ont même été résolus simplement en éliminant les 3-cycles et les 4-cycles. Résoudre la relaxation linéaire de

$DW(T3)$ en éliminant les $k$-cycles est un peu plus long (pour *bayg29* par exemple, résoudre $DW(T3)$ nécessite 4 secondes, alors que résoudre $DW(T3)$ en éliminant les 4-cycles en nécessite 25) mais les gains obtenus sur le saut d'intégrité justifient pleinement cet effort supplémentaire. Encore une fois, pour le problème $1|s_{ij}|\sum T_j$,

Tableau 4.6 – Bornes obtenues avec l'élimination de $k$-cycles

| Type | Problème | $k=2$ | $k=3$ | $k=4$ | $OPT$ |
|---|---|---|---|---|---|
| $1\|s_{ij}\|C_{max}$ | *gr*17 | 1882.7 | 2010.2 | 2074.5 | 2085 |
| | *gr*21 | 2707 | 2707 | 2707 | 2707 |
| | *gr*24 | 1243.3 | 1272 | 1272 | 1272 |
| | *bays*29 | 1952.5 | 2002.8 | 2020 | 2020 |
| | *bayg*29 | 1554.2 | 1597.4 | 1605.7 | 1610 |
| $1\|s_{ij}\|\sum C_j$ | *gr*17 | 11909.2 | 12994 | 12994 | 12994 |
| | *gr*21 | 23258.5 | 24345 | 24345 | 24345 |
| | *gr*24 | 13586.1 | 13795 | 13795 | 13795 |
| | *bays*29 | 25441.4 | 26121.4 | 26675.4 | 26862 |
| | *bayg*29 | 20869.8 | 21524.2 | 21814.6 | 22230 |
| $1\|s_{ij}\|\sum T_j$ | *prob*408 | 5357.2 | 5360.8 | 5367.9 | 5660 |
| | *prob*503 | 3480.7 | 3481.3 | 3481.3 | 3497 |
| | *prob*507 | 7149.3 | 7155.9 | 7170.9 | 7225 |
| | *prob*508 | 1649.9 | 1650.7 | 1654.4 | 1915 |

les gains obtenus avec l'élimination de $k$-cycles sont minimes. Cela semble confirmer le fait que la faiblesse du modèle proposé pour $1|s_{ij}|\sum T_j$ ne réside pas nécessairement dans les contraintes du problème de commis-voyageur dépendant du temps mais plutôt dans les contraintes :

$$
\begin{aligned}
& C_1 = 0 \\
& C_t = C_{t-1} + \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n}(s_{ij}+p_j)x_{ij}^{t-1}, \quad t = 2,\ldots,n \\
& T_t \geq C_t - \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} d_j x_{ij}^{t-1}, \qquad\qquad t = 2,\ldots,n \\
& T_t \geq 0, \quad t = 2,\ldots,n
\end{aligned}
$$

que l'on utilise afin de mesurer les retards. Pour cette raison, on a développé une nouvelle formulation pour $1|s_{ij}|\sum T_j$ dans laquelle on n'a pas besoin de contraintes supplémentaires afin de calculer le retard des pièces.

Tableau 4.7 – Sauts obtenus avec l'élimination de $k$-cycles (en %)

| Type | Problème | $k=2$ | $k=3$ | $k=4$ |
|---|---|---|---|---|
| $1\|s_{ij}\|C_{max}$ | $gr17$ | 9.70 | 3.59 | 0.50 |
| | $gr21$ | 0.00 | 0.00 | 0.00 |
| | $gr24$ | 2.25 | 0.00 | 0.00 |
| | $bays29$ | 3.34 | 0.85 | 0.00 |
| | $bayg29$ | 3.47 | 0.78 | 0.00 |
| $1\|s_{ij}\|\sum C_j$ | $gr17$ | 8.35 | 0.00 | 0.27 |
| | $gr21$ | 4.46 | 0.00 | 0.00 |
| | $gr24$ | 1.51 | 0.00 | 0.00 |
| | $bays29$ | 5.29 | 2.76 | 0.70 |
| | $bayg29$ | 6.12 | 3.17 | 1.87 |
| $1\|s_{ij}\|\sum T_j$ | $prob408$ | 5.35 | 5.29 | 5.16 |
| | $prob503$ | 0.47 | 0.45 | 0.45 |
| | $prob507$ | 1.04 | 0.96 | 0.75 |
| | $prob508$ | 13.84 | 13.80 | 13.61 |

### 4.3.4 Une nouvelle formulation pour le problème $1|s_{ij}|T_j$

La formulation que nous proposons peut être vue comme un cas particulier de la formulation unifiée de Desaulniers *et al.* (1998) pour les problèmes de routage de véhicules et généralise la formulation de génération de colonnes de Picard et Queyranne (1978) proposée pour le problème de commis-voyageur dépendant du temps. Dans cette formulation, au lieu de mesurer les retards à même le problème maître, comme avec la formulation $DW(T3)$ pour le problème $1|s_{ij}|T_j$, on mesure les retards directement dans le sous-problème.

On considère le graphe multiparti présenté à la section 1. Soit $\Omega$ l'ensemble des chemins légaux dans ce graphe et $c_p$ la somme des retards accumulés le long du chemin $p \in \Omega$. Alors, ce programme en nombres entiers permet de modéliser le problème $1|s_{ij}|\sum T_j$ :

$$z_{DW(T3')} = \min \sum_{p \in \Omega} c_p \theta_p \tag{4.40}$$

$$\sum_{p \in \Omega} (\sum_{i=1}^{n} \sum_{t=1}^{n} x_{ij}^{tp}) \theta_p = 1, \quad j = 1, ..., n \tag{4.41}$$

Ce modèle se distingue de celui de Picard et Queyranne (1978) pour le problème de commis-voyageur dépendant du temps dans la façon dont les colonnes de coût réduit négatif sont identifiées. Pour le problème de commis-voyageur dépendant du temps, on a vu qu'il suffisait de résoudre un problème de plus court chemin sur le graphe multiparti. Si on utilise l'objectif $\sum T_j$, on doit plutôt résoudre le problème auxiliaire suivant pour identifier les colonnes intéressantes :

$$z_{SP(T3')} = \min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} (f_{ij}^t - \pi_j) x_{ij}^t \tag{4.42}$$

$$\sum_{(1,j) \in \mathcal{A}} x_{1j}^1 = 1, \tag{4.43}$$

$$\sum_{(i,j) \in \mathcal{A}} x_{ij}^t - \sum_{(j,i) \in \mathcal{A}} x_{ji}^{t+1} = 0, \quad t = 2, \ldots, n-1, \quad j = 1, \ldots, n \tag{4.44}$$

$$\sum_{(i,1) \in \mathcal{A}} x_{i1}^n = 1, \tag{4.45}$$

$$C_1 = 0, \tag{4.46}$$

$$C_t = C_{t-1} + \sum_{i=1}^{n} \sum_{j=1}^{n} (s_{ij} + p_j) x_{ij}^{t-1}, \quad t = 2, \ldots, n \tag{4.47}$$

$$T_t \geq C_t - \sum_{i=1}^{n} \sum_{j=1}^{n} d_j x_{ij}^{t-1}, \quad t = 2, \ldots, n \tag{4.48}$$

$$f_{ij}^t = T_{t+1} x_{ij}^t, \quad i = 1, \ldots, n, \quad j = 1, \ldots n \quad t = 1, \ldots, n-1 \tag{4.49}$$

$$x_{ij}^t \in \{0,1\}, \quad i = 1,\ldots,n, \quad j = 1,\ldots n \tag{4.50}$$
$$t = 1,\ldots,n$$
$$T_t \geq 0, \quad t = 1,\ldots,n. \tag{4.51}$$

Dans ce modèle, en plus des traditionnelles variables $x_{ij}^t$, on retrouve les variables $C_t$ et $T_t$ donnant les temps de complétion et les retards pour chacune des positions ainsi que les variables $f_{ij}^t$ donnant le coût associé aux décisions $(i,j)^t$. Lorsque $x_{ij}^t$ vaut 1, cela signifie que la pièce $j$ est effectuée en position $t+1$ et le coût associé à cette décision est le retard de la pièce située en position $t+1$ tandis que lorsque $x_{ij}^t = 0$, alors $f_{ij}^t = 0$. Les contraintes (4.43)-(4.45) sont les contraintes de flot du problème de plus court chemin dans le graphe multiparti. Les contraintes (4.46)-(4.47) permettent de mesurer les temps de complétion en fonction des positions. Les contraintes (4.48) permettent de mesurer les retards en fonction de la position. Les contraintes (4.49) permettent d'évaluer le retard associé à la décision $(i,j)^t$. À première vue, le sous-problème pour la formulation $DW(T3')$ semble très difficile à résoudre puisqu'il s'agit d'un problème d'optimisation non-linéaire en nombres entiers. On peut cependant résoudre ce problème par programmation dynamique.

Soit $f(j,t,\tau)$ une fonction donnant le coût réduit de la séquence optimale dans laquelle la pièce $j$ est planifiée en position $t$ au moment $\tau$, où $\tau = 0,\ldots,\tau_{max}$. Alors on peut résoudre $SP(T3')$ en utilisant cette récursion :

$$f(1,1,0) = 0$$
$$f(j,t,\tau) = \min_{i \in N \setminus \{j\}} \{f(i,t-1,\tau - s_{ij} - p_j) + \max\{\tau - d_j, 0\} - \pi_j\}$$

La solution optimale à $SP(T3')$ consiste ainsi à trouver

$$f(1,n+1,-) = \min_{\tau=0,\ldots,\tau_{max}} f(1,n,\tau)$$

qui donne en fait la valeur de la meilleure séquence des $n+1$ pièces peu importe le moment de fin. Comme on génère dans le pire cas $O(\tau_{max} n^2)$ états et que pour chaque état, on doit effectuer $n-1$ évaluations de la fonction, on obtient une procédure

ayant comme complexité $O(\tau_{max} n^3)$ (contre $O(n^3)$ pour la formulation $DW(T3)$ de Picard et Queyranne, 1978). Cette complexité peut paraître à première vue élevée. En effet, s'il y a trente pièces dans le problème et que l'horizon de planification est de 1000 unités, on peut générer en théorie avec cette approche environ 27 millions d'états. En pratique, le recours à certaines règles de dominance peut heureusement permettre d'accélérer le processus. Par exemple, comme la fonction objectif $\sum T_j$ est non décroissante, on sait que l'état $(j, t, \tau_1)$ domine l'état $(j, t, \tau_2)$ si $f(j, t, \tau_1) \leq f(j, t, \tau_2)$ et $\tau_1 \leq \tau_2$ et que conséquemment l'état $(j, t, \tau_2)$ ne mérite pas d'être exploré. De plus, dans un contexte de génération de colonnes, on n'est obligé de résoudre ce sous-problème exactement qu'à la toute fin, lorsqu'on doit prouver qu'il n'y a plus de colonnes de coût réduit négatif (et qu'on a atteint la solution optimale). Cela implique qu'au cours des itérations intermédiaires on peut se contenter d'explorer partiellement l'espace des états de programmation dynamique sans que cela n'empêche le processus de génération de colonnes de converger vers la solution optimale. Dans notre implantation, cela a permis de diminuer considérablement le temps de calcul lié au *pricing* des colonnes.

Cependant, même en ayant recours à diverses stratégies d'accélération, calculer la relaxation linéaire de $DW(T3')$ demeure plus difficile que calculer celle de $DW(T3)$. Est-ce que le modèle $DW(T3')$ est beaucoup plus fort que $DW(T3)$ ? Le tableau 4.8, contenant les valeurs des bornes inférieures obtenues pour cinq exemplaires de $1|s_{ij}|\sum T_j$, indique que oui.

Tableau 4.8 – Bornes obtenues avec $DW(T3')$

| Type | Problème | $LP(DW(T3))$ | $saut(\%)$ | $LP(DW(T3'))$ | $saut(\%)$ | $OPT$ |
|---|---|---|---|---|---|---|
| $1\|s_{ij}\|\sum T_j$ | *prob*408 | 5343.5 | 5.61 | 5656.8 | 0.06 | 5660 |
| | *prob*503 | 3479.7 | 0.49 | 3494.4 | 0.07 | 3497 |
| | *prob*507 | 7137.9 | 1.21 | 7201.5 | 0.33 | 7225 |
| | *prob*508 | 1642.2 | 14.25 | 1900.0 | 0.78 | 1915 |
| | *prob*608 | 2276.6 | 51.89 | 4660.5 | 1.53 | 4732 |

En analysant les résultats, on s'aperçoit que le nouveau modèle donne de bien meil-

Tableau 4.9 – Temps en secondes pour calculer $LP(DW(T3))$ et $LP(DW(T3'))$

| Type | Problème | nbPièces | $LP(DW(T3))$ | $LP(DW(T3'))$ |
|---|---|---|---|---|
| $1\|s_{ij}\|\sum T_j$ | *prob*408 | 15 | $\leq 1$ | 2 |
| | *prob*503 | 25 | 3 | 46 |
| | *prob*507 | 25 | 3 | 157 |
| | *prob*508 | 25 | 3 | 75 |
| | *prob*608 | 35 | 70 | 2303 |

leures bornes. Par exemple, pour le problème *prob608*, le nouveau modèle a donné une borne de 4660.5, alors que la formulation $DW(T3)$ avait donné une borne de 2276.6. Pour cet exemplaire, on a donc fait passer le saut d'intégrité de 51.89% à 1.53 %. On obtient également une amélioration significative de la borne pour les problèmes *prob*508 et *prob*408 alors que pour *prob*503 et *prob*507 les gains sont beaucoup moins importants (quoique dans ces deux cas, la borne donnée par $DW(T3)$ était déjà très bonne).

Pour obtenir de petits sauts d'intégrité, la formulation $DW(T3')$ apparaît donc comme l'approche de choix. En plus, on peut modéliser avec celle-ci l'objectif $\sum w_j T_j$, ce qui n'est pas possible avec $DW(T3)$. Pour ce faire, il suffit de résoudre comme sous-problème cette récursion :

$$f(1,1,0) = 0$$
$$f(j,t,\tau) = \min_{i \in N \setminus \{j\}} \{f(i,t-1,\tau - s_{ij} - p_j) + w_j \max\{\tau - d_j, 0\} - \pi_j\}$$

Cette formulation possède cependant un bien grand défaut : résoudre sa relaxation linéaire peut être très couteux comme l'illustre le tableau 4.9. Pour les exemplaires comptant au-delà de 35 pièces, la formulation $DW(T3')$ est à peu près inutilisable, alors que $DW(T3)$ l'est parfois.

## 4.4 Algorithme d'évaluation et de séparation progressive

On a vu diverses formulations de programmation en nombres entiers pour le problème de commis-voyageur dépendant du temps de taille et de force différentes. Quelle formulation est la meilleure ? Dans les trois dernières sections, on a tenté de répondre à cette question en s'attardant à un bon critère théorique, soit la qualité de la relaxation linéaire. En effet, on s'entend généralement pour dire que plus une formulation est forte, meilleure elle est. Or en pratique, la meilleure formulation est encore celle qui est la plus facile à résoudre, qu'elle soit forte ou non. Dans cette section, on présente ainsi une règle de branchement compatible avec tous les modèles vus dans le chapitre. On présente ensuite des résultats numériques pour divers exemplaires de $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$ obtenus un algorithme d'énumération et de séparation progressive pouvant exploiter différentes bornes. Les résultats obtenus semblent confirmer le fait qu'il est souvent avantageux d'avoir recours à des bornes plus fortes pour restreindre l'espace des solutions à explorer explicitement avec l'algorithme d'évaluation et de séparation progressive.

### 4.4.1 Description du branchement

Pour toutes les formulations, on propose un branchement sur les variables $x_{ij}^t$. Pour $T2$ et $T3$, on connaît explicitement les valeurs de ces variables à la relaxation linéaire, alors que pour les formulations de génération de colonnes $DW(T3)$ et $DW(T3')$, elles peuvent être calculées comme suit :

$$x_{ij}^t = \sum_{p \in \Omega} x_{ij}^{tp} \theta_p$$

où $x_{ij}^{tp}$ est, rappelons-le, une constante valant 1 si le chemin $p$ emprunte l'arc $(i, j)$ en position $t$. Si les valeurs des variables $x_{ij}^t$ sont entières, alors on a résolu le problème. Autrement, on doit séparer.

Soit $x_{ij}^t$ une variable fractionnaire candidate à un branchement. Il existe de nombreux critères afin d'identifier cette variable : valeur plus proche de 0.5, valeur plus proche de 1, pénalités, etc. Lors de nos expériences, les approches suivantes sont cependant apparues supérieures aux critères plus proche de 0.5 et plus proche de 1 (nous n'avons pas testé d'approches de pénalités car elles sont assez délicates à implanter). Pour le problème $1|s_{ij}|\sum C_j$, on a choisi de brancher d'abord sur les variables fractionnaires $x_{ij}^t$ les plus tôt dans la séquence alors que pour $1|s_{ij}|\sum T_j$, on a choisi de brancher d'abord sur les variables fractionnaires les plus tard dans la séquence. En effet, en procédant comme suit, on fixe très tôt dans l'algorithme de branchement les variables qui sont susceptibles d'avoir le plus d'impact sur le coût de la solution (puisque pour le problème $1|s_{ij}|\sum C_j$, les plus gros coûts sont en début de séquence alors que pour $1|s_{ij}|\sum T_j$ ils sont plutôt en fin de séquence) et cela peut permettre d'identifier plus rapidement les séquences non prometteuses. Une fois la variable $x_{ij}^t$ identifiée, on impose la décision $x_{ij}^t = 1$ sur la branche de gauche et $x_{ij}^t = 0$ sur la branche de droite. On doit noter que ces décisions peuvent être imposées sans ajouter de contraintes au modèle. Pour appliquer la décision $x_{ij}^t = 0$, il suffit en effet de retirer du modèle la variable (ou l'arc) concernée alors que pour imposer $x_{ij}^t = 1$ il suffit de retirer du modèle toutes les variables (ou tous les arcs) associées à la position $t$ (sauf $x_{ij}^t$ évidemment).

En utilisant cette règle de branchement au sein d'un algorithme classique d'évaluation et de séparation progressive de programmation en nombres entiers, on obtient un algorithme pouvant résoudre en temps fini, mais exponentiel, n'importe lequel des programmes en nombres entiers présentés dans ce chapitre.

L'encadré 4.4 présente le pseudo-code décrivant l'algorithme *BranchAndCut* utilisé pour résoudre les problèmes $1|s_{ij}|\sum C_j$ et $1|s_{ij}\sum T_j$.

Dans l'algorithme *BranchAndCut*, on a choisi comme stratégie de sélection de noeud la méthode *depth until leaf* implantée dans *Gencol* (un mélange de profondeur d'abord et de meilleur d'abord). On a eu recours à la stratégie de dominance basée sur les coûts réduits pour éliminer des variables $x_{ij}^t$ du modèle. De plus, on a choisi d'ajouter une seule coupe à la fois pour chaque famille. Le paramètre *tsp* permet de spécifier

**Algorithme 4.4** Pseudo-code de l'algorithme *BranchAndCut*

Soit $\mathcal{L}$ la liste des noeuds, $X_0^i$ la liste des variables fixées à 0 au noeud $n_i$, $X_1^i$ la liste des variables fixées à 1, $\bar{z}$ une borne supérieure pour le problème, $\underline{z}^i$ la borne inférieure pour le noeud $n_i$, *tsp* un booléen qui indique si on ajoute des coupes de commis-voyageur et *maxRonde* le nombre de fois où on génère des coupes de cliques.

1. **(Initialisation)** $X_0^0 = \emptyset$, $X_1^0 = \emptyset$, $\mathcal{L} = \{n_0\}$, $\bar{z} = \infty$, $\underline{z}^0 = -\infty$, $compteur = 0$.

2. **(Critère d'optimalité)** Si $\mathcal{L} = \emptyset$, $\bar{z}$ est optimale. **STOP**.

3. **(Sélection d'un noeud)** Extraire de $\mathcal{L}$ le meilleur noeud $n_i$ à explorer en vertu du critère *depth until leaf*.

4. **(Dominance 4)** Appliquer la règle de dominance basée sur les coûts réduits et mettre à jour $X_0^i$.

5. **(Calcul de la borne)** Résoudre la relaxation linéaire du modèle considéré en tenant compte des ensembles $X_0^i$ et $X_1^i$ et mettre à jour $\underline{z}^i$ avec la valeur de la solution optimale de la relaxation linéaire, notée $\hat{x}$.

6. **(Évaluation de la solution)**
   **a)** Si $\underline{z}^i \geq \bar{z}$, retourner à l'étape 2.
   **b)** Si la solution $\hat{x}$ de la relaxation linéaire est entière, alors poser $\bar{z} = \underline{z}^i$, éliminer tous les noeuds $n_k \in \mathcal{L}$ tel que $\underline{z}^k \geq \bar{z}$ et retourner à l'etape 2.

7. **Identification de coupes**
   **a) Coupe de sous-tour** Si $tsp = VRAI$, trouver la coupe de sous-tour la plus violée par $\hat{x}$ et l'ajouter au modèle considéré.
   **b) Coupe de 2-couplages** Si $tsp = VRAI$, trouver la coupe de 2-couplages la plus violée par $\hat{x}$ et l'ajouter au modèle considéré.
   **c) Coupe de cliques** Si $compteur \leq MaxRonde$, trouver la coupe de clique la plus violée par $\hat{x}$ avec la procédure $MaxClique$ et l'ajouter au modèle. Incrémenter *compteur* si une coupe de clique violée a été trouvée.
   **d) Réoptimisation** Si une coupe a été ajoutée au modèle, retourner à l'étape 5.

8. **(Division)** Soit $x_f$ la variable sur laquelle on doit brancher (voir section 4.4). Créer les noeuds fils $k$ et $l$. Pour le fils $k$, poser $X_0^k = X_0^i \cup \{x_f\}$, $X_1^k = X_1^i$ et $\underline{z}^k = \underline{z}^i$. Pour le fils $l$, poser $X_0^k = X_0^i$, $X_1^k = X_1^i \cup \{x_f\}$ et $\underline{z}^l = \underline{z}^i$.

9. **(Mise à jour de la liste)**
Ajouter les noeuds $k$ et $l$ à $\mathcal{L}$ et retourner à l'étape 2.

si le modèle prévoit la séparation de coupes de commis-voyageur symétrique alors que le paramètre *maxRonde* permet de spécifier le nombre de fois où l'on génère des contraintes de cliques.

### 4.4.2 Résultats numériques : $1|s_{ij}|\sum C_j$

On considère la résolution du problème $1|s_{ij}|\sum C_j$ avec l'algorithme *BranchAndCut* utilisant les bornes de programmation linéaire obtenues à partir des formulations $DW(T3)^{tsp}_{cliques}$, $DW(T3)^{tsp,k=2}_{cliques}$ et $DW(T3)^{tsp,k=4}_{cliques}$. On a considéré cinq problèmes de petite ou moyenne taille tirés de *TSPLIB* et 10 problèmes de plus grande taille générés aléatoirement. Pour générer ces dix exemplaires, on a procédé selon la méthode proposée dans Fischetti *et al.* (1993) consistant à prendre au hasard $n$ points dans une grille 100 x 100 et à utiliser comme mesure de distance entre les points $(s_{ij})$ la norme euclienne arrondie à l'entier le plus près. Tous les tests ont été effectués sur une machine dotée d'un microprocesseur de 3.4 Ghz et de 2072 Megs de mémoire vive.

Le tableau 4.10 donne le temps en secondes et le nombre de noeuds qui ont été nécessaires afin de résoudre ces problèmes tests.

Les résultats du tableau 4.10 indiquent clairement que pour le problème $1|s_{ij}|\sum C_j$, il est très avantageux en pratique d'avoir recours à une élimination de $k$-cycles aggressive. En utilisant l'élimination des 4-cycles, on a été en mesure de résoudre tous les exemplaires tirés de *TSPLIB* en moins de quatre minutes, alors que pour les exemplaires *bayg29* et *bays29*, sans recours à l'élimination des $k$-cycles, plusieurs heures ont été nécessaires afin de résoudre ceux-ci.

Avec les formulations renforcées présentées dans ce chapitre, on est capable de résoudre en temps raisonnable des exemplaires de $1|s_{ij}|\sum C_j$ comptant au plus 50 pièces. Cela se compare favorablement aux approches de Lucena (1990) et Bianco *et al.* (1993) développées pour ce problème, utilisant une borne dérivée de $T3$ par relaxation lagrangienne calculable en $O(n^3)$, avec lesquelles ceux-ci ont respectivement

Tableau 4.10 – Résultats pour $1|s_{ij}|\sum C_j$ avec l'algorithme d'énumération et de séparation progressive

| Type | *nbPieces* | $DW(T3)^{tsp}_{cliques}$ | | $DW(T3)^{tsp,k=2}_{cliques}$ | | $DW(T3)^{tsp,k=4}_{cliques}$ | |
|---|---|---|---|---|---|---|---|
| | | temps | noeuds | temps | noeuds | temps | noeuds |
| 17 | *gr*17 | 72 | 500 | 4 | 33 | 3 | 1 |
| 21 | *gr*21 | 197 | 370 | 13 | 43 | 10 | 1 |
| 24 | *gr*24 | 258 | 453 | 10 | 14 | 15 | 1 |
| 29 | *bays*29 | 13977 | 2756 | 600 | 382 | 76 | 16 |
| 29 | *bayg*29 | 30014 | 3904 | 1571 | 648 | 191 | 51 |
| 40 | *dTsp*40.0 | – | – | – | – | 6473 | 377 |
| 40 | *dTsp*40.1 | – | – | – | – | 1452 | 50 |
| 40 | *dTsp*40.2 | – | – | – | – | 1068 | 28 |
| 40 | *dTsp*40.3 | – | – | – | – | 629 | 24 |
| 40 | *dTsp*40.4 | – | – | – | – | 299 | 4 |
| 50 | *dTsp*50.0 | | | | | 1364 | 5 |
| 50 | *dTsp*50.1 | – | – | – | – | 56240 | 571 |
| 50 | *dTsp*50.2 | – | – | – | – | 3668 | 8 |
| 50 | *dTsp*50.3 | – | – | – | – | 47771 | 541 |
| 50 | *dTsp*50.4 | – | – | – | – | 109586 | 1514 |

été en mesure de résoudre des problèmes de 30 et 35 pièces. Notre approche semble cependant dominée par celle de Fischetti *et al.* (1993) qui est basée sur le recours d'une borne assez faible mais facile à calculer (soit en $O(n^2)$ opérations). Avec leur approche, ceux-ci ont en effet été en mesure de résoudre certains exemplaires comptant 60 pièces en moins de 1800 secondes (sur un ordinateur datant d'avant 1991). Pour des exemplaires de cinquante pièces, générés de la même façon que nos exemplaires, ceux-ci ont obtenus des arbres comptant en moyenne plus de 100 000 noeuds, alors qu'on a obtenu des arbres comptant typiquement moins de 100 noeuds. En contrepartie, on doit faire remarquer qu'avec l'approche présentée ici, il peut être plus facile de tenir compte de contraintes supplémentaires (comme des fenêtres de temps ou de positions pour les pièces) qu'avec celle de Fischetti *et al.* (1993).

### 4.4.3 Résultats numériques : $1|s_{ij}|\sum T_j$

On considère la résolution du problème $1|s_{ij}|\sum T_j$ avec l'algorithme *BranchAndCut* utilisant les bornes données par les formulations $DW(T3)^{tsp}_{cliques}$, et $DW(T3')^{tsp}_{cliques}$, ainsi qu'avec cplex9.0 (modèle $T3$, paramètres par défaut avec *strong branching*).

On a utilisé les problèmes tests de Rubin et Ragatz (1995). Dans cette collection de problèmes, on retrouve 32 exemplaires de 15, 25, 35 et 45 pièces. Chaque exemplaire est caractérisé selon trois facteurs : la variance des temps de traitement (haute ($H$) ou basse ($L$)), la proportion de pièces en retard (modérée ($M$) ou basse ($L$)) et la plage des dates de livraison (étroite ($N$) ou large ($W$)). On peut par exemple s'attendre pour un exemplaire de type *HLW* à ce qu'il y ait beaucoup de variance dans les temps de traitement, qu'il y ait peu de pièces en retard dans sa solution optimale et que la plage des dates de livraison soit assez large. Ces exemplaires sont disponibles à l'adresse suivante :

*http ://www.bus.msu.edu/mgt/research.html*

Le tableau 4.11 donne le temps (en secondes) ainsi que le nombre de noeuds qu'a nécessité la résolution de chaque exemplaire. Les exemplaires dont on connaissait déjà la solution optimale sont notés d'une étoile, alors que ceux pour lesquels on a prouvé l'optimalité pour la première fois sont notés avec deux étoiles. Comme pour l'expérience précédente, tous les tests ont été effectués sur une machine dotée d'un microprocesseur de 3.4 Ghz et de 2072 Megs de mémoire vive. On a limité le temps de résolution à 100 000 secondes.

En observant le tableau 4.11, on remarque d'abord qu'on a été capable de trouver la solution optimale à de nombreux problèmes qui étaient encore ouverts (soit 11 problèmes sur 17 ). On doit cependant noter qu'hormis l'algorithme d'énumération implicite de Ragatz, décrit dans Tan *et al.* (2000), qui utilise une borne très faible mais facile à calculer, aucune approche exacte n'avait encore été utilisée sur ces exemplaires. De plus, pour les problèmes non résolus à l'optimalité, on a obtenu la plupart du temps

Tableau 4.11 – Résultats pour $1|s_{ij}|\sum T_j$ avec l'algorithme d'énumération et de séparation progressive

| nbPieces | Probleme | Type | $T3^{cplex}$ | | $DW(T3)^{tsp}_{cliques}$ | | $DW(T3')^{tsp}$ | | OPT |
|---|---|---|---|---|---|---|---|---|---|
| | | | temps | noeuds | temps | noeuds | temps | noeuds | |
| 15 | 401 | LLN | 24 | 79 | 4 | 27 | 4 | 4 | 90* |
| 15 | 402 | LLW | 38 | 101 | 2 | 55 | 1 | 27 | 0* |
| 15 | 403 | LMN | 4 | 10 | 14 | 251 | 2 | 3 | 3418* |
| 15 | 404 | LMW | 328 | 1111 | 2340 | 27277 | 1 | 1 | 1067* |
| 15 | 405 | HLN | 1 | 1 | 1 | 59 | 2 | 33 | 0* |
| 15 | 406 | HLW | 5439 | 29000 | 1 | 55 | 1 | 26 | 0* |
| 15 | 407 | HMN | 3 | 20 | 4 | 39 | 15 | 14 | 1861* |
| 15 | 408 | HMW | 145 | 555 | >100000 | | 5 | 3 | 5660* |
| 25 | 501 | LLN | 1061 | 238 | 56 | 48 | 179 | 13 | 261** |
| 25 | 502 | LLW | 192 | 51 | 161 | 229 | 13 | 102 | 0* |
| 25 | 503 | LMN | 96 | 39 | 55 | 176 | 137 | 13 | 3497** |
| 25 | 504 | LMW | >100000 | | 12 | 79 | 63 | 31 | 0* |
| 25 | 505 | HLN | 481 | 139 | 7 | 69 | 63 | 31 | 0* |
| 25 | 506 | HLW | 978 | 633 | 36 | 149 | 9 | 48 | 0* |
| 25 | 507 | HMN | 99 | 210 | 220 | 752 | 582 | 26 | 7225** |
| 25 | 508 | HMW | 2586 | 1976 | 4621 | 4368 | 243 | 4 | 1915** |
| 35 | 601 | LLN | 9686 | 550 | 473 | 35 | 1371 | 6 | 12** |
| 35 | 602 | LLW | 600 | 39 | 259 | 248 | 40 | 130 | 0* |
| 35 | 603 | LMN | 749 | 2215 | >100000 | | >100000 | | 17587** |
| 35 | 604 | LMW | >100000 | | >100000 | | >100000 | | 19092 |
| 35 | 605 | HLN | >100000 | | 828 | 93 | 15435 | 19 | 228** |
| 35 | 606 | HLW | >100000 | | 588 | 198 | 71 | 143 | 0* |
| 35 | 607 | HMN | 1956 | 1039 | 78768 | 39965 | 19760 | 75 | 12969** |
| 35 | 608 | HMW | > 100000 | | >100000 | | 47787 | 132 | 4732** |
| 45 | 701 | LLN | >100000 | | 3400 | 137 | 51430 | 51 | 97** |
| 45 | 702 | LLW | >100000 | | 2003 | 375 | 374 | 232 | 0* |
| 45 | 703 | LMN | >100000 | | >100000 | | >100000 | | 26533 |
| 45 | 704 | LMW | >100000 | | >100000 | | >100000 | | 16577 |
| 45 | 705 | HLN | >100000 | | 2636 | 359 | >100000 | | 200** |
| 45 | 706 | HLW | >100000 | | 1938 | 382 | 234 | 119 | 0* |
| 45 | 707 | HMN | >100000 | | >100000 | | >100000 | | 23797 |
| 45 | 708 | HMW | >100000 | | >100000 | | >100000 | | 22829 |

Tableau 4.12 – Exemplaires non résolus

| Problème | $BB_{Ragatz}$ | | Anciennes solutions | | | $DW(T3')$ | |
|---|---|---|---|---|---|---|---|
| | valeur | écart (%) | valeur | écart (%) | Approche | valeur | écart(%)) |
| prob604 | 19277 | 0.0 | [19114,19132] | -0.8 | ACO | 19092 | -0.9 |
| prob703 | 27097 | 0.0 | [26732,26758] | -1.3 | SA | 26533 | -2.1 |
| prob704 | 15941 | 0.0 | [15375,15391] | -3.5 | RSPI | 16577 | +3.9 |
| prob707 | 25070 | 0.0 | [24005, 24030] | -4.2 | ACO | 23797 | -5.1 |
| prob708 | 24123 | 0.0 | [22978,23002] | -4.7 | RSPI | 22829 | -5.4 |

des solutions de très bonne qualité. Le tableau 4.12 compare les solutions obtenues avec les meilleures solutions obtenues précédemment, recensées dans les articles de Tan *et al.* (2000) et de Gagné *et al.* (2001). Comme dans ces deux articles, on ne fournissait que l'amélioration en pourcentage obtenue par rapport à la solution obtenue avec l'algorithme d'énumération implicite tronqué de Ragatz, on retrouve dans le tableau une plage de valeurs possibles pour ces solutions. En plus de donner les anciennes meilleures solutions connues, on spécifie avec quel algorithme elles ont été trouvées (ACO : algorithme fourmi de Gagné *et al.* (2001), SA : recuit simulé de Tan *et al.* (1997) ; RSPI : algorithme de type multistart de Ragatz *et al.*, 1995) et on donne les solutions que l'on a obtenues avec notre algorithme d'évaluation et de séparation progressive. Dans presque tous les cas, on a obtenu de meilleures solutions avec notre algorithme d'évaluation et de séparation progressive. Le temps de calcul nécessité pour générer ces solutions peut sembler élevé (plus de 24 heures). Ces solutions ont cependant été obtenues très tôt dans l'algorithme (généralement à l'intérieur des deux premières heures), le reste du temps de calcul ayant été consacré à tenter de les améliorer ou à prouver leur optimalité.

Quelle est la meilleure des approches pour $1|s_{ij}|\sum T_j$ ? Même si aucune approche n'est complètement dominée, l'approche $DW(T3')$ semble être la plus efficace, ou du moins la plus robuste. En effet, seulement 2 problèmes non résolus avec $DW(T3')$ l'ont été avec $DW(T3)^{tsp}_{clique}$ (exemplaire 705) ou $T3^{cplex}$(exemplaire 603) alors que les approches $T3^{cplex}$ et $DW(T3)$ ont respectivement échoué 8 et 3 fois là ou d'autres approches ont réussi à résoudre le problème. L'approche $T3^{cplex}$ est apparue spécialement inefficace pour les problèmes sans retard, alors que pour les problèmes de type $HMN$ (variance des temps de traitement élévée, facteur de retard moyen, fenêtre des

Tableau 4.13 – Exemplaires sans retard

| Problème | $DW(T3)^{tsp}$ | | $DW(T3)$ | |
|---|---|---|---|---|
| | temps | noeuds | temps | noeuds |
| prob402 | 2 | 55 | 1 | 13 |
| prob405 | 1 | 59 | 1 | 14 |
| prob406 | 1 | 59 | 1 | 14 |
| prob502 | 161 | 229 | 4 | 25 |
| prob504 | 7 | 69 | 6 | 34 |
| prob505 | 7 | 69 | 8 | 25 |
| prob602 | 258 | 248 | 28 | 41 |
| prob606 | 588 | 198 | 20 | 34 |
| prob702 | 2003 | 375 | 173 | 55 |
| prob706 | 1938 | 382 | 131 | 57 |

date de livraison étroite), celle-ci a semblé être la plus efficace. Les résultats obtenus avec l'approche $DW(T3)^{tsp}_{cliques}$ sont dans l'ensemble décevants. L'ajout d'inégalités valides et l'utilisation d'un branchement spécialisé pour le problème n'a visiblement pas semblé apporté un gain quelconque par rapport à $T3^{cplex}$ (à part pour les exemplaires sans retard). Pire, l'ajout d'inégalités valides pour les exemplaires sans retard a nui plus que d'autres choses, comme le démontre le tableau 4.13, comparant les résultats obtenus avec et sans ajout d'inégalités valides. Pour les problèmes sans retard, on a observé que le nombre de contraintes de 2-couplages violées était assez grand et qu'en les ajoutant on ne faisait parfois que rendre la solution de plus en plus fractionnaire sans modifier le saut d'intégrité. D'où cette différence assez importante entre les temps de calcul obtenus avec les deux méthodes sur cette famille de problèmes.

## 4.5 Conclusion

Dans ce chapitre, on a vu comment des formulations de programmation en nombres entiers connues pour le problème de commis-voyageur dépendant du temps pouvaient être étendues pour modéliser certains problèmes d'ordonnancement sur une machine avec temps de réglage dépendant de la séquence, soit $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$. On a

ensuite montré comment on pouvait renforcer ces formulations en utilisant à la fois des techniques de coupes du problème de commis-voyageur, du problème de stable de poids maximum et des reformulations de Dantzig-Wolfe pour finalement confirmer lors d'expériences numériques qu'il était souvent utile d'avoir recours à ces techniques. En effet, on a pu résoudre certains exemplaires avec les formulations renforcées que l'on n'a pu résoudre avec les formulations de base.

Pour certains exemplaires, il peut malheureusement être très difficile de résoudre la relaxation linéaire des formulations renforcées. On pense cependant qu'en raffinant les règles utilisées pour restreindre l'espace des états visités lors du problème de *pricing* des itérations intermédiaires de génération de colonnes on pourrait grandement accélérer le calcul des relaxations linéaires.

De plus, on a parfois obtenu des arbres de recherche d'assez grande taille même en ayant recours aux formulations renforcées. Cela suggère qu'il serait probablement pertinent de resserrer davantage les formulations, en interdisant par exemple la formation de plus longs cycles ou en ajoutant d'autres types de coupes.

Malgré tout, on peut dire que les modèles présentés dans ce chapitre ont mené à des processus de résolution viables pour $1|s_{ij}|\sum C_j$ et pour $1|s_{ij}|\sum T_j$. Il serait donc intéressant de voir si on ne pourrait pas à partir de ces formulations construire des approches de résolution pour d'autres problèmes d'ordonnancement sur une machine avec temps de réglage dépendant de la séquence (comme $1|s_{ij}|\sum w_j T_j$ ou $1|s_{ij}|\sum U_j$ par exemple).

# CHAPITRE 5 : PROBLÈMES MILITAIRES

## 5.1 Introduction

Le groupe tactique des forces armées canadiennes est confronté à divers problèmes de confection d'horaires lors de la planification de ses activités de vol. Ce chapitre s'intéresse à deux de ces problématiques, soit la fabrication des lignes d'affectation et la confection des horaires du personnel navigant, et vise le développement d'une approche de résolution où ces deux problèmes sont traités de façon intégrée.

Ce problème de fabrication d'horaires peut être vu comme un cas particulier de la formulation unifiée de Desaulniers et *al.* (1998) pour les problèmes de routage de véhicules et d'horaires de personnel développée autour du principe de décomposition de Dantzig-Wolfe (1960).

Ce problème peut également être vu comme un problème d'ordonnancement sur machines parallèles avec contraintes supplémentaires où l'approche de stable vue au chapitre 2 peut être utilisée afin de concevoir une approche de résolution polyédrale.

On explore ainsi dans ce travail diverses formulations issues de ces deux paradigmes de modélisation et des stratégies de résolution envisagées pour les résoudre, basées sur le principe de décomposition de Dantzig-Wolfe (1960).

## 5.2 Généralités

Le problème considéré ici survient lors de la planification des missions pour la flotte des hélicoptères Griffon146. Dans ce problème, on cherche à affecter le personnel de

vol et les hélicoptères à des missions de façon à optimiser un critère donné (utilisation des hélicoptères, satisfaction du personnel) tout en respectant les contraintes auxquelles sont soumises les ressources (disponibilité du personnel et des hélicoptères, formation minimum du personnel pour certaines missions, nombre d'heures minimum accordées à la formation, priorité des missions, etc). Cette section se veut une présentation générale de divers éléments du problème (voir Desaulniers *et al.* 1996, pour une description plus approfondie).

### 5.2.1 Les missions

Le groupe tactique d'aviation des Forces Armées Canadiennes organise des missions de natures très variées. Les hélicoptères Griffon sont aussi bien utilisés pour les missions de sauvetage que pour le transport de fret et de dignitaires.

Ces missions sont effectuées pour le compte de différentes organisations de l'armée, suite à leur demande. Une telle requête contient un certain nombre d'informations comme :

- le nom du client ;
- les détails de mission ;
- le nombre de passagers ;
- le type de fret à transporter et son poids ;
- la fenêtre de temps à l'intérieur de laquelle la mission doit débuter ;
- la durée du temps de vol ;
- le nombre d'hélicoptères requis.

Une mission peut contenir une ou plusieurs étapes, liées entre elles par des contraintes de préséance. Une mission peut être obligatoire ou optionnelle. Les missions obligatoires sont toujours acceptées tandis que les missions optionnelles peuvent être rejetées, moyennant une pénalité calculée selon leur niveau de priorité. Avec cette politique, on ne peut pas, par exemple, exécuter une mission administrative optionnelle

si celle-ci empêche l'exécution d'une mission opérationnelle, car cette dernière possède un niveau de priorité plus élevé.

L'acceptation d'une requête de mission ou son rejet est l'un des éléments du problème de fabrication d'horaires de missions.

### 5.2.2 Le personnel navigant

L'exécution de ces missions nécessite évidemment du personnel navigant. Une équipe de Griffon CH146 est composée de deux pilotes (un capitaine et un premier officier). De plus, certains vols nécessitent un ingénieur de vol. Le niveau de formation requis peut être différent d'une mission à l'autre. Il faut donc tenir compte du niveau de formation des pilotes lors de la construction des horaires de vol afin de ne pas leur confier des positions dans une mission qui dépassent leurs compétences.

Le personnel navigant est assujetti à un ensemble de règles semblables à celles retrouvées dans le contexte de transport aérien civil. Il existe des règles concernant

- la durée maximum d'une journée de travail ;
- le temps de vol maximum accumulé au cours d'une journée ;
- le temps de vol maximum accumulé au cours d'une période ;
- le repos entre deux journées de travail.

En plus de ces règles gérant le stress au travail, il existe aussi des objectifs de formation à rencontrer via l'exécution de missions d'entraînement. Il existe en effet des seuils minimaux à respecter pour les heures consacrées à diverses séquences de formation. On doit ainsi prévoir des missions d'entraînement aux horaires des pilotes afin que ces objectifs soient tous rencontrés au cours d'une période donnée.

### 5.2.3 Les hélicoptères et les lignes d'affectation

Les hélicoptères utilisés pour effectuer les différentes missions au sein du groupe tactique sont des Griffon146. Ceux-ci peuvent transporter jusqu'à 15 personnes, à une vitesse maximale de 260 km/h. Comme les hélicoptères nécessitent un entretien périodique et peuvent être sujets à des bris, le nombre d'hélicoptères en fonction est généralement plus petit que le nombre d'hélicoptères détenus par le groupe.

Puisqu'on ne sait pas exactement quels seront les hélicoptères disponibles au cours d'une assez longue période, on fait la planification des missions en faisant appel au concept de lignes d'affectation. Une ligne d'affectation représente la suite de vols effectuée par un aéronef générique situé à une base donnée. On utilise le terme aéronef générique puisque ce n'est pas nécessairement le même aéronef qui effectue les missions effectuées sur une ligne d'affectation. Par contre, il s'agit toujours d'aéronefs de même type. Le nombre de lignes d'affectation pour une base dépend de la taille de la flotte, des équipages disponibles et des contraintes d'entretien. Une fois que l'on a construit les lignes d'affectations, on affecte les hélicoptères selon leur disponibilité respective au cours de l'horizon de planification. Ce problème d'affectation des hélicoptères aux lignes d'affectation n'est cependant pas considéré dans ce travail en raison notamment de la nature stochastique du problème de gestion de l'entretien.

### 5.2.4 Description du problème

Le problème de fabrication d'horaires de missions, noté FHM, revient à construire simultanément les horaires pour les aéronefs génériques (ou lignes d'affectation) et les horaires pour le personnel navigant. Le problème se divise naturellement en deux parties qui ne sont cependant pas indépendantes.

### 1ère partie : construction de lignes d'affectation

La première étape consiste à sélectionner, parmi les différentes missions possibles, celles qui seront effectuées au cours de la période de planification. Un ensemble de missions retenues est réalisable s'il existe un horaire de vols couvrant toutes les étapes de ces missions qui n'utilise pas, en tout temps, plus d'aéronefs que le nombre en service et pour lequel quelques contraintes relatives au personnel navigant sont respectées. On appelle cette étape le problème de fabrication des lignes d'affectation, une ligne d'affectation étant un horaire de vols pouvant être effectué par un aéronef d'un certain type.

Rancourt (1998) et Guigue (2000) ont travaillé sur ce problème de planification. Les deux formulations sont des variantes de la formulation unifiée de Desaulniers *et al.* (1994). Dans Rancourt (1998), on a un modèle de partionnement d'ensemble où les colonnes représentent les séquences de vols légales pour chacune des lignes d'affectation. Le nombre de colonnes étant très grand, celles-ci sont plutôt générées à l'aide d'un oracle. Comme ce sous-problème est capable de générer n'importe quelle séquence de vols légale, l'approche de Rancourt (1998) est exacte. Une des faiblesses de ce modèle réside dans la difficulté de résolution des sous-problèmes : trop de ressources sont nécessaires pour modéliser toutes les contraintes locales que l'on peut retrouver au sein d'une ligne d'affectation.

Suite à cela, Guigue (2000) a proposé une approche basée sur la génération de tâches *a priori*, une tâche étant une suite de vols débutant et se terminant à la même base qui est effectuée par le même équipage et le même aéronef. Au lieu de construire les lignes d'affectation à partir des vols, ce qui demande énormément de travail au niveau des sous-problèmes, on les construit plutôt à partir des tâches, dont la réalisabilité des contraintes locales (équipage, compatibilité de fret) a déjà été établie en prétraitement. Le sous-problème ne peut plus maintenant générer toutes les séquences de vols possibles, puisque certaines tâches ne sont pas considérées. Par contre, on a besoin d'utiliser moins de ressources. Cette approche est évidemment sous-optimale si on ne considère pas toutes les tâches possibles. La stratégie d'agrégation de vols utilisée par

Guigue (2000) est retrouvée dans bien d'autres applications, comme les problèmes de livraison et cueillette (Ioachim *et al.*, 1995).

### $2^{\text{ième}}$ partie : fabrication des horaires d'équipages

La deuxième partie consiste à construire les horaires pour les membres d'équipage en fonction de l'horaire de vol des aéronefs de façon à maximiser la satisfaction du personnel, tout en s'assurant que ces horaires respectent les diverses règles de travail régissant le personnel navigant (repos minimum entre deux journées de travail, durée maximum d'une journée de travail et heures de vol). De plus, on doit prévoir des missions d'entraînement pour les différents membres d'équipages de façon à rencontrer les objectifs de formation.

Ce problème de fabrication d'horaires d'équipage est similaire au problème classique de fabrication de rotations d'équipages et au problème de fabrication d'horaires mensuels de travail issus du contexte de l'aviation civile où les approches de génération de colonnes (Anbil *et al.*, 1998 ; Gamache *et al.*. 1999) ont connu de beaux succès. Ces approches reviennent essentiellement à trouver quels horaires légaux de personnel doivent être retenus afin de partitionner les différentes tâches de façon optimale. Comme l'ensemble des horaires légaux est de très grande taille, les horaires susceptibles d'améliorer la solution sont générés dynamiquement à l'aide d'un problème auxiliaire.

#### Une approche simultanée de génération de colonnes

On pourrait être tenté, en vue de résoudre le problème de fabrication d'horaires de missions (FHM), d'adopter une approche séquentielle où le problème de fabrication des lignes d'affectation serait résolu dans un premier temps et dont la solution servirait ensuite à contruire un problème de fabrication des horaires d'équipages, qui serait résolu dans un deuxième temps.

Or, le problème de fabrication de lignes d'affectation est fortement lié au problème de fabrication d'horaires d'équipage puisque 80 % des missions à planifier pour les lignes d'affectation sont des missions d'entraînement exécutées afin de rencontrer les seuils minimaux de formation du problème d'horaire de personnel. Les deux problèmes sont difficilement séparables : une approche traitant simultanément ces deux problèmes est donc souhaitable.

Desaulniers *et al.* (1996) et Rancourt et Savard (2001) ont proposé des formulations de génération de colonnes afin de résoudre le problème de fabrication simultanée des lignes d'affectation et des horaires d'équipage. Il existe certaines différences entre ces deux formulations, mais l'esprit est le même. Dans le problème maître, on retrouve des colonnes représentant les lignes d'affectation possibles et des colonnes représentant les horaires d'équipages possibles et celles-ci sont générées à l'aide de sous-problèmes de plus court chemin avec contraintes de ressources. Ces modèles peuvent être vus comme le produit d'une fusion entre la formulation de Guigue (2000) pour le problème de construction des lignes d'affectation et la formulation typique de partitionnement d'ensemble des problèmes de rotation d'équipages. Avec cette formulation, la plupart des contraintes sont gérées au niveau des sous-problèmes. Le problème maître gère les contraintes de synchronisation de ligne d'affectation-équipages, c'est-à-dire les contraintes forçant la couverture d'une tâche par un équipage si celle-ci a été retenue dans une ligne d'affectation, et des contraintes permettant de modéliser les priorités entre les missions.

Ces formulations de génération de colonnes s'inscrivent directement dans le paradigme de modélisation développé par l'équipe Desrosiers-Soumis pour les problèmes de tournées de véhicules et d'horaires d'équipages et formalisé dans Desaulniers *et al.* (1998).

### Une approche simultanée avec une formulation de stable

Le présent travail se distingue de ceux de Desaulniers *et al.* (1996) et Rancourt et Savard (2001) dans la mesure où on approche plutôt le problème FHM avec des formulations issues du problème de stable de poids maximum semblable à celle proposée au chapitre 2.

On propose donc, avec cette perspective, une nouvelle formulation pour le problème de fabrication d'horaires de missions (FHM). Ce nouveau modèle, construit autour des équipages légaux plutôt que des membres du personnel navigant, offre une plus grande flexibilité que les modèles de Desaulniers *et al.* (1996) et Rancourt et Savard (2001) afin de modéliser les contraintes de composition d'équipages.

Le reste du chapitre est divisé comme suit. La section suivante se veut une présentation du modèle de base sous-jacent aux formulations de Desaulniers *et al.* (1996) et Rancourt et Savard (2001). La quatrième section est consacrée au nouveau modèle développé pour le problème de fabrication d'horaires de missions et au processus de résolution utilisé.

## 5.3 Modèle de base

On a vu précédemment que Desaulniers *et al.* (1996) et Rancourt et Savard (2001) ont proposé des formulations pour le problème de fabrication d'horaires de missions. Or, on peut dire que ce sont des formulations très détaillées (surtout pour Desaulniers *et al.*, 1996). Afin de simplifier l'exposé, on a préféré présenter un modèle épuré (inspiré d'un modèle simplifié proposé par Rancourt et Savard, 2001) contenant uniquement les éléments essentiels du problème de fabrication d'horaires de missions. Les formulations de Desaulniers et *al.* (1996) et Rancourt et Savard (2001) peuvent donc être vues comme des dérivés de ce modèle.

L'ensemble des commodités dans le modèle est noté $K$ et est composé de l'ensemble

$K^A$, l'ensemble des membres d'équipage, l'ensemble $K^L$, l'ensemble des lignes d'affectation (ou aéronefs génériques). L'ensemble des types d'entraînement possible est noté $W$ et pour chaque type d'entraînement $w \in W$ est donné le minimum d'heures, noté $min_{kw}$ que le membre d'équipage $k \in K^A$ doit consacrer à $w$. L'ensemble des missions est noté $M$. À chaque mission $m \in M$ est associé $S^m$ l'ensemble des étapes de la mission. Un ensemble $V$ de tâches est généré à partir de ces étapes de missions et des séquences d'entraînement possible, une tâche étant définie comme un vol ou une séquence de vols qui débute et se termine à la même base, qui possède un moment de départ précis (et non une fenêtre de départ), et qui doit être effectuée par le même équipage avec le même aéronef. À l'intérieur d'une même tâche, il peut se dérouler différentes étapes de missions. Il est important que l'ensemble de tâches $V$ soit assez riche (notamment en opportunités d'entraînement) pour que le solveur puisse fournir des solutions de qualité. À chacune des tâches $v \in V$ sont associés $d_v^{sm}$, une constante qui prend la valeur de 1 si la tâche $v$ couvre l'étape $s \in S^m$ de la mission $m \in M$ et 0 autrement, ainsi que les ensembles $P_v$ contenant l'ensemble des positions devant être comblées par des membres d'équipages pour la tâche $v \in V$. Il y a typiquement deux ou trois positions à combler sur chacune des tâches : deux de pilotes (dans tous les cas), une d'ingénieur de vol (parfois). La constante $g_{wk}^v$ donne finalement le nombre d'heures qui sont créditées pour le membre d'équipage $k$ sur l'entraînement $w$ lorsque celui-ci participe à la tâche $v$.

Une solution du problème de fabrication d'horaires de missions peut être vue comme une collection de chemins : pour ce faire, on associe à l'horaire des aéronefs génériques et à l'horaire des membres d'équipage une séquence d'arcs entre un noeud origine (début de l'horizon de planification) jusqu'à un noeud de destination (fin de l'horizon de planification). Cette séquence d'arcs représente les transitions entre les divers états (repos, exécution d'une tâche, etc.) dans lesquels se trouvera la commodité concernée au cours de l'horizon de planification.

En énumérant tous les états $i \in N^k$ dans lesquels chaque commodité $k$ peut se trouver à un moment donné de l'horizon et si on identifie toutes les transitions $(i, j) \in A^k$ qui sont possibles entre les états $i \in N^k$ et $j \in N^k$, on obtient un graphe orienté

$G(N^k; A^k)$ qui capture une partie de FHM. L'ensemble des chemins légaux dans le graphe $G(N^k; A^k)$ pour la commodité $k \in K$ est noté $\Omega^k$.

On associe ensuite aux chemins légaux des graphes d'équipages une constante $a_p^{kqv}$, valant 1 si le chemin $p$ de commodité $k \in K^A$ couvre la position $q \in P_v$ de la tâche $v \in V$ et 0 sinon.

Pour les chemins légaux des commodités $k \in K^L$, on associe une constante $l_p^{kv}$ valant 1 si le chemin $p$ de la commodité $k \in K^L$ couvre la tâche $v \in V$.

On associe également aux chemins $p \in \Omega^k$ un coût $c_p^k$ reflétant sa qualité. On pourrait par exemple évaluer les chemins des commodités $k \in K^A$ selon une fonction de satisfaction dérivée à partir des préférences fournies par le membre d'équipage $k$. De même, on pourrait juger de la qualité des chemins des commodités $k \in K^L$ selon le nombre d'heures volées. Cependant, comme il existe plusieurs façons de juger de la qualité d'horaires, le squelette de modèle présenté ici laisse à l'utilisateur le soin de définir ses propres fonctions de coûts.

On associe à chacune des missions $m \in M$ une pénalité de rejet $c_m$ et une variable $Z_m$ prenant la valeur 1 lorsque la mission $m$ est rejetée à l'optimalité (et 0 autrement). Ces coûts servent à modéliser les priorités de mission.

En associant à chaque chemin la variable binaire $\theta_p^k$, valant 1 si le chemin $p \in \Omega^k$ est

retenu à l'optimalité pour la commodité $k \in K$, on peut formuler le modèle suivant :

$$\min \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k \theta_p^k + \sum_{m \in M} c_m Z_m \tag{5.1}$$

$$\sum_{k \in K^A} \sum_{p \in \Omega^k} a_p^{kqv} \theta_p^k - \sum_{k \in K^L} \sum_{p \in \Omega_k} l_p^{kv} \theta_p^k = 0, \quad v \in V, \quad q \in P_v \tag{5.2}$$

$$\sum_{v \in V} d_v^{sm} \sum_{k \in K^L} \sum_{p \in \Omega^k} l_p^{kv} \theta_p^k + Z_m = 1, \quad m \in M, \quad s \in S_m \tag{5.3}$$

$$\sum_{p \in \Omega^k} g_{wk}^v \theta_p^k \geq min_{wk}, \quad k \in K^A, \quad w \in W \tag{5.4}$$

$$\sum_{p \in \Omega^k} \theta_p^k = 1, \quad k \in K \tag{5.5}$$

$$\theta_p^k \in \{0,1\}, \quad k \in K, \quad p \in \Omega^k \tag{5.6}$$

$$Z_m \in \{0,1\}, \quad m \in M. \tag{5.7}$$

L'objectif (5.1) consiste à minimiser le coût relié aux horaires du personnel navigant et aux lignes d'affectation ainsi que le coût relié au rejet de missions.

L'ensemble (5.2) permet l'assignation du personnel navigant nécessaire à l'exécution d'une tâche donnée lorsque celle-ci est planifiée tandis que l'ensemble (5.3) force la planification de toutes les étapes d'une mission $m \in M$ lorsque celle-ci est acceptée. Les contraintes (5.4) permettent de faire respecter les seuils minimaux sur les différents types d'entraînement afin que le personnel puisse maintenir ses compétences. La contrainte (5.5) indique qu'il faut générer un chemin pour toutes les commodités. Les contraintes (5.6)-(5.7) restreignent les variables $\theta_p^k$ et $Z_m$ à prendre des valeurs binaires.

Comme la cardinalité des ensembles de chemins légaux pour les commodités $k \in K$ est très grande, on fonctionne plutôt avec des ensembles restreints et on génère les chemins au besoin à l'aide d'un problème de plus court chemin avec contraintes de ressources sur les graphes $G(N^k, A^k)$.

Cette formulation saisit bien la problématique de base du problème $FHM$. De nombreuses contraintes supplémentaires concernant les règles de travail ou les préférences du personnel peuvent également être intégrées efficacement dans cette formulation, de façon implicite dans la définition des chemins, ou de façon explicite à même le problème maître.

Ce type de formulation est malheureusement difficile à résoudre pour des exemplaires de taille réaliste en raison du grand nombre de contraintes dans le problème maître. Les contraintes (5.2), permettant d'assurer que les horaires du personnel navigant couvrent toutes les positions des tâches retenues dans les lignes d'affectation, sont en grande partie responsables de la lourdeur du processus.

En plus ce modèle offre peu de flexibilité pour modéliser un aspect non négligeable du problème $FHM$, soit la compatibilité des membres d'équipages. Pour ces deux raisons, on a décidé de développer une nouvelle modélisation pour $FHM$, basée sur le problème de stable de poids maximum, qui intègre plus naturellement les contraintes de compatibilité du personnel et qui est plus facile à résoudre.

## 5.4 Une formulation comme un problème de stable

### 5.4.1 Relaxation des règles de travail

En faisant l'hypothèse que les règles de travail des horaires ne sont pas trop contraignantes, on peut concevoir un modèle construit autour des équipages légaux plutôt qu'à partir du personnel, afin de faire respecter *a priori* les contraintes de synchronisation. Mais, en plus de se débarrasser des contraintes de type (5.2), on obtient un programme qui ne se résout plus obligatoirement par génération de colonnes.

Soit $E$ l'ensemble des équipages légaux, $e \in E$. Comme la taille des équipages légaux est limitée à 2 ou 3, il est possible de considérer $E$ dans sa forme explicite. Soit $x_v^e$ la

variable binaire valant 1 si la tâche $v$ est effectuée avec l'équipage $e$ et 0 autrement. Soit $c_v^e$, un coût composite pouvant intégrer à la fois le coût de satisfaction attribué à l'exécution de $v$ avec $e$ dans le problème de fabrication d'horaires d'équipage et le mérite attribué à l'exécution de $v$ dans le problème de fabrication des lignes d'affectation. Soit $\mathcal{C}$ l'ensemble des cliques d'incompatibilité temporelle du graphe d'incompatibilité temporelle induit par les tâches $V$. Comme le graphe d'incompatibilité temporelle est un graphe d'intervalles, on peut énumérer toutes ses cliques maximales en temps polynomial.

Avec cette notation, on peut écrire :

$$\min \sum_{e\in E}\sum_{v\in V} c_v^e x_v^e + \sum_{m\in M} c_m Z_m \tag{5.8}$$

$$\sum_{e\in E}\sum_{v\in V} d_v^{sm} x_v^e + Z_m = 1, \quad m \in M, \quad s \in S^m \tag{5.9}$$

$$\sum_{e\in E}\sum_{v\in C} x_v^e \leq |K^L|, \quad C \in \mathcal{C} \tag{5.10}$$

$$\sum_{e\in E:k\in e}\sum_{v\in C} x_v^e \leq 1, \quad C \in \mathcal{C}, \quad k \in K^A \tag{5.11}$$

$$\sum_{e\in E:k\in e}\sum_{v\in V} g_{kw}^v x_v^e \geq min_{kw}, \quad w \in W, \quad k \in K^A \tag{5.12}$$

$$x_v^e, Z_m \in \{0,1\}, \quad m \in M. \tag{5.13}$$

L'objectif consiste à maximiser la satisfaction du personnel tout en minimisant les pénalités reliées aux missions rejetées. Les contraintes (5.9) permettent d'appliquer la pénalité reliée au rejet d'une mission $m$. Les contraintes de cliques (5.10) permettent de faire respecter la contrainte sur le nombre de lignes d'affectation. Les contraintes de cliques (5.11) servent à empêcher qu'un pilote soit affecté à deux tâches différentes en même temps. Les contraintes (5.12) permettent finalement de faire respecter les seuils minimaux d'entraînement pour tous les membres d'équipages.

On a donc remplacé une formulation de génération de colonnes par un simple programme linéaire en nombres entiers d'assez grande taille. En effet, puisqu'il y a de

l'ordre de $|K^A|^3$ équipages et que dans le pire des cas, tous ces équipages peuvent accomplir chacune des tâches, alors il y a $\mathcal{O}(|K^A|^3\ |V|)$ variables. De même pour les contraintes, on peut dire qu'elles sont de l'ordre du nombre de contraintes de cliques sur les pilotes soit $|\mathcal{C}||K^A|$. Et dans le pire cas $|\mathcal{C}| = |V|$ car le graphe d'incompatibilité temporelle des tâches est un graphe d'intervalles (voir chapitre 2).

Même si ce modèle est susceptible de contenir un peu plus de contraintes que les modèles de génération de colonnes proposés précédemment, ce modèle est souvent plus facile à résoudre directement.

Le modèle (5.8)-(5.13) est également peu sensible à la taille de l'ensemble des tâches $V$. Là où la formulation (5.1)-(5.7) prévoyait au moins deux contraintes pour chaque élément de $v \in V$ (deux par position-tâche), (5.8)-(5.13) ne prévoit que des variables (une par équipage-tâche). Et c'est un fait reconnu que la performance des solveurs de programmation linéaire est plus influencé par le nombre de contraintes que par le nombre de variables. Des stratégies de calcul de coût de réduit efficace peuvent en effet être mises en place afin d'amoindrir les inconvénients associés à un grand nombre de variables.

Avec ce modèle, on gagne également de la flexibilité afin de modéliser des contraintes d'équipages. Ce qui était jadis difficile à modéliser avec les modèles de génération de colonnes de Rancourt et Savard (2001) et Desaulniers *et al.* (1996), comme les compatibilités et les incompatibilités entre membres de personnels, deviennent triviales avec le nouveau modèle : on n'a qu'à ne pas mettre dans le modèle les variables $x_v^e$ associées aux équipages $e$ qui sont incompatibles.

Ce modèle possède malheureusement un bien grand défaut : il n'y a absolument aucune contrainte empêchant de générer des horaires illégaux en ce qui concerne les règles de travail. On est donc susceptible de générer des solutions violant les contraintes sur les heures volées au cours du mois par un membre de personnel, sur les repos minimum entre deux journées de travail ou sur la durée maximum d'une journée de travail. Ces contraintes, qui étaient locales et faciles à formuler au moyen de ressources avec le précédent modèle de base, deviennent globales avec ce modèle

et difficiles à modéliser.

On pourrait certes ajouter ces contraintes de sac-à-dos en vue de faire respecter la règle du maximum d'heures volées :

$$\sum_{e \in E} \sum_{v \in V} f_{ev}^{k} x_{v}^{e} \leq F_k, k \in K^A$$

où $f_{ev}^{k}$ est le nombre d'heures volées par le pilote $k \in K^A$ lors de l'éxecution de la tâche $v \in V$ avec l'équipage $e \in E$ et $F_k$ est le maximum d'heures volées pour $k$.

Ce ne sont malheureusement pas toutes les règles de travail qui se modélisent aussi aisément. On ignore encore comment modéliser de façon concise les contraintes sur le repos minimum ou la règle sur la longueur d'une journée de travail.

Au groupe tactique de l'armée canadienne de Val Cartier, il semble cependant que les pilotes soient préaffectés à des quarts de travail respectant *a priori* les contraintes de repos minimum et de longueur des journées de travail. Les seules contraintes de règles de travail que l'on doit faire respecter explicitement sont donc des contraintes sur des minimum et des maximum d'heures volées au cours d'une journée et au cours du mois, que l'on peut modéliser facilement avec des contraintes de type sac-à-dos.

### 5.4.2 Stratégie de résolution

Pour résoudre ce programme en nombres entiers, on a utilisé un algorithme d'évaluation et de séparation progressive classique, où à chaque noeud de l'arbre on utilise la relaxation linéaire du modèle pour obtenir une borne valide sur la solution optimale entière. Cette borne peut nous aider de plusieurs façons dans l'algorithme, notamment pour fermer les régions non prometteuses de l'arbre ou pour prendre une décision de branchement. Dans notre algorithme, on a utilisé un branchement très simple sur les variables $x_t^e$ (*cfix*, consistant à fixer certaines colonnes à 1 de façon permanente) et on a exploré l'arbre selon le critère *depth until leaf* afin d'obtenir rapidement de bonnes

solutions réalisables.

On doit noter qu'on peut mettre en place des techniques efficaces pour l'identification de la variable entrante (problème de *pricing*) afin d'accélérer le calcul de la relaxation linéaire du modèle. Par exemple, si le coût associé au fait de placer l'ingénieur de vol $k$ sur la tâche $v$ est indépendant du pilote et du co-pilote choisi (i.e si $c_v^e = c_v^{(k_1,k_2,k_3)} = c_v^{(k_1,k_2)} + c_v^{k_3}$) et que la position d'ingénieur de vol est facultative pour $v$, on peut identifier pour chaque tâche $v \in V$ la variable $x_v^e$ de coût réduit le plus négatif en résolvant le problème :

$$f(v) = \min_{e \in E^2} \bar{c}_v^e + \min\{0, \min_{k \in K^I} \bar{c}_v^k\}$$

où $E^2$ est l'ensemble des équipages ne comptant qu'un pilote et un co-pilote, $\bar{c}_v^e$ est le coût réduit de la variable $x_v^{(k_1,k_2)}$, $\bar{c}_v^k$ est le coût réduit associé à la décision de placer l'ingénieur de vol $k$ sur la tâche $v$ et $K^I$ est l'ensemble des ingénieurs de vol. On peut résoudre cela comme un problème de plus court chemin dans un réseau acyclique comptant $O(|K^A|^2)$ arcs. Comme on doit résoudre ce problème pour chaque tâche $v \in V$, on peut en procédant ainsi abaisser la complexité du problème de *pricing* à $O(|K^A|^2|V|)$.

### 5.4.3 Résultats numériques

Afin de valider le modèle présenté dans ce chapitre, on a testé cette approche sur quatre scénarios inspirés de la situation actuelle de Val Cartier.

Dans le premier scénario, on considère un horizon de cinq jours, deux escadrons de 16 pilotes et 8 ingénieurs de vols, 8 lignes d'affectation, 7 missions et 70 souhaits d'entraînement à satisfaire. Pour couvrir ces missions et ces souhaits d'entraînement, on a fourni à l'optimiseur un ensemble de 743 tâches. Dans ce scénario, on recherche seulement une solution réalisable au problème.

Dans le deuxième scénario, on considère les mêmes données que dans le premier,

sauf que l'on doit maintenant trouver une solution réalisable qui maximise les heures volées.

Dans le troisième scénario, on résoud le problème de satisfaction du premier scénario sauf que l'horizon est allongé à dix jours, que le nombre de souhaits est de 138 et que l'optimiseur doit choisir les tâches parmi un ensemble en comptant 1184.

Dans le quatrième scénario, on considère finalement les même données qu'au troisième scénario sauf que l'on doit maintenant maximiser les heures volées.

Pour chaque scénario, on a utilisé quatre stratégies de résolution. Avec la stratégie 1, on n'utilise aucune stratégie de *pricing* efficace lors de la résolution linéaire et on considère explicitement toutes les variables $x_v^e$ alors qu'avec la stratégie 2, on fait appel à la méthode présentée à la section précédente pour trouver la colonne entrante. La stratégie 3 et la stratégie 4 ressemblent aux stratégies 1 et 2, sauf qu'au lieu de considérer toutes les variables $x_v^e$, on ne considère qu'un sous-ensemble de $E$, l'ensemble des équipages permis.

L'algorithme d'évaluation et de séparation progressive a été implanté à l'aide des librairies NetGen6.0, qui permettent de créer des applications pour Gencol4.3, un solveur de génération de colonnes et les tests ont été effectués sur une machine dotée d'un microprocesseur de 1.8 Ghz et de 1028 Meg de mémoire vive.

Le tableau 5.1 présente les résultats obtenus pour les quatre scénarios avec les quatre stratégies de résolution.

En observant le tableau, la première chose qu'on remarque est la faible amplitude des sauts d'intégrité. Dans à peu près tous les cas, on a obtenu des sauts d'intégrité nuls et on a été en mesure d'obtenir assez rapidement la solution optimale avec un branchement assez primitif. Dans le cas où on considère *FHM* comme un problème de faisabilité, il semble à première vue plus efficace d'avoir recours à la génération dynamique de colonnes (stratégies 2 et 4). On remarque finalement que pour les deux scénarios considérés, il n'a pas été trop désavantageux de ne pas considérer toutes les

Tableau 5.1 – Résultats

| Scén. | Stratégie | nbVariables | nbLignes | temps (s) | nbNoeuds | LP | $\bar{z}$ | saut (%) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 450450 | 3517 | 85 | 19 | 0 | 0 | 0 |
| | 2 | 450450 | 3517 | 3 | 8 | 0 | 0 | 0 |
| | 3 | 44792 | 3517 | 5 | 11 | 0 | 0 | 0 |
| | 4 | 44792 | 3517 | 2 | 16 | 0 | 0 | 0 |
| 2 | 1 | 450450 | 3517 | 249 | 99 | 274.5 | 274.5 | 0 |
| | 2 | 450450 | 3517 | 416 | 95 | 274.5 | 274.5 | 0 |
| | 3 | 44792 | 3517 | 12 | 44 | 274.5 | 274.5 | 0 |
| | 4 | 44792 | 3517 | 157 | 74 | 274.5 | 273.5 | 0.364 |
| 3 | 1 | 718545 | 5649 | 119 | 16 | 0 | 0 | 0 |
| | 2 | 718545 | 5649 | 10 | 13 | 0 | 0 | 0 |
| | 3 | 71423 | 7649 | 9 | 19 | 0 | 0 | 0 |
| | 4 | 71423 | 7649 | 4 | 9 | 0 | 0 | 0 |
| 4 | 1 | 718545 | 5649 | 2505 | 140 | 436.5 | 436.5 | 0 |
| | 2 | 718545 | 5649 | 765 | 157 | 436.5 | 436.5 | 0 |
| | 3 | 71423 | 5649 | 19 | 46 | 436.5 | 436.5 | 0 |
| | 4 | 71423 | 5649 | 275 | 123 | 436.5 | 434.2 | 0.527 |

variables $x_v^e$.

Les résultats obtenus avec cette approche se comparent avantageusement à ceux que l'on a obtenu avec l'approche de Rancourt et Savard (2001) consistant à résoudre la formulation construite autour des équipages légaux avec un algorithme utilisant de façon heuristique le principe de décomposition de Benders. Pour le scénario 1, leur approche a convergé en un peu moins d'une minute vers une solution alors que pour le scénario 3, leur approche a convergé en environ 5 minutes. Les solutions générées avec l'approche de Rancourt et Savard (2001) pour ces deux scénarios sont cependant de moins bonne qualité, dans la mesure où celles-ci, contrairement à nos solutions, ne satisfaisaient pas toutes les contraintes de souhaits.

## 5.5 Conclusion

Une formulation basée sur le problème du stable, semblable à celles déjà proposées dans la littérature pour les problèmes d'ordonnancement sur machines parallèles, a ainsi été développée pour le problème de fabrication d'horaires de mission (FHM), dans lequel on doit simultanément résoudre le problème de fabrication des lignes d'affectation et le problème de fabrication d'horaires de personnel.

Les expériences préliminaires conduites sur des exemplaires du problème de fabrication d'horaires de missions indiquent que le modèle construit autour des tâches-équipages légales semble adéquat. En plus de pouvoir tenir compte de la plupart des contraintes de base du problème (disponibilité des lignes d'affectation, disponibilité du personnel, règles de travail), ce modèle de programmation en nombres entiers semble assez serré, dans la mesure où on obtient avec celui-ci des sauts d'intégrité exploitables à l'intérieur d'un algorithme de branchement. Les résultats numériques indiquent d'ailleurs que la formulation construite autour des tâches-équipages légales semble plus facile à résoudre que la formulation basée sur les horaires d'équipages légaux, si on se fie aux résultats obtenus avec l'approche de Rancourt et Savard (2001). En plus d'être plus facile à résoudre, cette formulation permet de modéliser aisément les compatibilités entre les membres d'équipages. Cela ouvre la voie à un système d'aide à la décision pour le problème tactique qui pourrait aider les planificateurs à produire des solutions intégrant toute une gamme de préférences pour les membres d'équipages, comme des préférences de congés, des préférences sur les tâches à exécuter ou encore des préférences à travailler avec d'autres.

# Conclusion

Dans cette thèse, on a abordé divers problèmes d'ordonnancement sur une machine. Voici à notre avis nos principales contributions :

- Nous avons montré comment de nombreux problèmes d'ordonnancement sur une machine pouvaient être modélisés comme des problèmes de stable de poids maximum. Ce fait était connu pour les problèmes d'ordonnancement sur une machine avec temps de réglages indépendants de la séquence (Waterer *et al.*, 2002). Cependant, pour les problèmes d'ordonnancement sur une machine avec temps de réglage dépendants de la séquence, et notamment pour le célèbre cas particulier $1|s_{ij}|C_{max}$ (soit le problème de commis-voyageur classique), cette relation semble avoir été négligée. Cela a entre autres permis le développement d'une nouvelle famille de coupes pour le problème de commis-voyageur dépendant du temps basée sur les coupes de cliques du problème de stable.
- Nous avons montré comment obtenir de bonnes bornes inférieures pour les formulations de type temps indexé du problème d'ordonnancement sur une machine (avec temps de réglage indépendants de la séquence) en ayant recours au principe de décomposition de Dantzig-Wolfe et à un principe de décomposition temporelle. En intégrant cette borne à l'intérieur d'un algorithme d'énumération implicite (*CGTWT*), on a été en mesure de résoudre pour la première fois plusieurs exemplaires de 100 pièces du problème $1|\cdot|\sum w_j T_j$ retrouvés sur *OR-LIBRARY*.
- Nous avons également montré comment on pouvait renforcer les formulations de commis-voyageur dépendant du temps en utilisant des techniques de coupes de programmation en nombres entiers et des reformulations de Dantzig-Wolfe. En intégrant diverses coupes à des reformulations de génération de colonnes du problème de commis-voyageur dépendant du temps, on a été en mesure d'obtenir de très bonnes bornes pour le problème $1|s_{ij}|\sum C_j$ et $1|s_{ij}|\sum T_j$. Ces bornes

fortes ont permis le développement d'un algorithme exact (*BranchAndCut*) pour ces deux problèmes qui semble compétitif avec les approches existantes. Pour le problème $1|s_{ij}|\sum T_j$, de nombreux exemplaires de la libraire de Rubin et Ragatz (1995) ont même pu être résolus pour la première fois.

- Nous avons finalement proposé une nouvelle formulation pour le problème de fabrication d'horaires de missions retrouvé au sein des Forces Armées Canadiennes. En considérant ce problème comme un problème de couplage (entre le personnel et les véhicules) plutôt que comme un problème de routage (comme avec les formulations de Desaulniers *et al.*, 1996 et Rancourt, 2001), on a obtenu une formulation de programmation en nombres entiers qui a semblé, lors des expériences numériques, plus facile à résoudre. Cette formulation permet en plus d'intégrer des aspects du problème qui peuvent être difficilement pris en compte avec les autres formulations (comme les contraintes de compatibilité entre membres de personnel).

Certaines de ces contributions pourraient être mises à profit dans d'autres contextes.

- L'approche de décomposition temporelle vue au chapitre 3 a été testée sur le problème $1|\cdot|\sum w_jT_j$. Dans ce cas, on a vu qu'elle permettait de faciliter l'obtention de bornes de qualité. Il serait intéressant de tester cette approche sur d'autres problèmes d'ordonnancement sur une machine, comme $1|r_j|\sum w_jC_j$ ou même sur des problèmes d'ordonnancement sur machines parallèles.
- On a vu que la formulation $DW(T3')$ peut être utilisée en vue de trouver des bornes pour le problème $1|s_{ij}|\sum w_jT_j$. Il serait intéressant de développer un algorithme d'évaluation et de séparation progressive pour ce problème. En effet, aucune approche exacte ne semble encore avoir été proposée pour ce problème.
- On pense qu'on pourrait utiliser l'approche de stable ayant mené au développement des coupes de cliques pour le problème de commis-voyageur dépendant du temps dans d'autres problèmes, comme le problème de commis-voyageur avec fenêtres de temps (Ascheuer *et al.*, 2000). En effet, si on utilise une formulation basée sur la discrétisation des fenêtres de temps, on peut définir des relations d'incompatibilité entre les variables semblables à celles vues au chapitre 4.

Finalement, de nombreuses pistes de recherche restent encore à explorer.

- Huit exemplaires du problème $1|\cdot|w_jT_j$ tirés de OR-LIBRARY n'ont pu être résolus avec l'algorithme $CGTWT$. On pense qu'on pourrait améliorer cet algorithme en utilisant une règle de branchement faisant intervenir plusieurs variables et en raffinant les règles de dominance.
- Pour le problème $1|s_{ij}|\sum w_jT_j$, on a quelquefois obtenu de très bons résultats en résolvant la formulation $T3$ avec *cplex*9.0. À notre avis, les bonnes performances du solveur de programmation en nombres entiers dans ces cas sont dûes à l'utilisation du *strong branching*, qui consiste à évaluer à un même noeud plusieurs décisions de branchement. On pense qu'en implantant cette règle de branchement dans l'algorithme *BranchAndCut*, on pourrait résoudre certains problèmes non résolus dans cette thèse.
- On a vu comment on pouvait exploiter les coupes du problème de commis-voyageur symétrique dans le cadre du problème de commis-voyageur dépendant du temps. On pense qu'on pourrait obtenir encore de meilleures bornes en ayant recours également à des familles d'inégalités valides du problème de commis-voyageur asymétrique.

# Bibliographie

ABDUL-RAZAQ, T., POTTS, C. ET VAN WASSENHOVE, L. (1990). A Survey of Algorithms for the Single Machine Total Weighted Tardiness Scheduling Problem. *Discrete Applied Mathematics*, 26, 235-253.

ANBIL, R., FORREST, J.J. ET PULLEYBLANK, W.R. (1998). Column Generation and the Airline Crew Pairing Problem. *Documenta Mathematica*, Extra Volume ICM 1998, 3, 677-686.

APPLEGATE, D., BIXBY, R., CHVÁTAL, V. ET COOK, W. (1998). On The Solution Of Traveling Salesman Problems. *Documenta Mathematica*, Extra Volume ICM 1998, 3, 645-646.

ASCHEUER, N., FISCHETTI, M. ET GROETSCHEL, M. (2000). A Polyhedral Study of the Asymmetric Traveling Salesman Problem with Time Windows. *Networks*, 32, 69-79.

BALAS, E. (1985). On the facial structure of scheduling polyhedra. *Mathematical Programming Study*, 24, 179-219.

BALAS, E. (1989). The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. *SIAM Journal On Discrete Mathematics*, 2, 425-451.

BABU, P., PERIDY, L. ET PINSON, E. (2004). A Branch and Bound Algorithm to Minimize Total Weighted Tardiness on a Single Processor. *Annals of Operations Research*, 129, 33-46.

BELOUADAH, H., POSNER, M.E. ET POTTS, C.N. (1992). Scheduling with re-

lease dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36, 213-231.

BIANCO, L., MINGOZZI, A. ET RICCIARDELLI, S. (1993). The Traveling Salesman Problem with Cumulative Costs. *Networks*, 23, 81-91.

CONGRAM, R.K., POTTS, C.N. ET VAN DE VELDE, S.L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14, 52-67.

CONWAY, R.W., MAXWELL, W.L. ET MILLER, L.W. (1967). Theory of Scheduling. *Addison-Wesley.*

CRAUWELS, H.A.J., POTTS, C.N. ET VAN WASSENHOVE, L.N. (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10, 341-350.

DANTZIG, G.B. ET WOLFE, P. (1960). Decomposition principle for linear programs. *Operations research*, 8, 101-111.

DANTZIG, G., FULKERSON, D. ET JOHNSON, S. (1954). Solution of a Large Scale Traveling Salesman Problem. *Operations Research*, 2, 393-410.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOLOMON, M.M. ET SOUMIS, F. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In Crainic, T. and Laporte, G., editors, Fleet Management and Logistics, chapter 3, 57-93. *Kluwer Academic Publisher*, Boston, Dordrecht, London.

DESAULNIERS, G., DESROSIERS, J., LINGAYA, N., RANCOURT, E. ET SAVARD, G. (1996). Scheduling-Routing for Air Force Resource Management : Reports on Task I, II and III. Préparé pour DREV, Departement de la défense nationale, Valcartier, Québec.

DESROCHERS, M. ET LAPORTE, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10, 27-36.

DYER, M.E. ET WOLSEY, L.A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26, 255-270.

ELMAGHRABY, S.E. (1968). The One-Machine Sequencing Problem with Delay Costs. *Journal of Industrial Engineering*, 19, 105-108.

EMMONS, H. (1969). One-Machine Sequencing to Minimize Certain Functions of Jobs Tardiness. *Operations Research*, 17, 701-715.

ERLEBACH, T. ET SPIEKSMA, F.C.R. (2003). Interval selection : Applications, algorithms, and lower bounds. *Journal of Algorithms*, 46, 27-53.

FISCHETTI, M., LAPORTE, G. ET MARTELLO, S. (1993). The Delivery Man Problem and Cumulative Matroids. *Operations Research*, 31, 803-834.

FOX, K.R. (1973). Production Scheduling on Parallel Lines with Dependencies. Thèse de doctorat, The Johns Hopkins University, Baltimore.

FOX, K., GAVISH, B. ET GRAVES, S.C. (1980). An n-Constraint Formulation of the (Time-Dependant) Traveling Salesman Problem. *Operations Research*, 28, 1019-1021.

GAGNÉ, C., GRAVEL, M. ET PRICE, W.L. (2001). Scheduling a single machine with sequence dependent time using an ant colony optimization. Document de travail 2001-003, Faculté des sciences de l'administration, Université Laval.

GAMACHE, M., SOUMIS, F., MARQUIS, G. ET DESROSIERS, J. (1994). A column generation approach for large scale aircrew rostering problem. *Operations Research*, 47, 247-262.

GAREY, M.R. ET JOHNSON, D.S. (1979). Computers and Intractability : A Guide to the theory of NP-Completeness. *W.H. Freeman and Company*, New York, 1979.

GAVRIL, F. (1972). Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1, 180-187.

GELINAS, S. ET SOUMIS, F. (1996). A Dynamic Programming Algorithm for Single Machine Scheduling with Ready Times and Deadlines to Minimize Total Weighted Completion Time. *MIS Collection in the Annals of Operations Research*, 69, 135-156.

GEOFFRION, A. (1974). Lagrangean Relaxation for Integer Programming. *Mathematical Programming Study* 2, 82-114.

GOEMANS, M.X. (1997). Improved approximation algorithms for scheduling with release dates. *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 591-598.

GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.K. ET RINNOOY KAN, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5, 287-326.

GUIGUE, A. (2000). Fabrication d'horaires dans les forces armées canadiennes : une approche par génération de missions et réduction de réseaux. Mémoire de maîtrise (École Polytechnique de Montréal).

HADLEY, G. (1964). Nonlinear and Dynamic Programming. *Addison-Wesley Publishing Co.*, Inc.

HALL, L.A., SCHULTZ, A.S., SHMOYS, D.B. ET WEIN, J. (1997). Scheduling to minimize average completion time : off line and on line approximation algorithms. *Mathematics of operations research*, 22, 513-544.

HOFFMAN, K. ET PADBERG, M.W.(1993). Solving Airline Crew Scheduling Pro-

blems by Branch and Cut. *Management Science*, 39, 657-682.

HOUCK, D., PICARD, J., QUEYRANNE, M. ET VEGAMUNTI, R. (1980). The Travelling salesman as a Contrained Shortest Path Problem : Theory and Computational Experiment. *Opsearch*, 17, 93-109.

IOACHIM, L., DESROSIERS, J., DUMAS, Y., SOLOMON, M.M. ET VILLENEUVE, D. (1995). A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29, 63-79.

IRNICH, S. ET VILLENEUVE, D. (2003). The Shortest Path Problem with Resource constraints and $k$-cycle Elimination for $k \geq 3$. Les Cahiers du Gerad, G-2003-55.

JACKSON, J.R. (1955). Scheduling a Production Line to Minimize Maximum Tardiness. Research Report 43, Management Science Research Project, Los Angeles, CA : University of California.

LASDON, L.S. (1970). Optimization theory for Large Systems. *MacMillan*, New York.

LAWLER, E.L. (1964). On scheduling with deferral costs, *Management Science*, 11, 280-288.

LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G. ET SHMOYS, D.B.(1985). The traveling salesman problem. *New York : Wiley.*

LAWLER, E.L. (1977). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1, 331-342.

LAWLER, E.L. (1979). Efficient implementation of dynamic programming algorithms for sequencing problems. Rept. BW 106, Mathematisch Centrum, Amsterdam.

LUCENA, A. (1990). Time-Dependant Traveling Salesman Problem - The Deliveryman Case. *Networks*, 20, 753-763.

McNAUGHTON, R. (1959). Scheduling with Deadlines and Loss Functions. *Management Science*, 6, 1-12.

MILLER, C., TUCKER, A. ET ZEMLIN, R. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7, 326-329.

MOORE, J.M. (1968). An n-job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15, 102-109.

NEMHAUSER, G. ET WOLSEY, L. (1988). Integer and combinatorial optimization. *Wiley Interscience series in discrete mathematics and optimization.*

NEMHAUSER, G.L. ET TROTTER, L.E. (1975). Vertex packing : Structural properties and algorithms. *Mathematical Programming* 232-248.

ORMAN, A.J. ET WILLIAMS, H.P. (2004). A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem. Rapport technique LSEOR 04.67, The London School of Economics and Political Science.

PADBERG, M.W. (1975). On the facial structure of set packing polyhedra. *Mathematical Programming*, 5, 199-215.

PHILLIPS, C., STEIN, C. ET WEIN, J. (1998). Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82, 199-223.

PICARD, J.C. ET QUEYRANNE, M. (1978). The Time-Dependant Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling. *Operations Research*, 26, 86-110.

POTTS, C.N. ET VAN WASSENHOVE, L. (1985). A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33, 363-377.

RACHAMADUGU, R.M.V. (1987). A note on the weighted tardiness problem. *Operations Research*, 35, 450-451.

RANCOURT, E. (1998). La planification des vols pour le groupe transport aérien des Forces Armées Canadiennes. Mémoire de maîtrise (École Polytechnique de Montréal).

RANCOURT, E. ET SAVARD, G. (2001). The mission scheduling problem at Tactical Aviation Forces. Rapport technique. Préparé pour DREV, Departement de la défense nationale, Valcartier, Québec.

RINNOOY KAN, A., LAGEWEG, B. ET LENSTRA, J. (1975). Minimizing Total Costs in One-Machine Scheduling. *Operations Research*, 23, 908-927.

RUBIN, P.A. ET RAGATZ, G.L. (1995). Scheduling in a sequence dependent setup environment with genetic search. *Computers and Operations Research*, 22, 85-99.

SCHRAGE, L. ET BAKER, K.R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26, 444-449.

SCHULTZ, A.S. (1996). Polytopes and Scheduling. Thèse de doctorat. Technical University of Berlin.

SHWIMER, J. (1972). On the N-job, one-machine, sequence-independant scheduling problem with tardiness penalties : A branch-and-bound solution. *Management Science*, 18, B301-313.

SOUSA, J.P. ET WOLSEY, L.A. (1992). A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54, 353-367.

SMITH, W.E. (1956). Various Optimizers for Single-Stage Production. *Naval Research Logistic Quarterly*, 3, 59-66.

SPIEKSMA, F.C.R. (1999). On the approximability of an interval scheduling problem. *Journal of scheduling*, 2, 215-227.

TAN, K.C., NARASIMHAN, R., RUBIN, P.A. ET RAGATZ, G.L. (2000). A comparison of four methods for minimizing total tardiness on a single processor with sequence

dependent setup times. *Omega*, 28, 313-326.

TAN, K.C. ET NARASIMHAN, R. (1997). Minimizing tardiness on a single processor with sequence dependent setup times : a simulated annealing approach. *Omega*, 25, 619-634.

VAN DEN AKKER, J.M., HURKENS, C.A.J. ET SAVELSBERGH, M.W.P. (1995). Time-indexed formulations for single-machine scheduling problems : Branch and cut. Memorandum COSOR, 95-24. Eindhoven University of Technology.

VAN DEN AKKER, J.M., VAN HOESEL, C.P.M. ET SAVELSBERGH, M.W.P (1999). A Polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85, 541-572.

VAN DEN AKKER, J.M., HURKENS, C.A.J. ET SAVELSBERGH, M.W.P. (2000). Time-indexed formulations for single-machine scheduling problems : Column generation. *INFORMS Journal on Computing*, 12, 111-124.

VANDERBECK, F. ET WOLSEY, L.A. (1996). An Exact Algorithm for IP Column Generation. *Operations Research Letters*, 10, 151-160.

VANDER WIEL, R.J. ET SAHINIDIS, N.V. (1996). An Exact Solution Approach for the Time-Dependent Traveling-Salesman Problem. *Naval Research Logistics*, 43, 797-820.

VILLENEUVE, D. (1999). Logiciel de génération de colonnes. Thèse de doctorat (École Polytechnique de Montréal).

WATERER, H., JOHNSON, E.L., NOBILI, P. ET SAVELSBERGH, M.W.P. (2002). The Relation of Time Indexed Formulations of Single Machine Scheduling Problems to the Node Packing Problem. *Mathematical Programming*, 93, 477-494.

WOLSEY, L.A. (1976). Facets and Strong Valid Inequalities for Integer Programs. *Operations Research*, 24, 367-372.

# ANNEXE A : Notation

Afin de faciliter la compréhension des nombreux modèles présentés la thèse, on retrouve dans cette annexe la notation utilisée pour définir ceux-ci, regroupée par chapitre.

## A.1 Chapitre 2

| | |
|---|---|
| $G(\mathcal{N}, \mathcal{A})$ | un graphe (orienté ou non). |
| $\mathcal{N}$ | ensemble de noeuds. |
| $\mathcal{A}$ | ensemble d'arêtes (ou d'arcs dans le cas orienté). |
| $x_j$ | variable binaire valant 1 si le noeud $j$ est dans le stable optimal et 0 autrement. |
| $C$ | ensemble de noeuds formant une clique. |
| $\mathcal{H}$ | ensemble de noeuds formant un trou. |
| $I$ | ensemble d'intervalles. |
| $i$ | un intervalle donné. |
| $c_i$ | poids associé à un intervalle. |
| $\mathcal{C}$ | ensemble de cliques. |
| $x_{ij}^t$ | variable binaire valant un si à la période $t$ on utilise l'arc $(i, j)$ et 0 autrement. |
| $c_{ij}^t$ | coût associé à la variable $x_{ij}^t$. |
| $n$ | nombre de noeuds. |

## A.2 Chapitre 3

| | |
|---|---|
| $H$ | ensemble de périodes définissant l'horizon de planification. |
| $T$ | nombre de périodes dans $H$. |
| $t$ | une période de durée unitaire. |
| $p_j$ | durée de la pièce $j$. |
| $n$ | nombre de pièces. |
| $x_{jt}$ | variable binaire valant 1 si on décide de traiter la pièce $j$ à partir de la période $t$ et 0 autrement. |
| $\pi_j$ | variable duale associée à la contrainte de partionnement de la pièce $j$. |
| $e_t$ | variable binaire valant 1 si à la période $t$ on attend et 0 autrement. |
| $\Omega$ | ensemble des chemins légaux dans le réseau. |
| $\theta_p$ | variable binaire valant 1 si on choisit le chemin $p$ dans la solution optimale et 0 autrement. |
| $V$ | ensemble de tâches. |
| $v$ | une tâche. |
| $a_v^i$ | proportion du traitement de la pièce $i$ effectuée au sein de la tâche $v$. |
| $l(v)$ | période de début de la tâche $v$. |
| $u(v)$ | période de fin de la tâche $v$. |
| $x_v$ | variable binaire valant 1 si on exécute la tâche $v$ et 0 autrement. |
| $c_v$ | coût associé à la tâche $v$. |
| $H_k$ | un sous-horizon de planification. |
| $V_k$ | ensemble des tâches dans le sous-horizon $k$. |
| $T_k$ | période de fin de l'horizon $k$. |
| $V_c$ | ensemble des tâches compliquantes. |
| $H_c$ | ensemble des périodes compliquantes. |
| $V(t)$ | ensemble des tâches de l'ensemble $V$ se déroulant à la période $t$. |
| $\Omega_k$ | ensemble des chemins légaux pour l'horizon $k$. |
| $\theta_p^k$ | variable binaire valant 1 si le chemin $p$ est choisi pour le sous-horizon $k$ et 0 autrement. |
| $B_j$ | ensemble des pièces devant être effectuées avant la pièce $j$. |
| $A_j$ | ensemble des pièces devant être effectuées après la pièce $j$. |
| $edd(h)$ | position de la pièce $h$ dans l'ordre *edd* (pour *earliest due date*). |
| $smith(h)$ | position de la pièce $h$ dans l'ordre de Smith. |
| $d_{max}$ | date de livraison maximum du problème. |
| $d_{min}$ | date de livraison minimum du problème. |
| $LB(v)$ | borne inférieure pour le problème lorsqu'on force la tâche $v$ à être planifiée. |
| $\lambda_t$ | distance du noeud source au noeud associé à la période $t$. |
| $\mu_t$ | distance du noeud puits au noeud associé à $t$. |

## A.3 Chapitre 4

| | |
|---|---|
| $s_{ij}$ | temps de réglage si $j$ est traitée après $i$. |
| $G(\mathcal{N}, \mathcal{A})$ | un graphe orienté. |
| $\mathcal{N}$ | ensemble des noeuds du graphe $G$. |
| $\mathcal{A}$ | ensemble des arcs du graphe $G$. |
| $t$ | une période dans le problème de commis-voyageur dépendant du temps. |
| $n$ | nombre de noeuds (ou de pièces). |
| $x_{ij}^t$ | variable valant 1 si l'arc $(i,j)$ est placé à la position $t$ de la tournée et 0 autrement. |
| $c_{ij}^t$ | coût associé à la variable $x_{ij}^t$. |
| $\Omega$ | ensemble des chemins dans le graphe multiparti. |
| $\theta_p$ | variable binaire valant 1 si on choisit le chemin $p$ dans la solution optimale et 0 autrement. |
| $x_{ij}^t$ | constante valant 1 si le chemin $p$ emprunte l'arc $(i,j)$ en position $t$. |
| $\pi_j$ | variable duale de la $j$ième contrainte de la formulation $DW(T3)$. |
| $C_t$ | variable donnant le temps de complétion de la pièce planifiée en position $t$. |
| $T_t$ | variable donnant le retard de la pièce planifiée en position $t$. |
| $r_j$ | date de départ pour la pièce $j$. |
| $d_j$ | date de livraison pour la pièce $j$. |
| $U_j$ | variable binaire valant 1 si on traite la pièce $j$ et 0 autrement. |
| $\bar{d}_t$ | date de livraison à respecter pour la position $t$. |
| $y_{ij}$ | variable binaire valant 1 si l'arc $(i,j)$ est dans la solution optimale du problème de commis-voyageur et 0 autrement. |
| $c_{ij}$ | coût associé à la variable $x_{ij}$. |
| $H$ | ensemble de noeuds constituant la poignée dans une contrainte de 2-couplages. |
| $\hat{E}$ | ensemble d'arêtes constituant les dents dans une contrainte de 2-couplages. |
| $C$ | une clique du graphe d'incompatibilité. |
| $\mathcal{H}$ | un trou du graphe d'incompatibilité. |
| $\Gamma$ | ensemble des tournées du problème de commis-voyageur symétrique. |
| $\lambda_p$ | multiplicateur de combinaison convexe associé à la solution $p \in \Gamma$. |
| $\hat{y}$ | solution fractionnaire pour le problème de commis-voyageur. |
| $y_{ij}^p$ | constante valant 1 si la tournée $p$ utilise l'arête $(i,j)$ et 0 autrement. |
| $\pi_{ij}$ | variable duale associée à la contrainte $(i,j)$ du modèle $SPP(\hat{y})$. |
| $\alpha$ | variable duale associée à la contrainte de convexité du modèle $SPP(\hat{y})$. |

| | |
|---|---|
| $f(i,j,t)$ | fonction donnant le coût associé au fait de planifier la pièce $j$ en position $t$ après la pièce $i$. |
| $f(i_1,i_2,\ldots,i_{k-1},i_k,t)$ | coût associé au fait d'effectuer la pièce $i_k$ en position $t$ après la séquence $i_1,i_2,\ldots,i_{k-2},i_{k-1}$. |
| $\tau$ | un moment dans l'horizon de planification. |
| $\tau_{max}$ | moment de fin de l'horizon de planification. |
| $f(j,t,\tau)$ | fonction donnant le coût associé au fait de planifier la pièce $j$ en position $t$ au moment $\tau$. |
| $f_{ij}^t$ | coût associé à la variable $x_{ij}^t$ dans le modèle $SP(T3')$. |

## A.4 Chapitre 5

| | |
|---|---|
| $K$ | ensemble des commodités du modèle. |
| $K^L$ | ensemble des lignes d'affectation. |
| $K^A$ | ensemble des membres d'équipage. |
| $W$ | ensemble des entraînements possibles. |
| $w$ | un entraînement donné. |
| $min_{kw}$ | nombre d'heures minimum que le membre d'équipage $k$ doit consacré à $w$. |
| $M$ | ensemble des missions. |
| $m$ | une mission donnée. |
| $S^m$ | ensemble des étapes de missions de la mission $m$. |
| $V$ | ensemble de tâches. |
| $v$ | une tâche donnée. |
| $d_v^{sm}$ | constante prenant la valeur de 1 si la tâche $v$ couvre l'étape $s \in S^m$ de la mission $m$ et 0 autrement. |
| $P_v$ | ensemble des positions à couvrir pour la tâche $v$. |
| $g_{wk}^v$ | nombre d'heures qui sont créditées pour le membre d'équipage $k$ sur l'entraînement $w$ lorsque celui-ci participe à la tâche $v$. |
| $G(N^k, A^k)$ | réseau pouvant générer les horaires légaux de la commodité $k$. |
| $\Omega^k$ | ensemble des chemins admissible du réseau $G(N^k, A^k)$. |
| $\theta_p^k$ | variable binaire valant 1 si le chemin $p$ est retenu à l'optimalité pour la commodité $k$. |
| $c_p^k$ | coût reflétant la qualité du chemin $p$ de la commodité $k$. |
| $Z_m$ | variable binaire valant 1 si la mission $m$ est rejetée et 0 autrement. |
| $E$ | ensemble des équipages légaux. |
| $e$ | un équipage donné. |
| $x_v^e$ | variable binaire valant 1 si on décide de traiter la tâche $v$ avec l'équipage $e$ et 0 autrement. |
| $c_v^e$ | coût associé à $x_v^e$. |
| $\mathcal{C}$ | ensemble de cliques. |
| $C$ | une clique donnée. |
| $f_{ev}^k$ | nombre d'heures volées par le pilote $k \in K^A$ lors de l'exécution de $v$ avec l'équipage $e$. |
| $F_k$ | nombre maximum d'heures de vols permises pour $k$. |