Structuration de lignes d'usinage : méthodes exactes et heuristiques

Brigitte Finel

THESE

Présentée par

Brigitte FINEL

Ingénieur CNAM

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE DE METZ
(arrêté ministériel du 30 Mars 1992)

en AUTOMATIQUE, Spécialité : PRODUCTIQUE

Ecole doctorale Informatique, Electronique - Electrotechnique, Mathématiques (IAEM)

Structuration de lignes d'usinage : méthodes exactes et heuristiques

Soutenue le 1er Décembre 2004 devant le jury :

Président :

Henri PIERREVAL, *Professeur à l'IFMA, Clermont-Ferrand*

Rapporteurs :

Alain DELCHAMBRE, *Professeur à l'Université Libre de Bruxelles*

Gerd FINKE, *Professeur à l'Université Joseph Fourier de Grenoble*

Examinateurs :

Marie Claude PORTMANN, *Professeur à l'Ecole des Mines de Nancy*

Alexandre DOLGUI, *Professeur à l'Ecole des Mines de Saint Etienne,* Directeur de thèse

François VERNADAT, *Professeur à l'Université de Metz,* Directeur de thèse

Invités :

Pierre PADILLA, *Professeur, Directeur de l'Ecole Nationale d'Ingénieurs de Metz*

Jean-Marie PROTH, *Directeur de Recherche* à l'INRIA

Thèse préparée au sein du Laboratoire de Génie Industriel et de Production Mécanique
à l'Ecole Nationale d'Ingénieurs de Metz

THESE

Présentée par

Brigitte FINEL

Ingénieur CNAM

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE DE METZ
(arrêté ministériel du 30 Mars 1992)

en AUTOMATIQUE, Spécialité : PRODUCTIQUE

Ecole doctorale Informatique, Electronique - Electrotechnique, Mathématiques (IAEM)

Structuration de lignes d'usinage : méthodes exactes et heuristiques

Soutenue le 1er Décembre 2004 devant le jury :

Président :

Henri PIERREVAL, *Professeur à l'IFMA, Clermont-Ferrand*

Rapporteurs :

Alain DELCHAMBRE, *Professeur à l'Université Libre de Bruxelles*

Gerd FINKE, *Professeur à l'Université Joseph Fourier de Grenoble*

Examinateurs :

Marie Claude PORTMANN, *Professeur à l'Ecole des Mines de Nancy*

Alexandre DOLGUI, *Professeur à l'Ecole des Mines de Saint Etienne,* Directeur de thèse

François VERNADAT, *Professeur à l'Université de Metz,* Directeur de thèse

Invités :

Pierre PADILLA, *Professeur, Directeur de l'Ecole Nationale d'Ingénieurs de Metz*

Jean-Marie PROTH, *Directeur de Recherche* à l'INRIA

Thèse préparée au sein du Laboratoire de Génie Industriel et de Production Mécanique
à l'Ecole Nationale d'Ingénieurs de Metz

Remerciements

Les travaux de recherche présentés dans ce mémoire de thèse ont été réalisés dans l'équipe AGIP du LGIPM.

Je tiens à remercier Monsieur Pierre Padilla, Professeur, Directeur de l'ENIM, et Monsieur Gabriel Abba, Directeur du LGIPM pour m'avoir permis d'effectuer ses travaux de recherche dans de très bonnes conditions.

Je remercie tout particulièrement Monsieur Alexandre Dolgui, Professeur à l'Ecole des Mines de Saint Etienne et Monsieur François Vernadat, Professeur à l'Université de Metz pour m'avoir encadrée durant cette thèse. Sans eux, leurs conseils, leur patience résistant à toutes épreuves, je ne serai jamais parvenue à ce résultat.

Je voudrais remercier Monsieur Gerd Finke Professeur à l'Université Joseph Fourier de Grenoble et Alain Delchambre Professeur à l'Université Libre de Bruxelles pour avoir accepté d'évaluer mon travail et d'en être les rapporteurs.

J'associe à ces remerciements les autres membres du jury, Monsieur Henri Pierreval, Professeur à l'IFMA, Clermont-Ferrand, Mademoiselle Marie Claude Portmann, Professeur à l'Ecole des Mines de Nancy, Monsieur Pierre Padilla, Professeur, Directeur de l'Ecole Nationale d'Ingénieurs de Metz, Monsieur Jean-Marie Proth, Directeur de Recherche à l'INRIA.

Je tiens à exprimer mes remerciements à Mademoiselle Marie-Claude Portmann et à Monsieur Xiaolan Xie pour m'avoir intégrée comme invitée à l'équipe MACSI de l'INRIA, ce qui m'a permis d'utiliser tous les outils mis à la disposition des chercheurs de l'INRIA (en particulier le solveur Cplex et la base de données documentaire).

Je remercie également Monsieur Patrick Martin, Professeur à l'ENSAM, pour ses conseils et sa relecture assidue du premier chapitre de ce mémoire de thèse.

Merci encore à tous les membres du LGIPM et de l'équipe informatique de l'ENIM, qui m'ont soutenue durant tout ce temps.

Merci à ceux de mes collègues de l'ENIM et de l'Université de Metz qui m'ont aidé par leur bonne humeur et leur compréhension.

Je ne veux pas oublier de remercier mes collègues de Minsk et en particulier Nikolai Guschinsky, chercheur à l'Institut Unifié des Problèmes Informatiques, *Biélorussie*, pour leur aide et leurs conseils avisés.

Merci aussi à tous mes amis que je ne pourrai pas tous citer ici (ils se reconnaîtront), et à ma famille (en particulier mes neveux) qui m'ont supportée pendant ces trois ans grâce à leurs encouragements continus et à leur affection.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE

$\lceil x \rceil$	La plus petite valeur entière supérieure ou égale à x
$\lfloor x \rfloor$	Partie entière de x
$\|N\|$	Nombre d'opérations dans l'ensemble **N**
a	Temps de mise en route de l'outil
ch	Une caractéristique de l'outil
C	Coût de la ligne pour la solution courante
$C(x)$	Valeur de la fonction objectif
$C(k)$	Valeur de la fonction objectif pour la station k
C_1	Coût relatif d'une station
C_2	Coût relatif d'une tête d'usinage
CH	Nombre total de caractéristiques
C_i^o	Coût total de l'opération i
C_i^{ow}	Coût salarial pour effectuer l'opération i
C_{min}	Coût de la solution la meilleure
CNC	Nombre de contraintes donné par Cplex
$Coef1$	Paramètre du modèle BAL
$Coef2$	Paramètre du modèle BAL
$CouvArb$	Paramètre permettant de calculer le nombre de branchements autorisés
C^{sc}	Coût de capital par cycle pour une station
C_i^{sc}	Coût du capital pour l'opération i
C_k^s	Coût total de la station k
C_k^{sw}	Coût salarial sur la station k
$CTNV$	Nombre de variables donné par Cplex
D	Ensemble des arcs du graphe de précédence
$d[i,j]$	Distance entre deux opérations i et j dans le calcul des indices de blocs et de stations
D_j	Diamètre de l'outil
Dst	Distance de la solution courante à la solution idéale

ε_1	Une valeur non négative suffisamment petite
ε_2	Une valeur non négative suffisamment petite
EB	Ensemble des contraintes d'inclusion pour le regroupement des opérations en blocs
$\overline{EB}$	Ensemble des contraintes d'exclusion interdisant le regroupement des opérations en blocs
EF	Efficience ce le ligne
ES	Ensemble des contraintes d'inclusion pour le regroupement des opérations en stations
$\overline{ES}$	Ensemble des contraintes d'exclusion interdisant le regroupement des opérations en stations
ex	Numéro d'exemple
F_l	Une liste d'opérations dans la méthode COMSOAL
F_q	Variable auxiliaire servant à déterminer la durée du bloc q
F_i^*	Ensemble de tous les successeurs de l'opération i
g	Profit par unité de produit
G	Graphe de précédence
$j(e)$	Une opération de l'ensemble e
$K(j)$	Ensemble des indices k de stations où l'opération j peut être affectée
$k^+[j]$	Extrémité droite de $K(j)$.
$k^-[j]$	Extrémité gauche de $K(j)$.
$L(N)$	Longueur de la course d'outils de la tête d'usinage réalisant l'ensemble des opérations N
LB	Borne inférieure de la fonction objectif
$LB(T)$	Borne inférieure du cadencement
LF_i	Une liste d'opérations
L_j	Longueur (ou profondeur) de l'outil
l_j	Longueur additionnelle pour le fraisage ou longueur de la course d'outil suivant le contexte
LLB	Borne inférieure locale du critère
LLB^j_m	Borne inférieure locale du nombre de stations au nœud j
LP_i	Une liste d'opérations
m	Nombre de stations
$\overline{m}$	Borne supérieure du nombre de stations
$\underline{m}$	Borne inférieure du nombre de stations

$m(P)$	Nombre de stations pour la décision P
m_c	Coût de la matière première
$\mathbf{N}$	Ensemble de toutes les opérations à réaliser
N	Un sous-ensemble d'opérations de $\mathbf{N}$
n	Nombre de blocs dans une solution
$\underline{n}$	Borne inférieure du nombre de blocs
$\overline{n}$	Borne supérieure du nombre de blocs
NA	Nombre d'arcs dans le graphe de précédence
Nbl	Nombre minimum de blocs (nombre de niveaux) à remonter
$Nbra(v)$	Nombre maximum de branchements autorisés à partir du sommet v
NC	Nombre de contraintes avant le pré-processeur Cplex
$Ndbl$	Compteur du nombre de tentatives de déblocage
$NdblInit$	Paramètre donnant le nombre maximum de tentatives de déblocage
NDV	Nombre de variables de décision x_{jq} avant le pré-processeur Cplex
nEB	Nombre de sous-ensembles de EB
$n\overline{EB}$	Nombre de sous-ensembles de $\overline{EB}$
nES	Nombre de sous-ensembles de ES
$n\overline{ES}$	Nombre de sous-ensembles de $\overline{ES}$
$Niv(v)$	Numéro d'ordre de la dernière opération dans la séquence d'affectation menant de la racine de l'arbre d'énumération à v
N_k	Ensemble des opérations réalisées sur la station k
n_k	Nombre de têtes d'usinage (blocs d'opérations) de la station k
N_{kl}	Ensemble d'opérations réalisées par la tête l de la station k
N_k^*	Ensemble d'opérations qui peuvent potentiellement être assignées à la station k
$Nniv$	Compteur du nombre de niveaux remontés ;
NO	Nombre d'opérations
Op	Opération choisie
P	Décision de conception complète
p	Une décision partielle
pf	Taux de profit
p_k	Taux d'occupation de la station k
P_l	Une liste d'opérations de COMSOAL
P_i^*	Ensemble de tous les prédécesseurs de l'opération i

$Poids(i)$	Poids de l'opération i
pp	Plus grand nombre de prédécesseurs des opérations
PR	Pourcentage de temps mort
$Pred(j)$	Ensemble de tous les prédécesseurs de l'opération i
$Prob$	Probabilité de ne pas remonter vers le bloc précèdent
$ProbInit$	Paramètre de l'algorithme donnant la valeur initiale de $Prob$
$Q(j)$	Ensemble des indices q (blocs) où l'opération j peut être effectuée
$q^-[j]$	Extrémité gauche de $Q(j)$
$q^+[j]$	Extrémité droite de $Q(j)$
q_0	Valeur maximale possible pour q, $q_0=m_0n_0$
$Random$	Une variable aléatoire uniformément distribuée entre 0 et 1
rES	Taille maximum (nombre d'éléments) des sous-ensembles de EB
$r\overline{EB}$	taille maximum (nombre d'éléments) des sous-ensembles de $\overline{EB}$
rES	Taille maximum (nombre d'éléments) des sous-ensembles de ES
$r\overline{ES}$	taille maximum (nombre d'éléments) des sous-ensembles de $\overline{ES}$
RT^j_{mrt}	Temps mort au nœud j
r_u	Coût de vente
$S(k)$	Ensemble des blocs (leurs numéros) de la station k
$S(N)$	Avance par minute de la tête d'usinage réalisant l'ensemble d'opérations N
s_j	Taux de pénétration par dent, avance par tour, ou avance par minute suivant le contexte
t	Durée de production d'une unité de produit
T	Temps de cycle réel de la ligne
$T(P)$	Temps de cycle de la ligne pour une décision de conception P
T_0	Temps de cycle objectif (voulu)
$t^b(N)$	Temps du bloc N
t_e	Temps d'exécution
t_j	Temps de l'opération j
$t_j(v_j)$	Temps de l'opération j pour la vitesse d'usinage v_j
T_k	Temps libre de la station k
T^k_{mrt}	Temps mort de la station k
T_{mrt}	Temps mort total par pièce
TNV	Nombre total des variables avant le pré-processeur Cplex

t_p	Temps de préparation
t_r	Temps de remplacement
TR_{nimp}	Nombre d'itérations qui n'améliorent pas la solution
TR_{tot}	Nombre d'itérations déjà effectuées
$t^s(N_k)$	Temps de cycle de la station k
t_{sum}	Somme de tous les temps opératoires de **N**
u	Coût de production d'une unité de produit
UB	Borne supérieure
u_o	Coût de l'outil
u_p	Coût de production
u_r	Coût d'amortissement de la machine
v_j	Vitesse d'usinage
$W(N)$	Force d'avance totale pour le bloc N
W_{ch}	Valeur limite de la caractéristique c
W_{ich}	Apport de l'opération i à la définition de la caractéristique c
w_{ij}	Variable binaire
x_{jq}	Variable de décision binaire (x_{jq}= 1 si l'opération $j \in$**N** est affectée au bloc q et x_{jq} = 0 sinon)
Y_q	Variable auxiliaire indiquant si le bloc q existe ou non
z_j	Nombre de dents de l'outil de fraisage
Z_k	Variable auxiliaire indiquant si la station k existe ou non
a_j	Constante d'usinage
ρ	Taux de production
τ^b	Temps auxiliaire par bloc
t^s	Temps auxiliaire par station

Introduction

L'objectif de cette thèse est de développer des méthodes efficaces pour la structuration optimale des lignes d'usinage. Ce travail concerne la production de masse et en grandes séries de familles de produits proches en industrie mécanique. Le problème étudié dans cette thèse apparaît à l'étape de la conception lorsqu'un nouveau produit et ses gammes de fabrication sont connus. Il faut alors concevoir une nouvelle ligne pour fabriquer ce produit au moindre coût, tout en respectant des contraintes technologiques. Une fois que la ligne est conçue, il est difficile de changer sa structure pendant toute la période de son exploitation (3-5 ans). Etant donné l'effet d'échelle, une moindre erreur à ce stade provoque des pertes économiques importantes. Cela justifie que des études scientifiques approfondies soient menées pour trouver de très bonnes décisions, si possible optimales.

Historiquement, le premier problème de ce type a été connu sous le nom de Simple Assembly Line Balancing Problem (SALBP), i.e. équilibrage des lignes d'assemblage (il a été posé pour des chaînes d'assemblage dans l'industrie automobile). Pour ce problème classique, la chaîne est utilisée pour fabriquer un seul type de produit. L'ensemble des opérations élémentaires à réaliser sur la chaîne ainsi que leurs contraintes de précédence (obtenues en analysant toutes les gammes d'assemblage possibles) sont connues. Le temps de cycle de la ligne (productivité voulue) est également connu. Le but est de répartir les opérations sur les stations de travail de sorte que le temps mort des stations soit minimal. Le temps mort de chaque station est défini comme la différence entre le temps de cycle voulu (la charge objective) et la charge réelle (la somme des temps d'opérations affectées à cette station). Pour ce problème, la solution qui offre le plus petit temps mort donne le nombre minimum de stations (le coût minimal de la ligne).

Le problème d'équilibrage des lignes d'usinage qui est appelé dans ce mémoire Transfer Line Balancing Problem (TLBP) est moins connu et plus complexe. Des contraintes complémentaires apparaissent, comme, par exemple, celles liées à l'impossibilité de mettre

certaines opérations sur une même station à cause d'un outillage incompatible ou l'obligation de faire exécuter certaines opérations par la même tête d'usinage, pour pouvoir respecter une tolérance donnée. La recherche des solutions optimales se complique donc parce que les durées des opérations dépendent de leur affectation, plusieurs opérations s'exécutent en parallèle, etc.

Nous pouvons trouver plusieurs explications au fait que TLBP est moins connu que SALBP :

1. il est plus complexe : si pour l'assemblage nous pouvons facilement nous limiter aux contraintes de précédence (surtout pour l'assemblage manuel), dans l'usinage il faut tenir compte des caractéristiques multiples et variées liées aux modes d'usinage possibles, aux outils et têtes d'usinage réalisables, aux contraintes constructives et mécaniques d'usinage, etc ;
2. il concerne principalement les grands constructeurs de machines–outils, c'est-à-dire de grands groupes dont le nombre est limité et qui communiquent rarement leurs compétences et problèmes à l'extérieur de leurs bureaux d'études ;
3. si pour un assemblage manuel (assemblage des véhicules par exemple), le problème d'équilibrage se pose pratiquement au début de chaque mois, durant le cycle de vie, pour les lignes d'usinage, ce problème apparaît uniquement à l'étape de sa conception (il est donc résolu au sein de l'entreprise fabricant ces lignes et machines uniquement).

Un certain nombre de facteurs ont orienté cette thèse :

- l'augmentation du coût des machines-outils et des lignes d'usinage ; ce facteur a fait naître un vrai besoin en outils d'aide à la décision et d'optimisation pour l'industrie des machines-outils ;
- l'évolution de la puissance des ordinateurs et des logiciels d'optimisation : il est maintenant possible d'aborder des problèmes difficiles comme celui traité dans ce mémoire ; nous voulons particulièrement souligner un saut formidable dans le développement des outils de programmation linéaire en variables mixtes

soutenus par des produits logiciels du marché comme ILOG Cplex, XpressMP, GAMS, OSL, etc ;

- un autre facteur d'encouragement pour commencer ce travail est lié au fait que des nouvelles plates-formes d'ingénierie proposent déjà quelques solutions (voir les outils de Dassault Systèmes et de Tecnomatix) qui peuvent aider à résoudre ce type de problèmes, mais ce marché est loin encore de celui des outils de CAO ou de GPAO (ERP) ; un effort général est nécessaire aussi bien au niveau de la recherche académique, que des développements informatiques, surtout pour les problèmes de l'industrie mécanique comme celui de ce mémoire.

Dans le Chapitre 1 nous présentons les machines–outils et lignes d'usinages. Nous montrons leurs principales caractéristiques et leur spécificité. Nos donnons la liste des principaux constructeurs de machines-outils et lignes d'usinage, nous montrons certaines des tendances de développement du secteur. Nous terminons le chapitre par une présentation du problème que nous étudions dans ce mémoire.

Le Chapitre 2 est consacré à l'état de l'art pour l'équilibrage des lignes de fabrication et pour les méthodes de modélisation et d'optimisation dans le domaine. Nous montrons les principales familles de méthodes d'optimisation combinatoire utilisées, leurs propriétés et caractéristiques. Nous faisons une analyse des publications sur les méthodes d'équilibrage des lignes d'assemblage en présentant des méthodes exactes et des méthodes approchées. Nous mettons un accent sur les publications les plus proches de notre problème. Nous terminons en présentant les différences majeures entre les problèmes traités dans la littérature et celui de cette thèse. Nous montrons pourquoi les méthodes connues ne pourront pas être utilisées directement dans notre étude et nous donnons des voies de développement des méthodes dédiées au problème TLBP auquel cette thèse est consacrée.

Dans le Chapitre 3, nous proposons un modèle de programmation linéaire en variables mixtes pour l'optimisation TLBP. Nous illustrons les contraintes du modèle sur un exemple industriel. Nous testons ses performances pour quatre cas industriels et pour des exemples générés aléatoirement. Ensuite nous montrons les améliorations possibles du modèle. Le

chapitre se termine par une analyse approfondie des améliorations apportées et par des conclusions qui indiquent pour quel type et quelle taille de problème ce modèle exact peut être utilisé dans l'environnement industriel.

Dans le Chapitre 4, nous proposons plusieurs heuristiques capables de traiter les problèmes industriels de grande taille. Ces heuristiques se basent sur les techniques de COMSOAL et, de manière plus générale, sur une recherche aléatoire dans l'arbre d'énumération. Des tests numériques pour des cas industriels et pour les exemples générés aléatoirement montrent l'efficacité des algorithmes proposés.

Le Chapitre 5 est consacré au couplage de la méthode exacte avec les heuristiques. Nous proposons plusieurs schémas de décomposition des modèles permettant de traiter des exemples de grande taille et intégrant simultanément les deux types de méthodes développées dans les chapitres précédents.

Nous terminons ce mémoire par une conclusion générale et par une bibliographie.

Chapitre 1

Machines et Lignes d'usinage

1. Machines et lignes d'usinage

Dans ce chapitre, nous présentons de façon générale les systèmes d'usinage dans la production de masse. Nous présentons d'abord quelques caractéristiques des machines-outils pour donner ensuite une définition d'une ligne de transfert et de ses composants. Nous montrons les avantages et les inconvénients de ces lignes et posons la problématique de leur conception préliminaire. Une explication détaillée de l'objectif de la thèse, qui consiste à développer un ensemble de méthodes d'aide à la décision pour structurer de telles lignes, est ensuite donnée.

1.1 La machine-outil

Avant de décrire une machine-outil, rappelons qu'un solide possède six degrés de liberté couramment décrits par trois translations selon les axes 0X, 0Y et 0Z orthogonaux (dans un repère direct cartésien) et trois rotations autour de vecteurs parallèles à chaque axe comme désignés dans la Figure 1-1 par A, B et C.

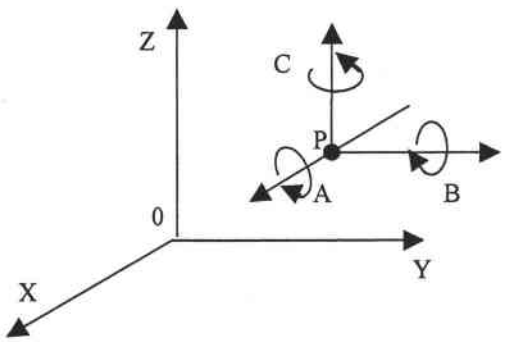

Figure 1-1 : Trois axes et six degrés de liberté dans l'espace 3D

Une machine-type pour l'usinage des métaux est composée de plusieurs éléments. Ils sont énumérés dans les paragraphes suivants, pour plus de détails voir (Pruvost, 1993).

Le bâti est l'intermédiaire entre le sol et les éléments actifs de la machine. Il tient en position correcte la pièce à usiner et l'outil. Il met en relation la zone de travail et l'opérateur. Le bâti doit être rigide car les outils et les pièces qui y sont fixés ne doivent pas subir de perturbations (si la pièce ou l'outil bouge un peu, l'usinage ne pourra pas s'effectuer correctement). Dans des lignes où les positions des pièces et outils doivent être adaptées à un opérateur, le bâti ne peut assurer parfaitement une telle rigidité alors que dans les lignes complètement automatisées, ce problème est résolu. Le bâti peut être constitué d'un ou plusieurs éléments.

La pièce doit être placée dans un certain sens face aux outils pour pouvoir être usinée correctement. Des dispositifs (fixations, brides...) doivent être utilisés pour que la pièce ne subisse pas de déformations statiques (défauts de forme et de dimension des surfaces usinées) ou dynamiques (défauts d'état de surface et des vibrations, ...).

L'outil peut avoir de multiples formes pour des usages multiples.

Le déplacement de l'outil par rapport à la pièce se réalise couramment par deux chaînes cinématiques comme expliqué dans (François, 1995), chaque élément ayant soit un seul degré de liberté commandé par rapport au produit (liaison cinématique), soit étant une liaison démontable (pièce sur porte pièce par exemple). Ces chaînes expliquent entre autres des contraintes de compatibilité pour des opérations.

La broche met en rotation l'outil pour usiner la pièce en fraisage (en tournage, c'est la pièce qui tourne). La broche peut être pilotée en tournage et ne l'est pas en fraisage. **Les paliers** doivent être rigides et ne pas avoir de jeu pour que la broche travaille de façon précise et pour permettre des rotations à grande vitesse en consommant le moins de puissance possible. Les efforts nécessaires pour réaliser un usinage sont principalement fournis par l'intermédiaire de la broche.

Le porte-outils assure la liaison démontable entre le cône broche et l'outil. Le changement de l'ensemble outil - porte-outils se fait couramment d'une façon automatique.

Des têtes multibroches peuvent être fixées afin d'augmenter la productivité mais au détriment de la flexibilité. L'ensemble outils et porte-outils constitue une **tête d'usinage** (Figure 1-2).

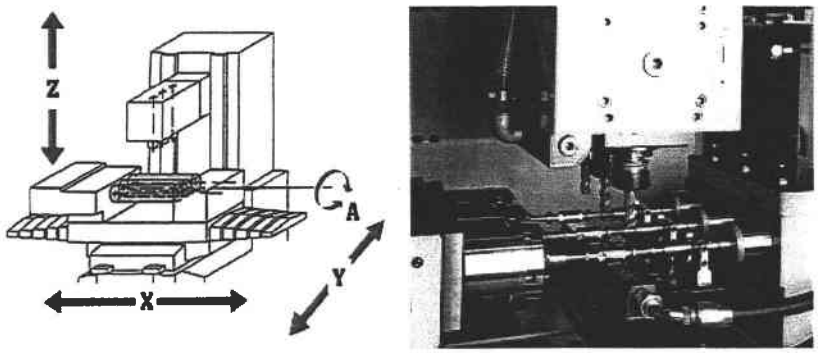

Figure 1-2 : Tête d'usinage à trois broches pour des pièces cylindriques (Wyssbrod-technologie)

Il existe de nombreuses variétés de têtes d'usinage. Certaines têtes d'usinage servant au travail du métal sont présentées ci-dessous :

- Les têtes d'usinage ajustables (Figure 1-3) servant pour des modèles de trous équidistants. Tous les outils tournent de la même manière ; ils peuvent faire des opérations de fraisage, d'alésage, de perçage et de tournage léger.

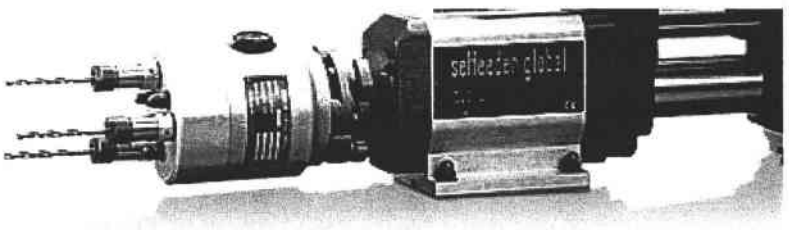

Figure 1-3 : Tête d'usinage ajustable

- Les têtes d'usinage fixes sont faites pour des modèles de trous irréguliers. Elles sont plus compactes et légères que leurs homologues ajustables.

- Les têtes d'usinage désaxées sur lesquelles les outils ne sont pas dans le même alignement que les portes-pièces. Elles permettent de résoudre les problèmes de localisation de trous situés à un endroit difficilement accessible.

Deux grandes classes de pièces existent :

- les pièces de révolution qui sont usinées sur un tour et montées directement dans le mandrin du tour (en l'air ou avec contre pointe) ;
- les pièces prismatiques qui sont fixées à l'aide d'un montage d'usinage spécifique (grande série) ou modulaire (petite série) sur la table ou plus généralement sur une palette porte-pièce (permet un montage-démontage en temps masqué).

Le mouvement des axes est assuré par **un actionneur** (électrique ou hydraulique) et **un asservissement.** Le mouvement de rotation est transformé à l'aide d'une chaîne cinématique en translation ou en rotation élémentaire.

La réalisation des surfaces quelconques nécessite la synchronisation des axes (contournage) donc des équipements très onéreux. Par contre, pour les opérations plus simples (perçage, ...), il est nécessaire de positionner l'outil et ensuite d'assurer un contrôle du mouvement d'avance. Dans ce cas, la commande peut être réalisée par un automate programmable, par contre, le nombre d'axes à piloter peut devenir important (lignes de fabrication dédiées à la grande série).

Il existe différentes configurations de structure de machines suivant l'ordre des degrés de liberté en translation ou en rotation (Bohez, 2002). Pour les machines courantes de trois à cinq axes, on peut identifier vingt quatre types de machines (Bernard, 2003).

1.2 Marché des machines-outils

Les machines-outils sont réparties en général en deux catégories selon leur procédé de production : a) machines enlevant de la matière et produisant des copeaux (usinage) et b)

machines transformant la matière (formatage). Les procédés les plus importants utilisés dans la première catégorie sont le tournage, le fraisage, le meulage et l'affûtage.

En ce qui concerne les procédés transformant la matière, les plus communs sont la forge, l'emboutissage et la fonderie d'assemblage. La forme est réalisée par l'outillage dont le mouvement est souvent limité à un axe de translation. A l'exception de quelques machines japonaises, les fabricants de machines-outils ne font pas de machines appartenant à ces deux catégories en même temps.

Les machines-outils peuvent également être classées selon leur morphologie.

Dans le Tableau 1-1, le nombre d'axes de la machine est indiqué dans la première colonne ; la deuxième colonne donne des exemples de type de machine ; la troisième colonne présente des exemples de types d'usinage et enfin la dernière colonne représente quelques exemples de réalisation.

Dans le Tableau 1-2, nous présentons l'évolution du marché des machines-outils (Felder, 1997) en Europe de 1989 à 1994.

Notons que l'Europe représentait 41% de la production mondiale des machines-outils. Dans ce tableau, nous constatons que les machines de transfert et centres d'usinage, objet de notre étude, représentent 16,3% de la production de machines-outils en 1989 et 19,2 % en 1994.

Au niveau Européen, le secteur des machines-outils emploie de nombreuses personnes (voir le Tableau 1-3) provenant des sources Eurostat (Eurostat, 2004).

Nb. d'axes	Type de machine	Type d'usinage	Exemples
1	Brocheuse	Usinage d'une surface par simple translation	Face de bloc cylindre, Face de culasse ou alésage de trous carrés
2	Tour	Usinage de pièces de révolution avec un outil à une seule arête de coupe	Pièce animée d'un mouvement de rotation, outil animé d'un mouvement de translation selon X et Z
3	Centre d'usinage à 3 axes	Usinage des faces perpendiculaires à l'axe de la broche, pièce selon axes X et Y, l'axe Z parallèle à l'axe de broche fournit l'avance du travail, cet usinage est peu rapide (outil doit retourner à sa place)	Fraisage de rainures, Usinage de formes cylindriques de révolution, Usinage de pièce de forme quelconque dans un demi-espace grâce à un outil sphérique ou torique
3	Centre de tournage	Tournage	Rainures de clavettes, Trous axiaux et radiaux et même inclinés
4	Centre d'usinage à 4 axes	Trois translations X, Y et Z et une rotation parallèle à Y en général	Pièces de formes très complexes
5	Machine à fraiser et à aléser	L'outil peut prendre n'importe quelle position angulaire dans l'espace cartésien	Moules, matrices, pales, voilures
5	Centre de tournage	Associe à la machine à 4 axes un axe B (voir Figure 1-1)	Inclinaison d'un usinage auxiliaire
6	Robot, Centre d'usinage (différents triplets rigidité/vitesse/ précision)	Souvent à 5 ou 6 axes mais de précision inférieure aux Machines-outils	Opérations de soudure sur carrosserie, peinture
>6	Tours multibroche	Machines à affûter	
X	Machines spéciales ou de transfert	Multi-axes	Voir plus loin

Tableau 1-1 : Classification des machines-outils

Type de machine-outil	1989	1990	1991	1992	1993	1994
Usinage non traditionnel électroérosion, laser	4	4	3,6	3,7	4,4	5,8
Centre d'usinage, machines-transferts	16,3	18,3	17,7	20	19,8	19,2
Tours	15,4	14,5	13,4	12,4	10,1	10,8
Fraiseuses, perceuses, aléseuses	14,2	13,9	11,9	11	9,8	9,4
Machines à rectifier et à polir	11,1	11,1	12	10,8	11,3	11,2
Autres machines d'usinage	10,4	9,7	9,3	8,8	9	9
Total des machines d'usinage (%)	**71,4**	**71,5**	**67,9**	**66,7**	**64,4**	**65,3**
Presse, emboutissage, forgeage, pliage, poinçonnage	20,6	19,2	21,8	23,4	24,3	23,6
Autres machines de formatage	8	9,3	10,3	9,9	11,3	11,1
Total des machines de formatage (%)	**28,6**	**28,5**	**32,1**	**33,3**	**35,6**	**34,7**
Production totale de machines outils (milliards d'écus)	**14,25**	**15,45**	**14,85**	**12,27**	**9,6**	**10,22**

Tableau 1-2 : Evolution du marché des machines-outils en Europe de 1989 à 1994

États membres de l'UE, 2000 Secteur	Emploi total dans l'UE (en 1000)	Valeur ajoutée totale de l'UE (en milliards d'euros)	Principale contribution à la valeur ajoutée de l'UE	État membre le plus spécialisé	État membre le moins spécialisé
Équipements mécaniques et machines d'usage général (29.1 + 29.2)	1 524,0	77,9	Allemagne	Danemark	Luxembourg
Machines agricoles (29.3)	170,7	7,8	Allemagne	Danemark	Luxembourg
Machines-outils (29.4)	277,1	15,0	Allemagne	Allemagne	Irlande
Autres machines d'usage spécifique (29.5)	803,0	41,4	Allemagne	Finlande	Irlande
Armes et munitions (29.6)	52,4	2,9	UK	Suède	Irlande
Appareils domestiques (29.7)	236,9	11,5	Allemagne	Italie	Belgique
Machines et équipements (DK)	3 069,6	156,7	Allemagne	Allemagne	Irlande
Industrie manufacturière (D)	28 253,0	1 453,1	---	---	---

Notes: EL: n.d.; IRL: 1999. Par État membre le plus spécialisé, il faut entendre le pays dont la part des machines et équipements dans le total de la valeur ajoutée de l'industrie manufacturière est la plus élevée par rapport à la part moyenne observée dans l'UE. L'État membre le moins spécialisé est celui dont le ratio dans ce domaine est le plus faible.

Source: Eurostat, sauf indication contraire.

Tableau 1-3 : Fabrication de machines dans les Etats de l'UE en 2000

Les principales compagnies mondiales fabricant et/ou commercialisant des lignes de transfert sont répertoriées dans le Tableau 1-4.

Dans notre travail, nous utilisons également certaines données de l'Usine des lignes de transfert de Minsk – MZAL (Biélorussie). C'est une entreprise leader dans l'espace post soviétique (pays de l'ex URSS) en fabrication de machines et de lignes d'usinage automatiques. Les données nous ont été fournies par nos collègues biélorusses travaillant avec nous sur ce problème dans le cadre d'un projet INTAS financé par la Commission Européenne.

Les lignes de transfert sont majoritairement utilisées dans l'industrie automobile et de biens d'équipement, l'évolution des principales industries manufacturières françaises est présentée par la Figure 1-4 (Sessi, 2004).

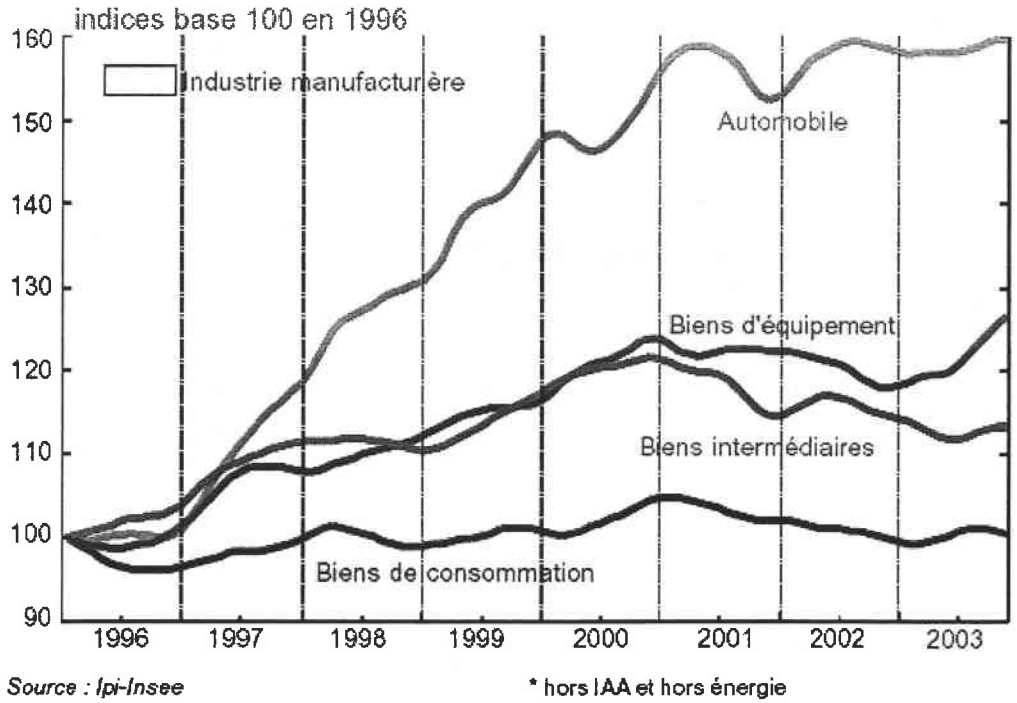

Figure 1-4 : La production manufacturière en France

Société	Web	Secteur d'activités	Industries
BURKHARDT-WEBER	www.burkhardt-weber.de	mesures, réajustement optique, liens inter-lignes par portiques, modules flexibles 3 axes	
COMAU (EX RENAULT AUTOMATION)	www.comau.fr	systèmes d'usinage, systèmes d'assemblage, robotique et montage final, études et ingénierie véhicules	Industrie automobile
DEARBORN MID-WEST CONVEYOR	www.dmwcc.com	systèmes et produits industriels pour l'automobile	Chrysler
ETXE-TAR	www.etxe-tar.com	création de systèmes d'usinage pour pièces de l'automobile	Audi, Citroen, Ford, Chrysler, Gm, Nissan, Opel, Peugeot, Renault, Nissan, WW, Seat, Opel
EX-CELL-O	www.ex-cell-o.de	conception de machines de transfert autonome pour la fabrication de pièces cubiques	Industrie automobile
EXPERT MASCHINENBAU	www.expert-international.com	planification, de l'ingénierie, de la gestion du projet/des coûts, du montage de machines spéciales	
GEORG	www.georg-gruppe.de	conception construction et fabrication de machines pour l'ingénierie mécanique	leader mondial
HELLER MACHINES-OUTILS	www.heller.co.uk	centres d'usinage, boîtiers multibroches, machines-transferts, machines d'usinage de vilebrequins / arbres à cames (un logiciel spécialement développé permet, en fonction de la pièce et du cycle d'usinage, de calculer les temps d'usinage et les temps morts de toutes les pièces courantes)	
I.M.V PRESSE	www.imvgalli.it	usinage de la tôle	automobile, électroménagers, casseroles et composants
MÜLLER WEINGARTEN	www.mwag.de	industrie automobile mondiale	
REXROTH BOSH GROUP	www.boschrexroth.com	systèmes de production automatisée et de technologie de groupe	
REKO	www.rekointl.com	conception et fabrication de moteurs d'injection en plastique, de pièces, de moteurs de compression, de machines spéciales, de centres de fabrication CNC	leaders pour l'automobile, les produits industriels et de ménage et pour l'industrie aérospatiale
WIEDEMANN GmbH & Co.KG	www.wiedemann.de	systèmes d'usinage, systèmes d'assemblage, robotique et montage final, études et ingénierie véhicules	

Tableau 1-4 : Principales sociétés fabriquant des lignes de transfert

1.3 La ligne de transfert

Nous pouvons distinguer trois types d'agencement d'atelier : l'agencement fonctionnel, l'agencement cellulaire et les lignes de transfert.

Dans **l'agencement fonctionnel**, les machines sont regroupées en sections spécialisées (sections homogènes) par fonctions, par exemple section de tournage, section de fraisage, etc. Les pièces circulent de section en section. Cela permet un très haut niveau de flexibilité. Cette forme est utilisée pour des entreprises ayant une production très diversifiée et en petites et moyennes séries. Elle présente certains inconvénients tels que le coût de manutention élevé, un large éventail d'encours et une gestion de flux complexe.

Dans **l'agencement cellulaire**, les processus, les hommes et les machines sont regroupés de manière à produire des familles de pièces similaires. L'objectif de ce regroupement est d'approcher l'efficience de la production en petite et moyenne série, pour laquelle cette forme d'agencement est utilisée, à celle de la production de masse. Cet agencement combine les avantages de la production de masse et de la production en petite série, par contre les ressources machines sont souvent sous utilisées à cause du dédoublement des machines dans plusieurs cellules.

La forme la plus économique en utilisation des ressources et qui permet la productivité la plus grande est **la ligne de transfert.** Pour pouvoir l'utiliser, il faut avoir des produits homogènes, ou un seul type de produit, fabriqués en grand volume. Elle est donc utilisée pour une production de masse : une grande quantité d'un même type de produit fini est usinée en suivant les mêmes séquences d'opérations sur des machines disposées en série. Elle convient à un écoulement régulier des encours avec un temps de cycle court. Les lignes de transfert étant l'objet de cette étude, nous détaillons leurs avantages et leurs inconvénients.

Les **principaux avantages** apportés par l'utilisation des lignes de transfert peuvent être énumérés comme suit :

- la précision : les lignes de transfert sont conçues pour apporter un maximum de précision dans l'usinage et l'assemblage des pièces ;

- la qualité : ce sont souvent les lignes automatisées, mais pour les lignes semi-automatisées, cela est également valable car les ouvriers font toujours les mêmes tâches et connaissent parfaitement leur travail (cela peut aussi être un inconvénient et amener une certaine lassitude pouvant provoquer des incidents par inattention) ;
- la productivité : la possibilité (voir la nécessité) de faire une production de masse (amortisseurs, pièces automobiles, ...), les volumes annuels de produits finis peuvent atteindre plusieurs millions ;
- l'interaction entre l'étude du produit et de ses moyens de production est simplifiée ;
- la production de déchets est minime (diminution du nombre de réglages) ;
- le coût de manutention est bas, les besoins en compétence moindres, la gestion de flux, le plan de production et de contrôle sont assez simples et réguliers.

Les **inconvénients** des lignes de transfert sont :

- les lignes de transfert demandent de gros investissements et doivent fonctionner suffisamment longtemps pour être rentables ;
- un produit va d'un bout à l'autre de la ligne ; s'il est défectueux, cela n'est détecté qu'en bout de ligne (il est très difficile de rompre le cycle d'une ligne de transfert pour y intervenir) ;
- l'arrêt d'une station est un problème important (plus le nombre de machines ou de stations augmente, plus la probabilité d'avoir une panne quelque part augmente), si une opération ne s'est pas terminée à temps à cause d'une panne, le produit est automatiquement défectueux ;
- le manque de flexibilité et la difficulté d'introduire de nouveaux produits (ou de changement dans la fabrication) sont des facteurs à prendre également en compte.

L'histoire des lignes de transfert est assez riche. Les premières machines transfert datent du début du XX^{e} siècle. Dans les figures suivantes, nous montrons trois exemples de lignes de transfert : une machine de transfert des années 60 dans la Figure 1-5 (Pruvost, 1997), une ligne d'usinage de blocs moteurs moderne dans la Figure 1-6 (www.samovie.com) et une autre ligne de ce type dans la Figure 1-7 (Quantinet, 2004). Les lignes modernes sont devenues plus complexes et plus performantes. Leur coût a également augmenté.

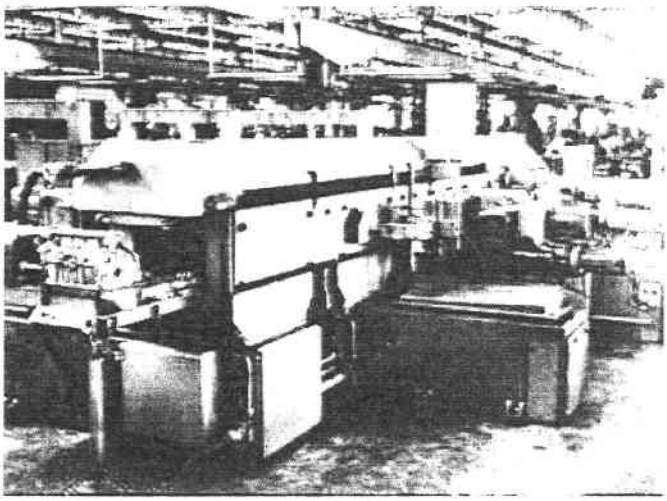

Figure 1-5 : Machine-transfert de chez Renault (1961)

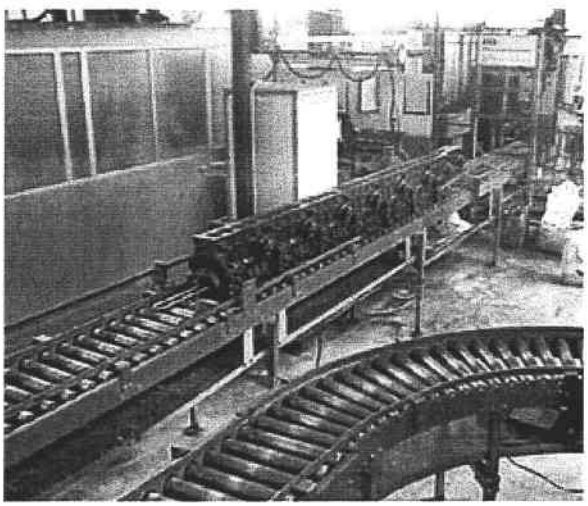

Figure 1-6 : Ligne d'usinage de blocs moteurs

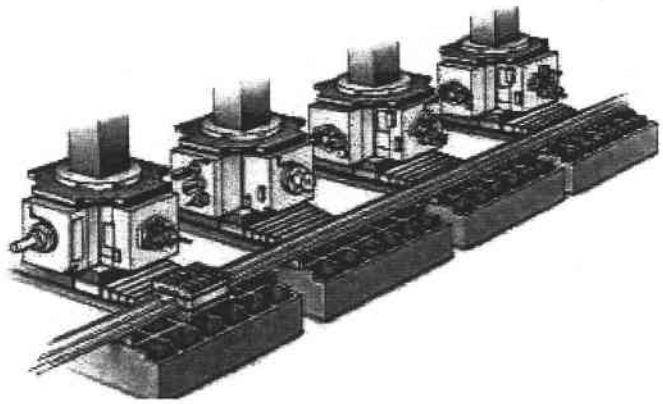

Figure 1-7 : Exemple de ligne de transfert

Les pays utilisant et produisant des machines-outils et des lignes de transfert sont ceux ayant une production manufacturière importante. Les lignes de transfert représentent un grand intérêt dans l'industrie automobile et chez les équipementiers automobiles. Les autres secteurs où les lignes de transfert sont largement utilisées sont ceux produisant des biens de grande consommation, entre autres les cycles, les motocyclettes, les équipements militaires, les machines à laver, à coudre, les lave-vaisselle. Nous les retrouvons aussi dans la fabrication de bouteilles en plastique ou d'emballage, etc.

1.4 Conception des machines outils et des lignes de transfert

De manière générale, il existe trois critères d'optimisation (Hitomi, 1995) en conception de machines-outils :

- le taux de production maximum (temps minimum) ;
- le coût minimum ;
- le taux de profit maximum (maximisation du profit dans un intervalle de temps donné).

La durée de production d'une unité de produit t (pour une seule machine elle est égale au temps de cycle T) est la somme du temps de préparation t_p (comprenant le temps de chargement et de déchargement de la pièce), du temps d'exécution t_e et du temps de remplacement de l'outil t_r :

$$T = t = t_p + t_e + t_r. \tag{1-1}$$

Le temps de remplacement de l'outil t_r est souvent prévu aux pauses, alors il est négligé pour les calculs. Dans certains cas, nous pouvons intégrer le temps de préparation au temps d'exécution, nous ne parlons alors que du temps d'exécution.

Le taux de production ρ est le nombre de pièces produites par unité de temps et est l'inverse de la durée de production d'une unité de produit, pour une seule machine-outil il est égal à :

$$\rho = 1/T = 1/t. \tag{1-2}$$

Le coût de production d'une unité de produit comprend le coût d'amortissement de la ressource (machine) u_r, le coût de production u_p et le coût de l'outil u_o (comprenant le coût d'achat, le coût de l'usure et d'amortissement et les coûts dérivés) :

$$u = u_r + u_p + u_o. \tag{1-3}$$

Le profit par unité de produit g est égal au prix de vente r_u moins le coût de la matière première m_c moins le coût de production u.

$$g = r_u - m_c - u. \tag{1-4}$$

Le taux de profit est calculé en utilisant la formule suivante :

$$pf = g / T. \tag{1-5}$$

Quelquefois, les deux premiers critères (le taux de production maximum et le coût minimum) sont utilisés en même temps. Dans notre étude, nous utilisons uniquement le deuxième critère, en ayant le taux de production comme contrainte. Lors de la conception préliminaire, nous pouvons retenir uniquement la composante u_r de ce coût, et partiellement la composante u_o (1-3) en les estimant par le nombre de têtes d'usinage.

Prenons maintenant une ligne de transfert avec plusieurs stations de travail, nous allons montrer de quoi dépendent son temps de cycle (cadencement) et son coût.

Au début de chaque cycle, chaque station est munie automatiquement d'une pièce qui vient de la station précédente.

Durant le cycle, chaque station doit effectuer un ensemble d'opérations données, et quand c'est fait, la pièce est transférée à la station suivante, le cycle suivant commence. Le produit fini est déchargé à la fin de la ligne. Si, pour fabriquer un produit, il faut faire l'ensemble **N** des opérations, alors la ligne correspondante peut contenir jusqu'à $m = |\mathbf{N}|$ stations (une opération par station au pire des cas). Bien sûr, nous pouvons avoir moins de m stations si nous regroupons certaines opérations. S'il y a m stations, la matière première de coût m_c passe à travers ces m stations pour donner un produit fini ayant un prix de vente r_u (voir Figure 1-8).

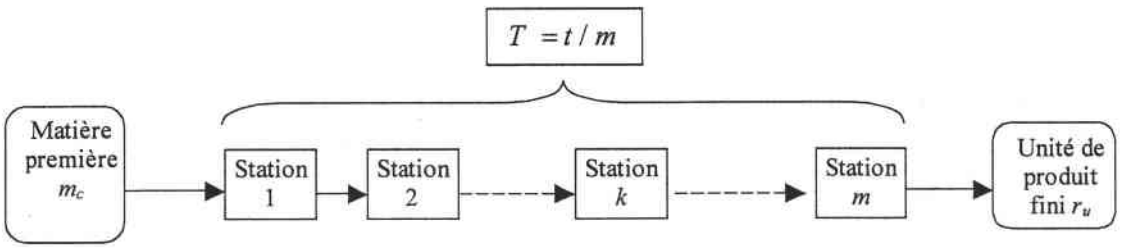

Figure 1-8 : Schéma de flux dans une ligne de transfert

Nous montrons maintenant certaines propriétés et caractéristiques des lignes de transfert simples utilisant un seul outil par tête d'usinage et des opérations en séquence.

Le chargement et déchargement des pièces et leurs passages sur les différentes stations de travail se fait automatiquement en fin de cycle. Le temps d'usinage dépend d'une constante d'usinage α_j ; cette dernière dépend de l'outil utilisé (voir Figure 1-9, Hitomi, 1995) :

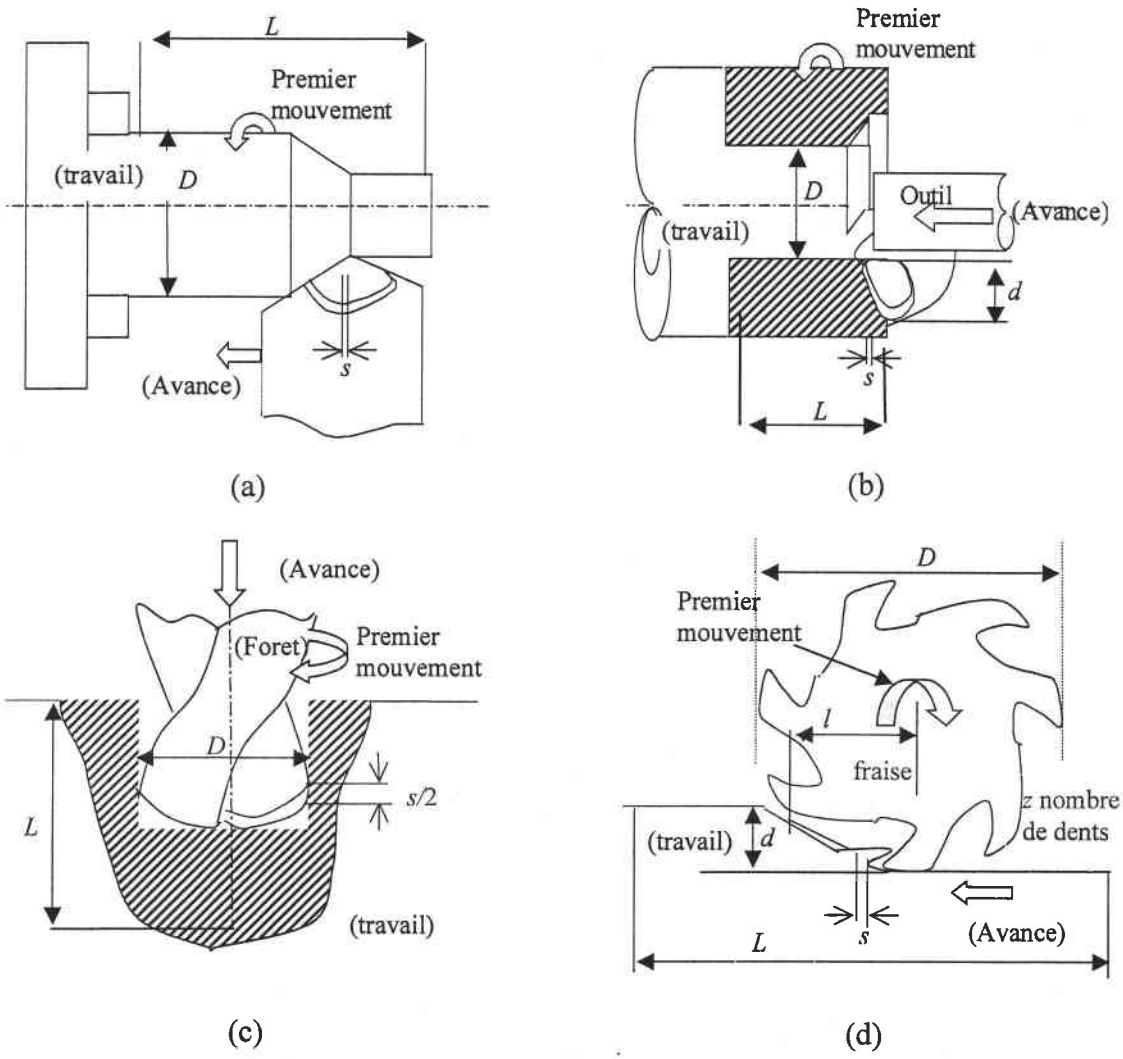

Figure 1-9 : Variables pour (a) le tournage, (b) le calibrage, (c) le perçage, (d) le fraisage

$$\alpha_j = \begin{cases} \pi D_j L_j / s_j & \text{pour le tournage, forage et alésage} \\ \pi D_j \left(l_j + L_j\right) / s_j Z_j & \text{pour le fraisage} \quad (j = 1, 2, \ldots, n) \end{cases} \qquad (1\text{-}6)$$

où (ici, nous avons gardé la notation de Hitomi)

D_j est le diamètre de l'outil en millimètres pour le tournage ou le calibrage et le diamètre opératoire pour le fraisage, le forage ou l'alésage,

L_j est la longueur de l'outil en millimètres pour le tournage ou le calibrage et la profondeur du trou en millimètres à atteindre pour le fraisage, le forage ou l'alésage,

l_j est ici une longueur additionnelle pour le fraisage, s_j est ici le taux de pénétration par dent pour le fraisage ou l'avance par tour (mm/tour) pour les autres opérations,

z_j est le nombre de dents de l'outil de fraisage j, j =1,2, .., m,

n est ici le nombre d'opérations.

Si a est le temps de mise en route de l'outil, v_j la vitesse d'usinage par minute pour l'outil utilisé, le temps opératoire t_j est :

$$t_j(v_j) = a + \alpha_j / v_j, \qquad j = 1, 2, \ldots, n. \tag{1-7}$$

Notons (par hypothèse) que les changements d'outils doivent aussi être inclus dans ce temps.

Le temps total minimum pour transformer un produit (de son chargement initial à son déchargement en fin de ligne), si toutes les opérations se font en séquence et s'il n'y a pas de temps mort sur les stations, est la somme des temps opératoires :

$$t(v) = \sum_{j=1}^{n} t_j(v_j) = an + \sum_{j=1}^{n} \alpha_j / v_j . \tag{1-8}$$

La borne inférieure $LB(T)$ du temps de cycle (du cadencement) est donc égale à :

$$LB(T) = t(v)/m, \tag{1-9}$$

où m est le nombre de stations.

Le temps de cycle réel T est toujours supérieur ou égal à la borne inférieure $T \geq LB(T)$.

Au cas où il n'y ait qu'une opération par station, c'est-à-dire que $m = n$, le temps de cycle de la ligne est égal à :

$$T = \max(t_j(v_j)) \geq LB(T) = t(v)/n. \tag{1-10}$$

Soit le temps de cycle objectif T_0 est un cadencement (productivité) voulue. D'habitude, à l'étape de la conception préliminaire de la ligne on essaie de minimiser le nombre de

stations, c'est-à-dire le coût de la ligne, en regroupant les opérations. Mais ce regroupement doit être fait de sorte que T reste inférieur ou égal à T_0, dans le cas contraire, la productivité voulue ne sera pas assurée.

Dans la suite de notre travail, nous supposons que la durée t_i d'une opération i tient déjà compte de toutes les composantes ci-dessus ; elle est la même quelle que soit la tête d'usinage qui la réalise. Nous ajoutons un temps auxiliaire par tête d'usinage τ^b et par machine t^s (pour tenir compte des autres temps).

1.5 Objectif de la thèse : optimisation des lignes de transfert

Une ligne de transfert (Askin et Standridge, 1993 ; Hitomi, 1995 ; McDonnel et *al.*, 2000) est un ensemble de machines (stations) en série pour fabriquer un type de produit ou une famille de produits proches. Chaque station de la ligne effectue un ensemble fixe d'opérations sur chaque pièce. Cet ensemble (cycle de la station) est défini, pour chaque station, à l'étape de la conception de la ligne. Le temps nécessaire pour réaliser les opérations du cycle s'appelle le temps de cycle de la station.

La station avec le temps de cycle le plus grand définit le **temps de cycle T de la ligne**, en d'autres mots, elle impose la cadence (cadencement) correspondante. A la fin de chaque cycle de la ligne, le système automatisé transporte chaque unité de produit de la station courante à la station suivante, pour toutes les stations en même temps ou par déplacement asynchrone avec attente au poste suivant.

Nous traitons dans ce mémoire de thèse le problème des lignes de transfert sans stocks tampons. En ce qui concerne le dimensionnement des stocks tampons voir par exemple (Buzacott et Hanifin,1978 ; Dallery et *al.*, 1988).

Les lignes de transfert sont chères, car les frais d'investissement concernant les stations et l'équipement de transfert sont très élevés. Le coût de revient des produits en dépend en

grande partie. Il est donc très important d'essayer de minimiser les coûts de telles lignes lors de leur conception.

Pratiquement tous les coûts d'une ligne sont définis à l'étape de sa conception. Une fois que la ligne est conçue, il est difficile de changer quoique ce soit, et si une erreur est faite, vue l'effet d'échelle (le nombre de pièces fabriquées par an multiplié par le nombre d'années de fonctionnement), elle représentera des pertes importantes. Inversement, un gain de coût même léger à l'étape de la conception permet d'obtenir des bénéfices importants sur le temps de cycle de vie de la ligne ; c'est pourquoi il est nécessaire de faire une analyse approfondie des solutions possibles pour trouver une bonne solution voire la meilleure.

Le processus de conception d'une ligne de transfert est assez long et complexe ; il est composé en étapes et sous étapes. Nous consacrons cette étude à *l'étape de leur conception préliminaire* (Proth, 1992). A ce stade, le concepteur connaît le produit à fabriquer, toutes les opérations à effectuer pour le faire et le volume annuel de fabrication pour lequel la ligne est prévue. Il doit prendre des décisions de base concernant la structuration de la ligne, à savoir : le nombre de stations de travail, la répartition préliminaire des opérations sur ces stations et la quantité et le type de têtes d'usinages à utiliser sur chaque station. Ces décisions seront détaillées, précisées et complétées à l'étape de la conception détaillée.

La différence entre l'étape de *conception préliminaire* et de *conception détaillée* réside dans le fait qu'à l'étape de la conception préliminaire l'objectif est de faire concorder la demande du marché (volume de production) avec une capacité de production dépendante de la structure de la ligne.

Par contre, l'étape de conception détaillée en se basant sur des résultats de la conception préliminaire, fait appel à des calculs complémentaires nécessaires à la conception physique de la ligne (choix des modes et des paramètres d'usinage, conception des têtes d'usinage en tenant compte des modes de fixation et accessibilités de pièces, conception des stations à partir des têtes obtenues, etc.) ; elle précise, détaille et met en oeuvre les décisions prises à l'étape de la conception préliminaire. Bien sûr, il est parfois nécessaire à l'étape de la

conception détaillée de revenir sur les décisions de la conception préliminaire pour non-respect de certaines contraintes. Nous sommes donc en présence d'un processus itératif.

L'étape de la **conception préliminaire** nécessite des outils interactifs d'aide à la décision, avec lesquels le concepteur puisse tester ses idées et chercher des solutions optimisées avant de prendre des décisions concernant la structuration de la ligne.

Dans cette thèse, nous développons une approche de ce type et nous proposons un ensemble de méthodes permettant d'aider le concepteur dans son choix.

Nous nous basons sur l'idée que les contraintes et les valeurs des paramètres de notre modèle sont données par le concepteur. Ces contraintes et paramètres peuvent avoir comme origine des contraintes technologiques réelles, connues par le concepteur, mais également des contraintes qui n'existent pas physiquement mais qu'il faut introduire d'après le concepteur pour aller vers des solutions qui, d'après lui, sont meilleures. Il s'agit alors des préférences du concepteur. Le modèle vérifie toutes les contraintes, trouve une solution optimale en tenant compte de ces contraintes et la propose au concepteur. Le concepteur examine la solution, s'il n'est pas satisfait, il change des paramètres et des contraintes et il relance le modèle. Cette procédure dure tant qu'une solution acceptable par le concepteur n'est pas trouvée (voir Figure 1-10).

Le choix du critère d'optimisation est basé sur les réflexions suivantes : chaque station et chaque tête d'usinage supplémentaire engendrent des coûts fixes importants (bâtis, tables et porte-pièces, porte-outils, actionneurs, asservisseurs, etc.), à ces coûts de base s'ajoutent des coûts dépendant de type de tête d'usinage et de stations, c'est-à-dire des coûts liés concrètement avec des combinaisons d'opérations affectées aux têtes et aux stations.

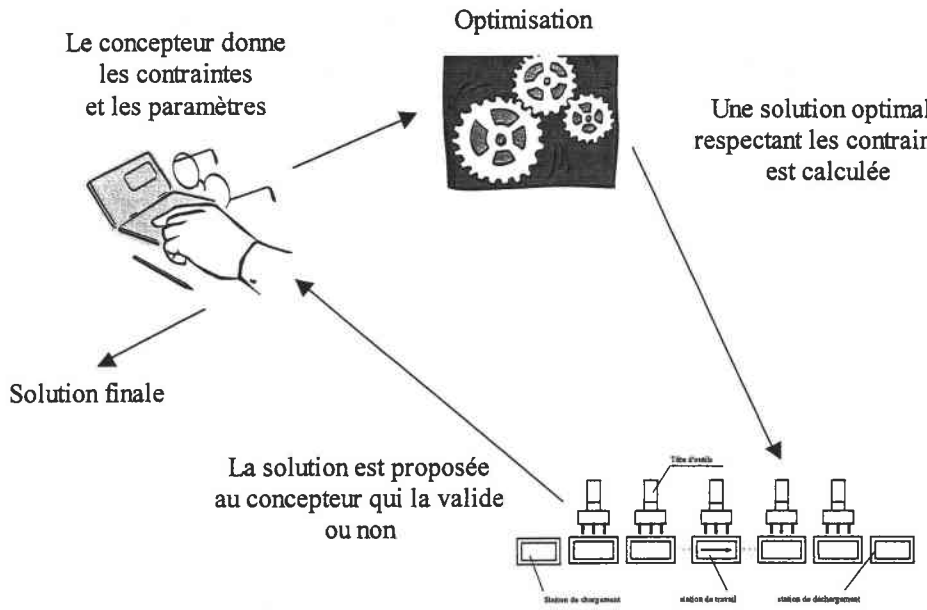

Figure 1-10 : Schéma de l'approche

A l'étape de la conception préliminaire il est difficile, voire impossible, de tenir compte de tous les coûts, certains d'entre eux sont liés aux décisions qui sont prises ultérieurement. En même temps, nous savons que les coûts fixes d'une station et d'une tête d'usinage sont souvent largement supérieurs aux autres coûts. Pour pouvoir aborder le problème de la conception préliminaire de manière efficace, nous avons donc opté pour un critère permettant de minimiser le nombre de stations de travail et le nombre de têtes d'usinage dans une telle ligne. Ce sera notre fonction objectif. La prise en compte des autres coûts sera faite ultérieurement mais aussi à travers des contraintes de nos modèles et une possibilité d'une analyse itérative.

La réduction du nombre de machines et de têtes d'usinage permettra de réduire le coût de la ligne et l'espace occupé. Cette réduction est possible si nous arrivons à regrouper les opérations de sorte qu'elles se passent en parallèle autant que faire se peut. Pour cela, nous utilisons la possibilité d'avoir des têtes d'usinages à outils multiples. Une telle tête permet de faire autant d'opérations que le nombre d'outils qu'elle possède, même plus, car nous pouvons avoir des outils composés (chaque outil composé fait plusieurs opérations en même temps).

Nous appelons bloc d'opérations toutes les opérations affectées à la même tête d'usinage (Dolgui *et al.* 1999b ; Dolgui *et al.* 2001b ; Dolgui *et al.* 2001b ; Dolgui et Finel 2002). Alors il est évident que plus nous pouvons mettre d'opérations par tête d'usinage, moins nous aurons de têtes d'usinage et de stations de travail ; en plus de la réduction du coût de la ligne, cette mise en parallèle des opérations nous permet d'assurer la productivité avec un nombre de stations limité : pour une illustration d'une telle ligne voir la Figure 1-11 (Martin, 2003).

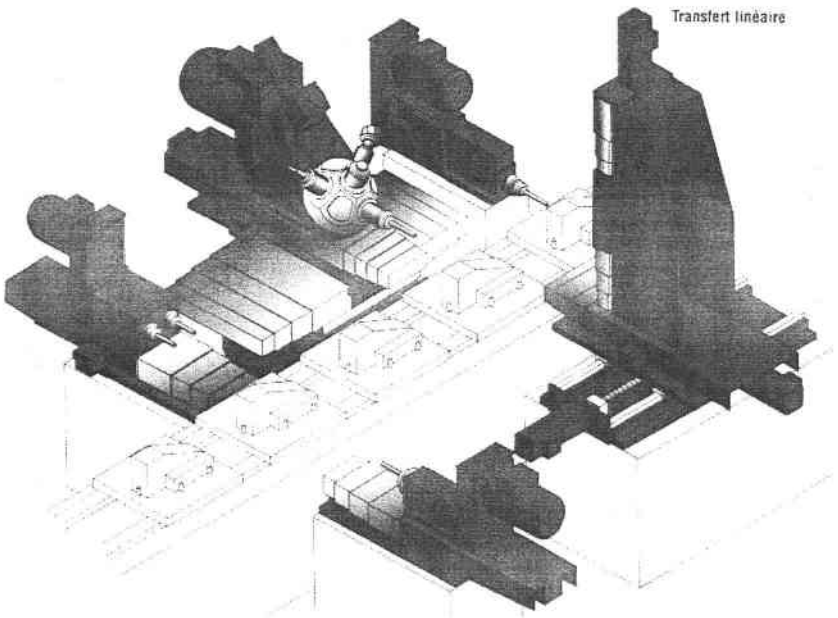

Figure 1-11 : Un exemple de ligne de transfert avec têtes d'usinage multiples
(documentation BSI)

Notons également que pour atteindre un certain niveau de tolérance, il est indispensable d'exécuter certaines opérations simultanément sans changement de position des outils et de la pièce (par la même tête usinage). La nécessité de regrouper des opérations en bloc ne vient donc pas seulement de notre objectif de réduction du coût, mais aussi d'un objectif de qualité du produit fabriqué.

A l'étape de la conception préliminaire, nous avons besoin de trouver une solution permettant de produire le volume voulu tout en respectant les contraintes technologiques et en minimisant le coût de production. Comme nous l'avons précisé, nous n'avons pas toute l'information nécessaire pour calculer exactement tous les éléments du coût de la ligne. Nous cherchons donc des paramètres clés permettant d'évaluer le plus exactement possible ce coût.

Une analyse des cas réels montre que le coût fixe de la ligne couvrant l'investissement pour la création des stations et des têtes d'usinage est structurant. Cela signifie que le coût d'une telle ligne est défini principalement *à partir du nombre de stations et du nombre de têtes d'usinage* utilisées. Une répartition concrète des opérations par stations et têtes d'usinage change bien sûr le coût de chaque station et tête, mais n'influe pas sensiblement sur les coûts relatifs de différentes décisions de conception. Nous pouvons donc utiliser, à l'étape de la conception préliminaire, la somme pondérée du nombre de stations et de blocs en tant que critère d'optimisation.

Nous introduisons deux paramètres C_1 et C_2 que nous appelons respectivement le *coût relatif d'une station* et le *coût relatif d'une tête d'usinage*. Nous partons de l'idée que le concepteur doit pouvoir changer les paramètres C_1 et C_2, en préférant la réduction du nombre de stations par rapport au nombre de têtes d'usinage ou inversement, du nombre de stations par rapport au nombre de têtes d'usinage pour chaque cas concret. Pour donner des valeurs initiales à ses paramètres, il peut se référer aux coûts de la partie fixe d'une tête d'usinage (porte outil, ...) et d'une station (bâti, ...) respectivement (qui ne dépend pas des opérations affectées) et/ou se baser sur son expérience dans le domaine. Dans la pratique, le rapport C_2/C_1 varie entre 0,2 et 0,6.

Les lignes étudiées sont conçues pour un volume de production annuel fixé. A partir de ce volume nous pouvons déduire le temps de cycle objectif T_0. Le temps de cycle réel dépend du nombre de têtes d'usinages, de l'ensemble des opérations à effectuer par chaque tête d'usinage, ainsi que du nombre de stations et de l'ensemble des têtes d'usinage affectés à chaque station (Figure 1-11).

Sur les lignes que nous considérons, les têtes d'usinage de la même station sont exécutées en séquence, l'une après l'autre, dans un ordre donné arrêté à l'étape de la conception. Nous connaissons l'ensemble **N** de toutes les opérations à réaliser pour fabriquer un produit. Le cardinal de cet ensemble |**N**| donne leur nombre. Les séquences des opérations possibles sont délimitées par un graphe de précédence (une séquence est possible si elle respecte les contraintes données par le graphe de précédence). Nous allons utiliser une interprétation particulière du graphe de précédence : comme nous pouvons, nous l'avons dit, effectuer plusieurs opérations simultanément, si le graphe montre que l'opération *i* précède l'opération *j*, cela veut dire que *i* et *j* peuvent être faites simultanément par le même outil composé, ou que *i* peut être réalisée avant *j* mais qu'en aucun cas *j* ne peut être effectuée avant *i*.

Nous savons calculer le temps opératoire t_i pour chaque opération $i \in \mathbf{N}$ réalisée séparément par son outil propre, voir la formule (1-7).

En général, la technique pour calculer le temps du bloc $t^b(N)$, c'est-à-dire le temps nécessaire pour réaliser simultanément comme un seul bloc par la même tête d'usinage un ensemble d'opérations $N \subseteq \mathbf{N}$, dépend de la spécificité des processus d'usinage choisis, des équipements de la ligne et des modes d'usinage, comme nous l'avons vu précédemment. Pour notre problème, deux approches différentes pour calculer les temps opératoires $t^b(N)$ sont possibles.

Dans la première approche, nous assumons que $t^b(N)$ est déterminé par la longueur de la course de l'outil $L(N)$ de la tête d'usinage, et par son avance par minute $S(N)$, qui sont communes pour toutes les opérations du bloc. Les valeurs de $L(N)$ et $S(N)$ sont :

$$L(N)=\max\{l_j | j \in N\}, \tag{1-11}$$

$$S(N)=\min\{s_j | j \in N\}, \tag{1-12}$$

où l_j et s_j sont ici la longueur de la course de l'outil (correspondant à la constante α_j de la formule (1-6) de Hitomi) et l'avance par minute pour l'opération j si elle était exécutée séparément par sa propre tête d'usinage et son propre outil. Alors

$$t^b(N)=L(N)/S(N)+\tau^b=\max\{t_{ij}|\ i,j\in N\}, \qquad (1\text{-}13)$$

où

τ^b est une constante donnant un temps additionnel auxiliaire (le même pour tous les blocs) ; elle correspond à a dans la formule (1-7),

$t_{ij}=\max\{l_i,l_j\}\ /\ \min\{s_i,s_j\}+\tau^b$.

Exemple 1. Soit un bloc auquel sont affectées deux opérations ($N=\{i,j\}$) pour percer deux trous différents (voir Figure 1-12) avec les paramètres suivants: l_i=45 mm, s_i=150 mm/min, l_j= 60 mm et s_j= 220 mm/min. D'après les équations (1-11) et (1-12), $L(N)$= 60 mm, $S(N)$= 150 mm/min et $t^b(N)= t_{ij}$ = 0,4 +τ^b minutes. Notons que les temps opératoires pour les exécutions séparées sont 0,3+τ^b minutes et 0,27+τ^b minutes, respectivement.

La seconde approche pour déterminer le temps d'exécution d'un bloc $t^b(N)$ est plus simple. Nous supposons que :

$$t^b(N)=\max\{t_i|i\in N\}+\tau^b, \qquad (1\text{-}14)$$

où $t_i=l_i/s_i$.

La première approche pour le calcul du temps d'exécution d'un bloc est une technique de base car elle se conforme aux spécificités de la plupart des lignes de transfert. La seconde approche correspond à quelques classes spéciales de lignes de transfert, mais elle est plus simple à expliquer. Dans les deux cas : $t^b(N')\leq t^b(N'')$ si $N'\subset N''\subseteq \mathbf{N}$. Dans la suite de ce mémoire, nous utiliserons uniquement la deuxième approche juste pour simplifier la présentation des résultats.

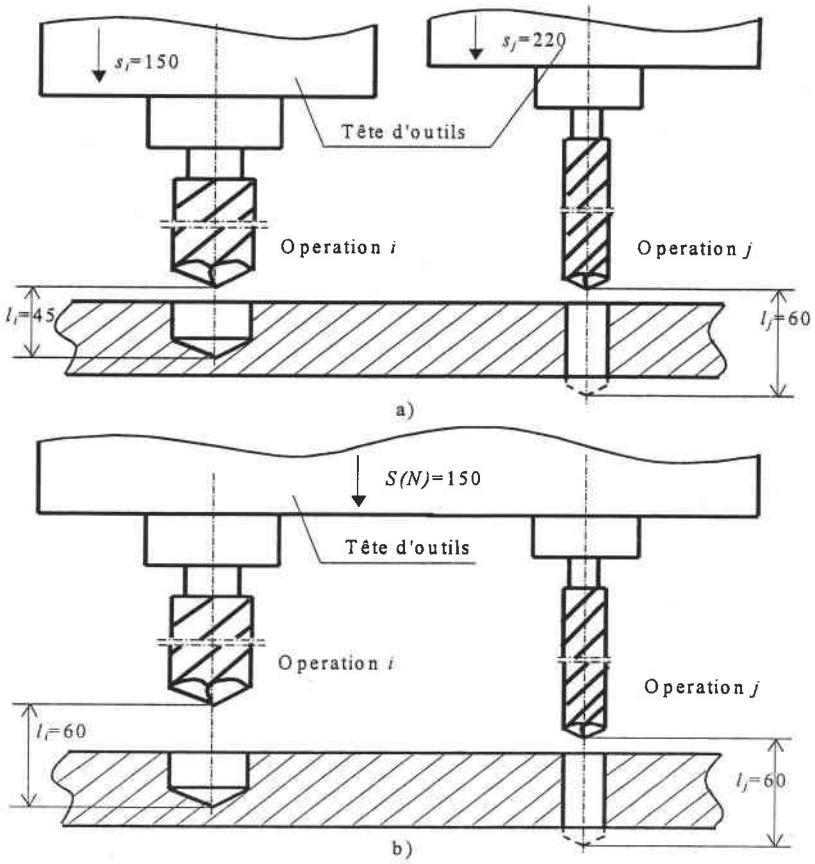

Figure 1-12 : Paramètres communs d'un bloc

a) les opérations sont exécutées individuellement;

b) les opérations sont exécutées dans le même bloc

Le temps de cycle $t^s(N_k)$ de la station k et le temps de cycle de la ligne $T(P)$ pour une décision de conception P sont calculés de la manière suivante :

$$t^s(N_k)=\sum_{l=1}^{n_k} t^b(N_{kl})+\tau^s, \tag{1-15}$$

$$T(P)=\max\{t^s(N_k)|\ 1\le k\le m\}, \tag{1-16}$$

où

τ^s est une constante qui donne un temps additionnel (correspondant au temps de fixation de chargement, déchargement etc) qui est le même pour toutes les stations (cette constante peut être omise si elle est soustraite auparavant de la valeur donnée T_0 du temps de cycle de la ligne ce qui est fait dans la suite du document),

n_k est le nombre de têtes d'usinage de la station k,

N_{kl} est l'ensemble d'opérations réalisées par la tête l de la station k,

N_k est l'ensemble d'opérations exécutées sur la station k.

En plus du graphe de précédence, nous connaissons également d'autres contraintes à prendre en compte :

1. **Contraintes d'exclusion** pour les opérations d'**une station**
2. **Contraintes d'exclusion** pour les opérations d'**une tête d'usinage**
3. **Contraintes d'inclusion** pour les opérations à affecter à **une tête d'usinage**
4. **Contraintes d'inclusion** pour les opérations à affecter à **une station**

Supposons que l'opération i produise de nombreux copeaux (tournage ou fraisage par exemple) et que l'opération j est une opération de finition, il est impossible d'envisager d'effectuer ces deux opérations sur la même station de travail : nous sommes en présence d'une *contrainte d'exclusion pour les opérations affectées à une station* (en l'occurrence les opérations i et j).

Si, par exemple, l'opération i nécessite que la pièce soit présentée dans un certain sens et que l'opération j nécessite que la pièce soit tournée de 90° par rapport à la position pour i, il sera impossible d'effectuer les opérations i et j par la même tête d'usinage. Nous sommes en présence de *contrainte d'exclusion pour les opérations d'une tête d'usinage*. Etant donné que la pièce ne change pas son positionnement et d'orientation à une station ; cela se fait durant le transport d'une station à la station suivante, cette contrainte est donc en même temps une contrainte d'exclusion pour les stations.

Il se peut aussi qu'une opération i doive être réalisée en même temps qu'une opération j (par exemple dans le cas où deux trous doivent être placés avec une haute précision l'un par rapport à l'autre). Dans ce cas l'opération i et l'opération j doivent être effectuées par la même tête d'usinage. Nous sommes en présence de contrainte *d'inclusion pour les opérations affectées à une tête d'usinage*.

Si deux opérations (ou plus) nécessitent le même environnement spécial et très coûteux (sous vide ou à une température très froide par exemple), dans ce cas il est nécessaire d'effectuer ces opérations sur la même station de travail. Nous sommes en présence de *contrainte d'inclusion pour les opérations affectées à une station.*

Le temps de cycle T_0 correspond au temps maximum que la pièce peut passer sur une station. A chaque temps de cycle, une pièce est transportée d'une station à la station suivante. Ce qui veut dire que si on peut mettre plusieurs têtes d'usinage en séquence sur une station, la somme des temps de toutes ces têtes d'usinages ne doit pas dépasser le temps de cycle T_0.

Comme nous l'avons dit précédemment ces contraintes peuvent être le reflet des contraintes physiques réelles et des règles générales engendrées par la conception de telles lignes mais également être complétées par des règles artificielles induites par l'expérience du concepteur.

La mise en place de procédures permettant de générer de telles contraintes dans un processus réel de conception de ligne est un problème très important mais indépendant de notre étude.

*Une **solution admissible** se caractérise par les propriétés suivantes : chaque opération est assignée à une et une seule station et à un et un seul bloc (tête d'usinage) ; les contraintes de précédence, d'inclusion et d'exclusion sont respectées ; le temps de chaque station ne dépasse pas le temps de cycle T_0. Une **solution optimale** est une solution admissible pour laquelle la somme pondérée du nombre de stations et de têtes d'usinage est minimale.*

Notre problème n'a pas encore été abordé dans la littérature, mais si chaque bloc d'opérations ne comprend qu'une opération et s'il n'y a pas de contraintes d'exclusion et d'inclusion, le problème considéré coïncide avec le problème classique d'équilibrage de lignes d'assemblage. Dans le chapitre suivant, nous présentons un état de l'art dans le domaine d'optimisation et d'équilibrage des lignes d'assemblage.

Chapitre 2

Conception et équilibrage des lignes de production : Etat de l'art et modélisation

2 Conception et équilibrage des lignes de production : Etat de l'art et modélisation

Le problème traité dans ce mémoire est un problème de conception de lignes d'usinage lié à celui de l'affectation des opérations à des têtes d'usinage et stations. Une fois que la décision est prise et que la conception est terminée, nous ne pouvons plus rien changer. Nous sommes donc devant un problème d'optimisation combinatoire complexe et, économiquement parlant, important. Dans ce chapitre, nous faisons l'état de l'art le concernant. Nous commençons par un bref aperçu des principales approches utilisées dans l'optimisation combinatoire de manière générale, pour ensuite passer à l'analyse des travaux en conception et équilibrage des lignes d'assemblage les utilisant. Puis nous présentons une classe des problèmes d'équilibrage des lignes d'assemblage les plus proches du problème traité. Nous montrons les difficultés propres à notre problème et pourquoi l'utilisation directe des méthodes existantes pour le résoudre est impossible. Enfin, nous terminons par donner des pistes réalisables pour trouver de nouvelles méthodes qui seront par la suite explorées dans les chapitres suivants.

2.1 Méthodes d'optimisation combinatoire

Les problèmes d'optimisation combinatoire sont caractérisés par un très grand ensemble de solutions potentielles et la fonction objectif est la fonction à minimiser ou maximiser. Les variables de décision ont un caractère combinatoire, cela veut dire que l'on cherche des combinaisons de leurs valeurs ; autrement dit, les valeurs des variables sont liées, et quand une variable de décision change de valeur, il faut en changer une ou plusieurs autres en conséquence. La complexité du problème est directement liée à un grand nombre de solutions possibles et la difficulté du problème est directement liée aux relations existant entre ses variables.

Prenons un exemple, soit (3, 2, 1, 4) l'ordre d'exécution de quatre opérations. Si l'opération 3 est déplacée à la fin de cette séquence, toutes les autres opérations seront décalées à gauche et on aura l'ordre (2, 1, 4, 3). Si x_i est la variable de décision donnant le numéro d'ordre de l'opération i, la solution initiale peut être écrite comme suit (x_1=3, x_2=2, x_3=1, x_4=4), et la solution finale sera (x_1=2, x_2=1, x_3=4, x_4=3). Le changement de la valeur de la variable x_4 de 4 en 3 a fait changer les valeurs de toutes les autres variables.

Les exemples de problèmes combinatoires sont nombreux : équilibrage de lignes de fabrication, agencement des machines et de cellules de production, séquencement et ordonnancement, problèmes de découpe et de remplissage, problèmes de transport, etc.

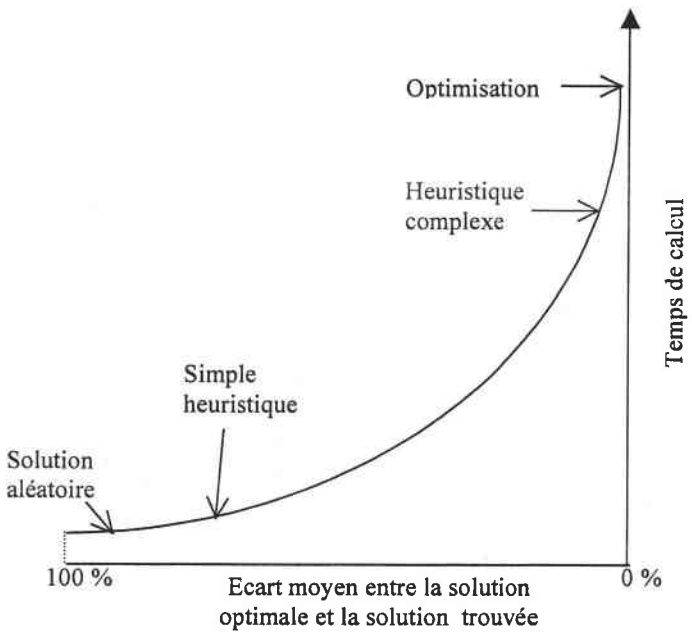

Figure 2-1: Liens entre le temps de calcul et la qualité du résultat

Nous distinguons deux classes de méthodes d'optimisation combinatoire, à savoir : les méthodes exactes et les méthodes approchées (ou heuristiques). Plus le problème est complexe plus les méthodes exactes se heurtent rapidement à l'effet d'explosion combinatoire du nombre de solutions et le temps de calcul croît de façon exponentielle. Les méthodes approchées ne garantissent pas la solution optimale, mais elles donnent en général de bonnes solutions. Un algorithme heuristique est souvent plus aisé à développer que des procédures

optimales et son temps de calcul reste acceptable. Plus les heuristiques sont élaborées et plus le temps de calcul est important, plus il y a de chances que l'optimum soit approché dans des limites acceptables.

La Figure 2-1 montre de manière schématique la dépendance entre la qualité de la solution obtenue et le temps de calcul en utilisant différentes méthodes (Askin, 1993).

Les méthodes exactes les plus connues sont celles de la programmation linéaire en variables mixtes (MIP), de la programmation dynamique et les Procédures par Séparation et Evaluation PSE (en anglais, Branch and Bound). Les méthodes approchées sont très nombreuses. Elles vont des heuristiques dédiées à des problèmes particuliers jusqu'aux méta-heuristiques qui sont des politiques généralement applicables à tous les problèmes d'optimisation combinatoire. Parmi les méta-heuristiques les plus connues nous pouvons citer les méthodes par voisinage comme la méthode du gradient, la recherche Tabou, ou le recuit simulé et les méthodes évolutionnistes comme les algorithmes génétiques.

2.1.1 Méthodes exactes

Nous pouvons utiliser les ***méthodes de programmation linéaire*** si la fonction objectif et les contraintes sont des fonctions linéaires des variables de décisions (Gondran et Minoux, 1979 ; Carlier et Chrétienne, 1988 ; Nemhauser et Wolsey, 1988 ; Williams, 1993). Les méthodes de programmation linéaire sont initialement développées pour des variables de décision réelles. La méthode la plus connue est celle du Simplex proposée par Dantzig en 1959 (Dantzig, 1990). Elle consiste à améliorer la fonction objectif initiale en explorant des solutions voisines. Toutes les contraintes connues sont représentées sous forme d'égalités ou d'inégalités. La fonction objectif correspond à la recherche d'un minimum ou d'un maximum. L'optimum est atteint quand il n'y a pas de meilleure solution voisine. Les méthodes de programmation linéaire à variables réelles sont très efficaces et rapides. Elles permettent de traiter des problèmes de très grande taille. Pour pouvoir utiliser ces méthodes pour des problèmes en variables mixtes (entières et réelles), les méthodes classiques de type Simplex

doivent être complétées par des procédures de type PSE pour trouver des valeurs optimales de variables entières. Ces méthodes sont supportées par des bibliothèques d'optimisation puissantes comme celles des logiciels ILOG Cplex, Xpress-MP, GAMS, OSL, etc. La complexité des calculs utilisant un de ces solvers réside donc principalement dans la recherche des valeurs optimales des variables de décision entières.

Une ***Procédure par Séparation et Evaluation*** (PSE) est une méthode d'énumération implicite qui évite de parcourir toutes les solutions (Carlier, 1982 ; Carlier et Pinson, 1989 ; Demeulemeester et Herroelen, 1992). L'ensemble des solutions réalisables E est divisé en sous-ensembles de solutions E_i qui sont à leur tour divisés en sous ensembles, etc., pour construire un arbre d'énumération multi-niveaux (Figure 2-2) : c'est la **séparation**. Le premier niveau est le niveau racine, il représente l'ensemble de départ. Les nœuds du niveau suivant sont créés par la division des nœuds ancêtres (niveau précédent).

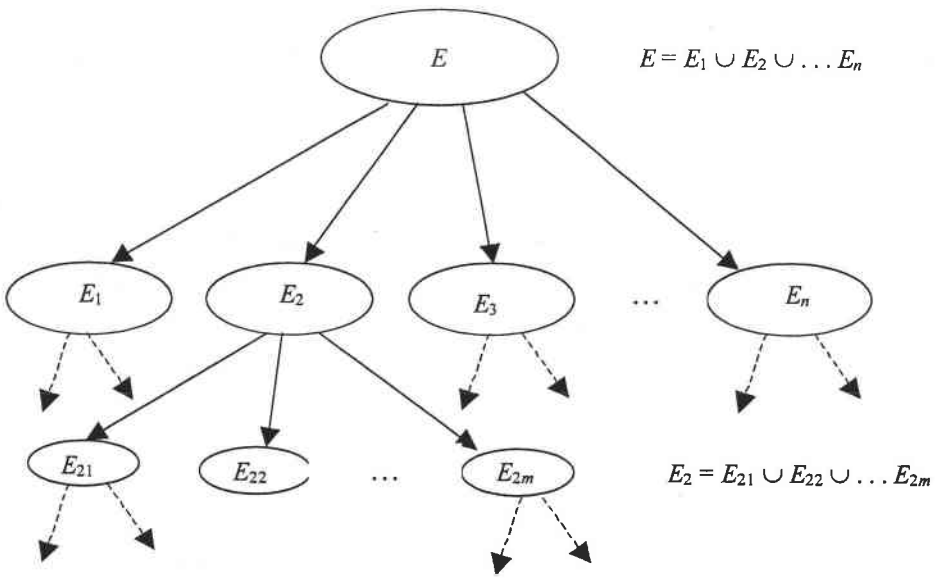

Figure 2-2 : Arbre d'énumération

Pour éviter de passer par toutes les branches, il est nécessaire de faire une **évaluation** à chaque nœud. Pour ce faire, une borne inférieure et une borne supérieure de la fonction objectif sont calculées (les solutions optimales de la fonction objectif se situent entre ces deux bornes). Dans le cas de minimisation de la fonction objectif, la borne supérieure est

généralement calculée au début de l'algorithme par une heuristique (ou une autre méthode). Si au cours du parcours de l'arbre une meilleure solution réalisable est trouvée, elle devient la nouvelle borne supérieure. La borne inférieure peut être calculée à chaque nœud en relaxant des contraintes. Si au cours du parcours de l'arbre, la borne inférieure du nœud courant (le mieux de ce qu'on espère obtenir en continuant cette branche) est supérieure à la borne supérieure, alors le nœud courant (et tous ses descendants) est supprimé. En effet, dans ce cas, la valeur de la meilleure solution finale obtenue en passant par ce niveau sera moins bonne que la meilleure solution réalisable déjà obtenue. L'arrêt des branches dans lesquelles il n'est pas possible d'obtenir une solution meilleure que celle déjà obtenue permet de ne pas parcourir tout l'arbre. L'efficacité de cette approche dépend donc de la qualité des bornes. Meilleures sont les bornes (c'est-à-dire plus elles sont proches des solutions optimales), plus vite des branches inutiles sont éliminées.

Il existe deux types de méthodes de parcours de l'arbre qui se distinguent par le principe du choix du prochain nœud à "brancher" (explorer). Les procédures par **Séparation et Evaluation Séquentielle** sont celles où les nœuds sont explorés soit en largeur d'abord (width first search), c'est-à-dire que tant que tous les nœuds d'un niveau ne sont pas parcourus, on ne passe pas au niveau suivant, ou alors en profondeur d'abord (depth first search), c'est-à-dire que le nœud choisi pour le branchement est le dernier nœud créé (voir Dolgui et Pashkevich, 2001) ; si le branchement ne peut pas être fait à partir de ce nœud Figure 2-3, alors on revient au nœud précédent (backtracking).

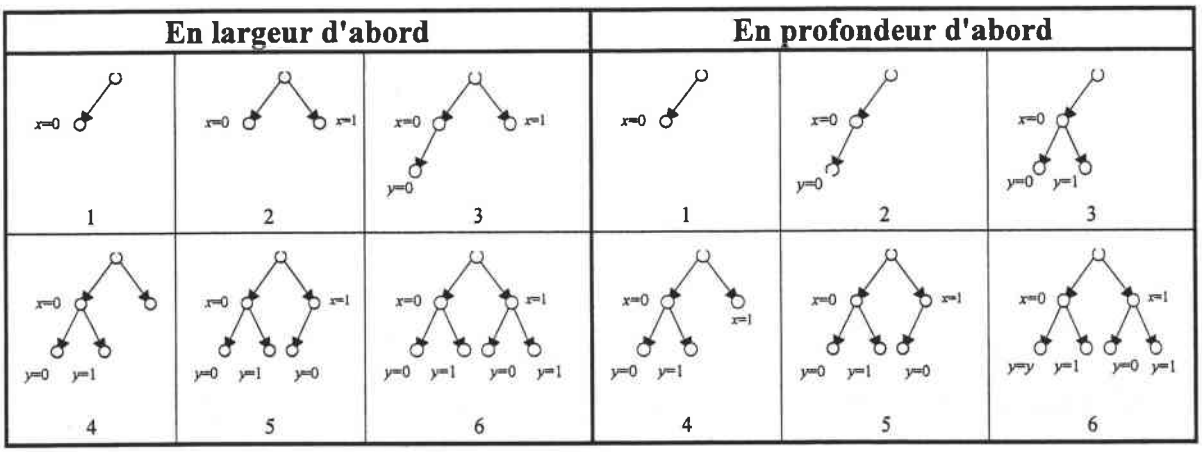

Figure 2-3 : Exploration des nœuds pour les PSE

Les procédures par **Séparation et Evaluation Progressive** ou **Mixte** sont celles où la séparation se fait d'abord par les nœuds les plus prometteurs (ayant par exemple la borne inférieure la plus faible).

La programmation dynamique n'est applicable que si le problème de départ est décomposable en phases et que la fonction objectif est additive. La valeur optimale est obtenue par l'énumération de solutions partielles complétée à chaque étape par une nouvelle décision. L'utilisation de la récurrence permet de réduire l'énumération. Le résultat donne la valeur optimale du critère. En effectuant un parcours arrière, on obtient les valeurs des variables de décision pour la solution optimale (Schrague et Baker, 1978).

L'utilisation de la **programmation par contraintes** est une autre voie prometteuse (Finel, 1991 ; Dincbas *et al.*, 2002 ; Baptiste *et al.*, 2003). Au cœur de cette approche se trouve la propagation des contraintes. Son objectif essentiel consiste à définir un problème équivalent au problème initial mais avec un espace de recherche plus restreint. La propagation des contraintes a ainsi pour but d'éliminer les valeurs des domaines de variables conduisant à une infaisabilité.

2.1.2 Méthodes approchées

Les heuristiques ou méthodes approchées ne garantissent pas que la solution obtenue soit optimale, mais de manière générale, elles donnent de bons résultats souvent plus ou moins proches de l'optimum. Elles sont utilisées pour des problèmes de grande taille ou lorsque le temps de calcul alloué est fortement limité.

Les méthodes par construction construisent progressivement une solution à partir d'algorithmes travaillant sur des listes en tenant compte de règles de priorité. Ces méthodes sont rapides mais ne permettent pas d'éviter les optimums locaux (si l'algorithme trouve un optimum local, il s'arrête et le considère comme l'optimum global).

Les méthodes par voisinage sont basées sur la génération itérative des solutions voisines. Parmi ces méthodes, le **recuit simulé** propose d'accepter une dégradation du critère en vue de pouvoir sortir d'un optimum local (Kirkpatrick *et al.*, 1983 ; Van Laarhoven *et al.*, 1992). Il choisit une solution voisine de la solution courante. Même si cette nouvelle solution dégrade le critère, elle est acceptée avec une probabilité dépendant de l'ampleur de la dégradation et d'autres paramètres. On fait décroître cette probabilité au fur et à mesure de l'avancement de l'algorithme. La **méthode Tabou** (Hertz et Widmer, 1995 ; Glover et Laguna, 1997) explore toutes les solutions voisines et choisit la meilleure. Pour éviter de visiter plusieurs fois les mêmes voisins, cette méthode gère une liste tabou qui inclut les voisins déjà visités. La qualité de la solution obtenue par ces méthodes dépend de la définition du voisinage et du réglage des paramètres utilisés.

Les algorithmes génétiques sont basés sur la notion de l'évolution (Goldberg, 1989 ; Portmann, 1996 ; Dolgui *et al.*, 2002a ; Pierreval *et al.*, 2003 ; Bloch *et al.* 2003). Ils traitent plusieurs solutions en même temps (une génération de solutions). Le passage d'une génération à la génération suivante est assuré par des techniques de recombinaison et de mutation inspirées de la biologie et du comportement des chromosomes (individus). Ici la difficulté est de bien coder les solutions, de trouver des bonnes méthodes de recombinaison, de traiter les problèmes des solutions non réalisables obtenues après les recombinaisons, et, de manière générale de bien gérer la diversification et l'optimisation locale.

Les méthodes de décomposition visent à faire face à des problèmes de grande taille en les décomposant en sous problèmes de taille réduite pouvant être plus facilement résolus (Portmann, 1988). On distingue la décomposition spatiale, la décomposition temporelle (Levy, 1996) et la décomposition paramétrique (Dolgui *et al.*, 1999a ; Dolgui *et al.*, 2003).

Les méthodes utilisant l'intelligence artificielle sont basées sur la présentation de toutes les contraintes sous forme de règles de production. Un moteur d'inférence est utilisé par la suite (Lecomte, 1993).

La simulation propose de construire un algorithme imitant le fonctionnement du système étudié sur la base d'un modèle qui peut être programmé. La simulation comporte trois étapes (Feuvrier, 1971 ; Zeigler, 1984 ; Zeigler, 2000) : la modélisation du problème, l'écriture du programme informatique et l'expérimentation avec une analyse des résultats. L'efficacité de l'analyse peut être améliorée par un couplage des modèles de simulation avec des méta-heuristiques ou méthodes de plan d'expériences (Dolgui et Ofitserov, 1997 ; Andradottir, 1998 ; Dolgui et Thirion, 1999 ; Swisher *et al.*, 2000).

2.2 Problèmes d'équilibrage des lignes d'assemblage

Le problème d'affectation des opérations aux stations de travail est connu dans la littérature sous le nom ***d'équilibrage des lignes d'assemblage***. Il consiste à affecter les opérations nécessaires pour fabriquer un produit aux stations de travail de sorte que toutes les contraintes soient respectées, que le cadencement (productivité) voulu soit assuré et que le coût de produit fini soit le plus petit possible. Ce problème apparaît à l'étape de la conception préliminaire d'une nouvelle ligne, mais également au moment de changement de produit fabriqué. Une erreur ici (une mauvaise répartition par station) peut entraîner un temps mort non justifié et des coûts supplémentaires inutiles par pièce.

Prenons un exemple impliquant 5 opérations. Dans le Tableau 2-1, nous donnons les temps et les contraintes de précédence : la première colonne représente les numéros d'opérations, la deuxième leur temps opératoire (en Unité de Temps : heures, minutes, secondes ...), la troisième montre la liste de leurs prédécesseurs directs. Nous devons répartir ces opérations par station sur une ligne de production. Supposons que le temps de cycle objectif de la ligne T_0 est égal à 4 UT, il donne le temps maximum autorisé pour chaque station de la ligne pour chaque cycle (par pièce fabriquée). Le temps mort d'une station correspond au temps de son inactivité par cycle. Par exemple, pour T_0 =4, si la somme des temps des opérations affectées à une station est égale à 3, alors le temps mort sur cette station sera de 1 UT par cycle (par pièce fabriquée).

Opération	Temps opératoire	Prédécesseurs
1	3	-
2	2	1
3	1	1
4	3	2, 3
5	2	2, 3

Tableau 2-1 : Données de l'exemple

Si les opérations du Tableau 2-1 sont placées dans l'ordre de leur numérotation, la ligne doit avoir 4 stations et le temps mort total de la ligne est de 5 UT (voir la solution 1 de la Figure 2-4). Dans la Figure 2-4, les opérations sont représentées par des rectangles gris.

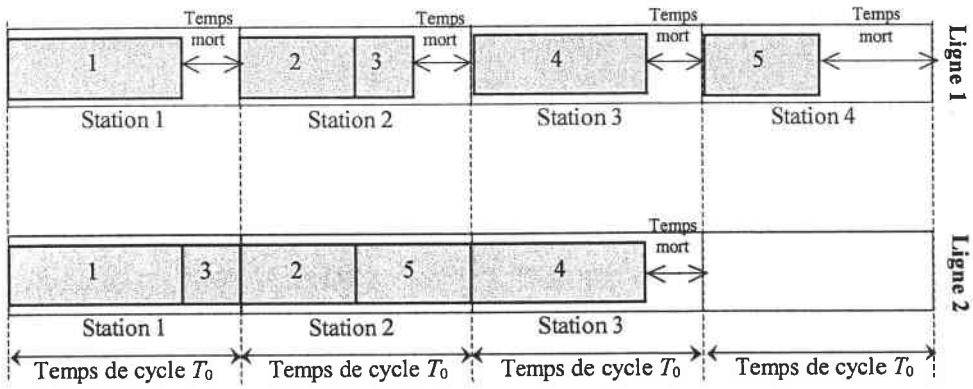

Figure 2-4 : Exemples de placement d'opérations

Mais nous pouvons trouver une affectation plus intéressante des opérations aux stations, tout en respectant les contraintes de précédence. Comme le montre la solution 2 de la Figure 2-4, la ligne peut être constituée de trois stations seulement, avec un temps mort total égal à 1 UT par cycle.

Plusieurs auteurs nous présentent leur vision de l'état de l'art du problème d'équilibrage des lignes d'assemblage (Ignall, 1965 ; Baybars, 1986 ; Ghosh et Gagnon, 1989 ; Erel et Sarin, 1998 ; Scholl, 1999 ; Rekiek *et al.*, 2002 ; Becker et Scholl, 2004 ; Scholl et Becker, 2004).

Les problèmes traités dans la littérature peuvent être classés selon différents critères illustrés par la Figure 2-5 (Scholl, 1999) : le nombre de types de différents produits traités (*simples*, c'est-à-dire mono-produit, ou *mixtes/multiples*, c'est-à-dire plusieurs types de produits différents), les données concernant les temps opératoires (déterministes, stochastiques, dynamiques), etc.

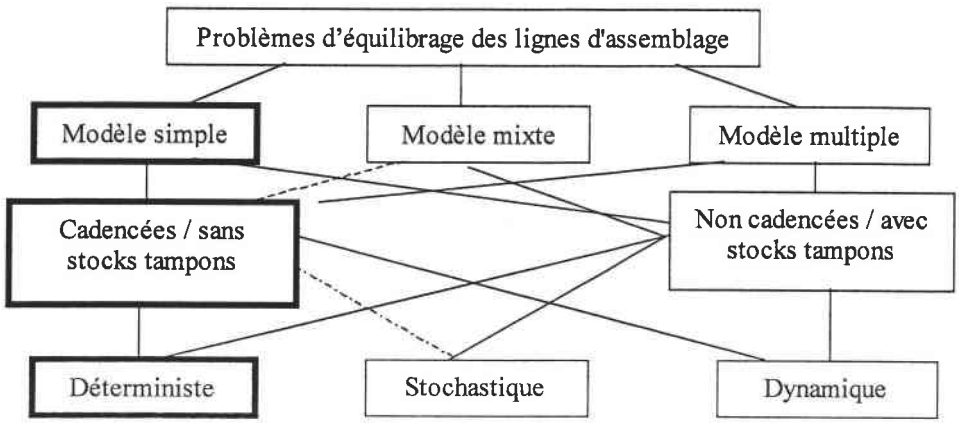

Figure 2-5 : Classification des problèmes d'équilibrage

Le modèle "simple" concerne l'équilibrage des lignes avec un seul type de produit, la plupart des travaux traitent ce type de problème. Le modèle "mixte" intègre plusieurs types de produits avec, pour chaque type, ses propres contraintes et ses propres temps opératoires. Les produits de différents types sont lancés sur la ligne en respectant des ratios donnés, mais sans regroupement obligatoire par type, c'est-à-dire qu'à chaque instant un mélange de produits de différents types est présent sur la ligne.

Le modèle "multiple" suppose qu'il y a plusieurs types de produits, mais les différents types de produits sont lancés sur la ligne par lots. Pour le modèle multiple, deux problèmes se posent en même temps, le problème d'équilibrage de la ligne pour chaque lot et le problème de séquencement des lots. Pour plus de détails sur les modèles mixtes et multiples, voir (Van Zante-de Fokkert et Kok, 1997 ; Rekiek *et al.*, 2000 ; Rekiek, 2000 ; Bukchin *et al.*, 2002 ; Boutevin, 2003).

Les variations des temps dans une ligne d'assemblage sont principalement dues à l'instabilité des performances des opérateurs humains. Pour les lignes d'assemblage manuel, il est donc parfois nécessaire de considérer les temps opératoires aléatoires ou flous. En comparaison avec des modèles déterministes, relativement peu de travaux de recherche ont été effectués pour des modèles avec des temps stochastiques ou flous (Vrat et Virani, 1976 ; Suresh et Sahu, 1994 ; Nkasu et Leung, 1995). Dans (Sotskov *et al.*, 2004), une méthode est proposée pour estimer la stabilité (robustesse) des solutions optimales d'équilibrage lorsque les temps d'opérations varient.

Le principal critère d'optimisation est la minimisation du *temps mort* (ce qui revient à la minimisation du coût des produits fabriqués). Cette approche porte le nom de l'équilibrage *orienté temps* (Time oriented). Il y a d'autres travaux où le coût n'est pas estimé à travers le temps mort, mais directement sur la base des salaires des opérateurs et des investissements (Amen, 2001). Il s'agit alors de l'équilibrage *orienté coût* (Cost oriented). Cette dernière approche est justifiée surtout pour les lignes d'assemblage manuel (dans le cas de ressources différentes pour des lignes hybrides) avec une grande dispersion de qualification et de salaire des opérateurs.

Certains auteurs étudient le problème d'équilibrage des lignes comme un problème d'optimisation multi-objectifs (Malakooti, 1994 ; McMullen et Frazier, 1998).

L'approche « equal piles » est utilisée pour traiter des lignes d'assemblage mono-produit avec un nombre de stations fixe. Il s'agit de regrouper les opérations dans des piles (stations de travail) de tailles les plus proches possibles (équilibrage de la charge au sens propre du terme). Rekiek et Delchambre (2001) proposent un algorithme génétique pour résoudre ce problème.

Par la suite, nous ne parlerons que des publications proches de notre problème, c'est-à-dire de celles qui ne considèrent qu'un seul type de produit et qui traitent les lignes cadencées sans stocks tampons avec des temps opératoires déterministes (en gras dans la Figure 2-5).

Ces hypothèses se justifient par le fait que nous nous intéressons à la conception des lignes automatisées pour la production de masse et en grandes séries de familles de produits proches.

2.3 Le problème SALB

2.3.1 Hypothèses et contraintes

Les **principales hypothèses** du problème classique d'équilibrage des lignes d'assemblage énoncées par Baybars (1986) sont :

1. Les temps opératoires sont déterministes.
2. Le splitting est interdit.
3. Il y a des contraintes de précédence.
4. Il faut réaliser toutes les opérations.
5. N'importe quelle station peut faire n'importe quelle opération.
6. Les temps opératoires ne dépendent pas de la station.
7. N'importe quelle opération peut être affectée à n'importe quelle station.
8. Les stations sont disposées en série.
9. La ligne est conçue pour un seul type de produit et elle a un seul mode de fonctionnement.

Enfin, soit

10a. Le temps de cycle T_0 est donné et fixe,

soit

10b. Le nombre de stations est donné et fixe.

Le splitting est le découpage, s'il le faut, des opérations en opérations plus petites. Pour SALB, il est interdit.

Si l'hypothèse 10a est choisie, le problème d'équilibrage des lignes d'assemblage s'appelle **SALB-1** et revient à minimiser le nombre de stations (équivalent à un temps mort

minimal). Si c'est l'hypothèse 10b qui est choisie, on parle de **SALB-2** qui a comme objectif de minimiser le temps de cycle (maximiser la productivité). A l'étape de la conception d'une ligne, on est le plus souvent dans le cas de SALB-1 (le nombre de stations n'est pas fixé a priori).

Pour SALB, **les temps opératoires** *sont considérés comme déterministes.* Notons qu'en réalité les temps opératoires peuvent varier (différente allure des opérateurs, pour les lignes d'assemblage manuel ; une pièce se présente mal et doit être replacée, etc.) ; les technologies de fabrication avancées réduisent cette variabilité par l'accroissement de l'automatisation.

Les premières contraintes à respecter sont des ***contraintes de précédence.*** L'ensemble **N** des opérations à réaliser pour fabriquer un produit et ses contraintes de précédence sont souvent représentés pour SALB par un **graphe de précédence** G = (**N**, D), où D est l'ensemble des arcs du graphe. Par exemple, la Figure 2-6 montre le graphe de précédence pour le problème dont les contraintes de précédence sont présentées dans le Tableau 2-1. Les opérations sont représentées par les sommets du graphe. Si un arc relie l'opération $i \in \mathbf{N}$ à l'opération $j \in \mathbf{N}$, c'est-à-dire $(i, j) \in D$, cela signifie que i doit être exécutée avant j, et qu'il faut l'assigner sur la même station que j ou sur une station antérieure (en amont).

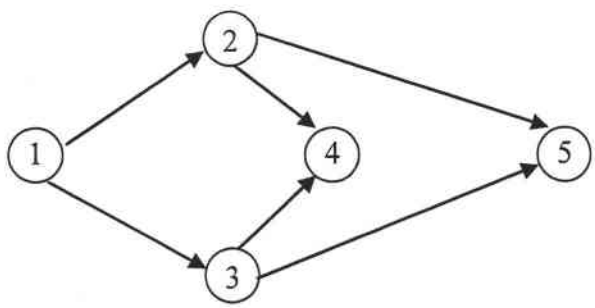

Figure 2-6 : Graphe de précédence

Le graphe de précédence $G(\mathbf{N},D)$ est donc un graphe orienté sans circuits avec un ensemble de nœuds $\mathbf{N} = \{1, 2, ..., n\}$ (les opérations) et un ensemble d'arcs $D = \{(i,j) \mid j \in \mathbf{N}$ et $i \in Pred(j)\}$ (relations de précédence entre les opérations), où $Pred(j)$ représente l'ensemble

des prédécesseurs immédiats de l'opération j. A chaque nœud j de ce graphe nous mettons en correspondance t_j qui donne le temps opératoire de l'opération j.

Le problème SALB-1 consiste en la répartition de l'ensemble de toutes les opérations $\mathbf{N} = \{1, 2, ..., n\}$ en m sous-ensembles (stations) N_k, $k = 1, 2, ..., m$. L'opération j peut être assignée dans le sous-ensemble N_h (à la station h), c'est-à-dire $j \in N_h$, si est seulement si, pour toute opération $i \in N_k$ pour laquelle $(i,j) \in D$, la condition suivante est respectée : $k \leq h$ (l'opération j est assignée à une station et une seule et tous ses prédécesseurs sont assignés à la même station ou à une station en amont).

L'affectation d'une opération à une station est également soumise à la contrainte de temps de cycle T_0. Le temps de cycle $t(N_k)$ de la station k, à laquelle l'ensemble d'opérations N_k est assigné, est égal à la somme de t_j pour tous les $j \in N_k$. L'opération i peut être ajoutée dans N_k si et seulement si le temps de cycle $t(N_k) + t_i$ reste inférieur ou égal à T_0.

$T_t = m\ T_0$ représente le temps total de séjour de chaque pièce sur la ligne. T_t ne peut pas être plus petit que la somme de tous les temps opératoires $t_{sum} = \sum_{j \in \mathbf{N}} t_j$.

Les décisions d'équilibrage sont évaluées par :

- l'efficience de la ligne $EF = t_{sum}/T_t$;
- le temps mort total par pièce $T_{mrt} = T_t - t_{sum}$;
- le pourcentage de temps mort $PR = (1 - EF)\ 100\%$.

Le temps mort de la station k est égal à :

$$T^k_{mrt} = T_0 - t(N_k). \tag{2-1}$$

Le taux d'occupation p_k de la station k est :

$$p_k = \sum_{j \in N_k} \frac{t_j}{T_0}. \tag{2-2}$$

Le temps mort total est égal à :

$$T_{mrt}=\sum_{k=1}^{m} T_{mrt}^{k} = mT_0 - \sum_{k=1}^{m} t(N_k) = mT_0 - \sum_{j\in N} t_j \,. \qquad (2\text{-}3)$$

La formule (2-3) montre que pour SALBP-1 minimiser le temps mort revient à minimiser le nombre de stations.

La solution idéale est celle où toutes les stations sont chargées de la même manière et au maximum de leur capacité. La distance à cette solution idéale peut être mesuré par :

$$Dst = \sqrt{\sum_{k=1}^{m} (T_0 - t(N_k))^2} \,. \qquad (2\text{-}4)$$

2.3.2 Bornes inférieure et supérieure

Dans la littérature, différentes *bornes* ont été trouvées pour le *nombre de stations* du problème SALB-1. Ces bornes sont souvent utilisées dans les PSE. Elles peuvent être également utiles pour accélérer des calculs ou mesurer la qualité des solutions pour d'autres méthodes.

Une borne inférieure (en anglais, Lower Bound – *LB*) pour le nombre de stations nous donne le nombre de stations minimum qu'il faut avoir dans la ligne. Nous ne sommes jamais sûrs que nous pouvons obtenir une solution faisable avec *LB* stations, mais nous savons qu'il n'est pas possible d'en avoir moins. Par ailleurs, une solution avec *LB* stations est une solution optimale. La borne la plus simple et fréquente est la suivante (Scholl, 1999) :

$$LB_m = \underline{m} = \lceil t_{sum}/T_0 \rceil = \left\lceil \sum_{j=1}^{m} p_j \right\rceil , \qquad (2\text{-}5)$$

où $\lceil x \rceil$ est la plus petite valeur entière supérieure ou égale à x.

La borne supérieure (en anglais, Upper Bound – *UB*) la plus simple est obtenue en affectant une seule opération par station. Elle est donc égale au nombre d'opérations :

$$UB_m^1 = \overline{m} = |\mathbf{N}|. \tag{2-6}$$

Une autre borne est (Hackman *et al.*, 1989 ; Scholl, 1999) :

$$UB_m^2 = \overline{m} = \lfloor (t_{sum} - 1)/(T_0 + 1 - t_{\max}) \rfloor + 1, \tag{2-7}$$

où t_{max} est le plus grand temps opératoire, $\lfloor x \rfloor$ est la partie entière de x.

D'autres bornes inférieures et supérieures pour le nombre de stations pour SALB-1 sont données dans (Scholl, 1999).

Nous pouvons également trouver des *bornes* pour les *indices de station* des opérations. La borne $k^-[i]$ donne le numéro (indice) de la station à partir de laquelle l'opération i peut être placée (station au plus tôt). La borne $k^+[i]$ nous donne l'indice de la dernière station sur laquelle peut être effectuée l'opération i (station au plus tard).

Les bornes des indices de station peuvent être déduites des relations de précédence de la manière suivante (Scholl, 1999) :

$$k^-[i] = \left\lceil \left(t_i + \sum_{h \in P_i^*} t_h \right) / T_0 \right\rceil, \text{ pour tout } i \in \mathbf{N}, \tag{2-8}$$

$$k^+[i] = m + 1 - \left\lceil \left(t_i + \sum_{h \in F_i^*} t_h \right) / T_0 \right\rceil, \text{ pour tout } i \in \mathbf{N}, \tag{2-9}$$

où P_i^* et F_i^* sont l'ensemble de tous les prédécesseurs et l'ensemble de tous les successeurs (pas seulement directs) de l'opération i, respectivement.

L'ensemble des indices des stations où l'opération i, i=1, 2, …, |**N**| peut être assignée est donc $K(i) = \{ k \mid k^-[i] \le k \le k^+[i] \}$,. C'est-à-dire que l'opération i ne peut en aucun cas être affectée à une station dont l'indice n'appartient pas à l'ensemble $K(i)$. Et inversement, pour chaque station k = 1, 2, …, m, on peut trouver l'ensemble N_k^* contenant toutes les opérations qui peuvent potentiellement être assignées à la station k : $N_k^* = \{i \mid k \in K(i)\}$.

2.3.3 Méthodes d'optimisation exactes

Programmation linéaire en nombres entiers

Un programme linéaire de base pour SALBP-1 est présenté dans (Patterson et Albracht, 1975), pour cela les auteurs introduisent les variables binaires x_{ik} suivantes :

$$x_{ik} = \left\{ \begin{array}{ll} 1 & \text{si l'}\textit{opération } i \text{ est affectée à la station } k \\ 0 & \text{sinon} \end{array} \right\}, \text{ pour } i \in \mathbf{N} \text{ et } k \in K(i), \qquad (2\text{-}10)$$

La fonction objectif s'écrit donc :

$$\text{Min } C(x) = \sum_{i \in \mathbf{N}} \sum_{k \in K(i)} k x_{ik} \qquad (2\text{-}11)$$

Sous les contraintes :

$$\sum_{k \in K(i)} x_{ik} = 1, \text{ pour tout } i \in \mathbf{N}, \qquad (2\text{-}12)$$

$$\sum_{i \in N_k^*} t_i x_{ik} \le T_0, \text{ pour } k \in K(i), \qquad (2\text{-}13)$$

$$\sum_{k=1}^{UB_m} k x_{hk} \le \sum_{k=1}^{UB_m} k x_{ik}, \text{ pour tout } (h, i) \in D, \qquad (2\text{-}14)$$

$$x_{ik} \in \{0,1\}, \text{ pour tout } i \in \mathbf{N} \text{ et pour } k = 1, 2, \ldots, UB_m. \qquad (2\text{-}15)$$

Les contraintes (2-12) assurent que chaque opération n'est affectée qu'une seule fois. Les contraintes (2-13) garantissent que le temps de cycle n'est pas dépassé. Les contraintes (2-14)

imposent les relations de précédence. Les contraintes (2-15) font que les variables de décision soient binaires.

D'autres formulations existent, ainsi Scholl (1999) a proposé un programme linéaire en variables mixtes (MIP). Il combine les idées de Bowman (1960) avec celles de Talbot et Patterson (1984). Dans le modèle de Scholl, la variable entière z_i représente le numéro de la station à laquelle l'opération i est affectée et la variable réelle y_i correspond à la date de début de l'opération i dans chaque cycle (l'intervalle de temps entre le lancement d'une pièce sur la ligne et l'instant où l'opération i commence). Les opérations sont numérotées dans l'ordre croissant de leur rang dans le graphe de précédence $G(\mathbf{N}, D)$.

Les variables binaires w_{ij} sont utilisées :

$$w_{ij}=\left\{\begin{array}{ll}1 & \text{si l'opération } i \text{ est effectuée avant l'opération } j \\ 0 & \text{sinon}\end{array}\right\}, \qquad (2\text{-}16)$$

pour $i<j$, $(i,j) \notin D$, $K(i) \cap K(j) \neq \varnothing$.

La fonction objectif est :

Min m,

et les contraintes sont :

$$y_i \geq T_0 (z_i - 1), \qquad (2\text{-}17)$$

$$y_i + t_i \leq T_0 z_i, \qquad (2\text{-}18)$$

$$(1- w_{ij})\, T_t + y_j \geq y_i + t_i, \text{ pour } i<j,\ (i,j) \notin D \text{ et } K(i) \cap K(j) \neq \varnothing, \qquad (2\text{-}19)$$

$$w_{ij}\, T_t + y_i \geq y_j + t_j, \text{ pour } i<j,\ (i,j) \notin D \text{ et } K(i) \cap K(j) \neq \varnothing, \qquad (2\text{-}20)$$

$$y_j + t_j \leq y_i, \text{ pour } (j,i) \in D \text{ et } k^{+}[j] \geq k^{-}[i], \qquad (2\text{-}21)$$

$$k^{-}[i] \leq z_i, \text{ et } z_i \leq k^{+}[i], \tag{2-22}$$

$$z_i \geq 0 \text{ entière, et } y_i \geq 0, \text{ réelle,} \tag{2-23}$$

$$w_{ij} \in \{0, 1\}, \text{ pour } i < j, (i,j) \notin D \text{ et } K(i) \cap K(j) \neq \varnothing, \tag{2-24}$$

où $T_t = m\, T_0$, et $i, j \in \mathbf{N}$.

Les contraintes (2-17) et (2-18) assurent qu'une opération est exécutée sur une et une seule station. Les contraintes (2-19) et (2-20) précisent que si deux opérations n'ont pas de relations de précédence, alors si l'une d'elle est commencée avant l'autre, elle est terminée avant l'autre. Les contraintes (2-21) assurent que les relations de précédence sont respectées. Les contraintes (2-22) restreignent les intervalles d'assignation des opérations aux stations et les contraintes (2-23) et (2-24) définissent les variables de décision.

Procédures par Séparation et Evaluation (PSE)

Plusieurs procédures PSE ont été proposées pour SALBP-1 (Scholl, 1999). Les algorithmes les plus connus sont "Fable" (Johnson, 1988), "Eureka" (Hoffmann, 1992) et "Salome" (Scholl et Klein, 1997).

Fable

Dans Fable, la construction de nouveau nœud (le branchement) se fait par ajout d'une opération candidate (qui n'a plus de prédécesseurs non assignés) à la station courante *k*, c'est-à-dire que l'énumération est **orientée opération** (opération par opération). S'il n'existe pas de telle opération, alors la charge de la station courante ne peut plus être augmentée et une nouvelle station est créée, elle devient la station courante. Une fois toutes les opérations affectées, une solution faisable est obtenue. Un retour en arrière est opéré jusqu'à trouver un branchement où il y a encore des choix d'opérations candidates.

La *règle de dominance* des nœuds utilisée par Fable identifie des solutions partielles qui contiennent le même ensemble d'opérations. Un nœud est supprimé s'il a le même nombre de stations ou plus qu'un nœud avec le même ensemble d'opérations considéré précédemment.

Pour réduire les possibilités d'énumération, Fable utilise également des règles de dominance des opérations : par exemple, si deux opérations i et j peuvent être assignées à la station courante, alors nous assignons d'abord celle qui domine l'autre. La plus connue des règles appliquées est la *règle de dominance de Jackson* (Jackson, 1956) qui dit que l'opération j domine potentiellement l'opération i si $F_i \subseteq F_j$ et $t_i \leq t_j$, où les ensembles F_i et F_j donnent tous les successeurs immédiats des opérations i et j respectivement. Pour plus de détails sur l'algorithme Fable, voir (Johnson, 1988).

Eureka

Dans Eureka, l'énumération est **orientée station**, c'est-à-dire qu'au branchement nous cherchons des combinaisons d'opérations réalisables formant une station. Nous en choisissons une et affectons toutes les opérations appartenant à cette combinaison d'un coup à la nouvelle station pour former un nouveau nœud de l'arbre de l'énumération.

Eureka applique une méthode appelée méthode de l'incrémentation de la borne inférieure. L'énumération est commencée pour trouver une solution avec un nombre de stations égal à la borne inférieure LB_m. Si la procédure ne trouve pas une telle solution, la borne est incrémentée et la procédure est relancée dès le début avec cette nouvelle borne. Un exemple est traité par la procédure Eureka dans la Figure 2-7 (Dolgui et Pashkevich, 2001).

Pour cet exemple, la borne inférieure du nombre de stations est $LB_m = \lceil 41/14 \rceil = 3$ et $T_{mrt} = 3 \cdot 14 - 41 = 1$. La branche #1 est arrêtée après St1, la branche #2 est arrêtée après St1 et St2, la branche #3 représente la solution optimale.

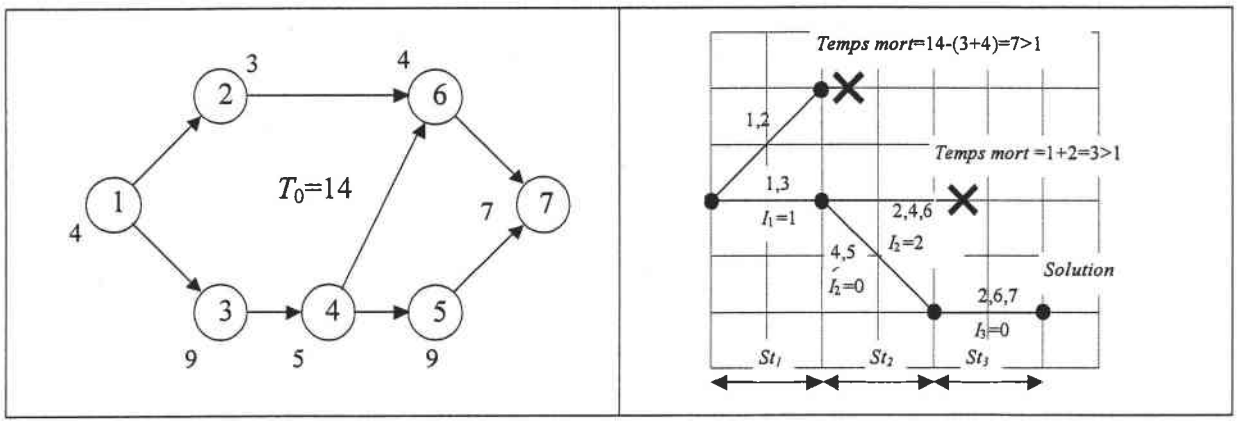

Figure 2-7 : Exemple pour l'algorithme Eureka

Salome-1

La méthode Salome-1 (**S**imple **A**ssembly **L**ine balancing **O**ptimization **M**ethod de type 1) est proposée par (Scholl et Klein, 1999), voir également (Scholl, 1999). Elle utilise une approche combinant celles d'Eureka et de Fable. A chaque nœud j, une borne inférieure locale du nombre de stations est calculée (LLB^j_m). Ensuite, le temps mort RT^j_{mrt} à ne pas dépasser pour les nœuds suivants est obtenu :

$$RT^j_{mrt}=LLB^j_m T_0 - t_{sum} - \sum_{k=1}^{j}(T_0 - t(N_k)),$$

où N_k est la charge de la station du nœud k (l'ensemble des opérations affectées à la station k).

Si RT^j_{mrt} est négatif, alors le nombre de stations égal à la borne inférieure LLB^j_m ne suffit plus. Dans ce cas, au lieu de supprimer le nœud comme dans Eureka (ne plus le considérer pour des branchements), dans Salome-1 nous rajoutons une station (comme dans Fable) et continuons les calculs. Ensuite, une fois qu'une solution faisable est construite, il y a un retour au dernier nœud où des branchements sont encore possibles.

Salome utilise également "Dynamic prefixing" (Scholl, 1999) qui est une règle exploitant directement le concept des bornes $k^+[i]$ et $k^-[i]$ pour l'indice de station (station au plus tôt et station au plus tard pour l'opération $i \in \mathbf{N}$. Avant de charger une station par énumération, cette règle affecte déjà les opérations ne pouvant pas être affectées à une station postérieure (la station courante est la station au plus tard pour ces opérations).

D'autres PSE sont présentées dans (Klein et Scholl, 1996) et (Sprecher, 1999), entre autres.

2.3.4 Méthodes approchées

Heuristiques

Talbot *et al.* (1986) puis Ponnambalam *et al.* (1999) présentent différentes heuristiques pour l'équilibrage des lignes d'assemblage. Expliquons ici brièvement quelques heuristiques les plus connues.

La méthode **RPW** (**R**anked **P**ositioned **W**eight) utilise une pondération des opérations liée à leur temps d'exécution et temps opératoires de leurs successeurs (Helgerson et Birnie, 1961). Les opérations sont choisies dans l'ordre décroissant de leur poids ; l'opération choisie est affectée à la station courante si tous ses prédécesseurs ont été déjà affectés et s'il reste encore assez de temps pour l'exécuter (le temps de l'opération est inférieur ou égal au temps restant de la station courante). S'il n'y a plus suffisamment de temps, nous passons à l'opération suivante. Si aucune opération ne peut être rajoutée sur la station, la station courante est fermée et une nouvelle station est ajoutée, elle devient courante. Cette procédure est recommencée en partant du début de la liste des opérations non allouées. Cette méthode donne de bons résultats dans des cas simples. Elle a été utilisée à plusieurs reprises dans le cadre de l'équilibrage des lignes d'assemblage (Helgerson et Birnie, 1961 ; Dar-El Mansoor, 1964 ; Boutevin, 2003).

La méthode **RRPW** (**R**eversed **R**anked **P**ositional **W**eight) propose d'inverser les contraintes de précédence, puis d'appliquer RPW. Une fois la solution trouvée, la numérotation des stations est inversée.

Kilbridge and **W**ester (1961) ont proposé une heuristique basée sur le rang d'opérations dans le graphe de précédence. La somme des temps des opérations de chaque rang est

calculée. Nous affectons à la station courante toutes les opérations du rang courant si la somme des temps ne dépasse pas la capacité restante de la station courante. Dans le cas contraire, nous n'affectons qu'un sous-ensemble d'opérations du rang dont le temps total ne dépasse pas le temps restant de la station courante. Les opérations du rang qui restent sont affectées à la nouvelle station qui devient la station courante. Le temps restant de la station courante est recalculé en tenant compte des opérations qui viennent d'être affectées. Cette méthode donne de bons résultats pour des graphes uniformes et denses. Elle est facile à implanter.

COMSOAL (COmputer Method of Sequencing Operations for Assembly Lines) est une méthode qui génère des solutions en suivant l'algorithme ci-dessous (Arcus, 1966) :

1. Créer la première station de la ligne, elle devient la station courante.
2. Le temps restant sur la station courante est égal à T_0.
3. Créer (ou mettre à jour) la liste P_l qui contient les opérations qui n'ont plus de prédécesseur non assignés. Si P_l est vide, fin d'algorithme.
4. Créer la liste F_l qui contient les opérations de la liste P_l dont le temps est inférieur ou égal au temps restant de la station courante.
5. Si la liste F_l est vide, alors créer une nouvelle station qui devient courante. Aller à 2.
6. Tirer au hasard une opération de la liste F_l et la placer sur la station courante.
7. Mettre à jour le temps restant de la station courante en le diminuant du temps de l'opération qui vient d'être assignée. Aller à 3.

Cet algorithme est répété un nombre suffisamment grand de fois (un grand nombre d'itérations). La meilleure solution de toutes les itérations est retenue. Quelques règles de pondération des opérations de la liste F_l existent permettant d'améliorer la qualité du résultat.

Nous proposons plusieurs méthodes dérivées de la méthode COMSOAL dans la suite de ce travail (voir Chapitre 4).

Méta-heuristiques

Plusieurs auteurs utilisent la recherche **Tabou** (Soriano et Gendreau, 1997 ; Wen-Chyuan Chiang, 1998). Lapierre *et al.* (2004) ont testé une modification de cette méthode sur de vrais cas industriels de lignes d'assemblage (162 opérations et 264 contraintes de précédence).

D'autres auteurs, par exemple, Jones et Beltramo (1991), Rubinovitz et Levitin (1995), Rekiek *et al.* (2000), Kim *et al.* (2001) utilisent les ***algorithmes génétiques***. Le principal avantage des algorithmes génétiques réside dans le fait qu'ils ne travaillent pas avec une seule solution mais avec toute une famille (une population) de solutions en même temps. Pourtant, pour des problèmes fort contraints, comme celui étudié dans ce mémoire, l'aspect lié avec la non faisabilité des solutions recomposées (après croisements), nécessitant leur réparation, peut peser lourdement sur les performances de ce type de techniques.

Minzu et Henrioud (1997a ; 1997b) proposent d'utiliser la méthode ***Kangourou*** pour traiter le problème d'équilibrage des lignes d'assemblage avec un nombre donné de stations. Le but est de minimiser la charge maximale de travail des stations ce qui conduit à une ligne bien équilibrée. La méthode Kangourou remplace la solution courante par une meilleure solution trouvée par un tirage aléatoire dans le voisinage. Si plus aucune solution améliorant la solution courante n'est trouvée dans son voisinage, alors une procédure « saut » est utilisée afin d'éviter de rester dans un optimum local. La stratégie adoptée est d'explorer le plus d'optimum locaux possibles dans un temps donné, ce qui permet d'augmenter de façon significative la chance d'atteindre un optimum global.

Surech et Sahu (1994) proposent une approche basée sur le **recuit simulé** qui prend en compte le problème des durées aléatoires des opérations. McMullen et Frazier (1998) utilisent le recuit simulé pour un problème d'équilibrage multiobjectif.

La remarque générale sur l'application des méta-heuristiques pour l'équilibrage des lignes d'assemblage est que, dans beaucoup de travaux, les auteurs de publications ne tiennent pas suffisamment compte de la spécificité du problème ; la recherche stochastique (Monte

Carlo) est intégrée dans la plupart des approches utilisées, pour éviter les optima locaux, mais même dans ce cas, l'obtention de l'optimum global n'est garantie que lorsque le temps de calcul tend vers l'infini.

2.4 Equilibrage orienté coût

Un nouveau problème vient d'apparaître. Il s'agit de l'équilibrage orienté coût. Amen (2000a, 2000b, 2001) propose un état de l'art, différentes formulations, des méthodes exactes et heuristiques concernant ce problème.

L'objectif d'une telle approche (CALBP : **C**ost oriented **A**ssembly **L**ine **B**alancing **P**roblem) est de ne pas se limiter à la minimisation du temps mort, mais de travailler directement sur les coûts, en essayant les intégrer dans les calculs. Cela peut être des coûts de machines et des outils, des salaires, des coûts d'installation, etc. L'hypothèse de SALBP qui dit que le coût est minimum lorsque le temps mort est minimum est abandonnée.

Les postulats de base de CALBP sont les suivants. Etant plus ou moins complexes les opérations nécessitent plus ou moins de savoir-faire. Elles demandent donc la présence d'un opérateur plus ou moins spécialisé. Cela revient donc trop cher d'employer un ouvrier très qualifié (donc plus payé) pour faire les opérations très simples. Le CALBP préconise de regrouper les opérations non plus de manière à ne pas avoir un temps mort trop important, mais de façon à ce que leurs coûts soient les plus proches possibles (coût salarial, coût d'équipement utilisé,...) : en d'autres termes, en regroupant ensemble les opérations demandant une qualification proche. L'opérateur est payé pour tout le temps de cycle indépendamment de la durée des opérations et de leur difficulté. Ainsi, avoir une station supplémentaire sur la ligne peut coûter moins cher que de rassembler sur une station des opérations aux coûts (et à la complexité) très différents.

Le coût de la main d'œuvre par une unité de produit fini est égal à la somme des salaires par unité de temps sur toutes les stations multipliée par le temps de cycle T_0. Le coût d'investissement et de l'amortissement (le capital) est proportionnel à la longueur de la ligne

en terme de nombre de stations. Tous les autres coûts (matières premières par exemple) sont indépendants de l'affectation des opérations aux stations et de la longueur de la ligne.

Le coût total par unité de produit est donc (Amen, 2000a) :

$$C = \left(\sum_{k=1}^{m} T_0 \, C_k^{sw} \right) + m C^{sc},$$

où C_k^{sw} est le coût salarial sur la station k par unité de temps, et C^{sc} est le coût de capital par cycle pour une station.

$$C_k^{sw} = \max_{i \in N_k} \left\{ C_i^{ow} \right\}, \text{ pour tout } k = 1, 2, \ldots, m,$$

où C_i^{ow} est le coût salarial pour effectuer l'opération i.

La notion du coût mort est introduite. Le coût mort par station dépend du temps mort et des différences des coûts salariaux unitaires pour les opérations assignées à la station. Il faut minimiser le coût total par produit, ce qui revient à minimiser le coût mort total.

L'indice supérieur o veut dire "opération", s "station", c "capital" et w "salaire". C'est-à-dire C_i^{ow} est le coût salarial de l'opération i, C_i^c est le coût du capital pour l'opération i, C_i^o est le coût total de l'opération i, C_k^s est le coût total de la station k par unité de temps.

Le problème peut donc être formulé en terme de programmation linéaire de la façon suivante (Amen, 2000 ; Scholl et Becker, 2003).

Les variables binaires x_{ik} sont utilisées :

$$x_{ik} = \left\{ \begin{array}{l} 1 \text{ si l'opération } i \text{ est affectée à la station } k \\ 0 \text{ sinon} \end{array} \right\}. \qquad (2\text{-}25)$$

La fonction objectif :

$$\text{Min } C=\sum_{k=1}^{m} C_k^s T_0=T_0\sum_{k=1}^{m}\left[C_k^{sw}+\left(C^{sc}/T_0\right)\right], \tag{2-26}$$

et les contraintes :

$$\sum_{k=1}^{m} x_{ik}=1 \text{ , pour } i = 1, 2, ..., |\mathbf{N}|, \tag{2-27}$$

$$\sum_{i\in\mathbf{N}} t_i x_{ik}\leq T_0 \text{ , pour } k = 1, 2, ..., m, \tag{2-28}$$

$$\sum_{k=1}^{m} k x_{jk}\leq\sum_{k=1}^{m} k x_{ik} \text{ , pour } i = 1,2, ..., |\mathbf{N}| \text{ et } j \in Pred(i), \tag{2-29}$$

$$C_k^s\geq C_i^o \cdot x_{ik} \text{ , pour } i = 1, 2, ..., |\mathbf{N}| \text{ et } k = 1, 2,.., m, \tag{2-30}$$

$$x_{ik}\in\{0,1\}, \text{ pour } i = 1, 2,..., |\mathbf{N}| \text{ et } k = 1, 2,..., m. \tag{2-31}$$

La fonction objectif minimise le coût total par unité de produit fini. Les contraintes (2-27) et (2-31) assurent que chaque opération est affectée une et une seule fois. Les contraintes (2-28) garantissent qu'il n'existe pas de stations avec un temps de travail plus grand que le temps de cycle objectif de la ligne. Les contraintes de précédence (2-29) garantissent qu'une opération ne peut pas être affectée à une station antérieure à celles de ses prédécesseurs. Les contraintes (2-30) imposent que le coût de station par unité de temps soit égal au coût par unité de temps de l'opération la plus chère affectée à cette station.

2.5 Le temps masqué

Le *temps masqué* représente des parties d'opérations qui sont exécutées en même temps que d'autres opérations appartenant à la même station. Il y a peu de travaux concernant cet aspect important d'équilibrage des lignes lié avec un chevauchement possible des opérations.

Minzu et Henrioud (1997b) proposent un algorithme général pour la détermination des stations pour SALBP. Le graphe d'assemblage combine les avantages d'un arbre d'assemblage et d'un graphe de précédence (combinaison d'informations sur les contraintes de précédences et d'information géométrique comme l'orientation des pièces). Une méthode d'allocation des opérations aux équipements basée sur ce graphe, des séquences opérationnelles pour les équipements et le calcul du temps d'exécution sur une station ont été proposés. Cet algorithme est intéressant, spécialement pour l'évaluation du temps de processus d'une station après l'équilibrage, tenant compte des temps masqués, des mouvements des opérateurs et du temps de mise en route. Des essais ont été faits sous Prolog en utilisant la programmation par contraintes mais le temps masqué n'apparaît que a posteriori, les tests effectués ne tiennent pas compte des facteurs de coûts et de faisabilité, n'ont pas été rapprochés d'une base de données d'équipements et ne tiennent pas compte réellement du format technique de la station.

2.6 Comparaison de SALBP-1 et TLBP

Dans ce mémoire, nous étudions le problème TLBP (voir Chapitre 1). C'est un nouveau problème qui n'est pas encore suffisamment étudié. Nous présentons donc une analyse comparative de SALBP-1 et TLBP. Les différences et ressemblances de ces deux classes de problèmes sont résumées dans le Tableau 2-2.

En fait, si dans un TLBP nous imposons une seule opération par bloc pour tous les blocs et qu'aucune contrainte d'inclusion ou d'exclusion n'existe, nous sommes dans le cas de SALBP-1.

	SALBP-1	TLBP
SIMILITUDES	Existence des contraintes de précédence	
	Opérations avec temps opératoires déterministes	
	Temps de cycle objectif de la ligne à respecter	
	Objectif de repartir les opérations par station	
DIFFERENCES	Contraintes de précédence : Si *i* précède *j*, alors *i* est faite avant *j*	Contraintes de précédence : Si *i* précède *j*, alors *i* peut être effectuée avant *j* ou en même temps que *j* (en parallèle)
	Toutes les opérations sont exécutées en séquence	Certaines opérations sont regroupées en bloc, toutes les opérations d'un même bloc sont exécutées simultanément (en parallèle)
	Il n'y a pas de temps masqués	Temps d'un bloc = maximum des temps des opérations du bloc
	Temps de la station= somme des temps opératoires de la station	Temps de la station = somme des temps des blocs
	Pas de contraintes d'exclusion et d'inclusion (dans le modèle de base)	Contraintes d'exclusion et d'inclusion obligeant de mettre ensemble ou séparer certaines opérations dans un bloc ou sur une station

Tableau 2-2 : Comparaisons entre SALBP-1 et TLBP

Mais dans le cas général, TLBP ne peut pas être réduit à SALBP-1. Par exemple, la propriété classique de SALBP qui dit que le minimum de temps mort est équivalent au minimum de stations (simplifiant l'application de plusieurs méthodes) n'est plus valable pour TLBP à cause de l'exécution en parallèle des opérations du même bloc.

Le regroupement des opérations en blocs peut être rapproché du problème du temps masqué dans les lignes d'assemblage, mais à notre connaissance il n'y a pratiquement pas de travaux sur SALBP avec un temps masqué.

Certaines ressemblances existent entre TLBP et CALBP. Comme dans CALBP, pour le TLBP, l'objectif n'est pas de minimiser le temps mort, mais le coût par produit fabriqué. Mais les modèles existants de CALBP sont basés sur exactement les mêmes hypothèses que SALBP ; ils ne tiennent donc pas compte de possibilité de regrouper des opérations en blocs. Les opérations sont donc considérées comme uniquement séquentielles, etc. Les différences entre SALBP et TLBP présentées dans le Tableau 2-2 sont donc valables pour CALBP et TLBP. En plus de cela, l'utilisation des méthodes de CALB s'avère difficile même pour des cas particuliers de TLBP à cause de la difficulté d'estimer les paramètres relatifs aux coûts dans ces modèles pour le cas de lignes d'usinage.

CALBP a une certaine justification pour des lignes d'assemblage manuel, où le principal coût est celui de la main d'œuvre. Le coût fixe d'une station est relativement petit par rapport au coût salarial. Dans ce cas, le principal gain se trouve dans une bonne répartition des opérations en tenant compte des qualifications des opérateurs et par conséquent de leur salaire. Les lignes de transfert sont automatisées, les opérations se font en régime automatique par des équipements très chers, les coûts fixes d'une station et d'une tête d'usinage sont donc prépondérants. Comme nous l'avons montré dans le Chapitre 1, dans la plupart de cas, la variation des coûts variables, c'est-à-dire ceux qui sont liés avec l'affectation (la combinaison) concrète des opérations aux stations et aux têtes d'usinage, peut être négligée à l'étape de la conception préliminaire. Ces coûts existent, mais pour les différentes solutions

ayant le même nombre de stations et de têtes d'usinage, ils sont approximativement les mêmes, même si leur répartition par station est différente.

Pour terminer mentionnons également des problèmes proches dans l'ordonnancement par lot (*batch scheduling*). Dans (Boudhar et Finke, 2000), le problème de minimisation du makespan pour les lignes de production avec des machines qui peuvent traiter plusieurs tâches en même temps sous forme de batchs est étudié. Les contraintes de comptabilité des tâches dans le batch sont données sous forme de graphe. Les contraintes de précédence entre les tâches sont absentes. Le temps de traitement d'un batch sur une machine est égal au temps de la tâche la plus longue. Le même problème sans contraintes de compatibilité et pour uniquement deux machines est étudié dans (Oulamar et Finke, 2001). Notre problème s'avère plus difficile que le problème de batch scheduling car nous avons des contraintes de précédence ainsi que quatre catégories de contraintes de compatibilité. Les contraintes d'exclusion et d'inclusion pour les blocs et les stations sont définies sur des ensembles d'opérations et pas pour des opérations deux à deux.

2.7 Conclusions

La plupart des travaux connus sont consacrés à l'équilibrage et à la conception des lignes d'assemblage avec des contraintes et des spécificités propres à ce type d'environnement. Notre travail se situe dans le cadre de la conception préliminaire des lignes d'usinage. C'est de la que viennent le nombre et la spécificité de contraintes et des paramètres des modèles que nous développons. L'objectif de cette thèse est de proposer des méthodes d'optimisation efficaces pour ce nouveau problème qui est l'équilibrage des lignes de transfert - **TLBP** (Transfer Line Balancing Problem).

Dans ce chapitre, nous avons présenté les approches classiques d'optimisation combinatoires qui peuvent être utiles pour le problème considéré. Puis, nous avons expliqué la spécificité de l'équilibrage des lignes d'assemblage : les objectifs, les contraintes et les méthodes utilisées. Ensuite, nous avons présenté les méthodes exactes les plus performantes, les heuristiques dédiées et les méta-heuristiques pour le problème le plus proche de TLBP, à

savoir : SALBP-1. Cette analyse nous a permis de conclure que nous ne pouvons pas utiliser directement toutes les méthodes existantes de SALBP-1, le développement de nouvelles méthodes est nécessaire.

Ce travail de thèse porte sur le développement de nouvelles méthodes traitant le TLBP pour des lignes d'usinage hautement automatisées pour lesquelles les investissements sont très importants. Une telle ligne est conçue pour une période de minimum trois à cinq ans. Si dans le cas des lignes d'assemblage manuel étudiées avec le modèle SALBP, le rééquilibrage peut être fait à n'importe quel instant (souvent au début de chaque mois), pour les lignes de transfert que nous étudions une fois que la ligne est conçue, nous ne pouvons plus rien changer (ou presque). Le coût d'une erreur est donc beaucoup plus important. Nous devons donc nous assurer de la qualité de la solution proposée et de trouver *si possible une solution optimale* ou très proche de l'optimum.

Ce constat, nous a tout d'abord conduit vers la nécessité de développer des méthodes d'*optimisation exactes*. L'existence sur le marché des bibliothèques d'optimisation puissantes comme celles de ILOG Cplex nous a encouragé à aller dans cette direction. Une attention particulière dans cette thèse est donc portée sur l'approche de programmation linaire en variables mixtes (MIP). Dans le Chapitre 3, nous proposons un modèle MIP et un ensemble de techniques améliorant ses performances. Nous montrons les caractéristiques et limites de ce modèle en utilisant des cas industriels et des exemples de test générés aléatoirement.

Pour pouvoir traiter des problèmes de grande taille, nous complétons le modèle MIP par plusieurs heuristiques présentées dans le Chapitre 4. Le choix de développer des heuristiques dédiées au lieu d'utiliser des méta-heuristiques se justifie par le caractère extrêmement contraint du problème traité et, par conséquent, par notre souci de nous tenir le plus près possible de la structure et de la spécificité de ses contraintes.

Enfin, notre souci d'utiliser autant que faire se peut des modèles exacts, nous a conduit aux couplages de MIP et des heuristiques développées. Le Chapitre 5 propose quelques approches de ce type, nous montrons les techniques permettant un couplage efficace et un certain nombre de tests ouvrant des perspectives intéressantes pour la future recherche.

Chapitre 3

Programme linéaire en variables mixtes (MIP)

pour la

conception et l'équilibrage de lignes de transfert

3 Programme linéaire en variables mixtes (MIP) pour la conception et l'équilibrage de lignes de transfert

Dans ce chapitre, nous faisons d'abord quelques rappels concernant le problème traité, puis nous illustrons les données et les contraintes ainsi que les objectifs à atteindre. Ensuite, nous présentons un modèle de programmation linaire en variables mixtes, nous détaillons toutes les techniques utilisées pour la construction du modèle et nous présentons les modifications de ce modèle qui permettent de diminuer le temps de calcul. Enfin, nous étudions les performances du modèle sur un ensemble de tests en utilisant le solver Cplex 7.0 (ILOG www.ilog.com).

3.1 Données et objectifs

3.1.1 Temps opératoires

Rappelons que sur les lignes de production étudiées, plusieurs opérations se font en parallèle par la même tête d'usinage (nous parlons alors de blocs d'opérations). Cela est possible grâce à l'utilisation de têtes d'usinage munies de plusieurs outils (simples ou composés). Chaque tête d'usinage réalise un ensemble d'opérations donné. Les têtes d'usinage de la même station sont activées séquentiellement. Les opérations assignées à chaque tête d'usinage et l'ordre d'activation des têtes pour chaque station sont fixés lors de la conception de la ligne.

Il n'y a qu'une seule pièce à chaque station. La station réalise toutes les opérations de son cycle (active séquentiellement toutes ses têtes d'usinage) sans changement de positionnement de la pièce ; ensuite la pièce est déplacée par un moyen de transport commun (convoyeur) vers la station suivante. Pendant ce déplacement, des mécanismes spéciaux

peuvent tourner la pièce pour avoir accès à de nouvelles faces, s'il le faut. La pièce est positionnée sur la nouvelle station et un nouveau cycle commence. Les débuts de cycle sur toutes les stations sont synchronisés. La ligne est complètement automatisée. Il n'y a pas de stock tampon entre les stations.

Une telle ligne de transfert est illustrée par la Figure 3-1.

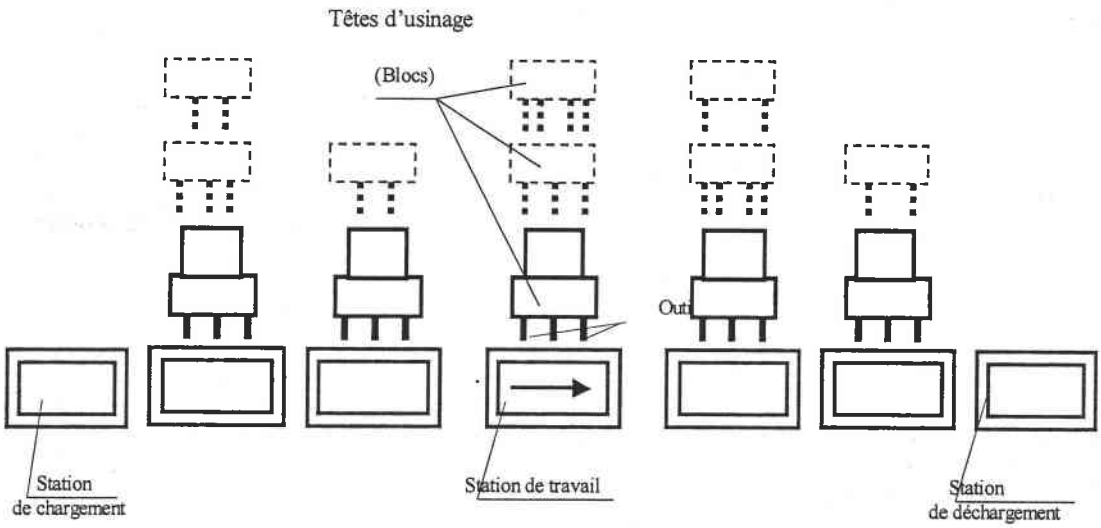

Figure 3-1 : Schéma d'une ligne de transfert

La première station est la station où le produit semi-fini est chargé. A chaque station de travail, la pièce est usinée par toutes les têtes d'usinage présentes sur la station. Chaque tête effectue une ou plusieurs opérations en parallèle (un bloc d'opérations) en fonction du nombre et du type d'outils dont elle dispose. Les têtes de la même station sont activées l'une après l'autre dans un ordre fixe défini à la conception de la ligne. A la fin du cycle, la pièce est transportée par un convoyeur vers la station suivante. Le convoyeur est commun à toutes les stations, toutes les pièces qui sont sur la ligne commencent donc leur cycle en même temps. Après avoir parcouru toutes les stations de travail de la ligne, la pièce (le produit fini) est déchargée à la dernière station. Le cycle de la ligne (soit le temps de la station la plus chargée) définit la cadence à laquelle les pièces finies sortent de la ligne (soit la productivité de la ligne).

Comme nous l'avons expliqué au Chapitre 1 (voir le paragraphe 1.5), nous mesurons le temps d'un bloc à partir du maximum des temps des opérations du bloc. En effet, la tête d'usinage correspondante ne termine son travail que lorsque toutes les opérations du bloc sont terminées. Si nous connaissons le temps d'exécution de chaque opération lorsqu'elle est réalisée toute seule par sa propre tête d'usinage, nous pouvons en déduire le temps pour une tête d'usinage réalisant plusieurs opérations simultanément en utilisant la formule suivante :

$$t^b(N) = \max \{ t_i \mid i \in N \} + \tau^b, \tag{3-1}$$

où

N est l'ensemble des opérations appartenant au bloc,

t_i est le temps de l'opération i lorsqu'elle est réalisée séparément (toute seule, c'est-à-dire sans être incluse dans un bloc avec d'autres opérations),

τ^b est une constante (un temps additionnel supposé le même pour tous les blocs).

Le temps de cycle d'une station est égal à la somme des temps de ses blocs (la somme des temps de fonctionnement des têtes d'usinage dont la station est équipée) :

$$t^s(N_k) = \sum_{l=1}^{n_k} t^b(N_{kl}) + \tau^s, \tag{3-2}$$

où

$t^s(N_k)$ est le temps de cycle de la station k,

N_k est l'ensemble des opérations réalisées sur la station k,

n_k est le nombre de têtes d'usinage (blocs d'opérations) de la station k,

N_{kl} est l'ensemble des opérations regroupées en bloc l (réalisées par la l^e tête d'usinage) de la station k,

τ^s est un temps additionnel pour les stations (le même pour toutes les stations).

Pour une décision de conception $P = \{N_1(P), N_2(P), \ldots, N_{m(P)}(P)\}$ donnant la répartition des opérations par station (c'est-à-dire définissant le nombre de stations et leur contenu), le temps de cycle de la ligne T est égal au plus grand temps de cycle des stations composant la ligne :

$$T(P) = \max\{\, t^s(N_k(P)) \mid 1 \le k \le m(P)\}, \tag{3-3}$$

où $m(P)$ est le nombre de stations pour la décision de conception P. Dans le texte qui suit, nous utiliserons souvent T, m et N_k sans P, pour simplifier la présentation.

Connaissant le volume annuel de production pour lequel la ligne est conçue et le temps d'ouverture de la ligne, tenant compte de tous les arrêts réglementaires et de la maintenance, nous calculons le temps de cycle objectif T_0. Si le temps de cycle obtenu T est supérieur au temps de cycle objectif, nous ne pouvons pas assurer la productivité voulue, c'est une contrainte de notre modèle.

C'est donc le temps de cycle de la ligne T qui nous intéresse, car il définit sa productivité. En le connaissant et en connaissant le nombre de stations m dans la ligne, nous pouvons également connaître le temps de séjour d'une pièce dans la ligne : $T_t = m\ T$. Le temps total minimum pour transformer un produit est $t(v)$ (voir le Chapitre 1). Un exemple de calcul des temps est présenté dans la Figure 3-2 où UT signifie « Unité de Temps » (une minute, une seconde, etc.).

Dans cet exemple, nous avons 8 opérations qui sont regroupées en 3 blocs repartis sur 2 stations, les temps additionnels pour les stations τ^s et pour les blocs τ^b sont respectivement 0,2 et 0,3.

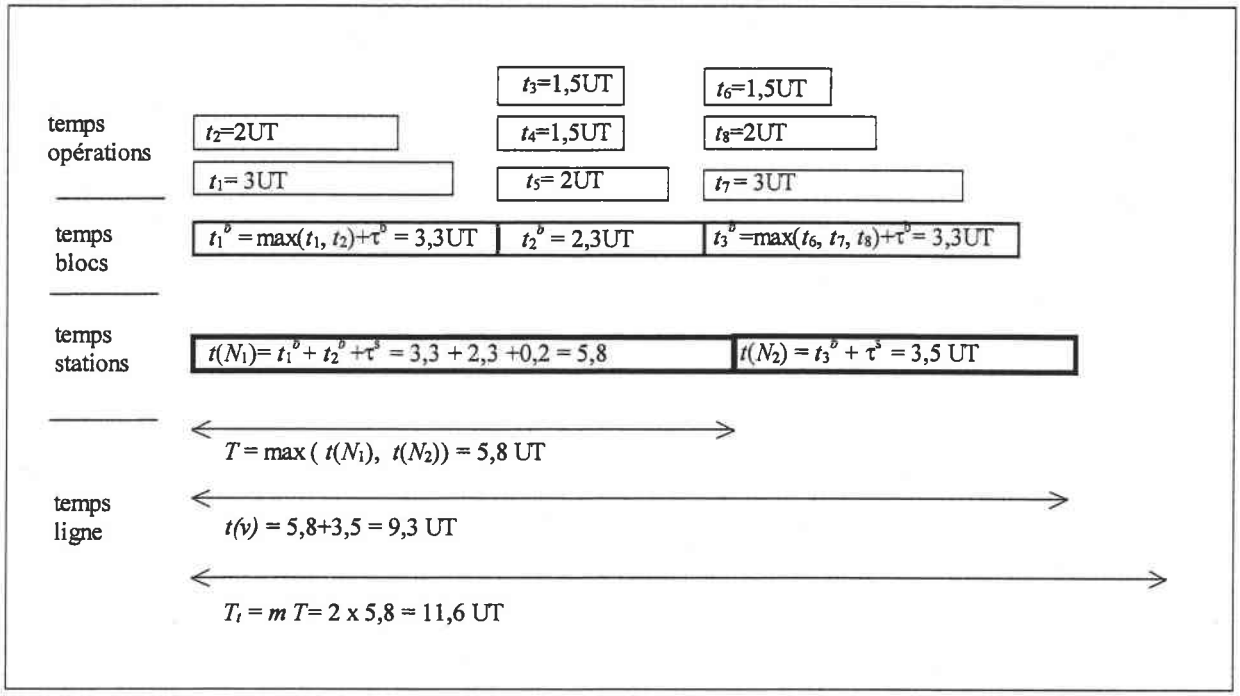

Figure 3-2 : Exemple de calcul des temps

Dans le calcul des temps, nous avons sciemment mis de côté quelques détails concernant la spécificité de différents types de processus manufacturiers et d'équipements (voir Chapitre 1). Cela permet de dégager et d'expliquer plus facilement la structure mathématique du problème. Par exemple, nous ne tenons pas compte dans les formules des détails du temps de transport d'une station à une autre ou du changement d'orientation des pièces. Nous incluons ces temps dans les autres temps du modèle et dans les constantes τ^s et τ^b. Par ailleurs, les constantes τ^s et τ^b peuvent ne pas être présentes explicitement.

3.1.2 Contraintes de précédence et de compatibilité

Description des contraintes

Comme nous l'avons expliqué dans le Chapitre 1, nous devons tenir compte de nombreuses contraintes de conception et de production. Parmi celles-ci, nous trouvons :

- Des relations d'ordre entre les opérations. Ces relations définissent des séquences réalisables d'opérations (gammes).
- La nécessité (obligation=inclusion) d'effectuer des groupes d'opérations fixes dans le même bloc ou sur la même station (à cause de la tolérance donnée, par exemple).
- L'impossibilité (interdiction=exclusion) d'effectuer certains sous-ensembles d'opérations dans le même bloc ou sur la même station (à cause de la morphologie des blocs ou à cause de l'incompatibilité des opérations à réaliser).

Comme pour SALBP, pour TLBP les *contraintes de précédence* peuvent être représentées par un graphe orienté sans circuits $G(\mathbf{N}, D)$. Mais étant donné la spécificité de l'usinage d'un certain type d'opérations, nous utilisons une interprétation particulière du graphe $G(\mathbf{N}, D)$. Dans notre cas (Dolgui *et al.* 2004), lorsqu'une opération i est reliée par un arc de précédence à une opération j, cela signifie qu'elle peut être effectuée avant j ou en même temps que j, c'est-à-dire que les deux peuvent s'exécuter en parallèle (mais en aucun cas j ne peut être exécutée avant i).

L'arc $(i, j) \in \mathbf{N}\times\mathbf{N}$ appartient à l'ensemble D si et seulement si j ne peut pas précéder l'opération i. Il s'agit donc de contraintes de précédence « non strictes » admettant l'exécution simultanée des opérations. Si nous avons besoin que l'opération i soit liée par une contrainte de précédence « stricte » à l'opération j (contrainte classique dans SALBP), il suffit de compléter la contrainte non stricte entre i et j que nous utilisons, par une contrainte interdisant l'exécution simultanée des opérations i et j dans le même bloc (voir ci-dessous).

Pour TLBP, de manière générale, les contraintes liées avec la compatibilité et l'incompatibilité des opérations et blocs sont divisées en *quatre catégories* :

1) les *contraintes d'exclusion pour le regroupement des opérations en blocs* ($\overline{EB}$), c'est-à-dire que les opérations concernées par une telle contrainte ne doivent pas être dans le même bloc ;

2) les *contraintes d'inclusion des opérations en blocs* (*EB*), obligeant les opérations concernées par une telle contrainte à être regroupées dans le même bloc ;

3) les *contraintes d'exclusion pour le regroupement des opérations dans les stations* ($\overline{ES}$), signifiant que les opérations concernées par une telle contrainte ne doivent pas être dans la même station ;

4) les *contraintes d'inclusion des opérations dans les stations ES*, imposant que les opérations concernées par une telle contrainte soient absolument effectuées dans la même station.

Nous modélisons ces contraintes par les collections de sous-ensembles d'opérations de **N** qui portent les noms correspondants, c'est-à-dire : $\overline{EB}$, *EB*, $\overline{ES}$ et *ES*. La collection $\overline{EB}$ contient les sous-ensembles d'opérations ne devant pas être ensemble dans le même bloc (si un sous ensemble de $\overline{EB}$ est constitué de plus de deux opérations, il suffit que l'une d'entre elles soit assignée à un autre bloc pour que la contrainte soit respectée). La collection *EB* contient les sous-ensembles d'opérations devant obligatoirement être regroupées dans le même bloc. La collection $\overline{ES}$ regroupe les sous-ensembles dont les opérations ne peuvent pas se faire ensemble dans la même station (si un sous-ensemble de $\overline{ES}$ est constitué de plus de deux opérations, il suffit que l'une d'entre elles soit effectuée à une autre station pour que la contrainte soit respectée). La collection *ES* regroupe les sous-ensembles dont, pour chacun d'eux, les opérations doivent être exécutées dans la même station.

Les opérations devant impérativement être exécutées dans le même bloc (formant donc un ensemble qui fait partie de *EB*) peuvent être regroupées en macro-opérations. Le temps opératoire d'une telle macro-opération est le temps le plus long des opérations la constituant. Une procédure permettant d'effectuer ce regroupement en tenant compte (et en modifiant) des contraintes de précédence est proposée dans (Dolgui *et al.*, 2000). Nous supposons que ces transformations en macro-opérations sont toutes opérées en amont de notre approche, et nous n'utilisons donc pas dans nos modèles de contraintes de type *EB*.

Notons que nous pouvons également tenir compte des contraintes liées avec des valeurs admissibles de certaines caractéristiques des têtes d'usinage (par exemple la puissance totale, la vitesse d'avance maximale, etc.). Pour les blocs, ces contraintes s'expriment de la manière suivante :

$$W(N) = \sum_{i \in N} W_{ich} \leq W_{ch},\ ch=1, 2, ..., \underline{CH}, \tag{3-4}$$

où

W_{ich} est l'apport de l'opération i à la définition de la caractéristique ch de la tête d'usinage, N est l'ensemble des opérations réalisées par la tête d'usinage, et W_{ch} est une valeur limite admissible de la caractéristique ch pour la tête d'usinage, $\underline{CH}$ est le nombre total de caractéristiques à prendre en compte. Les mêmes contraintes existent également pour les stations. Ce type de contraintes est relativement facile à formuler en les transformant en contraintes $\overline{ES}$ et $\overline{EB}$.

Un exemple industriel

Maintenant, prenons un exemple réel de pièce à usiner présentée dans la Figure 3-3 et utilisons le pour illustrer les différentes données et contraintes de notre problème (nous l'appelons l'exemple i0).

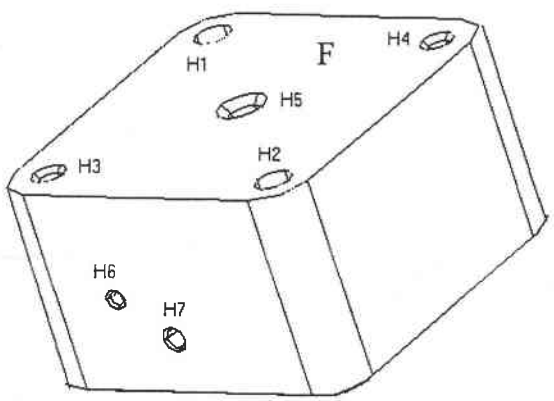

Figure 3-3 : Un exemple de pièce à usiner

Dans le Tableau 3-1, nous donnons les vingt premières opérations permettant d'usiner une partie de la pièce de la Figure 3-3. Les caractéristiques des opérations sont les suivantes : l_j est la longueur de la course de l'outil utilisée pour réaliser l'opération j, s_j est son avance par minute et t_j est son temps opératoire ($t_j = l_j / s_j$).

Nous avons comme données supplémentaires : $T_0 = 2{,}75$; $\tau^b = 0{,}3$; $\tau^s = 0{,}2$.

Les contraintes d'exclusion et d'inclusion peuvent être mieux comprises si nous regardons la nature des opérations dans le Tableau 3-1. En effet, il est impossible par exemple de concevoir une tête d'usinage réalisant du dégrossissage (opération 1) et de la finition (opération 2), les opérations 1 et 2 ne peuvent donc pas être dans le même bloc.

$\overline{ES}$ = {{2, 5}, {2, 9}, {2, 13}, {2, 17}, {3, 4}, {3, 5}, {3, 9}, {3, 13}, {3, 17}} ;

$\overline{EB}$ = {{1, 2}, {2, 3}, {6, 7}, {7, 8}, {10, 11}, {11, 12}, {13, 14}, {13, 18}, {13, 19}, {14, 15}, {14, 17}, {14, 18}, {14, 19}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {17, 18}, {18, 19}, {19,20}} ;

EB= {{16, 20}} ;

ES= {{7, 11}}.

Dans cette exemple ces contraintes de compatibilité concernent des opérations deux à deux. Notons que notre modèle sera plus général, permettant de traiter les contraintes de compatibilité représentées par des sous-ensembles de plus que 2 opérations.

Elément usiné	Opération	Numéro d'opération	Paramètres de l'opération		
			l_j	s_j	t_j
Plan F	Laminage dégrossissage	1	130	240	0,54
	Laminage finition	2	130	210	0,62
Rainure	Laminage dégrossissage	3	110	220	0,50
	Laminage finition	4	110	190	0,58
Trou H_1	Perçage	5	32	34	0,95
	Fraisage en bout	6	6	28	0,21
	Fraisage	7	28	95	0,29
	Taraudage	8	25	280	0,09
Trou H_2	Perçage	9	32	34	0,95
	Fraisage en bout	10	6	28	0,21
	Fraisage	11	28	95	0,29
	Taraudage	12	25	280	0,09
Trou H_3	Perçage	13	45	38	1,18
	Lamage	14	10	35	0,29
	Fraisage	15	38	88	0,43
	Alésage	16	35	44	0,80
Trou H_4	Perçage	17	45	38	1,18
	Lamage	18	10	35	0,29
	Fraisage	19	38	88	0,43
	Alésage	20	35	44	0,80

Tableau 3-1 : Opérations et leurs paramètres

La Figure 3-4 représente le graphe de précédence pour cet exemple.

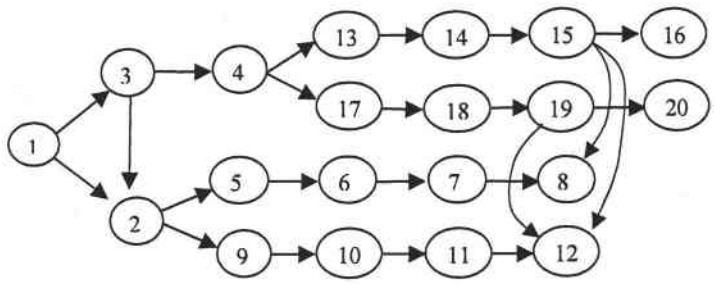

Figure 3-4 : Graphe *G* représentant les contraintes de précédence

Les contraintes d'exclusion et d'inclusion peuvent s'expliquer également en tenant compte de l'équipement nécessaire pour effectuer les opérations . Il est impossible de faire du laminage de finition de la face F et de commencer à percer un trou. L'équipement de la station nécessaire pour les effectuer est très différent ; les opérations 2 et 5 ne peuvent donc pas être effectuées sur la même station.

L'opération 16 de H3 est technologiquement identique à l'opération 20 de H4. La longueur de l'outil et sa vitesse d'avance sont les mêmes. Les résultats de ces deux opérations doivent être strictement identiques. La distance entre les axes de ces deux trous doit être réalisée avec une haute précision. Il est donc souhaitable (voire nécessaire) de les placer dans le même bloc. Par ailleurs, les trous H3 et H4 sont sur la même face et, en plus, ils sont suffisamment éloignés l'un de l'autre pour pouvoir être travaillés par une seule tête d'usinage (il n'y aura pas de conflit entre les outils). Le concepteur a donc décidé d'imposer, pour ces deux opérations, la contrainte d'inclusion : le sous-ensemble {16, 20} appartient à *EB*, c'est-à-dire que les opérations 16 et 20 doivent être faites par la même tête d'usinage, elles sont regroupées en une seule macro-opération (voir la Figure 3-5).

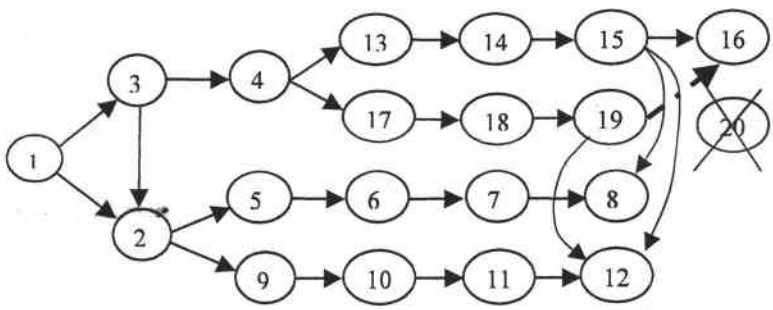

Figure 3-5 : Nouveau graphe de précédence

S'il n'y avait que les contraintes de précédence données par le graphe G ci-dessus, l'opération 3 pourrait être exécutée en même temps que l'opération 1 ou après cette opération. Pour l'opération 2, si nous ne regardions que le graphe de précédence, il y aurait trois possibilités : elle pourrait être exécutée en même temps que les opérations 1 et 3 ou en même temps qu'uniquement l'opération 3 ou encore après les opérations 1 et 3. Mais si nous regardons, en plus, les contraintes d'exclusion $\overline{EB}$, nous nous apercevons que les opérations 1 et 2 ne peuvent pas être dans le même bloc. Ces opérations ne peuvent pas être effectuées simultanément. L'opération 1 doit donc être exécutée avant l'opération 2. Selon $\overline{EB}$, les opérations 2 et 3 ne peuvent également pas être exécutées simultanément. L'opération 3 doit donc être effectuée avant l'opération 2. Nous constatons donc que les contraintes d'exclusion et d'inclusion affinent les contraintes de précédence.

3.1.3 Objectifs

En prenant en compte les *données* suivantes :

a) l'ensemble **N** de toutes les opérations à réaliser sur la ligne pour un produit fabriqué ;

b) le temps de cycle objectif de la ligne T_0 à ne pas dépasser ;

c) le nombre maximum de stations pouvant être placées sur la ligne m_0 ;

d) le nombre maximum de blocs possibles sur chaque station n_0 ;

e) les coûts relatifs d'une station C_1 et d'un bloc C_2 ;

f) les contraintes qui définissent :

   des relations de précédence entre les opérations,

   la nécessité (inclusion) et l'impossibilité (exclusion) de regrouper les opérations en blocs et de les exécuter sur la même station ;

g) les paramètres des opérations qui définissent leur temps (par exemple, la longueur de la course de l'outil et l'avance par minute) ;

h) les paramètres complémentaires pour limiter les regroupements des opérations (par exemple, la puissance, la force d'avance totale maximum, etc).

Une *décision réalisable* est celle qui respecte l'ensemble des contraintes du problème énumérées ci-dessus et qui nous donne :

- le nombre de stations m ;
- l'affectation des opérations aux m stations où l'ensemble des opérations placées sur la station k est noté N_k, k= 1, 2, ..., m ;
- le nombre n_k de blocs de chaque station k ; ce nombre correspond au nombre de têtes d'usinage dont la station k sera munie pour pouvoir effectuer l'ensemble N_k des opérations qui lui sont affectées, k= 1, 2, ..., m ; le nombre total de blocs sur la ligne est $n= \sum_{k=1}^{m} n_k$;
- la répartition des opérations de l'ensemble N_k en n_k blocs : $N_{k1}, N_{k2}, ..., N_{kn_k}$, k= 1, 2, ..., m.

Si $P= \{\{N_{11}, N_{12}, ..., N_{1n_1}\}, ..., \{N_{m,1}, N_{m,2}, ..., N_{m,n_m}\}\}$ est une collection qui représente une décision réalisable de conception, nous recherchons une telle décision P qui *optimise le critère* exprimé comme une somme pondérée du nombre de stations et du nombre de blocs :

$$\textbf{Min } C(P) = C_1 m + C_2 n = C_1 m + C_2 \sum_{k=1}^{m} n_k . \qquad (3\text{-}5)$$

3.2 Modèle MIP

3.2.1 Fonction objectif et contraintes

Nous résumons ci-dessous les notations utilisées :

- **N** est l'ensemble de toutes les opérations à affecter ;
- T_0 est le temps de cycle à ne pas dépasser ;
- m_0 est le nombre maximum autorisé de stations ;
- m est le nombre de stations dans une solution concrète ;
- n_0 est le nombre maximum autorisé de têtes d'usinage (blocs) par station ;
- n_k est le nombre de blocs à la station k pour une solution concrète ;
- n est le nombre total de blocs pour toute la ligne dans une solution ;
- C_1 est le coût relatif d'une station (coût fixe, *c.-à-d.* le même pour toutes les stations) ;
- C_2 est le coût relatif d'une tête d'usinage (coût fixe, *c.-à-d.* le même pour toutes les têtes) ;
- q est l'indice général de bloc, *c.-à-d.*, pour le l^e bloc de la station k, $q = (k\text{-}1)\, n_0 + l$;
- q_0 est la valeur maximum possible pour q, $q_0 = m_0 n_0$;
- $S(k)=\{(k\text{-}1)n_0 + 1, \ldots, kn_0\}$ est l'ensemble des numéros de blocs réservés pour la station k ;
- $Q(j)$ est l'ensemble des indices (numéros) de blocs auxquels l'opération j peut être assignée (l'opération j ne peut pas être effectuée dans un bloc $q \notin Q(j)$ à cause des contraintes du problème) ;
- $K(j)$ est l'ensemble des indices de station k où l'opération j peut être affectée (l'opération j ne peut pas être faite sur une station $k \notin K(j)$ à cause des contraintes du problème) ;
- $Pred(j)=\{i \in \mathbf{N} | (i, j) \in D\}$ est l'ensemble des prédécesseurs directs de l'opération j ;
- $j(e)$ est une opération de l'ensemble e ;
- x_{jq} est une variable de décision binaire (x_{jq}= 1 si l'opération $j \in \mathbf{N}$ est affectée au bloc q et $x_{jq} = 0$ sinon) ;

- F_q est la variable auxiliaire servant à déterminer la durée du bloc q, notons que par définition $0 \le F_q \le T_0$;
- Y_q est la variable auxiliaire indiquant si le bloc q existe (Y_q=1) ou non (Y_q=0) ;
- Z_k est la variable auxiliaire indiquant si la station k existe (Z_k =1) ou non (Z_k =0) ; les variables Y_q, $Z_k \in \{0,1\}$ sont utilisées pour compter le nombre de blocs et de stations, respectivement.

Présentons maintenant notre *modèle MIP*.

La ***fonction objectif*** du modèle est la suivante (Dolgui *et al.*, 2000b ; Dolgui *et al.*, 2003 ; Dolgui *et al.*, 2002d ; Dolgui *et al.*, 2004) :

$$\textbf{Min } C(P) = C_1 \sum_{k=1}^{m_0} Z_k + C_2 \sum_{q=1}^{q_0} Y_q, \tag{3-6}$$

c'est-à-dire que nous avons comme objectif de minimiser la somme pondérée du nombre de stations et du nombre de blocs, les poids étant le coût relatif d'une station C_1 et d'un bloc C_2.

Les ***contraintes*** sont exprimées par les équations suivantes :

$$\sum_{q \in Q(j)} x_{jq} = 1, \; j \in \mathbf{N}. \tag{3-7}$$

L'équation (3-7) précise que chaque opération j appartenant à **N** doit être affectée à un et un seul bloc ; l'indice de ce bloc doit appartenir à l'ensemble $Q(j)$.

$$\sum_{i \in Pred(j)} \sum_{q' \le q,\, q' \in Q(i)} x_{iq'} \ge x_{jq} \left| Pred(j) \right| \;, \; j \in \mathbf{N}, q \in Q(j). \tag{3-8}$$

L'équation (3-8) modélise les contraintes de précédence, c'est-à-dire qu'elle oblige d'affecter tous les prédécesseurs d'une opération j soit avant j soit dans le même bloc que l'opération j ; $|Pred(j)|$ est le cardinal de l'ensemble des prédécesseurs directs de l'opération j.

$$\sum_{j\in e\setminus\{j(e)\}} \sum_{q\in S(k)\cap Q(j)} x_{jq} = \left(|e|-1\right) \sum_{q\in S(k)} x_{j(e)q}\ ,\ e\in ES\,,\ k\in K(j(e)). \qquad (3\text{-}9)$$

L'équation (3-9) représente les contraintes d'inclusion pour les stations. Elle oblige d'inclure certains groupes d'opérations (donnés par les éléments de ES) dans la même station. C'est-à-dire que si n'importe quelle opération $j(e)$ d'un ensemble d'opérations $e\in ES$ est assignée à la station k, toutes les opérations de e doivent être assignées à la même station k.

$$\sum_{j\in e} x_{jq} \le |e|-1, \qquad e\in \overline{EB}, q\in \bigcap_{j\in e} Q(j). \qquad (3\text{-}10)$$

L'équation (3-10) définit les contraintes d'exclusion pour les blocs, c'est-à-dire l'impossibilité d'inclure certains groupes d'opérations toutes ensemble dans le même bloc. Pour chaque $e\in \overline{EB}$ et pour chaque bloc q possible, le nombre des opérations de l'ensemble e affectées au bloc q doit être inférieur au nombre total des opérations de e (si la somme est égale au cardinal de e, alors toutes les opérations de e sont incluses dans le bloc q, la contrainte n'est donc pas respectée).

$$\sum_{j\in e} \sum_{q\in S(k)\cap Q(j)} x_{jq} \le |e|-1, \quad e\in \overline{ES}, k\in \bigcap_{j\in e} K(j). \qquad (3\text{-}11)$$

L'expression (3-11) représente les contraintes d'exclusion pour les opérations affectées à la même station. Elle définit l'impossibilité d'inclure certains groupes d'opérations en entier sur la même station ; c'est-à-dire que pour chaque $e\in \overline{ES}$ et pour chaque station k possible, le nombre des opérations de l'ensemble e affectées à la station k doit être inférieur au nombre des opérations de e (si la somme est égale au cardinal de e, alors toutes les opérations de e sont sur la station k : la contrainte n'est donc pas respectée).

$$F_q \ge t_j x_{jq},\ q = 1, 2, ..., q_0 \text{ et } j\in \mathbf{N}. \qquad (3\text{-}12)$$

L'expression (3-12) calcule la durée d'un bloc donné q en accord avec (3-1). C'est-à-dire que le temps du bloc q est supérieur ou égal au temps opératoire le plus long des opérations faisant partie du bloc. Pour simplifier la présentation, la constante τ^b est absente, elle est supposée être intégrée dans les temps opératoires.

$$\sum_{q \in S(k)} F_q \leq T_0, \quad k=1, 2, \ldots, m_0. \tag{3-13}$$

L'expression (3-13) permet d'atteindre la productivité voulue ; c'est-à-dire que les temps des stations n'excèdent pas le temps de cycle objectif de la ligne T_0. Pour simplifier la présentation, la constante τ^s est omise (en modifiant T_0 en conséquence).

$$Y_q \geq x_{jq}, \quad j \in \mathbf{N}, \; q= 1, 2, \ldots, q_0. \tag{3-14}$$

L'équation (3-14) donne les conditions d'existence du bloc q, c'est-à-dire que s'il existe une opération j appartenant au bloc q (la variable de décision x_{jq} est égale à 1), alors Y_q doit être égale à 1 ; si au contraire aucune opération n'est affectée au bloc q, alors Y_q sera égale à zéro (à cause de la forme de la fonction objectif) ; dans ce dernier cas, le bloc est donc vide (inexistant).

$$Z_k \geq Y_q, \quad k = 1, 2, \ldots, m_0, \; q \in S(k). \tag{3-15}$$

L'équation (3-15) représente les conditions d'existence de la station k ; c'est-à-dire que s'il y a un bloc q non vide appartenant à la station k (Y_q=1, $q \in S(k)$), alors Z_k doit être supérieur ou égal à 1, (en tenant compte de la fonction objectif, Z_k sera strictement égal à 1).

Dans le modèle (3-6) - (3-15), les variables de décision x_{iq} sont *primaires*, elles déterminent les valeurs de toutes les autres variables.

Les contraintes du modèle sont construites d'une telle manière que nous pouvons déclarer les variables Z_k et Y_q comme les variables réelles $Z_k \in [0,1]$, $Y_q \in [0,1]$, k=1, 2, .., m_0, q= 1, 2, …, q_0, leur valeur reste de toutes les façons binaire {0,1} ; les variables F_q, q=1, 2, …, q_0 peuvent également être déclarées comme réelles.

Si $Q(j)$ ={1, 2, …, q_0} et $K(j)$ ={1, 2, … m_0} pour tout $j \in \mathbf{N}$, nous parlons du modèle MIP de base (MB). Pour MB, le nombre total de variables du modèle est de : $|\mathbf{N}|\, n_0\, m_0 + n_0\, m_0$

$+ m_0 + n_0 m_0$. Le nombre total de contraintes (de lignes de MIP) est de : $|\mathbf{N}| + 3 |\mathbf{N}| n_0 m_0 + n_0 m_0 + m_0 + |ES| m_0 + n_0 m_0 | \overline{EB} | + m_0 | \overline{ES} |$.

Pour l'exemple ci-dessus avec 19 opérations, n_0= 2, m_0= 6, et les contraintes données, nous avons 228 variables binaires x_{jq} (le nombre total des variables x_{jq}, F_q, Y_q et Z_k est égal à 258). Pour le même exemple, le nombre de contraintes du modèle MIP est égal à 1021. (Notons que si nous avions eu 20 opérations le nombre de variables serait égal à 270 dont 240 binaires x_{jq}).

3.2.2 Réduction du nombre de variables et contraintes

Intervalles des indices de blocs et de stations

Le temps de résolution du modèle MIP croit très fortement avec sa taille exprimée en nombre de variables binaires. Dans le paragraphe précédent, nous avons déjà montré que certaines variables binaires peuvent être remplacées par des variables réelles pour diminuer le temps de calcul. Nous avons également étudié d'autres possibilités pour augmenter les performances de ce modèle (Dolgui *et al.*, 2000b, Dolgui *et al.*, 2004). Entre autres, nous avons essayé de réduire le nombre de variables de décision binaires x_{jq} et de contraintes par des calculs plus précis des ensembles $Q(j)$ et $K(j)$ pour $j \in \mathbf{N}$.

Etant donné que les éléments des ensembles $Q(j)$ et $K(j)$ pour $j \in \mathbf{N}$ sont des indices consécutifs de blocs et de stations, par commodité, nous les appelons « *intervalles* ». Pour tout $j \in \mathbf{N}$, la valeur la plus petite de l'intervalle $Q(j)$ sera appelée l'*extrémité gauche* de $Q(j)$ et la valeur la plus grande sera appelée l'*extrémité droite* de l'intervalle $Q(j)$. Nous désignerons l'*extrémité gauche* de $Q(j)$ par $q^-[j]$ et l'*extrémité droite* de $Q(j)$ par $q^+[j]$.

De même pour l'ensemble $K(j)$, pour tout $j \in \mathbf{N}$, la valeur la plus petite de l'intervalle $K(j)$ sera appelée l'*extrémité gauche* de $K(j)$ et la valeur la plus grande sera appelée

l'*extrémité droite* de l'intervalle $K(j)$. Nous désignerons l'*extrémité gauche* de $K(j)$ par $k^-[j]$ et l'*extrémité droite* de $K(j)$ par $k^+[j]$.

Par définition des valeurs $q^-[j]$ et $q^+[j]$, nous ne pouvons pas mettre l'opération j dans un bloc dont l'indice est inférieur à $q^-[j]$, sans enfreindre les contraintes du problème ; nous ne pouvons pas mettre non plus l'opération j dans un bloc dont l'indice est supérieur à $q^+[i]$, sans perdre de l'optimalité de la solution. De même pour $k^-[j]$ et $k^+[j]$, mais pour les stations.

Nous essayons de réduire les cardinalités (tailles des intervalles) des ensembles $Q(j)$ et $K(j)$ par une analyse plus approfondie des contraintes du modèle.

Les valeurs $q^-[j]$ et $q^+[j]$ ont les *propriétés suivantes* :

pour chaque $i,\ j \in \mathbf{N}$,

si $(j, i) \in D$, alors $q^-[i] \geq q^-[j]$ et $q^+[i] \geq q^+[j]$,

de plus,

si $\{j, i\} \in \overline{EB}$ (ou $\{j, i\} \in \overline{ES}$), alors $q^-[i] > q^-[j]$ et $q^+[i] > q^+[j]$.

Les valeurs $k^-[j]$ et $k^+[j]$ ont les *propriétés suivantes* :

i) pour chaque $i,\ j \in \mathbf{N}$,

si $(j, i) \in D$, alors $k^-[i] \geq k^-[j]$ et $k^+[i] \geq k^+[j]$,

de plus,

si $\{j, i\} \in \overline{EB}$ (ou $\{j, i\} \in \overline{ES}$), alors $k^-[i] > k^-[j]$ et $k^+[i] > k^+[j]$;

ii) pour tout $i,\ j \in e \in ES$, $k^-[i] = k^-[j]$ et $k^+[i] = k^+[j]$.

Pour tout $j \in \mathbf{N}$, nous calculons la valeur $q^-[j]$ en supposant que les opérations de **N** sont classées dans l'ordre non décroissant de leur rang dans G. Nous utilisons une procédure pas à pas qui se base sur la notion de distance $d[i,j]$ entre deux opérations i et j. La distance $d[i,j]$ est définie de la manière suivante :

1) $d[i,j] = 2$, si $\{i,j\} \in \overline{ES}$,

2) $d[i,j] = 1$, si $\{i,j\} \in \overline{EB}$,

3) $d[i,j] = 0$, si $\{i,j\} \notin \overline{ES} \cup \overline{EB}$.

Les valeurs $q^-[j]$, $q^+[j]$ et $k^-[j]$, $k^+[j]$ pour tout $j \in \mathbf{N}$, sont liées de la manière suivante :

$k^-[j] = \lceil q^-[j]/n_0 \rceil$,

$k^+[j] = \lceil q^+[j]/n_0 \rceil$,

où $\lceil x \rceil$ est la plus petite valeur entière supérieure ou égale à x.

L'équation suivante est vérifiée :

$q^-[j] \geq \max \{\max \{q^-[i] \oplus d[i,j] \mid i \in Pred(j)\}, \max \{(k^-[i] - 1)\, n_0 + 1 \mid i, j \in e \in ES\}\}$,

où

$q \oplus d = \lceil q/n_0 \rceil n_0 + 1$, si $d = 2$,

$q \oplus d = q + d$, si $d \in \{0, 1\}$.

Présentons maintenant la formulation ci-dessus sous forme d'un algorithme. La variable *jcur* est utilisée dans l'algorithme pour savoir si une nouvelle itération doit être faite ou non.

Rappelons que les opérations sont classées dans l'ordre non décroissant de leur rang dans le graphe de précédence G.

Algorithme 3-1 :

Etape 1. Affecter $q^-[j]=1$ pour tout $j=1, 2,\ldots, |\mathbf{N}|$ et mettre $jcur=1$.

Etape 2. Mettre $jmin=jcur$ et $jcur=|\mathbf{N}|$.

Etape 3. Pour $j=jmin+1, \ldots, |\mathbf{N}|$ faire $q^-[j]= \max\{ q^-[i] \oplus d[i,j] \mid i \in Pred(j) \}$.

Etape 4. Pour chaque $e \in ES$

1. Calculer $M(e)= \max\{(\lceil q^-[j]/n_0 \rceil -1)n_0+1 \mid j \in e\}$.
2. Si $q^-[j] < M(e)$ pour $j \in e$ alors mettre $q^-[j] = M(e)$ et $jcur=\min\{jcur, j\}$.

Etape 5. Si $jcur=|\mathbf{N}|$ alors *Fin*, sinon aller à *Etape* 2.

Nous utilisons la même technique pour déterminer les valeurs $q^+[j]$, mais de la manière suivante :

- le graphe de précédence $\overline{G}$ est créé en inversant les arcs du graphe G (la Figure 3-6 montre le graphe $\overline{G}$ obtenu à partir du graphe de précédence de la Figure 3-5) ;
- pour tout $i \in \mathbf{N}$, l'extrémité gauche $\overline{q^-}[i]$ de l'intervalle $Q(i)$ est calculée pour $\overline{G}$. L'extrémité droite de l'intervalle $Q(i)$ pour le graphe initial G est alors égale à :

$q^+[i] = q_0 - \overline{q^-}[i] + 1$.

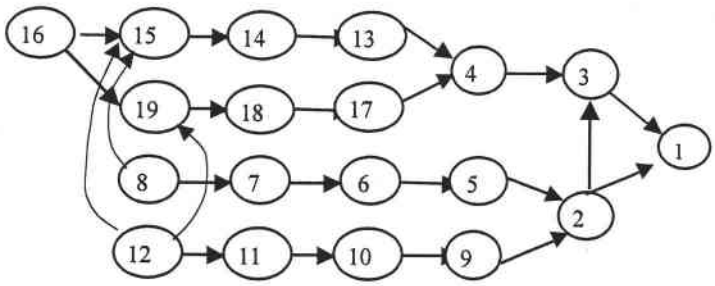

Figure 3-6 : Graphe de précédence inversé $\overline{G}$

Exemple de calcul des intervalles d'indices

Reprenons l'exemple ci-dessus avec 19 opérations, rappelons que les contraintes d'inclusion pour les blocs ont été déjà traitées et que les opérations 16 et 20 constituent une seule macro-opération 16. Dans les ensembles $\overline{ES}$, $\overline{EB}$ et ES, l'opération 20 est remplacée par l'opération 16. Seul un sous-ensemble de $\overline{EB}$ est concerné : {19, 20} devient {19, 16}. Le graphe de précédence est représenté par la Figure 3-7.

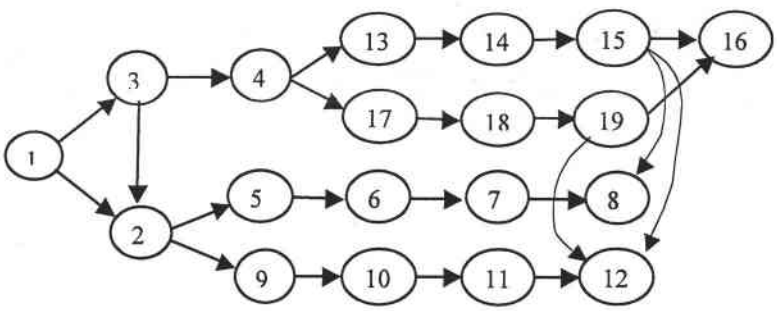

Figure 3-7 : Graphe de précédence G

Les contraintes de compatibilité (inclusion et exclusion) sont :

$\overline{ES}$ = {{2, 5}, {2, 9}, {2, 13}, {2, 17}, {3, 4}, {3, 5}, {3, 9}, {3, 13}, {3, 17}} ;

$\overline{EB}$ = {{1, 2}, {2, 3}, {6, 7}, {7, 8}, {10, 11}, {11, 12}, {13, 14}, {13, 18}, {13, 19}, {14, 15}, {14, 17}, {14, 18}, {14, 19}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {17, 18}, {18, 19}, **{19, 16}**} ;

ES = {{7, 11}}.

Pour cet exemple, les intervalles des indices de blocs et de stations, pour n_0= 2, m_0= 6, sont représentés dans le Tableau 3-2.

j	$q^-[j]$	$q^+[j]$	$k^-[j]$	$k^+[j]$
1	1	7	1	4
2	2	8	1	4
3	1	7	1	4
4	3	9	2	5
5	3	10	2	5
6	3	10	2	5
7	4	11	2	6
8	5	12	3	6
9	3	10	2	5
10	3	10	2	5
11	4	11	2	6
12	5	11	3	6
13	3	9	2	5
14	4	10	2	5
15	5	11	3	6
16	6	12	3	6
17	3	9	2	5
18	4	10	2	5
19	5	11	3	6

Tableau 3-2 : Intervalles des valeurs des indices de blocs et de stations

En appliquant les calculs des intervalles pour cet exemple, nous avons réduit le nombre de variables de décision binaires x_{jq} de 228 à 141. Le nombre de contraintes est réduit de 1021 à 790.

3.2.3 Modification de la fonction objectif

Bornes inférieures du nombre de blocs et de stations

Nous allons calculer maintenant des bornes inférieures du nombre de stations et du nombre de blocs. Il est facile de voir qu'une *borne inférieure* $\underline{m}$ du nombre de stations est égale à :

$$\underline{m} = LB_m = \max_i k^-[i]. \tag{3-16}$$

De même, une *borne inférieure* du nombre de blocs $\underline{n}$ est facile à obtenir.

Si nous mettons n_0=1 et $ES = \{\varnothing\}$, et calculons tous les $q^-[i]$ en utilisant l'Algorithme 3-1, la borne inférieure du nombre de blocs $\underline{n}$ sera égale à :

$$\underline{n} = LB_n = \max_i q^-[i], \text{ pour } n_0=1, ES = \{\varnothing\}. \tag{3-17}$$

Notons que nous ne pouvons pas obtenir une borne inférieure de n si nous prenons n_0>1, car dans ce cas il est possible d'avoir des blocs vides. Par contre, en utilisant l'Algorithme 3-1 pour une valeur de n_0 différente de 1, nous pouvons obtenir une borne inférieure de la valeur maximale de l'indice q (borne inférieure du numéro du dernier bloc dans une solution optimale) :

$$\underline{\max q} = \max_i q^-[i], \text{ pour } n_0>1. \tag{3-18}$$

Modification de la fonction objectif et agrégation des variables

Pour améliorer d'avantage les performances du modèle, nous modifions le critère et nous remplaçons toutes les variables Z_k par une seule variable $Z \geq 0$ totalisant le nombre de stations. Les contraintes (3-15) et la fonction objectif (3-6) deviennent alors (3-19) et (3-20) respectivement :

$$Z \geq kY_q, \quad k=\underline{m}, \underline{m}+1, \ldots, m_0, q \in S(k), \tag{3-19}$$

$$\textbf{Min } C(P) = C_1 Z + C_2 \sum_{q=1}^{q_0} Y_q + \varepsilon_1 \sum_{q=1}^{q_0} qY_q \quad , \tag{3-20}$$

où

ε_1 est une valeur non négative suffisamment petite choisie comme suit :

$\varepsilon_1 < \varepsilon / (q_0 (q_0+1))$,

$\varepsilon = 2 \min\{C_1, C_2, \text{abs}(C_1-C_2)\}$,

abs(a) représente la valeur absolue de a.

La fonction objectif modifiée, pour des variables binaires, permet d'éviter des énumérations inutiles menant vers des solutions identiques (du point de vue de la fonction objectif) à celles qui sont déjà obtenues se différenciant seulement par les numéros de blocs et de stations non vides. Parmi les solutions identiques, elle privilégie celle qui a des blocs avec des numéros plus petits. La numérotation des blocs dans la solution optimale devient consécutive (il n'y a pas de blocs vides). Par conséquent, l'introduction de cette nouvelle fonction objectif accélère les calculs.

Notons qu'en utilisant (3-19) et (3-20), nous pouvons aboutir à une distribution irrégulière des blocs. Les blocs sont concentrés sur les premières stations. Si cet effet est non désirable, il peut être éliminé en utilisant la fonction objectif suivante :

$$\textbf{Min } C(P) = C_1 Z + C_2 \sum_{q=1}^{q_0} Y_q + \varepsilon_2 \sum_{q=1}^{q_0} q^* Y_q , \tag{3-21}$$

où

$q^*=q-\lfloor q/n_0 \rfloor n_0$,

$\lfloor a \rfloor$ est la partie entière de a,

ε_2 est une autre valeur non négative suffisamment petite obtenue comme suit :

$\varepsilon_2 < \varepsilon/((n_0+1)(q_0+1))$.

La fonction objectif (3-21) permet de répartir les blocs sur les stations de manière plus uniforme.

3.2.4 Ajout de coupes

Utilisation des coupes standard de Cplex

Pour accélérer les calculs des modèles MIP, plusieurs types de coupes sont proposés dans la littérature (Gill *et al.*, 1981 ; Nemhauser et Wolsey, 1988 ; Williams, 1993 ; Cplex, 2001 ; John, 2003). Les coupes sont des contraintes supplémentaires ajoutées au modèle ; elles permettent de réduire l'espace de recherche des solutions intermédiaires lors de l'étape de relaxation de la programmation linéaire. L'ajout de coupes réduit le nombre de branches de l'arbre à parcourir pour résoudre le problème en nombres entiers. Les coupes types sont intégrées dans les solvers du commerce comme Cplex (ILOG).

Les coupes standard les plus utilisées sont (Cplex, 2001) :

Utilisation des cliques : une clique est une relation entre un ensemble de variables binaires impliquant qu'une seule variable de cet ensemble soit non nulle pour toute solution faisable entière.

Utilisation des contraintes de sac à dos : pour les contraintes de type sac à dos (la somme de variables binaires ayant un coefficient non-négatif est inférieure ou égale à la partie droite non négative), un recouvrement minimal est recherché ; un recouvrement minimal est un sous-ensemble de ces variables tel que si toutes les variables de ce sous-ensemble valent 1, la contrainte du sac à dos est violée alors que si au moins une de ces variables est égale à zéro, la contrainte est respectée.

Coupe par recouvrement GUB (Generalized Upper Bound) : cette coupe appliquée à un ensemble de variables binaires représente une somme de variables inférieure ou égale à un.

Coupe de Gomory : cette coupe arrondit la valeur optimale fractionnelle pour une variable entière se trouvant en base sur la ligne pivot de la solution optimale du problème relaxé.

Les algorithmes PSE enrichis par des coupes sont appelés les algorithmes de Séparation et Coupe (Branch & Cut). A chaque sommet de l'arbre de branchement de son PSE, Cplex teste plusieurs coupes (www.iro.umontreal.ca/~gendron/IFT6551/CPLEX/HTML), pour essayer de réduire l'espace de recherche, et résout les problèmes LP relaxés correspondants. Et quand il n'est plus possible d'apporter des améliorations par des coupes, un nouveau branchement est fait sur une variable entière.

Par défaut, Cplex détermine combien de fois il doit utiliser chacune de ses coupes. Il est possible de le forcer à les utiliser plus souvent ou au contraire d'interdire l'application des coupes standard, en totalité ou en partie.

Pour les tests nous utilisons les coupes Clique, GUB et Gomory qui sont considérées comme meilleures pour les problèmes MIP de notre type (John, 2003).

Cplex offre également deux options pour le mode d'optimisation : la première garantit l'optimum, mais le temps de calcul (c'est-à-dire la démonstration que la solution obtenue est

optimale) peut être long et la deuxième privilégie la recherche la plus rapide possible d'une très bonne solution, mais elle ne garantit pas que la solution obtenue soit optimale. Notons que la deuxième option (elle est choisie en utilisant le paramètre EMPHASIS) donne pourtant dans la plupart de cas l'optimum (ou une solution très proche de l'optimum). Cette option est conseillée par ILOG (John, 2003).

Nouvelles coupes proposées

En plus des coupes standard, pour des problèmes spécifiques, nous pouvons trouver d'autres coupes tenant compte de la structure concrète du modèle.

Pour notre modèle, nous proposons les deux coupes suivantes :

1. une coupe basée sur la spécificité des variables Y et Z, nous l'appelons la coupe YZ ;
2. une coupe basée sur l'analyse des contraintes $\overline{EB}$, nous l'appelons la coupe ES.

***Coupe* YZ.**

La première coupe, la coupe YZ, consiste à ajouter les contraintes suivantes :

$$Y_{q-1} - Y_q \geq 0, \text{ pour tout } q \in S(k)\backslash\{(k\text{-}1)n_0+1\}, k=1, 2, \ldots, m_0, \tag{3-22}$$

c'est-à-dire que le bloc q ne peut être créé que si le bloc q-1 existe pour la station k ;

$$Z_{k-1} - Z_k \geq 0, \quad k=\underline{m}+1, \underline{m}+2, \ldots, m_0, \tag{3-23}$$

c'est-à-dire que la station k ne peut être créée que si la station k-1 existe dans la solution construite.

L'idée de ces coupes vient d'approches assurant la continuité des numéros de blocs et de stations. Aucun bloc sur une station ou aucune station n'est laissé vide. Leur effet doit donc être le même que celui d'introduction des ε dans la fonction objectif (voir le paragraphe 3.2.3).

En appliquant les coupes YZ ci-dessus, nous utiliserons la contrainte (3-15) modifiée :

$$Z_k \geq Y_q, \quad k=\underline{m}+1, \underline{m}+2, \ldots, m_0, q=(k-1)n_0+1, \tag{3-24}$$

c'est-à-dire que la station k est créée si au moins le premier bloc de cette station est créé.

***Coupe* ES.**

Notre *deuxième coupe* consiste à rajouter des contraintes dans $\overline{ES}$ suite à une analyse de $\overline{EB}$. Pour cela nous partons de la réflexion suivante : lorsqu'il y a des opérations qui constituent un sous-ensemble $e \in \overline{EB}$, nous savons déjà qu'au minimum, elles doivent être réparties dans deux blocs différents pour respecter cette contrainte. Dans le meilleur des cas, le temps total pour ces deux blocs sera égal à la somme du temps de l'opération la plus longue et de l'opération la plus courte de e. La répartition en deux blocs donnant un temps minimum revient donc à avoir dans un premier bloc l'opération la plus courte et dans un deuxième bloc toutes les autres opérations (ou inversement) : nous ne pouvons pas mieux faire. Si cette somme est supérieure à T_0, il sera impossible de respecter la contrainte sans créer une nouvelle station. Dans ce cas, nous pouvons donc, sans modifier le problème rajouter e dans $\overline{ES}$. En d'autres termes :

$$\text{si } e \in \overline{EB} \text{ et } mint(e) + maxt(e) > T_0, \text{ alors } e \in \overline{ES}. \tag{3-25}$$

où

$$mint(e) = \min_{i \in e} (t_i),$$

$$maxt(e) = \max_{i \in e} (t_i),$$

t_i est le temps de l'opération i.

3.3 *Tests numériques*

3.3.1 Déroulement des tests

Pour tester notre modèle, nous avons utilisé le solver Cplex version 7.0 sur un PC Compaq W6000 bi-processeurs, Intel Xeon avec 1.70 Ghz de CPU et 1 Giga de RAM.

Nous commençons par donner les résultats détaillés d'optimisation pour l'exemple que nous traitons tout au long de ce chapitre. Puis, nous résumons les statistiques des tests pour :

- quelques cas industriels (MZAL) ;
- des tests individuels générés aléatoirement ;
- des séries de tests générées aléatoirement.

Nous avons utilisé un générateur d'exemples aléatoires pour créer des séries d'exemples (tests) type. Chaque série comporte 10 exemples ayant un nombre fixe d'opérations et soit le temps de cycle, soit les contraintes sont modifiés d'un exemple à un autre de la même série. Pour les séries, nous ne présentons les résultats qu'en termes de valeurs moyennes de chaque série.

Nous commençons par le modèle représenté par les équations (3-6) à (3-15). Nous appelons modèle de base (MB), le modèle sans les intervalles d'indices c.-à-d. quand $K(j)=\{1, 2, \ldots, m_0\}$, $Q(j)=\{1, 2, \ldots, q_0\}$, $j \in \mathbf{N}$. Nous montrons d'une part ses capacités en termes de taille de problème qu'il peut traiter dans un temps alloué, et d'autre part, ses capacités en termes de temps de calcul pour les problèmes connus avec les caractéristiques données. Ensuite, nous refaisons les mêmes tests en utilisant les différentes améliorations du modèle expliquées dans ce chapitre.

Les résultats de nos tests permettent de situer notre modèle par rapport à la problématique industrielle de la conception des lignes de transfert en donnant des éléments de réponse aux questions suivantes : Quelle est la taille des problèmes réels qui peuvent être traités avec ce modèle ? Quelle est la ressource de temps nécessaire pour des cas réels ? Les tests donnent également des idées sur des voies possibles d'amélioration du modèle.

3.3.2 Résultats détaillés pour un exemple

Nous commençons par l'exemple i0 que nous traitons tout au long de ce chapitre concernant la fabrication de la pièce de la Figure 3-1. Pour cet exemple C_1= 10, C_2= 2, n_0=2, m_0=6, T_0= 575.

L'analyse des contraintes pour cet exemple nous a permis d'obtenir les intervalles des indices de blocs et de stations présentés dans le Tableau 3-2. Le nombre de chaque type de contraintes (chaque équation du modèle) pour cet exemple est présenté dans le Tableau 3-3.

3-7	3-8	3-9	3-10	3-11	3-12	3-13	3-14	3-15	Total
19	141	5	124	27	228	6	228	12	790

Tableau 3-3 : Nombre des contraintes

La Figure 3-8 représente la solution optimale obtenue avec Cplex en utilisant le modèle MIP (3-6) – (3-15) avec les intervalles d'indices de blocs et de stations calculés au préalable suivant l'Algorithme 3-1. La valeur du critère est égale à 56.

Dans la Figure 3-8, les flèches montrent la séquence des opérations sur la ligne (dans une station c'est l'ordre d'activation des têtes d'usinage). La première station a une tête d'usinage pour effectuer simultanément les opérations 1 et 3 et une autre tête pour effectuer l'opération 2. La deuxième station a une tête d'usinage exécutant cinq opérations et une tête réalisant deux opérations. La troisième station a deux têtes d'usinage avec deux opérations

chacune et enfin la dernière station a 2 têtes d'usinage, la première fait une seule opération et la deuxième réalise trois opérations.

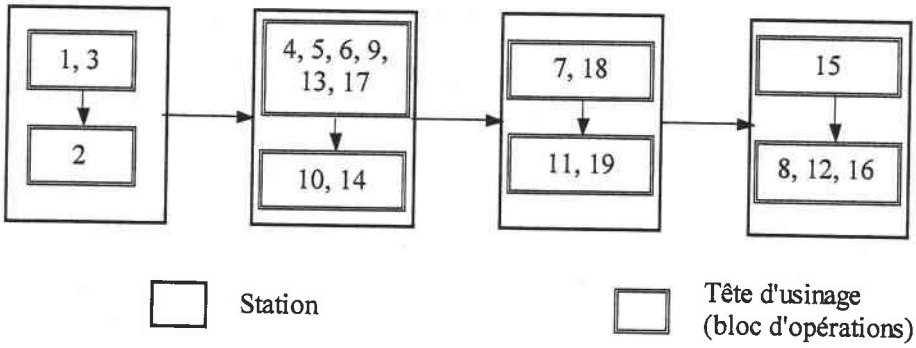

Figure 3-8 : Solution optimale

La solution optimale est obtenue en 2 minutes 20 secondes en utilisant le modèle (3-6) – (3-15) avec les intervalles d'indices. Pour obtenir l'optimum en nombres entiers, 48600 nœuds ont été parcourus et 673449 itérations ont été exécutées. Le temps de relaxation est seulement de 0,05 secondes. Cplex a effectué de lui-même 15 coupes de Gomory, et 14 coupes GUB.

Pour montrer la sensibilité des performances du modèle aux options (améliorations) que nous avons proposées, nous avons effectué quelques tests complémentaires (voir le Tableau 3-4). La première colonne donne le numéro de test. La deuxième montre le type de modèle utilisé. La troisième colonne indique le nombre de variables de décision x_{jq} du modèle correspondant. La quatrième colonne donne le nombre total de variables. La dernière colonne donne le nombre de contraintes. Le nombre de variables x_{jq} et le nombre de contraintes sont ceux du modèle avant l'utilisation de Cplex. Nous verrons plus loin qu'après le pré-processeur de Cplex (pré-traitement de tous les modèles intégré à Cplex), ce nombre peut diminuer.

En ajoutant les coupes YZ et ES, nous avons donc 9 contraintes de plus, mais en utilisant en même temps (3-24) au lieu de (3-15), nous faisons l'économie de 9 contraintes, car la borne inférieure $\underline{m}$ pour cet exemple est égale à 3. Le nombre total de contraintes ne change donc pas.

Test N°	Modèle	Nombre de variables x_{jq} (*NDV*)	Nombre total de variables (*TNV*)	Nombre de contraintes (*NC*)
1	Modèle de base (MB)	218	246	846
2	MB + intervalles indices	141	171	790
3	MB + intervalles indices + EMPHASIS	141	171	790
4	MB + intervalles indices + EMPHASIS + Coupes YZ+ ES	141	171	790

Tableau 3-4 : Caractéristiques des tests effectués

Le Tableau 3-5 présente les résultats de ces quatre tests. Ce tableau donne le temps de calcul total, la durée de la relaxation, le type de coupe et le nombre de fois que ces coupes ont été utilisées (Clique, Gomory, Gub), le nombre total de variables (*CTNV*), le nombre de contraintes (*CNC*) après le pré-processeur, le nombre de nœuds parcourus par l'algorithme de Branch & Cut et le nombre d'itérations. Le nombre d'itérations de Cplex est le nombre d'itérations cumulées faites pour résoudre tous les sous-problèmes.

N°	Temps CPU	Temps relax	Clique	Gomory	Gub	*CTNV*	*CNC*	Nombre nœuds	Nombre itérations
1	2'20"	0,05"	-	15	14	176	573	48600	673449
2	20"	0,02"	-	10	5	133	417	12500	115690
3*	5"	0,02"	-	9	-	133	417	2300	32717
4*	3"	0,02"	1	11	4	133	426	800	11467

* la solution n'est pas optimale

Tableau 3-5 : Résultats des tests

Notons que les tests 3 et 4 n'ont pas abouti à la solution optimale. La procédure EMPHASIS utilisée a arrêté la recherche avant de trouver l'optimum. L'écart entre la solution obtenue et la solution optimale pour le test 4 est de 21,43 % pour la fonction objectif.

3.3.3 Tests pour des cas industriels

Description des instances

Pour la présentation des données des tests, introduisons quelques notations :

ex est le numéro d'exemple ;

NO est le nombre d'opérations ;

NA est le nombre d'arcs dans le graphe de précédence ;

$n\overline{EB}$ est le nombre de sous-ensembles de $\overline{EB}$;

$n\overline{ES}$ est le nombre de sous-ensembles de $\overline{ES}$;

nES est le nombre de sous-ensembles de *ES* ;

$r\overline{ES}$ est la taille maximum (nombre d'éléments) des sous-ensembles de $\overline{ES}$;

rES est la taille maximum des sous-ensembles de *ES* ;

$r\overline{EB}$ est la taille maximum des sous-ensembles de $\overline{EB}$;

NDV est le nombre de variables de décision x_{jq} avant le pré-processeur Cplex ;

TNV est le nombre total des variables avant le pré-processeur Cplex ;

NC est le nombre de contraintes avant le pré-processeur Cplex ;

CTNV est le nombre total de variables donné par Cplex après l'élimination par le pré-processeur des variables inutiles pour le calcul ;

CTNC le nombre de contraintes donné par Cplex après l'élimination par le pré-processeur des contraintes inutiles pour le calcul ;

n_0 et m_0 sont les nombres maximum autorisés de blocs et stations, respectivement.

Introduisons également le paramètre *pp* qui est le plus grand nombre de prédécesseurs des opérations :

$$pp=\max\{|Pred(j)| \, j \in \mathbf{N}\} ; \qquad (3\text{-}26)$$

Dans la première partie des tests ci-dessous, nous avons déclaré les variables Y_q, Z_k *et* F_q comme entières. Nous montrons par la suite l'effet de leur déclaration en tant que réelles.

Dans le Tableau 3-6 sont présentées les données correspondant aux exemples industriels que nous traitons dans ce paragraphe. Nous avons déjà présenté un quatrième exemple industriel (i0) au paragraphe précèdent (voir également le paragraphe 3.1.2).

ex	*NO*	n_0	m_0	T_0	*NA*	*pp*	$\overline{nEB}$	$\overline{nES}$	*nES*	$r\overline{EB}$	$r\overline{ES}$	*rES*
i1	9	2	6	200	19	6	12	8	-	2	2	-
i2	17	9	3	300	51	14	47	9	-	2	2	-
i3	15	8	3	400	52	12	35	34	-	2	2	-

Tableau 3-6 : Caractéristiques des contraintes des exemples industriels

Des exemples industriels de cette taille peuvent être traités par MIP même avec le modèle de base, c'est-à-dire sans améliorations liées avec la réduction du nombre de variables, sans l'agrégation des variables et sans les coupes autres que celles de Cplex. Un petit récapitulatif de l'optimisation des cas i1, i2 et i3 par MIP de base est donné dans le Tableau 3-7.

ex	*NDV*	*TNV*	*NC*	*CTNV*	*CNC*	**Nombre nœuds**	**Nombre itérations**	**Temps relax**	**Critère (stations, blocs)**	**Temps CPU**
i1	112	138	543	124	321	76900	996291	0,01	42 (3,6)	2'14"
i2	459	516	2720	217	608	2200	35928	0,34	30 (2,5)	11"
i3	360	411	2064	196	1263	4600	161366	0,48	44 (3,7)	2'26"

Tableau 3-7 : Résultats donnés par MIP de base

3.3.4 Un ensemble de tests générés aléatoirement

Compléterons maintenant les exemples industriels par un certain nombre d'exemples générés aléatoirement (voir le Tableau 3-8).

En plus des notations du paragraphe précédent, nous utilisons ici les suivantes :

- $T(S)$ est le temps de calcul CPU total,
- *Non Zero* est le total de croisements variables utiles / contraintes utiles ayant une valeur non nulle dans Cplex,
- n est le nombre de blocs dans la solution optimale trouvée,
- m le nombre de stations dans la solution optimale trouvée.

ex	*NO*	*NA*	*pp*	T_0	$n\overline{EB}$	$n\overline{ES}$	*nES*	$r\overline{EB}$	$r\overline{ES}$	*r ES*
1	6	6	2	7	2	1	1	2	2	2
2	9	10	2	60	4	2	2	2	2	2
3	11	11	2	9	2	1	1	2	3	2
4	13	17	3	11	3	3	2	3	2	3
5	15	15	2	7	5	1	1	2	3	2
6	18	26	3	15	3	3	3	2	3	3
7	23	39	4	70	4	4	3	3	3	3
8	25	38	4	16	3	3	3	3	3	3
9	30	44	4	80	3	4	4	3	3	2
10	35	46	3	70	12	4	5	4	5	4
11	38	62	3	20	4	6	5	2	2	3
12	45	67	3	100	5	4	4	3	3	3
13	100	194	4	35	6	7	5	2	2	3
14	120	214	5	27	14	7	8	4	2	3
15	150	286	13	80	26	13	9	3	3	3

Tableau 3-8 : Caractéristiques des exemples

Pour ces exemples, les résultats de tests avec le modèle MIP de base (MB) sont présentés dans le Tableau 3-9. Nous ne donnons plus le temps de relaxation LP, car il est négligeable. La phase « > 5h » signifie que nous avons interrompu le test au bout de 5 heures de calcul sans avoir obtenu une solution optimale.

ex	*NO*	*NDV*	*TNV*	*NC*	*T(S)*	*CTNV*	*CNC*	*Non zéro*	n_0	m_0	*Solution*	
											n	*m*
1	6	36	51	139	0,3"	48	118	361	2	3	2	2
2	9	81	102	314	1"	63	163	583	3	3	3	3
3	11	198	240	669	41"	237	612	4540	3	6	3	2
4	13	234	273	826	1"	154	428	3650	6	3	3	3
5	15	270	309	952	54"	171	283	2983	6	3	2	2
6	18	270	305	912	5'52"	215	621	8673	3	5	5	3
7	23	414	453	1407	13'32"	445	1260	38847	6	3	5	3
8	25	600	652	1966	1'17"	454	1339	29686	6	4	4	2
9	30	1080	1158	3472	13h23'	1137	3230	144238	6	6	4	3
10	35	630	669	2223	22h56'	655	2063	101170	6	3	5	3
11	38	798	847	2579	>10h	-	-	-	3	7	-	-
12	45	945	994	3037	>10h	-	-	-	3	7	-	-
13	100	2100	2149	6593	>10h	-	-	-	3	7	-	-
14	120	2520	2569	8050	>10h	-	-	-	3	7	-	-
15	150	3600	3656	11680	>10h	-	-	-	3	8	-	-

Tableau 3-9 : Résultats des tests avec MIP de base

3.3.5 Séries de tests

Récapitulatif pour le modèle de base

Pour évaluer plus finement les performances des modèles proposés, nous avons généré de manière aléatoire des séries d'exemples dans lesquelles chaque série contient 10 exemples avec le même nombre d'opérations, mais pour lesquels les contraintes sont différentes. Dans

le texte qui suit, les noms de tests représentant des séries d'exemples commençant par « s ». Les noms des tests pour les cas industriels commencent par la lettre « i ». Les exemples individuels sont donnés par leur numéro.

Le Tableau 3-10 donne les résultats d'optimisation. Dans ce tableau, les colonnes 5 à 11 présentent les valeurs moyennes (arrondies). Nous ne montrons pas le nombre moyen de blocs et de stations dans les solutions obtenues car ces valeurs n'ont pas de sens.

ex	*NO*	n_0	m_0	*NDV*	*TNV*	*NC*	*CTNV*	*CNC*	*Non Zéro*	*T(S)*
s1	10	5	5	250	305	845	285	742	7925	34"
s2	10	6	5	300	366	1465	363	948	9032	2'25"
s3	13	5	5	325	380	1473	358	992	32898	38"
s4	15	6	6	540	618	2397	571	1669	25306	16'32"
s5	17	6	5	510	576	2466	521	1532	25306	2'16"
s6	18	5	5	450	505	2025	343	1006	21663	2'28"
s7	20	3	5	300	333	1200	279	845	13318	3'21"
s8	23	4	6	552	604	2169	445	1260	38847	1h23'20"
s9	25	4	6	600	652	2355	581	1819	47142	5h04'55"
s10	25	3	6	450	489	1699	414	1167	30833	3h17'50"
s11	29	3	6	522	561	1967	452	1330	29277	24'31"
s12	40	2	6	480	506	1818	456	1366	29062	36'03"

Tableau 3-10 : Résultats pour les séries d'exemples avec MIP de base

Une première conclusion que nous pouvons faire consiste à dire que les paramètres n_0 et m_0 influencent beaucoup la taille du modèle et le temps de calcul. Nous devons donc les estimer au plus juste. Une deuxième conclusion est que plusieurs problèmes réels de taille moyenne peuvent être résolus en temps alloué ne dépassant pas 3 heures. Pour les problèmes de taille plus importante ou avec un budget de temps plus réduit, il faut faire appel à des améliorations du modèle de base.

Réduction du nombre de variables

Pour savoir quel est le gain de l'introduction des intervalles $Q(j)$ et $K(j)$, nous avons fait quelques exemples et trois séries sans et avec ces intervalles.

ex	*NO*	**Modèle de base (MB)**				**Réduction du nombre de variables, i.e. MB + intervalles (MBI)**			
		CTNV	*CNC*	*Non Zéro*	*T(S)*	*CTNV*	*CNC*	*Non zéro*	*T(S)*
3	11	237	612	4599	41"	237	608	4532	26"
5	15	171	283	6165	54"	171	279	5942	11"
6	18	215	621	8673	5'52"	215	615	8667	1'38"
7	23	445	1260	38847	13'32"	445	1260	22032	9"
8	25	454	1339	29686	1'17"	421	1250	22837	52"
s2	10	363	948	9032	2'25"	283	928	8543	1'25"
s4	15	571	1669	25306	16'32"	552	618	20397	6'35"
s9	25	581	1819	47142	5h04'55"	411	1306	31896	2'58"

Tableau 3-11 : Effet de l'utilisation des intervalles $Q(j)$ et $K(j)$

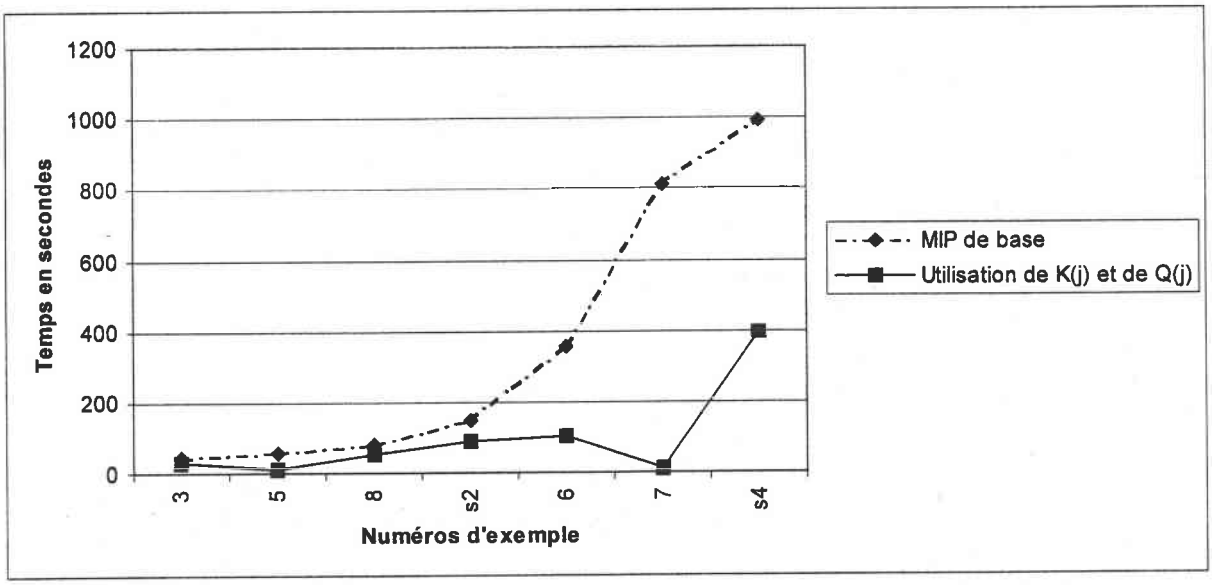

Figure 3-9 : Effet de l'utilisation des intervalles sur le temps de calcul

Une première conclusion est que Cplex dans sa procédure de pré-traitement détecte lui-même certaines choses liées avec les intervalles que nous avons introduits, mais pas toutes. Une deuxième conclusion est que l'effet de l'introduction des intervalles dans le modèle de

base est d'autant plus grand que le nombre d'opérations et le nombre de contraintes sont grands, la **Figure 3-9** le montre. Dans cette figure pour une question de lisibilité nous n'avons pas inclus l'exemple 9 et la série s9 car leurs temps d'exécution sont beaucoup plus grands que pour les autres exemples et ils auraient tassé les courbes.

Ajout de coupes

Pour la suite de cette étude, nous avons choisi la série s7 (20 opérations), car cette série est celle ayant le nombre moyen d'opérations. Le Tableau 3-12 donne les caractéristiques des exemples de cette série.

Les Tableau 3-13, Tableau 3-14 et le Tableau 3-15 ont la même structure. La première colonne donne le code de l'exemple, la deuxième colonne présente le nombre d'arcs du graphe de précédence, la troisième colonne, le nombre de contraintes pour le modèle de base, les quatrième et cinquième colonnes donnent les temps CPU des deux méthodes comparées, et la dernière colonne le gain en temps CPU obtenu (le temps est arrondi à la seconde).

ex	*NA*	*pp*	m_0	n_0	T_0	$n\overline{EB}$	$n\overline{ES}$	*nES*	r $\overline{EB}$	r $\overline{ES}$	r *ES*
s7-1	66	8	5	3	60	4	9	1	4	3	2
s7-2	66	7	5	3	60	7	9	1	4	3	3
s7-3	59	6	5	3	60	7	6	1	2	3	3
s7-4	52	5	5	3	60	6	9	1	3	3	2
s7-5	40	4	5	3	60	7	8	0	5	3	4
s7-6	40	5	5	3	60	6	6	1	4	2	3
s7-7	51	6	5	3	60	5	6	1	4	4	2
s7-8	39	4	5	3	60	6	8	1	4	4	3
s7-9	40	3	5	3	60	4	8	1	3	4	0
s7-10	43	7	5	3	60	5	7	1	4	3	3

Tableau 3-12 : Caractéristiques des exemples de la série s7

Les effets de l'introduction des coupes de Cplex et du choix de l'option "Emphasis" sont représentés dans le Tableau 3-13. L'option Emphasis de Cplex accélère beaucoup le temps de calcul, mais pour notre problème le nombre de fois (51%) que le résultat n'est pas optimal est trop important. Nous ne retenons pas cette solution.

ex	*NA*	*NC*	**Temps CPU - *T(S)***		
			MBI + coupes Cplex	**MBI +coupes Cplex + EMPHASIS**	***Gain***
s7-1	66	2065	5'8"	34"	4'34"
s7-2	66	2018	2'9"	52"*	1'17"
s7-3	59	1759	10"	4"	6"
s7-4	52	1595	1'11"	34"*	37"
s7-5	40	1683	15"	4"*	11"
s7-6	40	1511	10"	4"	6"
s7-7	51	1954	37"	4"*	33"
s7-8	39	1873	20"	7"*	13"
s7-9	40	1838	2'45"	1'5"	1'4"
s7-10	43	1597	8"	5"	3"

* la solution n'est pas optimale (Emphasis)

Tableau 3-13 : Effets de l'introduction d'Emphasis et des coupes Cplex

Dans le Tableau 3-14, nous présentons l'effet apporté par l'introduction des deux coupes que nous avons proposées. Nous constatons que l'introduction de nos coupes améliore de façon très significative le temps de calcul (voir les courbes dans la Figure 3-10). La courbe en pointillée représente les temps de calcul en utilisant le modèle de base avec les intervalles (MBI) et l'autre courbe les temps pour les mêmes exemples en utilisant MBI plus toutes les coupes proposées. En même temps nous pouvons constater (en comparant la colonne MBI avec le Tableau 3-13) que, pour ces exemples, l'utilisation des coupes standard de Cplex toutes seules ralentit les calculs.

ex	*NA*	*NC*	Temps CPU - *T(S)*		
			MBI	**MBI+ coupes Cplex +YZ +ES**	**Gain**
s7-1	66	2065	3'	11"	2'49"
s7-2	66	2018	2'1"	15"	1'46"
s7-3	59	1759	28"	5"	23"
s7-4	52	1595	7'15"	22"	6'53"
s7-5	40	1683	16'44"	13"	16'31"
s7-6	40	1511	13"	5"	8"
s7-7	51	1954	26"	4"	22"
s7-8	39	1873	45"	5"	40"
s7-9	40	1838	2'15"	19"	1'56"
s7-10	43	1597	17"	6"	11"

Tableau 3-14 : Effet de l'introduction des coupes ES et YZ

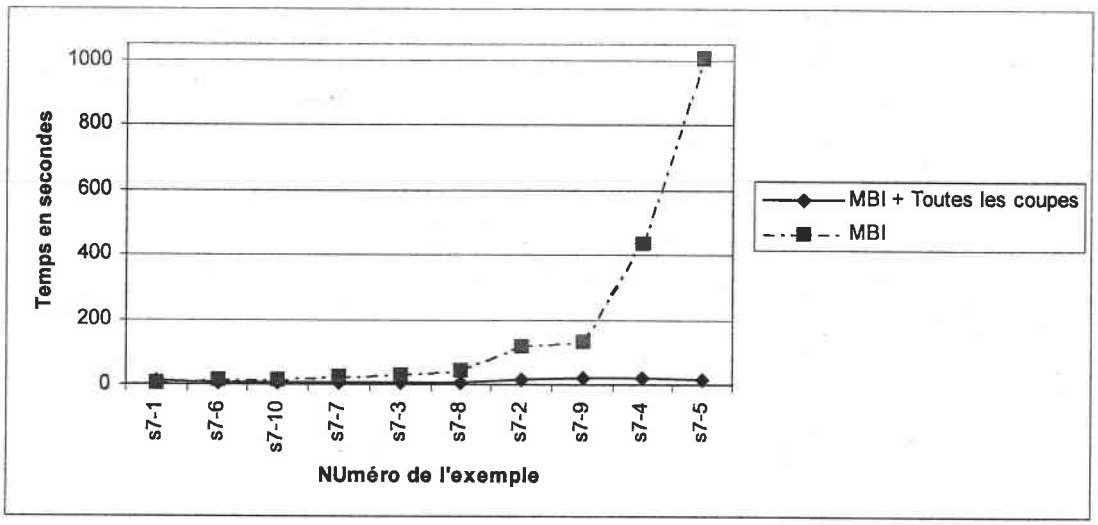

Figure 3-10 : Effet de l'introduction des coupes ES et YZ

Modification de la fonction objectif et changement des variables

Dans le Tableau 3-15, nous étudions l'effet de la modification de la fonction objectif et d'agrégation des variables (remplacement des variables Z_k par une seule variable Z), c'est-à-dire lorsque nous passons du modèle (3-6) – (3-15) au modèle modifié avec la fonction objectif (3-20) et les contraintes (3-19).

ex	*NA*	*NC*	Temps CPU – *T(S)*		
			MBI	**Modèles avec (3-19), (3-20)**	**Gain**
s7-1	66	2046	3'	16"	2'44"
s7-2	66	1999	2'1"	49"	1'12"
s7-3	59	1740	28"	13"	15"
s7-4	52	1576	7'15"	1'24"	6'51"
s7-5	40	1664	16'44"	12"	16'32"
s7-6	40	1492	13"	7"	6"
s7-7	51	1935	26"	10"	16"
s7-8	39	1854	45"	14"	31"
s7-9	40	1819	2'15"	1'11"	1'4"
s7-10	43	1578	17"	14"	3"

Tableau 3-15 : Résultats comparatifs pour la modification de la fonction objectif

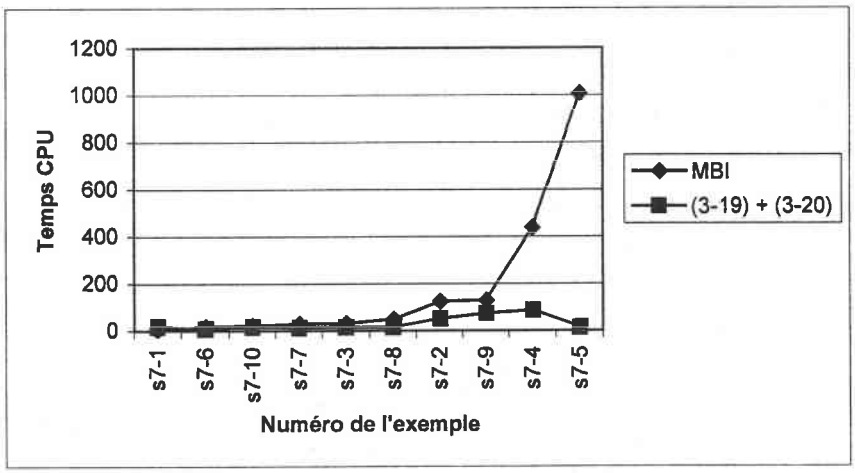

Figure 3-11 : Résultats comparatifs pour la modification de la fonction objectif

La modification de la fonction objectif conduit à une bonne amélioration du temps de calcul, la Figure 3-11 le montre. Pour cet exemple, les améliorations apportées par nos coupes sont meilleures (voir Figure 3-12).

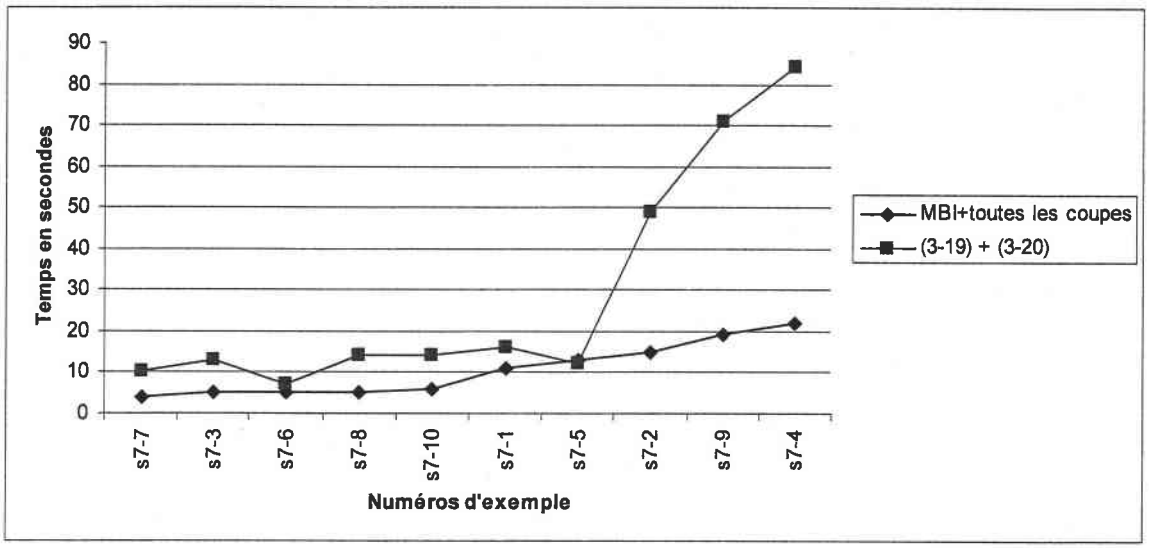

Figure 3-12 : Comparaison entre MBI avec les coupes et la fonction objectif modifiée

Dans le Tableau 3-16, nous étudions l'effet de la déclaration des variables Y_q et Z dans le dernier modèle avec (3-19) et (3-20) en tant que des variables réelles. Rappelons que le modèle utilisé fait appel aux intervalles $Q(j)$ et $K(j)$. Nous présentons les résultats pour la série s9 qui était la plus difficile pour le modèle de base. La première colonne du tableau donne le numéro de l'exemple, la deuxième le temps CPU lorsque l'on déclare ces variables comme binaires et la dernière colonne le temps CPU lorsque ces variables sont déclarées comme réelles. Le pourcentage de gain a été calculé selon la formule :

$$Gain = \frac{T(s) - T_{\min}(s)}{T(s)} \times 100\%.$$

Nous avons pu utiliser des variables réelles à la place de variables binaires pour Y_q et Z grâce à la structure du modèle que nous avons proposée. La Figure 3-13 présente deux courbes : en abscisses sont représentées les exemples triés par temps CPU croissant, en ordonnée, le temps (exprimé en secondes). La courbe supérieure représente les valeurs obtenues en utilisant des variables Y_q et Z binaires, l'autre courbe représente le comportement du temps de calcul en déclarant ces variables réelles.

ex	Temps CPU – *T(S)*		
	Y_q et Z binaires	Y_q et Z réelles	*Gain, %*
s9-1	22"	18"	18%
s9-2	9"	8"	11%
s9-3	6'53"	1'44"	75%
s9-4	10'28"	1'53"	82%
s9-5	4'37"	1'07"	76%
s9-6	1'12"	11"	85%
s9-7	50"	33"	34%
s9-8	1'01"	31"	49%
s9-9	1'13"	16"	78%
s9-10	2'54"	1'2"	64%

Tableau 3-16 : Résultats comparatifs de remplacement de variables sur la série s9

Nous avons pourtant constaté également, que dans certains autres exemples (celui par exemple de la série s7) le temps de calcul n'a pas été amélioré. L'effet de l'emploi de variables réelles à la place des variables binaires se manifeste quand l'exemple est d'une taille relativement grande avec un grand nombre de variables de ce type.

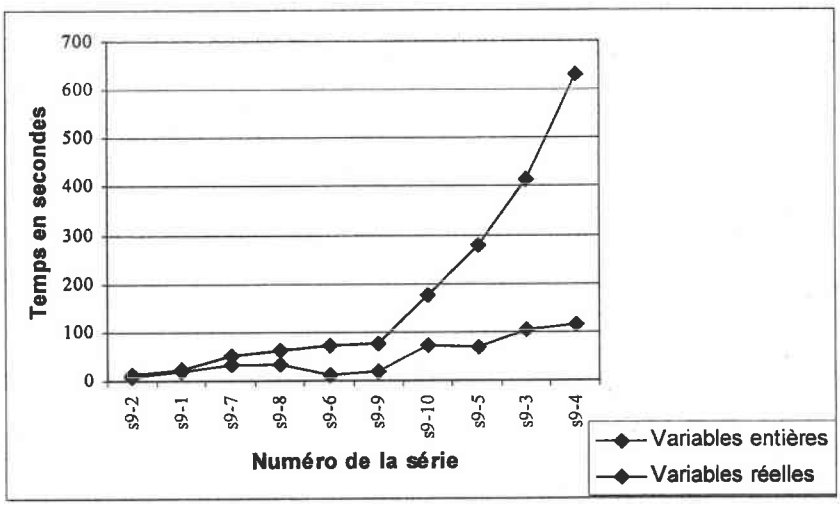

Figure 3-13 : Effet de substitution des variables entières par variables réelles (série s9)

Bilan des améliorations

Pour faire un bilan, nous commençons par donner les résultats obtenus en appliquant nos différentes coupes et améliorations sur les trois exemples industriels et deux exemples générés aléatoirement.

Les scénarios sont les suivants (dans tous les scénarios les coupes classiques de Cplex sont activées et les variables F_q, Y_q et Z_k sont déclarées comme réelles) :

Scénario 1 : modèle de base (MB) avec les intervalles $Q(j)$ et $K(j)$;

Scénario 2 : MB avec les intervalles $Q(j)$ et $K(j)$ et forçage des coupes Cplex ;

Scénario 3 : MB avec les intervalles $Q(j)$ et $K(j)$ et les coupes YZ ;

Scénario 4 : MB avec les intervalles $Q(j)$ et $K(j)$ et les coupes ES ;

Scénario 5 : MB avec les intervalles $Q(j)$ et $K(j)$, les coupes YZ et les coupes ES ;

Scénario 6 : Modèle avec la fonction objectif modifiée et les intervalles $Q(j)$ et $K(j)$;

Scénario 7 : Modèle avec la fonction objectif modifiée, les intervalles $Q(j)$ et $K(j)$ et les coupes Cplex forcées.

Le Tableau 3-17 présente les résultats de tests des scénarii ci-dessus. Les résultats de ces tests montrent que la meilleure solution est d'utiliser le modèle de base complété par les intervalles $Q(j)$ et $K(j)$ avec les coupes classiques de Cplex et les deux nouvelles coupes que nous avons proposées (scénario 5).

Scenario	$T(s)$				
	i1	**i2**	**i3**	**3**	**9**
1	2'14"	11"	6"	11"	2'26"
2	57"	3"	2'26"	3"	33"
3	10"	2"	33"	2"	7"
4	2'13"	11"	2'25"	11"	2'25"
5	**2"**	**1"**	**7"**	**1"**	**7"**
6	3"	4"	7"	4"	32"
7	9'24"	5"	13'21"	5"	34"

Tableau 3-17 : Résultats des différents scénarii

Nous montrons dans le Tableau 3-18, les résultats comparatifs de 10 exemples optimisés avec le modèle de base et avec le meilleur modèle, à savoir : scénario 5. Grâce à ces résultats numériques, nous pouvons mieux appréhender les améliorations que nous avons su apporter au modèle initial. Dans le Tableau 3-18, le temps CPU n'est pas systématiquement arrondi à la seconde, la valeur 0,03" par exemple signifie trois centièmes de secondes.

ex	*NO*	*NA*	*NC*	**Modèle de base** *T(S)*	**Meilleure méthode** *T(S)*	**Gain, temps**	**Gain, %**
1	6	6	139	0,3"	0,05"	0,25"	83%
2	9	10	314	1"	0,03"	0,97"	97%
3	11	11	669	41"	1"	40"	99,9%
4	13	17	826	1"	0,08"	0,92"	92%
5	15	15	952	54"	1"	53"	98%
6	18	26	912	5'52"	1"	5'51"	99,9%
7	23	39	1407	13'32"	3"	13'29"	99,9%
8	25	38	1966	1'17"	9"	1'8"	99,9%
9	30	44	3472	13h23'	7"	13h22'22"	99,9%
10	35	58	2223	22h56'	4'35"	22h51'25"	99,9%
s1	10	22	845	34"	6"	28"	93%
s4	15	15	2397	16'32"	1'14"	15'18"	88%
s7	20	50	1200	1'31"	11"	1'20"	94%
s9	25	87	2355	5h04'55"	1'59"	5h2'56"	99,9%
11	38	62	2579	>5h	17'45"	-	-
12	45	67	3037	>5h	35'23"	-	-
13	100	194	6593	>10h	>10h	-	-
14	120	214	8050	>10h	>10h	-	-
15	150	286	11680	>10h	>10h	-	-

Tableau 3-18 : Résultats avec la meilleure méthode

Le Tableau 3-18 montre que, pour tous les exemples que nous avons étudiés, la méthode MBI avec l'introduction de toutes les coupes est vraiment la plus efficace. Les gains en temps sont très importants. La dernière colonne du tableau donne le pourcentage de gain

par rapport à la méthode de base. Le pourcentage de gains est dans tous les cas supérieur à 80%, c'est-à-dire que le gain de temps est environ de 80% du temps de calcul du modèle de base (le temps est au minimum réduit de cinq fois). Malgré cela, le modèle amélioré reste trop lent pour pouvoir donner un résultat optimal pour les grands exemples (avec une taille supérieure à 100 opérations) en un temps inférieur à 10h.

La Figure 3-14 montre l'influence du nombre d'opérations sur le temps de calcul du meilleur modèle MIP que nous avons proposé. Bien sûr, l'exponentielle de la courbe va dépendre des densités des contraintes.

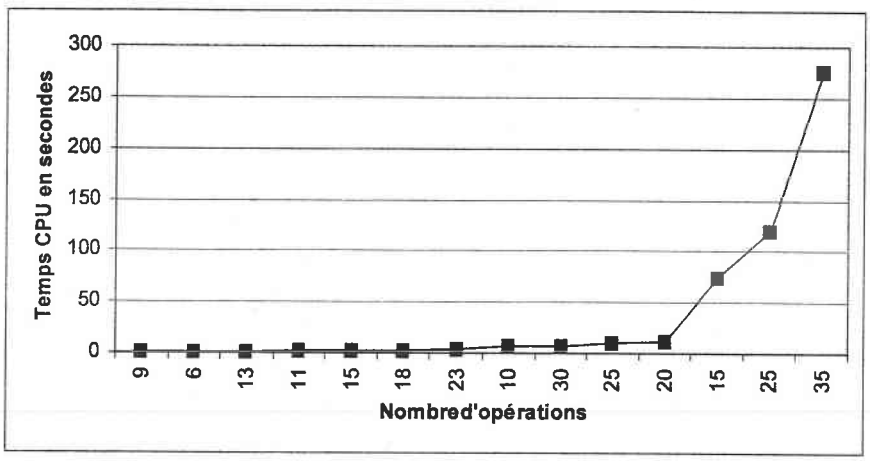

Figure 3-14 : Influence du nombre d'opérations sur le temps de calcul

Pour terminer, la Figure 3-15 montre une relation empirique entre le nombre de contraintes et le temps de calcul. Plus le nombre de contraintes est grand, plus le temps de calcul augmente. Pour rappel le nombre de contraintes pour le modèle de base MB est égal à $|N| + 3\,|N|\,n_0\,m_0 + n_0\,m_0 + m_0 + |ES|\,m0 + n_0\,m_0\,|\overline{EB}| + m_0\,|\overline{ES}|$ (voir le paragraphe 3.2.1). Le nombre de contraintes dépend à son tour du nombre de variables de décision. Il est donc très difficile de séparer les effets correspondants. L'analyse plus fine de ces paramètres reste à réaliser et cela fait partie des perspectives à étudier.

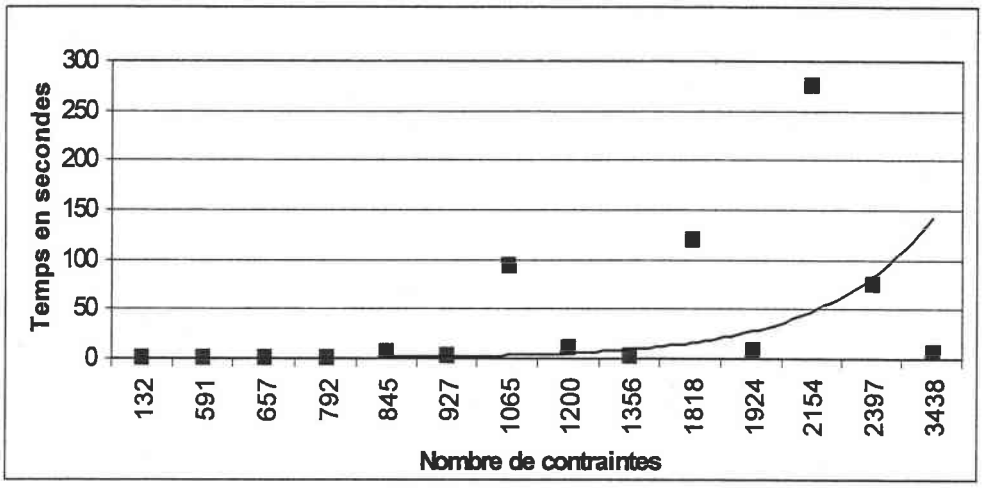

Figure 3-15 : Influence du nombre de contraintes sur le temps de calcul

3.4 Conclusions

Dans ce chapitre nous avons utilisé la programmation linéaire pour résoudre un nouveau problème de conception des lignes de transfert en usinage. Pour cela nous avons proposé un modèle en variables mixtes qui tient compte de l'ensemble de contraintes et qui est formulé de manière à optimiser le nombre de stations et têtes d'usinage. Nous avons montré les performances de ce modèle sur quatre exemples industriels, mais aussi sur un ensemble d'exemples de tests générés aléatoirement.

Les exemples industriels que nous avons étudiés sont plutôt de petite taille (19 opérations maximum). Pour pouvoir utiliser notre modèle pour les problèmes de taille plus grande, nous avons proposé et testé un ensemble de techniques. Parme ces techniques, nous avons utilisé :

- le remplacement de certaines variables binaires par des variables réelles ;
- l'agrégation des variables binaires ;
- la fixation des intervalles des indices possibles de numéro de bloc et de station limitant le nombre de variables binaires pour chaque opération ; ces intervalles sont obtenus

par une analyse plus approfondie des toutes les contraintes du problème en utilisant l'algorithme que nous avons développé ;

- les modifications de la fonction objectif forçant la recherche arborescente (pour les variables binaires) à ne pas parcourir des branches identiques ;
- l'utilisation forcée des coupes standard de Cplex ;
- l'ajout de nouvelles coupes dédiées à ce problème ;
- les modifications des paramètres du solver Cplex.

L'ensemble des techniques d'amélioration que nous avons proposées a permis d'accélérer sensiblement les calculs. Nous pouvons citer ici le meilleur résultat de nos tests : pour un exemple, nous sommes passé de plus de 13 heures de calcul à seulement 4 secondes. Le résultat le moins bon est celui pour lequel nous sommes descendus à 17 % du temps de calcul initial (c'est-à-dire que même dans ce cas, nous avons divisé par plus de 5 le temps de calcul). Ce résultat nous encourage à continuer à explorer les pistes possibles d'amélioration du modèle.

Nous avons obtenu l'amélioration la plus performante du modèle de base en utilisant le modèle MIP avec la fonction objectif de base, l'emploi de toutes les coupes standard de Cplex et deux coupes que nous avons proposées, et bien sûr en réduisant le nombre de variables en fixant les intervalles des indices de blocs et de stations et en définissant toutes les variables Y, Z et F en variables réelles. Notons également que les paramètres n_0 et m_0 influencent également la taille du modèle et donc le temps de calcul. Nous montrerons dans le chapitre 5 comment définir des bornes supérieures du nombre de stations et de ce fait réduire davantage la taille du problème.

L'emploi de notre modèle MIP permet de résoudre des problèmes de taille moyenne (jusqu'à 30 opérations) en un temps inférieur à quelques minutes. Si le budget de temps est fixé à 3 heures, nous pourrons résoudre les problèmes de taille environ 80 opérations.

Mais la complexité de problème fait qu'à un moment donné en augmentant la taille du problème traité, nous ne pourrons plus utiliser le modèle MIP proposé. En effet pour les exemples 13, 14 et 15 (c.-à-d. avec 100, 120 et 150 opérations), pour un temps alloué de 10h, notre modèle MIP n' a pas terminé la recherche d'une solution optimale. Nous avons donc besoin de développer comme un complément de ce modèle des méthodes heuristiques efficaces. C'est l'objectif du chapitre suivant.

Chapitre 4

Approches heuristiques

4. Approches heuristiques pour la conception et l'équilibrage de lignes de transfert

Pour des problèmes de taille moyenne le modèle MIP exposé dans le chapitre précédent donne des résultats exacts en un temps raisonnable. Mais si le temps alloué pour les calculs est critique ou si le problème traité est à grande échelle, les méthodes exactes ne suffisent plus. Dans ce cas, une solution possible consiste à utiliser des algorithmes heuristiques. Même s'ils ne garantissent pas l'optimalité, ces algorithmes (voir le Chapitre 2) donnent souvent de bons résultats en relativement peu de temps de calcul. Ils sont beaucoup moins sensibles à la taille du problème traité.

Dans ce chapitre, nous proposons des algorithmes heuristiques pour TLBP basés sur la méthode COMSOAL (Arcus, 1966). Cette méthode s'est avérée très performante pour le problème classique SALBP-1, de nombreux exemples de son utilisation efficace sont connus pour des instances industrielles et académiques d'une taille assez importante (Buffa et Taubert, 1972 ; Johnson et Montgomery, 1974 ; Nof *et al.*, 1997 ; Giard, 2003 ; Boutevin, 2003). Nous montrons comment l'approche de COMSOAL peut être adaptée et développée pour le problème TLBP qui se présente comme plus difficile avec des propriétés et contraintes nouvelles (Dolgui *et al.*, 2002d, Finel *et al.* 2003).

4.1. Heuristique COMSOAL pour SALBP-1

COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) est une procédure heuristique d'équilibrage des chaînes d'assemblage mono-produit (SALBP-1). Elle combine une construction progressive et déterministe de solutions, sans remise en cause des décisions antérieurement prises, avec une approche de type Monté Carlo : lorsqu'une décision à prendre pendant la construction comporte plusieurs possibilités, un choix aléatoire est

effectué. La méthode est itérative, elle génère un grand nombre de solutions faisables en conservant tout au long des itérations la meilleure solution trouvée.

Dans le Chapitre 2, nous avons donné une interprétation générale de l'algorithme COMSOAL, ici nous présentons son diagramme détaillé (Figure 4-1). Une itération de l'algorithme correspond à la construction d'une solution faisable. L'algorithme s'arrête au bout d'un nombre donné d'itérations.

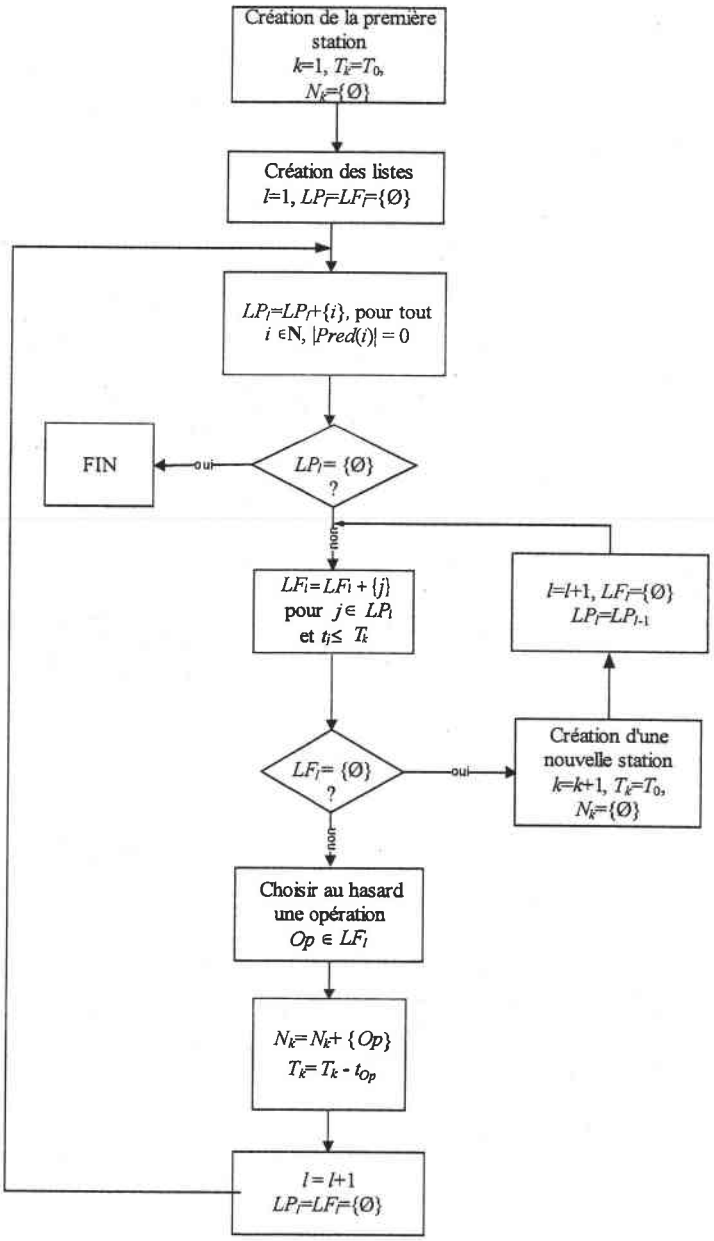

Figure 4-1: Diagramme de l'heuristique COMSOAL

Dans la Figure 4-1, T_k est *le temps libre de la station k.*

Montrons le déroulement d'une *itération* de l'algorithme COMSOAL pour l'exemple de la Figure 4-2 (Dolgui et Pashkevich, 2000). Les chiffres dans les cercles représentent les numéros des opérations et les chiffres au-dessus des cercles, les temps opératoires.

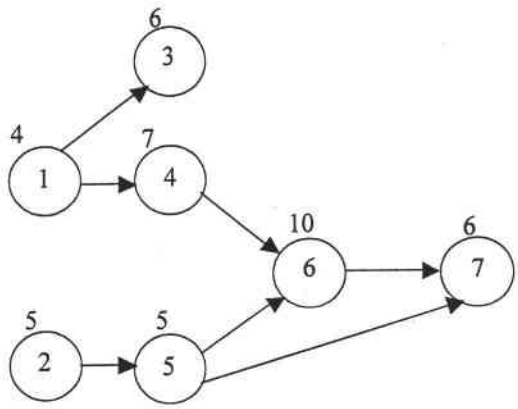

Figure 4-2 : Graphe de précédence

Au début de l'itération, une station vide est créée (la première station de la ligne). Le temps libre sur cette station T_1 est égal au temps de cycle $T_0 = 16$. Le temps libre est le temps de cycle moins la charge de la station correspondante, c'est-à-dire que $T_k = T_0 - t^s(N_k)$, $k= 1, 2, \ldots, m$ où $t^s(N_k)$ est le temps nécessaire pour réaliser l'ensemble N_k d'opérations affectées à la station k.

La liste LP_1 est constituée des deux opérations n'ayant pas de prédécesseurs $LP_1 = \{1, 2\}$, $N_1= \{\emptyset\}$. Les deux opérations de la liste LP_1 ont un temps inférieur au temps libre de la station courante T_1, alors $LF_1= \{1, 2\}$. Supposons que l'opération 1 est choisie aléatoirement dans la liste LF_1. Nous affectons cette opération à la première station, c'est-à-dire $N_1= \{1\}$. Le temps libre de la station est diminué de 4, $T_1 = 16 - t_1 = 16 - 4 = 12$.

La liste LP_2 est devenue $LP_2= \{2, 3, 4\}$ parce que les opérations 3 et 4 ont comme seul prédécesseur l'opération 1. Toutes les opérations de la liste LP_2 ont un temps inférieur au

temps libre de la station. Nous avons donc LF_2= {2, 3, 4}. Soit l'opération 3 choisie aléatoirement dans la liste LF_2. Alors $T_1 = 12 - t_3 = 12 - 6 = 6$, $N_1 = \{1, 3\}$.

La liste $LP_3 = \{2, 4\}$ et la liste $LF_3 = \{2\}$, car le temps de l'opération 4 est supérieur à T_1. L'opération 2 est choisie dans la liste LF_3. Alors $N_1 = \{1, 3, 2\}$, $T_1 = 6 - t_2 = 6 - 5 = 1$.

La liste $LP_4 = \{4, 5\}$, mais la liste LF_4 est vide. Il faut donc créer une deuxième station : $N_2 = \{\emptyset\}$, $T_2 = 16$, $LP_5 = \{4, 5\}$ et $LF_5 = \{4, 5\}$. Soit l'opération 5 choisie aléatoirement dans la liste LF_5. Alors $N_2 = \{5\}$, $T_2 = 16 - t_5 = 16 - 5 = 11$.

Les listes $LP_6 = \{4\}$ et $LF_6 = \{4\}$, l'opération 4 est choisie dans la liste LF_6. Alors $N_2 = \{4, 5\}$, $T_2 = 11 - t_4 = 11 - 7 = 4$.

La liste $LP_7 = \{6\}$, mais la liste LF_7 est vide. Nous créons une troisième station : $N_3 = \{\emptyset\}$, $T_3 = 16$ et les listes $LP_8 = \{6\}$ et $LF_8 = \{6\}$. L'opération 6 est choisie dans la liste LF_8. Alors $N_3 = \{6\}$, $T_3 = 16 - t_6 = 16 - 10 = 6$.

La liste $LP_9 = \{7\}$ et la liste $LF_9 = \{7\}$, nous affectons cette dernière opération à la troisième station (station courante) $N_3 = \{6, 7\}$. L'itération est terminée. Dans la solution obtenue (solution réalisable) le nombre de stations est égal à trois. Le temps mort total de la ligne est égal à 5.

Cette itération est présentée sous la forme de branche d'arbre dans la Figure 4-3. Chaque cercle noirci correspond à une étape (un pas) de l'itération.

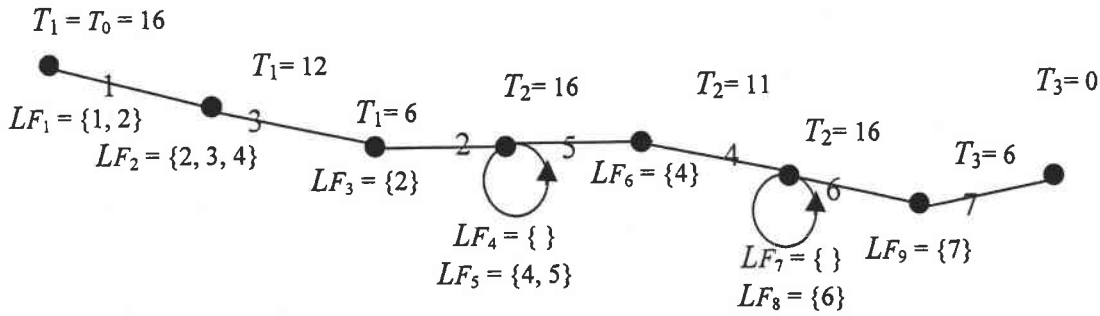

Figure 4-3 : Déroulement d'une itération de l'algorithme COMSOAL

Pour avoir le résultat final de l'algorithme, il faut répéter les itérations depuis le début un nombre suffisamment grand de fois en gardant toujours la meilleure solution.

4.2. *Heuristiques pour TLBP basées sur COMSOAL*

4.2.1. Présentation générale

La différence (et la difficulté) de l'application de l'approche COMSOAL pour TLBP par rapport à SALBP consiste à prendre en compte de la notion de blocs (opérations parallèles) et de contraintes complémentaires relativement complexes souvent en interaction. Nous proposons deux heuristiques pour TLBP basées sur COMSOAL.

Soient les notations suivantes :

n_k est le nombre de blocs de la station k ;

B_q est l'ensemble des opérations du bloc q, c'est-à-dire $B_q = N_{kl}$, pour $q = (k\text{-}1)\ n_0 + l$;

$C(k)$ est le coût de la station k avec tous ses blocs, c'est-à-dire $C(k) = C_1 + C_2 n_k$;

k est le numéro de station, k=1, 2, …, m.

Nous appelons « station courante » et « bloc courant » la station et le bloc auxquels nous affectons des opérations à l'étape courante de l'algorithme. Tant que le bloc courant n'est pas

rempli complètement (tant que nous pouvons y ajouter des opérations), nous ne créons pas de bloc suivant. Il en va de même pour les stations. Une fois qu'un bloc (une station) est créé(e), il (elle) devient courant(e). Par définition, le bloc courant appartient à la station courante.

Tous les algorithmes que nous proposons utilisent les trois listes suivantes (ici nous ne donnons plus l'indice de l'étape l comme pour LP_l et LF_l dans la description de COMSOAL ci-dessus, mais uniquement le type de la liste par son numéro) :

L_1 associe à chaque opération i l'ensemble $L_1(i)$ de ses successeurs directs ;

L_2 associe à chaque opération i le nombre courant $L_2(i)$ de ses prédécesseurs directs non affectés ;

L_3 regroupe toutes les *opérations candidates* à l'affectation, c'est-à-dire :

- les opérations qui n'ont plus de prédécesseurs directs non affectés,
- tous leurs successeurs s'il n'y a pas de contraintes d'exclusion de station avec une opération déjà dans la liste L_3,
- les opérations devant être sur la même station qu'une opération déjà dans la liste L_3 (notons que nous avons déjà regroupé les opérations devant être dans le même bloc en macro-opérations au préalable à l'optimisation).

En d'autres termes, les opérations de la liste L_3 sont celles n'ayant plus de contraintes de précédence (comme dans COMSOAL), mais également les opérations pouvant ou devant être affectées à la station courante avec ces dernières (en tenant compte de la spécificité des contraintes de précédence et de contraintes complémentaires du problème TLBP).

Dans tous les algorithmes proposés, l'opération i est placée dans L_3 si :

(a) tous les prédécesseurs de i sont soit déjà affectés soit dans la liste L_3 ;

(b) le temps opératoire t_i est inférieur au temps libre T_k de la station courante k, c'est-à-dire :

$$t_i \leq T_k = T_0 - \sum_{q' \in S(k),\, q' < q} F_{q'},$$

où q est l'indice du bloc courant ;

(c) les contraintes d'exclusion dans les stations $\overline{ES}$ ne risquent pas d'être violées par ce placement (si nous ajoutons l'opération i à la station courante k qui a déjà un certain nombre d'opérations affectées, ces contraintes sont respectées).

Donnons le schéma général d'utilisation de nos algorithmes. Soient TR_{tot} le nombre d'itérations déjà effectuées et TR_{nimp} le nombre d'itérations qui n'améliorent pas la solution. Soient C_{min} le coût de la solution la meilleure et C le coût de la solution courante.

Notons que dans nos algorithmes, vu l'interaction des contraintes du problème traité, nous pouvons avoir des *itérations non concluantes*, c'est-à-dire des itérations au cours desquelles nous ne pouvons pas aller jusqu'au bout de la construction de la solution, à cause des contraintes qui ne pourront plus être respectées, étant donnée la solution partielle obtenue. Nous obtenons donc l'algorithme suivant.

Algorithme 4-1 :

***Etape* 0.** $C_{min} = \infty$, $TR_{tot} = 0$, $TR_{nimp} = 0$.

***Etape* 1.** Faire une itération de l'algorithme en essayant d'affecter toutes les opérations de l'ensemble **N** avec le respect de toutes les contraintes ; déterminer le coût C de la solution construite ($C = \infty$ si l'itération n'est pas concluante).

***Etape* 2.** Si $C_{min} > C$ alors $C_{min} = C$, garder la solution courante comme étant la meilleure solution, sinon incrémenter le compteur $TR_{nimp} = TR_{nimp} + 1$.

***Etape* 3.** Incrémenter le compteur $TR_{tot} = TR_{tot} + 1$.

***Etape* 4.** Arrêter l'algorithme si une des conditions suivantes est atteinte :

un temps d'exécution donné est dépassé,

TR_{tot} est supérieur au nombre maximum d'itérations autorisé,

TR_{nimp} est supérieur à une valeur donnée,

C_{min} est inférieur à une valeur du coût donné.

Sinon aller à *Etape* 1.

L'étape 1 ci-dessus fait appel à un des deux algorithmes que nous proposons dans ce chapitre. Ces algorithmes sont basés sur une séquence de choix aléatoires et cette dernière dépend des règles de constitution des listes des opérations candidates à l'affectation. Comme dans COMSOAL, dans les algorithmes de base que nous proposons (RAP et FSIC), dans chaque itération, nous ne remettons pas en cause les choix faits tant que nous n'obtenons pas une solution réalisable complète (par la suite, nous montrerons comment nous pouvons nous affranchir de cette règle de COMSOAL). L'ensemble de tous les choix possibles peut être représenté sous la forme d'un arbre d'énumération (Figure 4-4).

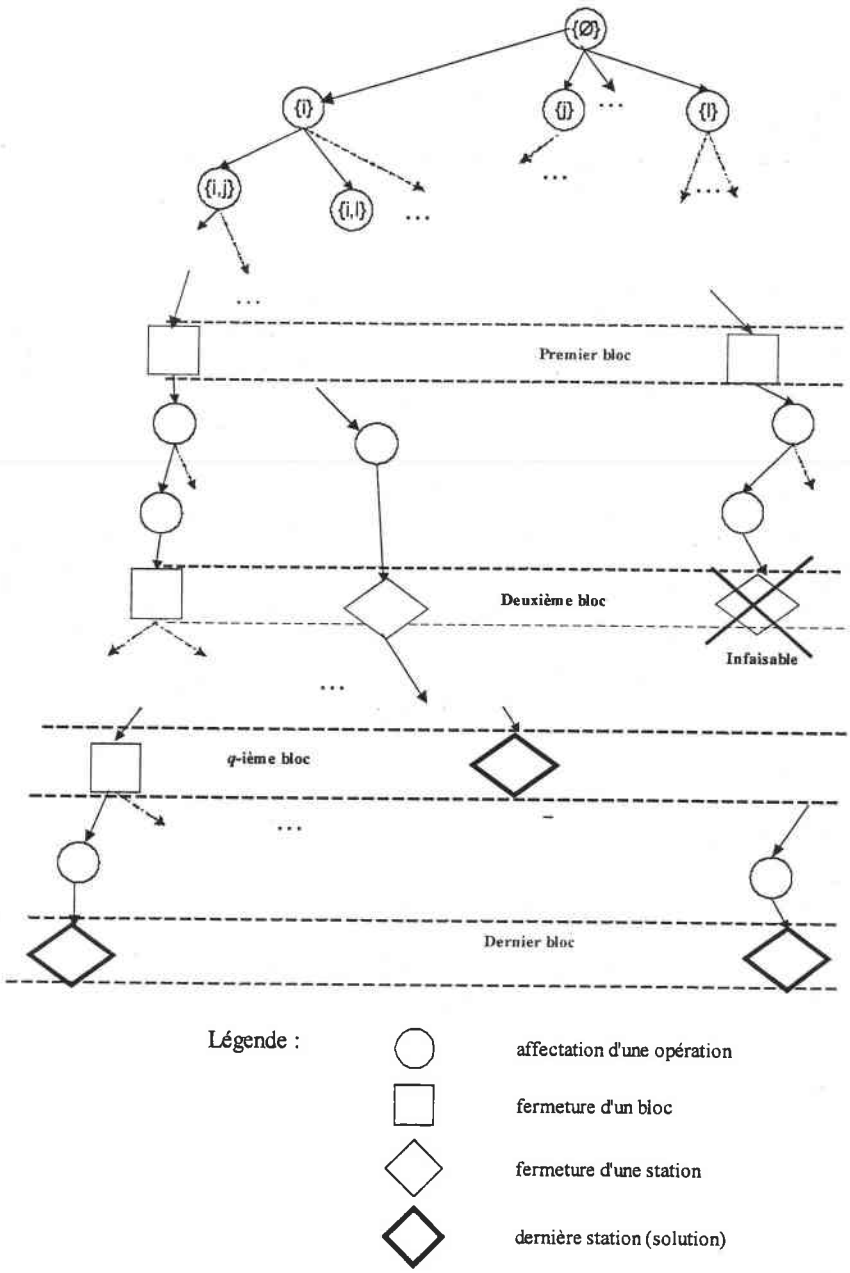

Figure 4-4 : Illustration de l'arbre d'énumération

A la racine de l'arbre nous associons un ensemble vide d'opérations. Chaque arc signifie l'affectation d'une opération au bloc courant de la station courante. Chaque sommet de l'arbre d'énumération est donc caractérisé par l'ensemble des opérations qui sont déjà affectées.

Les sommets représentés par des carrés (dans la Figure 4-4) marquent les passages d'un bloc à un autre (fermeture du bloc courant et ouverture d'un nouveau bloc qui devient courant). Les sommets représentés par des losanges marquent la création d'une nouvelle station qui devient la station courante.

Les branchements sont liés au choix de l'opération à affecter parmi les opérations candidates. Chaque feuille de l'arbre pour laquelle l'ensemble des opérations affectées est l'ensemble **N** donne une *solution réalisable*. Chaque feuille de l'arbre pour laquelle il reste encore des opérations non affectées, représente un arrêt de la construction : on ne peut plus avoir de solution réalisable sur cette branche (présentée dans la Figure 4-4 par un losange barré). En d'autres termes, si une branche est parcourue jusqu'à ce que toutes les opérations soient affectées en respectant toutes les contraintes, une solution est trouvée ; si aucune opération ne peut être rajoutée à la branche, et que toutes les opérations ne sont pas affectées, nous parlons d'un échec et d'une *itération non concluante*.

Les deux algorithmes de base que nous proposons (RAP et FSIC) construisent, à chaque itération, une seule branche de cet l'arbre. Une fois l'itération terminée (concluante ou non), ils reviennent à la racine. Les algorithmes RAP et FSIC se différencient principalement par la manière de sélectionner une opération dans la liste des opérations candidates quand il y a un choix à faire (c'est-à-dire par la façon d'effectuer le branchement dans l'arbre d'énumération).

4.2.2. Heuristique RAP

La première heuristique, nommée RAP (Recursive Assignment of Predecessors), tente de généraliser la méthode COMSOAL directement.

Une des particularités de TLBP est que les opérations liées par des contraintes de précédence peuvent se trouver ensemble dans la liste des opérations candidates (c'est-à-dire que dans la liste L_3, qui est en quelque sorte l'équivalent de F_l pour COMSOAL, peuvent se trouver une opération et ses prédécesseurs). Alors, le déroulement de l'algorithme est le suivant :

- à chaque itération, une opération appartenant à L_3 est choisie aléatoirement pour être affectée au bloc courant. Si cette opération a des prédécesseurs dans L_3, alors il faut commencer par affecter ses prédécesseurs (RAP - Recursive Assignment of Predecessors) puis l'opération elle-même ;
- si l'opération qui a été choisie pour être affectée à la station courante est liée par une contrainte d'inclusion concernant les stations (elle fait partie d'un ensemble $e \in ES$), alors nous devons nous assurer que toutes les autres opérations appartenant à e sont affectées à la même station avant d'en créer une autre (rappelons qu'elles sont dans L_3 au début de l'étape correspondante).

La deuxième remarque ci-dessus explique ce que nous appelons « *itération non concluante* ». Il s'agit du cas où il reste dans L_3 des opérations liées par des contraintes d'inclusion avec des opérations déjà affectées. Il n'est plus possible de les rajouter à la station courante. Une nouvelle station doit être créée. Nous constatons donc un échec de l'itération et nous lançons une nouvelle itération depuis le début.

L'affectation de l'opération choisie (soit aléatoirement dans la liste L_3 soit en tant que prédécesseur d'une opération choisie aléatoirement) se fait :

- dans le bloc courant (s'il n'y a pas de contrainte d'exclusion entre cette opération et les opérations déjà affectées au bloc courant) ;
- si ce n'est pas possible, alors dans un nouveau bloc qu'il faut créer sur la même station (s'il y a une contrainte d'exclusion pour le bloc courant mais que le temps libre sur la station est supérieur au temps opératoire de l'opération et qu'il n'y a pas de contrainte d'exclusion de station entre cette opération et les opérations déjà affectées à la station courante) ;
- sinon dans un nouveau bloc sur une nouvelle station (si les conditions précédentes ne sont pas respectées) ; dans ce cas, avant de fermer la station courante et de créer une

nouvelle station, les contraintes d'inclusion sont vérifiées et le cas échéant, les opérations concernées par ces contraintes sont rajoutées à la station courante.

Quand une opération est affectée à un bloc, les listes L_2 et L_3 sont mises à jour. Une itération de cet algorithme donne soit une solution réalisable soit une solution non réalisable (itération non concluante).

Nous présentons l'heuristique RAP sous forme d'une procédure principale (Algorithme 4-2) et de cinq procédures (*NouvelleStation, NouveauBloc, ChercherAssigner, Assignment*1 et *AffecterAuBloc*).

Algorithme 4-2 :

***Etape* 0.** Initialiser $k = 1$, $C = 0$.

***Etape* 1.** Lancer la **procédure** *NouvelleStation* pour créer la station k.

***Etape* 2.** Construire la liste L_3.

***Etape* 3.** Si L_3 n'est pas vide, alors tirer aléatoirement une opération Op dans L_3 et aller à l' *étape* 4 ; sinon aller à l' *étape* 7.

***Etape* 4.** Si l'opération Op a des prédécesseurs non affectés ($L_2(Op) > 0$), alors faire un appel récursif permettant d'affecter tous ses prédécesseurs d'abord, et cette opération ensuite, en appelant la **procédure** *ChercherAssigner*. Si cette affectation n'est pas possible, alors aller à l'*étape* 5, sinon aller à l'*étape* 2.

***Etape* 5.** Vérifier, pour chaque opération non affectée de la liste L_3, si elle est liée ou non par une contrainte d'inclusion dans la station courante (devant être exécutée en même temps qu'une opération déjà sur la station). Si oui, essayer de l'affecter à la station courante. Si ce n'est pas possible, alors l'algorithme s'arrête (l'itération est non concluante).

***Etape* 6.** Augmenter le coût C du coût de la station k et de ses blocs, i.e. de $C(k)$.

***Etape* 7.** Si toutes les opérations de **N** sont placées, alors l'itération se termine, nous avons une nouvelle solution faisable ; sinon, $k=k+1$. Si $k > m_0$, alors l'itération s'arrête (l'itération est non concluante) ; sinon, aller à l'*étape* 1.

La procédure *NouvelleStation* crée une nouvelle station (station k) :

NouvelleStation

$n_k = 0, T_k = T_0, S(k) = \{\varnothing\}, C(k) = C_1, NS = 1, q = (k - 1)n_0$

appeler *NouveauBloc*(0)

finNouvelleStation

La procédure *NouveauBlock* crée un nouveau bloc (la variable *NB* est égale à 1 si c'est possible, elle est égale à 0, sinon).

NouveauBloc(τ)

si ($n_k > 0$) **alors**

$T_k = T_k - F_q$ (la durée du bloc est soustraite du temps libre de la station)

finsi

$n_k = n_k + 1$

si ($n_k > n_0$ or $T_k < \tau$) **alors** $NB = 0$

sinon

$q = q+1, F_q = 0, B_q = \{\varnothing\}, S(k) = S(k) \cup \{q\}, C(k) = C(k) + C_2, NB = 1$

finsi

finNouveauBloc

La procédure *ChercherAssigner* permet d'affecter de manière récursive tous les prédécesseurs de l'opération *Op* en les cherchant et les traitant comme opération courante (*Op* étant la dernière à être affectée).

ChercherAssigner

(*Op* est un prédécesseur non affecté)

si ($L_2(Op) \neq 0$) **alors**

(tant que *Op* a un prédécesseur non affecté, faire un appel récursif pour chercher et affecter tous ses prédécesseurs)

Appeler ChercherAssigner

finsi

Appeler Assignement1

finChercherAssigner

Assignement1

Appeler *AffecterAuBloc*(*Op*, 1) (voir plus loin)

si (*ASGN* ≠ 1) **alors**

si (*ASGN* = -1) **alors** (l'opération *Op* ne peut pas être dans le bloc courant mais peut être affectée sur la station courante)

Appeler *NouveauBloc*(t_{Op})

si (*NB* ≠ 1) **alors** retourner 0

sinon (un nouveau bloc est créé)

Appeler *AffecterAuBloc*(*Op*, 0)

finsi

sinon retourner 0

finsi

finsi

si (l'opération *Op* doit être affectée à la même station qu'un ensemble d'opérations de L_3 appartenant également à $e \in ES$) **alors**

pour toutes les opérations $Op' \in e \setminus \{Op\}$ telles que $L_2(Op') = 0$

Appeler *AffecterAuBloc*(*Op'*, 1)

si (*ASGN* ≠ 1) alors

si (*ASGN* = - 1) **alors** (l'opération ne peut pas être dans le bloc courant)

Appeler *NouveauBloc*($t_{Op'}$)

si (*NB* = 0) **alors** retourner 0

sinon

Appeler *AffecterAuBloc* (*Op'*, 0)

finsi

sinon retourner 0

finsi

finsi

finpour

finsi

retourner 1

finAssignement1

La procédure *Assignment*1 permet d'affecter l'opération *Op* à la station *k*. Cette procédure retourne 1 si l'affectation est possible, 0 sinon.

La procédure *AffecterAuBloc* permet d'affecter l'opération *Op* au bloc *q*. Dans cette procédure la variable auxiliaire *ASGN* = 1, s'il est possible d'affecter l'opération *Op* au bloc *q*, *ASGN* = - 1, si les contraintes d'exclusion pour le bloc *q* ne sont pas respectées, et *ASGN* = - 2 si les contraintes d'exclusion pour la station *k* ne sont pas respectées.

AffecterAuBloc (*Op, check*)

si (*check* = 0) **alors** *ASGN* = 1

sinon

ASGN = 1

si ($T_k < t_{Op}$) ou $\exists\ e \in \overline{ES}$ tel que $e \subseteq (\{Op\} \cup \bigcup_{q \in S(k)} B_q)$ **alors**

ASGN = -2

sinon

si $\exists e \in \overline{EB}$ tel que $e \subseteq B_q \cup \{Op\}$ **alors**

ASGN = -1

finsi

finsi

finsi

si (*ASGN* = 1) **alors**

$F_q = \max(t_{Op}, F_q)$, $B_q = B_q \cup \{Op\}$

pour tout $i \in L_1(Op)$, $L_2(i) = L_2(i) - 1$

finsi

finAffecterAuBloc

4.2.3. Heuristique FSIC

L'algorithme RAP ne limite pas les choix possibles de l'opération à affecter à chaque étape. Comme nous l'avons déjà souligné, cela peut conduire à des itérations non concluantes. Un nombre trop important d'itérations non concluantes peut ralentir les calculs de manière significative. Par conséquent, la qualité du résultat diminue : pour un temps alloué, la

meilleure solution trouvée n'est pas suffisamment bonne à cause d'une très grande perte de temps pour des itérations non concluantes.

Pour essayer de réduire le nombre des itérations non concluantes et donner plus de temps à la génération des solutions faisables (pour un temps alloué fixe), nous proposons un second algorithme (Dolgui *et al.*, 2002c) que nous appelons FSIC (First Satisfy Inclusion Constraints). Dans cet algorithme, toutes les opérations devant se trouver sur la même station ainsi que tous leurs prédécesseurs non encore affectés sont traités en priorité. L'algorithme FSIC utilise *une quatrième liste* L_4. Cette liste regroupe toutes les opérations qui doivent être affectées à la même station.

Soit *Op* une opération choisie aléatoirement dans la liste L_3 parmi celles de cette liste qui n'ont plus de prédécesseurs non affectés. Nous plaçons *Op* au début de la liste L_4 et nous la supprimons de la liste L_3. Ensuite, nous analysons les contraintes d'inclusion et de précédence de manière récursive. Nous mettons dans la liste L_4 toutes les opérations de chaque ensemble $e \in ES$ dont *Op* fait partie ; et par récursivité, toute opération, non encore affectée, liée par des contraintes d'inclusion ou/et de précédence avec des opérations déjà dans la liste L_4 (les prédécesseurs sont placés avant l'opération qu'elle précèdent). Si une opération que nous mettons dans L_4 appartient à L_3, nous la supprimons de L_3.

L'algorithme tente alors de placer sur la station courante les opérations de la liste L_4 une par une, dans l'ordre dans lequel elles sont dans la liste. Si cela s'avère impossible pour une opération, alors l'algorithme *annule toutes les affectations d'opérations faites de cette liste.* Ensuite l'algorithme ferme le bloc et la station courants. Il crée une nouvelle station qui devient courante et il essaie d'affecter toutes les opérations de la liste L_4 de nouveau, cette fois-ci à la nouvelle station où il n'y a pas d'autres opérations affectées avant celles de la liste L_4. Si cette deuxième tentative d'affecter toutes opérations de L_4 sur la même station échoue également, alors (et seulement alors) nous disons que l'itération n'est pas concluante, nous la terminons et nous commençons l'itération suivante.

Quand L_4 est vide, nous revenons à la liste L_3 pour en tirer au hasard une nouvelle opération *Op* (voir ci-dessus) qui initie la création d'une nouvelle liste L_4, etc.

L'algorithme FSIC peut donc être représenté de la manière suivante :

Algorithme 4-3 :

***Etape* 0.** Initialiser k=1, C=0 et lancer la **procédure** *NouvelleStation* pour créer une nouvelle station k.

***Etape* 1.** Construire la liste L_3.

***Etape* 2.** Choisir aléatoirement dans L_3 une opération *Op* telle qu'elle n'ait pas de prédécesseurs non affectés, la supprimer de L_3 et la placer dans la liste L_4. Si aucune opération *Op* de L_3 ne répond à ce critère, aller à l'étape 6.

***Etape* 3.** Pour tous les ensembles $e \in ES$ qui contiennent *Op*, placer toutes les autres opérations et leurs prédécesseurs non affectées dans la liste L_4 en les supprimant de L_3.

***Etape* 4.** *Sauvegarder* l'état courant de la station k. Essayer d'affecter à la station k toutes les opérations de la liste L_4 une par une en utilisant **la procédure** *Assignment2*. Dès qu'une opération de L_4 ne peut pas être affectée, aller à l'étape 5. Aller à l'étape 1.

***Etape* 5.** *Restaurer* l'état précédent de la station k (sauvegardé au début de l'étape 4). Ajouter le coût $C(k)$ à C. Incrémenter k de 1. Si $k > m_0$, alors aucune solution ne peut être trouvée dans cette itération, l'itération est terminée, sinon créer une nouvelle station k par la **procédure** *NouvelleStation*. Essayer d'affecter à la nouvelle station k toutes les opérations de la liste L_4 une par une en utilisant **la procédure** *Assignment2.* Si ce n'est pas possible, alors aller à l'étape 6, sinon aller à l'étape 1.

***Etape* 6.** L'itération est terminée ; si toutes les opérations de **N** sont affectées, alors une solution faisable est trouvée, sinon l'itération est non concluante.

La procédure *Assignment2* affecte toutes les opérations de la liste L_4 sur la station k. Cette procédure retourne 1 si l'affectation de toutes les opérations de L_4 est possible, 0 sinon.

Une modification possible de l'algorithme *Assignment2* consiste à choisir l'opération *Op* de L_4 de manière aléatoire.

Assignment2

Répéter

Choisir l'opération suivante *Op* de L_4 telle qu'elle n'ait plus de prédécesseurs non affectés ($L_2(Op) = 0$)

Appeler *AffecterAuBloc*(*Op*, 1)

si ($ASGN \neq 1$) **alors**

si ($ASGN = -1$) **alors** (l'opération ne peut pas être dans le bloc courant)

Appeler *NouveauBloc*(t_{Op})

si ($NB \neq 0$) **alors** retourner 0

sinon Appeler *AffecterAuBloc*(*Op*, 0)

finsi

sinon retourner 0

finsi

finsi

Jusqu'à ce que toutes les opérations de L_4 aient été affectées

retourner 1

fin*Assignment2*

4.3. Améliorations possibles des algorithmes

4.3.1. Affectation des opérations « au plus tôt » (ESB)

La règle la plus simple qui vient à l'esprit est d'essayer de ne pas affecter systématiquement les opérations dans le bloc courant mais chercher un bloc le plus en amont qui peut encore l'accueillir. Dans ce cas, à chaque fois qu'une opération *Op* est choisie pour être affectée, nous regardons s'il est possible ou non de l'affecter à un bloc au plus tôt même si ce dernier est déjà fermé. Pour cela nous tentons d'affecter l'opération *Op* à partir du bloc $q^-[Op]$, en respectant toutes les contraintes. Si cela n'est pas possible, nous essayons de

l'affecter au bloc immédiatement supérieur jusqu'à atteindre le bloc courant. Puis nous continuons notre algorithme normalement à partir du bloc courant. Nous espérerons ainsi améliorer la solution en minimisant le critère (ou le temps de calcul pour la même valeur du critère). Nous appelons ce type d'amélioration ESB (*Earliest Start Block*).

4.3.2. Remontée dans l'arbre d'énumération avec apprentissage (BAL)

Une autre possibilité d'améliorer les algorithmes RAP et FSIC que nous voulons tester ici est liée avec l'utilisation des retours en arrière dans l'arbre d'énumération. Dans ce cas, nous avons des algorithmes par construction mais avec possibilité de remise en cause des décisions prises antérieurement.

Recherche dans l'arbre d'énumération

Schéma général de l'algorithme

Nous utilisons la notation suivante des paramètres de l'algorithme que nous proposons dans ce paragraphe :

Nbl est le nombre minimum de blocs (nombre de niveaux) à remonter ;

Prob est la probabilité de ne pas remonter vers le bloc précèdent ;

Random est une variable aléatoire uniformément distribuée entre 0 et 1 ;

ProbInit est le paramètre de l'algorithme donnant la valeur initiale de *Prob* ;

Nniv est le compteur du nombre de niveaux remontés ;

Nbra(v) est le nombre maximum de branchements autorisés à partir du sommet v ;

NdblInit est un paramètre donnant le nombre de tentatives de déblocage autorisées ;

Ndbl est le compteur du nombre de déblocages ;

CouvArb est un paramètre permettant de calculer le nombre de branchements autorisés à partir de chaque sommet ;

Niv(v) est le numéro d'ordre de la dernière opération dans la séquence d'affectation menant de la racine à v ;

Poids(i) est le poids de l'opération i (utilisé pour calculer la probabilité de tirage de l'opération i) ;

Coef1 et *Coef2* sont des paramètres du modèle.

La remontée dans l'arbre d'énumération peut utiliser l'historique de la recherche pour pouvoir « favoriser » les retours dans les sommets à partir desquels nous avons déjà trouvé de bonnes solutions et rendre moins probables les retours dans les branches où nous avons eu des échecs (itérations non concluantes, par exemple). Il s'agit alors de l'« apprentissage ».

Nous appelons BAL (*Backtracking And Learning*) la remontée dans l'arbre d'énumération avec « apprentissage ». Nous présentons maintenant quelques techniques de BAL que nous utiliserons par la suite conjointement avec nos algorithmes RAP et FSIC.

Nous disons qu'une branche est parcourue jusqu'au bout, en d'autres termes que l'*itération courante est terminée*, si :

- soit la borne inférieure (*LB*) du sommet courant de la branche est supérieure à la valeur du critère pour la meilleure solution connue à ce stade (borne supérieure –*UB*) ; il est alors sûr que si le parcours de la branche se poursuit, la solution finale (quand toutes les opérations sont affectées), sera moins bonne que *UB* : par conséquent cette *branche est stoppée* ;
- soit toutes les opérations ne sont pas affectées mais aucune opération ne peut plus être rajoutée à la branche courante en respectant les contraintes, nous avons une *itération non concluante* ;
- soit toutes les opérations sont affectées, c'est-à-dire qu'*une solution faisable* est trouvée.

Dans le premier cas de figure ci-dessus, nous commençons l'itération suivante dès la racine de l'arbre d'énumération (c.-à-d. là où aucune opération n'est affectée) quel que soit le sommet dans lequel la branche courante est stoppée. Pour le deuxième et troisième cas de figure, nous proposons quelques techniques de remontée bloc par bloc.

Dans la Figure 4-5 nous montrons le schéma général de branchement avec BAL :

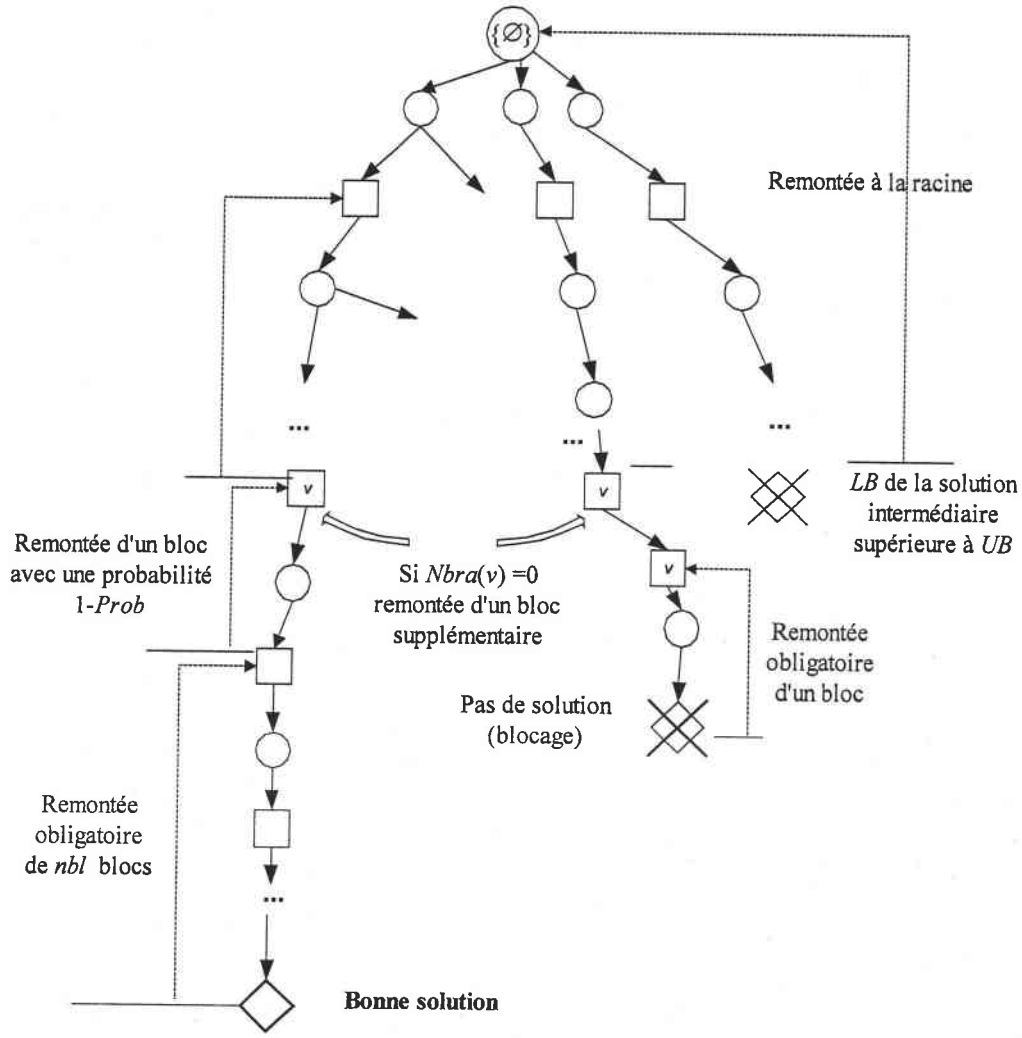

Figure 4-5 : Schéma de branchement avec BAL

Principe de la remontée

Dans le cas de figure où nous avons une solution faisable, pour la nouvelle itération, nous essayons d'en garder une partie, nous remontons donc la branche courante. Quand la décision d'arrêter la remontée est prise, un nouveau branchement est fait à partir du sommet de l'arbre où nous sommes remontés. Une telle nouvelle itération a des chances d'être plus rapide que celle qui commence de la racine, car elle ne nécessite pas l'affectation de toutes les opérations. Elle réutilise également des bonnes décisions prises au début de l'itération précédente. Le *taux de réutilisation* doit dépendre de la qualité des solutions précédentes obtenues sur la branche. Plus les solutions sont bonnes, plus tôt nous arrêtons la remontée (moins de blocs de la dernière solution sont remis en cause).

Nous remontons la branche bloc par bloc (nous ne nous arrêtons pas à tous les sommets mais uniquement là où un bloc a été fermé). Le *nombre minimum de blocs* (nombre de niveaux) *Nbl* à remonter est un paramètre de l'algorithme. De manière générale, nous *devons remonter au minimum* 2 *blocs* pour revenir à un état à partir duquel nous pouvons obtenir une solution différente de celle que nous avons. L'utilisateur doit pouvoir choisir la valeur de *Nbl* en la fixant plus grande ou égale à celle de ce seuil : $Nbl \geq 2$.

Pour les remontées supplémentaires, nous introduisons une nouvelle variable *Prob* qui définit la probabilité de ne pas remonter vers le bloc précèdent (bloc au-dessus). Il n'y a qu'une seule variable *Prob* dans l'algorithme. Nous tirons au hasard une variable *Random* (uniformément distribuée entre 0 et 1). Si $Random<(1-Prob)$ et que les autres conditions d'arrêt ne sont pas remplies, nous remontons d'un niveau complémentaire (nous annulons un bloc et prenons son prédécesseur) et ainsi de suite.

Le paramètre *Nbl* donne le nombre de niveaux à remonter (blocs à annuler) *obligatoire*. Tandis que la probabilité *Prob* introduit un choix aléatoire pour avoir la possibilité de remonter un certain nombre de niveaux supplémentaires. La valeur initiale de la variable *Prob* est donnée par *ProbInit* qui est un paramètre de l'algorithme. Chaque nouvelle solution faisable change la valeur courante de la probabilité *Prob*. Plus la solution obtenue est bonne, plus la probabilité *Prob* croît, et inversement, plus la solution obtenue est mauvaise plus la probabilité *Prob* décroît (voir Figure 4-6). Si $(1-Prob)$ est égale à 1, nous remontons automatiquement à la racine. Quand nous remontons à la racine, la probabilité *Prob* prend sa valeur initiale *ProbInit*. Nous expliquons plus en détails les règles de calcul de la probabilité *Prob* plus loin dans le texte.

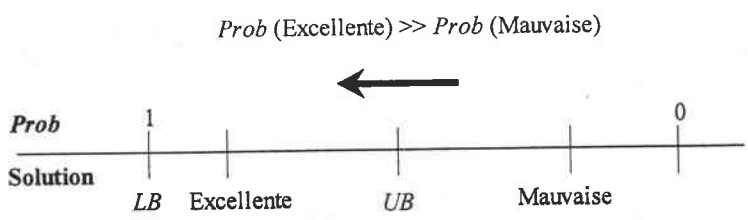

Figure 4-6 : Echelle des valeurs de *Prob*

Introduisons un compteur du nombre de niveaux remontés à partir de la dernière solution (nombre de blocs annulés), nous l'appelons *Nniv*. Ce compteur est mis à zéro dès que nous commençons un nouveau branchement. Il est incrémenté à chaque fois que nous remontons d'un niveau (annulons un bloc).

Les *conditions d'arrêt* pour la remontée sont donc les suivantes :

1) nous sommes arrivés à la racine de l'arbre ;

2) $Nniv \geq Nbl$ et $Random > Prob$.

Nombre de branchements maximum

Introduisons la variable $Niv(v)$ représentant le niveau auquel le sommet v appartient, et qui est le numéro d'ordre de la dernière opération dans la séquence d'affectation menant de la racine à v ; en construisant une branche nous passons du sommet v de niveau $Niv(v)$ au sommet v' de niveau $Niv(v') = Niv(v)+1$ par l'affectation d'une opération ; pour la racine $Niv(v_0) = 1$.

Pour chaque sommet de l'arbre d'énumération, nous introduisons encore un paramètre. Il s'agit de celui qui limite le nombre maximum de branchements autorisés à partir du sommet. Nous notons $Nbra(v)$ la valeur de ce paramètre pour le sommet v. Dans nos algorithmes, nous ne gardons en mémoire qu'une branche, nous avons donc au maximum $|\mathbf{N}|$ valeurs $Nbra(v)$ à gérer. Evidement, $Nbra(v)= 0$, si v est une feuille de l'arbre. Nous le fixons à la valeur infinie pour la racine, c.-à-d. $Nbra(v_0)=\infty$.

La valeur de $Nbra(v)$ doit dépendre du niveau $Niv(v)$ dans lequel nous nous trouvons dans l'arbre. Plus nous approchons d'une solution complète (plus nombre d'opérations placées est grand), moins nous avons besoin de parcourir de branches.

Pour calculer $Nbra(v)$, introduisons le paramètre *CouvArb* correspondant au taux de couverture désiré de l'arbre. La valeur de *CouvArb* est fixée par l'utilisateur et dépend du temps alloué à l'optimisation et du nombre d'opérations. Elle doit être choisie dans l'intervalle]0, 1].

Pour les sommets non racine, $Nbra(v)$ est donc calculé de la manière suivante :

$$Nbra(v)=\lceil CouvArb\left(|\mathbf{N}| - Niv(v)\right)! \rceil. \tag{4-1}$$

Le nombre de branchements $Nbra(v)$ croît très vite avec l'augmentation de $CouvArb$. Pour un sommet v avec $Niv(v)=2$ et $CouvArb = 1$, $Nbra(v)=(|\mathbf{N}| - 2)!$. Si nous prenons $CouvArb = 1/(|\mathbf{N}| - w)!$, alors pour $Niv(v)\geq w$, le nombre de branchements autorisés $Nbra(v)=1$, tandis que pour $Niv(v)<w$, nous avons la formule suivante :

$$Nbra(v)=\prod_{l=0}^{w-Niv(v)-1}\left(|\mathbf{N}|-Niv(v)-l\right). \tag{4-2}$$

Quand nous créons un nouveau sommet v, nous calculons la valeur initiale de $Nbra(v)$ en utilisant la formule (4-1). Nous modifions le paramètre $Nbra(v)$ du sommet v comme suit : à chaque fois qu'un branchement est fait à partir de v, le paramètre $Nbra(v)$ est décrémenté de 1 : $Nbra(v)=Nbra(v)-1$. Quand $Nbra(v)=0$, la condition d'arrêt de branchement pour le sommet v est remplie, nous ne pouvons plus faire de branchement à partir de ce sommet, nous remontons donc d'un niveau supplémentaire (nous passons au bloc précédent).

Probabilité de choisir une opération

Rappelons que lorsque nous sommes à un sommet de l'arbre, nous avons une liste d'opérations candidates L_3 dans laquelle nous tirons au hasard une opération pour un nouveau branchement. La probabilité d'être tirée pour chaque opération i de la liste L_3 à l'étape l de l'algorithme, dépend de son poids (importance) : $Poids_i^l$. L'indice supérieur donne le numéro du branchement courant. A chaque fois qu'un branchement est fait à partir du même sommet, cet indice est incrémenté. L'indice inférieur représente le numéro de l'opération.

Pour chaque nouveau sommet créé par un branchement normal, le poids de toutes les opérations est mis à 1. Par contre, quand nous nous sommes arrêtés à un sommet en remontant, nous souhaitons ne pas générer exactement la même branche. Nous connaissons l'opération j qui a conduit à fermer ce bloc à l'itération précédente et nous voulons avoir

moins de chance de la tirer à nouveau, nous diminuons donc son poids (mais en tenant compte de la qualité des résultats précédents, c.-à-d. de *Prob*) :

$$Poids_j^l = Coef1 \times Prob \times Poids_j^{l-1}, \tag{4-3}$$

où $Coef1$ est un coefficient (paramètre) du modèle, $0 \leq Coef1 \leq 1$.

En tenant compte du nouveau poids de l'opération j, $Poids_j = Poids_j^l$, nous calculons la probabilité d'être tirée pour toute opération $i \in L_3$:

$$p_i = \frac{Poids_i}{\sum_{l \in L_3} Poids_l}. \tag{4-4}$$

Traitement des itérations non concluantes

Un traitement particulier est appliqué pour des itérations non concluantes. Dans le cas d'une itération non concluante (blocage) nous remontons d'un bloc pour RAP, et d'une station pour FSIC ; nous modifions les probabilités d'être choisies des opérations candidates et nous refaisons un branchement aléatoire en espérant continuer la branche de sorte à ne plus avoir de blocage. Ici, nous modifions les poids de toutes les opérations i du dernier bloc annulé (et pas seulement d'une opération comme précédemment) en utilisant la formule suivante :

$$Poids_i^l = Coef2 \times Poids_i^{l-1}, \tag{4-5}$$

où

$Poids_i^l$ est le poids de l'opération i pour la $l^{ème}$ tentative de débloquer la situation (tentative d'obtenir une itération concluante), $Poids_i^1 = 1$;

$Coef2$ est un coefficient (paramètre) du modèle, $0 \leq Coef2 \leq 1$;

$i \in L_3$.

Les poids des opérations ainsi obtenus sont recopiés dans tous les sommets descendants tant que nous ne créons pas une station valable (pour laquelle toutes les contraintes sont respectées), c'est-à-dire que la règle "mettre les poids à 1 à la création de chaque sommet" est suspendue tant que nous n'arrivons pas à créer une station complète.

Pour éviter une boucle infinie, nous introduisons un nouveau paramètre *NdblInit* donnant le nombre maximum de tentatives de déblocage et un compteur *Ndbl*. Le compteur *Ndbl* est mis à zéro quand nous constatons un échec. Il est incrémenté à chaque nouvelle tentative de débloquer la situation. Quand le compteur *Ndbl* devient égal à *NdblInit*, nous arrêtons les tentatives de déblocage et commençons la remontée dans l'arbre suivant la procédure normale.

Apprentissage (calcul de la probabilité Prob)

Notre technique d'apprentissage se base sur la probabilité *Prob* caractérisant la qualité de la branche. Si *Prob* est égale à 1, alors l'algorithme s'arrête, la solution optimale est trouvée. Si la probabilité est inférieure à 1, alors avec la probabilité (1-*Prob*) nous remontons de bloc en bloc dans l'arbre. Par conséquent, la probabilité *Prob* doit diminuer significativement en cas d'une itération non concluante ou d'une mauvaise solution ; elle doit augmenter en cas de bonne solution trouvée. L'écart doit être proportionnel à la qualité de la solution (voir Figure 4-6).

Pour calculer *Prob*, nous utilisons les bornes supérieures : initiale (UB_0) et courante (*UB*), et les bornes inférieures : initiale (LB_0) et locale (*LLB*), de la fonction objectif. La différence entre les 2 bornes initiales UB_0 et LB_0 correspond à une estimation de la distance maximum (*DstMax*) entre les solutions possibles :

$$DstMax = UB_0 - LB_0. \tag{4-6}$$

Une borne supérieure initiale UB_0 du critère est égale à :

$$UB_0 = C_1 m_0 + C_2 q_0. \tag{4-7}$$

Une borne inférieure initiale LB_0 du critère est :

$$LB_0 = C_1 \underline{m} + C_2 \underline{n} = C_1 \max_{i \in \mathbf{N}} (k^-[i]) + C_2 \max_{i \in \mathbf{N}} (q^-[i] \mid n_0=1, ES = \{\varnothing\}). \quad (4\text{-}8)$$

Si une branche n'est pas parcourue jusqu'au bout (toutes les opérations ne sont pas encore placées), alors nous avons une solution partielle p avec $N \subset \mathbf{N}$ opérations affectées ; pour cette solution, le critère $C(p)$ est égal à :

$$C(p) = C_1\, m(p) + C_2\, n(p),$$

où $m(p)$, $n(p)$ sont le nombre de stations et le nombre de blocs, respectivement, obtenus pour le placement de l'ensemble N d'opérations.

Nous recalculons LLB à chaque fois qu'un bloc est fermé. Pour obtenir une borne inférieure locale LLB pour p, après avoir placer un ensemble N d'opérations, nous fixons $q_p^-[j] = q_p^+[j] = q[j]$ et $k_p^-[j] = k_p^+[j] = k[j]$, pour $j \in N$ et nous recalculons $q_p^-[i]$ et $k_p^-[i]$ pour $i \in \mathbf{N} \backslash N$ en utilisant l'algorithme 3-1. Ici, $q[j]$ et $k[j]$ sont les numéros de bloc et station pour l'opération j dans la solution partielle p.

$$LLB = C_1 \max_{i \in \mathbf{N}} (k_p^-[i]) + C_2 \max_{i \in \mathbf{N}} (q_p^-[i] \mid n_0=1, ES = \{\varnothing\}). \quad (4\text{-}9)$$

Quand toutes les opérations de **N** sont affectées, nous parlons d'une solution complète faisable P avec la fonction objectif :

$$C(P) = C_1\, m + C_2\, n.$$

Dès qu'une nouvelle valeur du critère $C(P)$ est meilleure que la borne supérieure courante UB, elle devient la nouvelle borne supérieure courante $UB = C(P)$.

Pour calculer *Prob*, nous avons donc quatre cas de figure :

1. Si $LLB \geq UB$, c'est-à-dire que la borne inférieure construite à partir de la solution partielle p est déjà supérieure ou égale à la meilleure solution trouvée, alors $Prob = 0$ ($1\text{-}Prob = 1$), l'algorithme remonte donc automatiquement à la racine.

2. S'il n'est plus possible d'affecter des opérations à cause des contraintes non respectées (itération non concluante), dans ce cas, nous corrigeons la valeur de la probabilité $Prob$ en la diminuant de la manière suivante :

$$Prob_s = Prob_{s-1} - Prob_{s-1}\left(1 - \frac{UB - LLB}{UB_0 - LB_0}\right), \tag{4-10}$$

où $Prob_s$ est la nouvelle valeur de $Prob$.

3. Si une solution faisable est trouvée mais qu'elle n'améliore pas la borne supérieure courante UB alors :

$$Prob_s = Prob_{s-1} - Prob_{s-1}\left(\frac{C(P) - UB}{UB_0 - UB}\right). \tag{4-11}$$

4. Si une bonne solution est trouvée (toutes les opérations sont placées et le critère est meilleur que la borne supérieure courante) alors :

$$Prob_s = Prob_{s-1} + (1 - Prob_{s-1})\frac{UB - C(P)}{UB - LB_0}\ ; \tag{4-12}$$

Notons que si $C(P) = LB_0$ alors $Prob = 1$ ($1\text{-}Prob = 0$), donc l'algorithme s'arrête, la solution optimale est trouvée.

4.4. Résultats de tests numériques

Ces résultats sont obtenus à partir des jeux d'essais (cas industriels et cas générés aléatoirement) présentés dans le chapitre précédent. Les tests ont été faits (comme ceux du Chapitre 3) sur un ordinateur bi-processeur Compaq W6000 avec 1.70 Ghz de CPU et 1 Giga RAM.

Nous nous servons, le cas échéant, des résultats obtenus dans le Chapitre 3 (méthodes exactes) pour étudier l'écart de la fonction objectif obtenue par les heuristiques aux meilleures solutions possibles (méthodes exactes) ainsi que pour comparer le temps de calcul dans deux approches.

4.4.1. Algorithmes RAP et FSIC

Nous avons d'abord testé les algorithmes FSIC et RAP dans les conditions suivantes. Les calculs ont été stoppés :

- soit quand la valeur optimale du critère donné par MIP est atteinte,
- soit après 60 000 itérations ;

c'est-à-dire qu'une nouvelle itération d'un algorithme heuristique est lancée si la valeur du critère est moins bonne que la solution obtenue par MIP ou si le nombre d'itérations est inférieur à 60 000.

Les résultats obtenus sont présentés dans le Tableau 4-1. La première colonne donne le numéro de l'exemple, la deuxième le nombre d'opérations dans cet exemple, la troisième la valeur optimale du critère obtenue par MIP (meilleurs résultats du Chapitre 3), et les trois dernières présentent les temps de calcul des algorithmes MIP, RAP et FSIC. Si une heuristique est stoppée au bout de 60 000 itérations sans atteindre l'optimum, nous indiquons cela dans le tableau comme « pas atteint ».

Puis, nous avons testé les algorithmes RAP et FSIC sur 10 000 itérations. Chaque test a été répété 10 fois. Les meilleurs résultats sont présentés dans Tableau 4-2. Dans ce tableau ~100% signifie proche de 100% (>99%).

ex	*NO*	Critère	MIP	RAP	FSIC
3	11	26	1"	2"	25"
4	13	36	0,08"	1"	0,5"
5	15	24	1"	44"	0,5"
6	18	40	1"	20"	32"
7	23	40	3"	31"	8"
8	25	28	9"	33'41"	pas atteint
9	30	38	7"	pas atteint	37"
10	35	40	4'35"	6"	5"
11	38	40	17'45"	pas atteint	pas atteint
12	45	42	35'23"	pas atteint	pas atteint

Tableau 4-1 : Comparaison du temps de calcul des méthodes RAP, FISC et MIP

Exemple		Critère (stations, blocs)		Temps CPU		% de bonnes itérations	
ex	*NO*	RAP	FSIC	RAP	FSIC	RAP	FSIC
2	9	36 (3,3)	36 (3,3)	4"	25"	33	~100
3	11	38 (3,4)	26 (2,3)	4"	41"	72	~100
4	13	36 (3,3)	36(3,3)	36"	1'3"	100	~100
5	15	36 (3,3)	36 (3,3)	55"	6'30"	28	44
6	18	40 (3,5)	40 (3,5)	21"	34"	12	89
7	23	40 (3,5)	40 (3,5)	2'19"	7'10"	0,1	99
8	25	52 (5,6)	40 (3,5)	18"	47"	12	99
9	30	40 (3,5)	54 (4,7)	5"	2"	1	74
10	35	52 (4,6)	40 (3,5)	2"	5"	1	23
11	38	42 (3,6)	42 (3,6)	1'2"	46"	1	1
12	45	54 (4,7)	54 (4,7)	1'04"	10'12"	8	82
13	100	82(6,11)	84 (7,7)	2'9"	3'6"	0,01	87
14	120	58 (4,9)	58 (4,9)	10'38"	22'3"	6	13
15	150	88 (7,9)	90 (7,10)	5'42"	41'59"	6	6

Tableau 4-2 : Résultats des tests pour RAP et FSIC après 10.000 itérations

La première colonne donne le numéro de l'exemple, la deuxième le nombre d'opérations, la troisième et la quatrième présentent la valeur du critère obtenu par RAP et

FSIC, les deux suivantes donnent les temps d'exécution et les deux dernières le pourcentage de bonnes itérations obtenues.

Nous avons ensuite testé différentes valeurs de TR_{tot} (nombre d'itérations) : TR_{tot} varie de 1000 à 10 000 itérations avec un pas de 1000 pour les exemples pour lesquels le nombre d'opérations est inférieur ou égal à 25 et TR_{tot} varie de 10 000 à 100 000 avec un pas de 10 000 pour les autres. Les résultats sont présentés dans les Tableau 4-3 et Tableau 4-4. Dans ces deux tableaux, les colonnes a, b représentent les valeurs du critère pour RAP et FSIC, respectivement ; les colonnes c, d représentent les temps de calcul pour RAP et FSIC, respectivement. Quand les valeurs du critère changent, nous les mettons en gras.

Ces tableaux montrent que la méthode FSIC améliore plus rapidement le critère, si nous regardons le nombre d'itérations et même si nous regardons le temps total de calcul (CPU). Si la première conclusion peut également être directement liée avec les itérations généralement plus longues pour FSIC que pour RAP, la deuxième montre clairement que la méthode FSIC est plus performante pour ces exemples-là. L'écart pourtant n'est pas grand et les performances comparatives des méthodes dépendent très fortement des exemples.

TR_{tot}	**Example 4**				**Example 5**				**Example 7**			
	a	b	c	d	a	b	c	d	a	b	c	d
1 000	**38**	**38**	1"	2"	**38**	**36**	1"	0,5"	**52**	**48**	2"	10"
2 000	38	38	2"	5"	38	36	4"	30"	52	48	7"	30"
3 000	38	**36**	5"	9"	38	36	7"	1"	52	**40**	14"	1'
4 000	38	36	8"	14"	38	36	12"	1'30"	52	40	24"	1'30"
5 000	38	36	12"	20"	38	36	18"	2'	52	40	36"	2'10"
6 000	38	36	17"	26"	38	36	26"	2'40"	52	40	51"	3'
7 000	38	36	22"	34"	38	36	34"	3'30"	**40**	40	1'12"	3'50"
8 000	38	36	29"	43"	**36**	36	44"	4'20"	40	40	1'27"	4'50"
9 000	**36**	36	32"	53"	36	36	50"	5'20"	40	40	1'49"	6'
10 000	36	36	36"	1'3"	36	36	55"	6'30"	40	40	2'19"	7'10"

Tableau 4-3 : Résultats des tests pour RAP et FISC (petit nombre d'opérations)

Nbr Iter.	Example 9				Example 10				Example 13			
	a	b	c	d	a	b	c	d	a	b	c	d
10 000	**40**	**54**	5"	2"	**52**	**40**	2"	5"	**82**	**84**	2'9"	3'6"
20 000	40	54	15"	7"	**40**	40	7"	14"	82	84	2'28"	3'15"
30 000	40	54	31"	13"	40	40	14"	25"	82	84	2'57"	3'27"
40 000	40	54	52"	20"	40	40	23"	40"	82	84	3'30"	3'42"
50 000	40	54	1'18"	29"	40	40	35"	57"	82	84	4'	4'
60 000	40	**38**	1'50"	39"	40	40	51"	1'17"	82	84	4'38"	4'21"
70 000	40	38	2'24"	51"	40	40	1'7"	1'40"	82	84	5'45"	4'46"
80 000	40	38	2'55"	1'3"	40	40	1'22"	2'15"	82	**78**	6'42"	5'15"
90 000	40	38	2'9"	1'18"	40	40	1'31"	2'35"	82	78	7'31"	6"
100 000	**38**	38	3'14"	1'34"	40	40	1'45"	3'5"	**78**	78	8'36"	6'19"

Tableau 4-4 : Résultats des tests pour RAP et FISC (grand nombre d'opérations)

4.4.2. FSIC avec utilisation d'ESB

Maintenant, prenons deux modifications de l'algorithme FSIC. La première consistera à faire le choix dans la liste L_4 parmi les opérations qui n'ont plus de prédécesseurs non affectés de manière aléatoire (notons que dans FSIC de base ces opérations sont choisies dans l'ordre de leur enregistrement dans la liste L_4, voir la première ligne de la procédure *Assignment2* du paragraphe 4.2.3). Nous appelons cette modification de FSIC dans les tableaux suivants "FSIC +RDI" où RDI vient de *RanDom Improvement*.

Nous comparerons "FSIC+RDI" avec l'algorithme FSIC sur lequel nous appliquons la méthode d'affectation au plus tôt, c'est-à-dire ESB (voir le paragraphe 4.3.1), nous appellerons cette dernière "FSIC+ESB". Dans le Tableau 4-5, nous présentons quelques tests des algorithmes FSIC+ESB et FSIC+RDI.

Nous avons utilisés trois exemples avec un nombre d'opérations relativement grand (100, 120 et 150). Le nombre d'itérations va de 10 000 à 60 000 avec un pas de 10 000. La première colonne donne le numéro de l'exemple, la deuxième, le nombre d'opérations. La

troisième colonne présente le nombre d'essais (tentatives) d'affectations au plus tôt réalisées par l'algorithme FSIC+ESB. La colonne suivante donne le nombre d'itérations effectuées (le même nombre pour les deux méthodes : FSIC+ESB et FSIC+RDI). Les deux colonnes suivantes donnent les valeurs du critère, le nombre de stations et le nombre de blocs pour les solutions obtenues par FSIC+ESB et FSIC+RDI, respectivement. Enfin, les deux dernières colonnes donnent le temps CPU pour les deux méthodes.

ex	*NO*	Nbre Retours	Nombre Itérations	Critère (Stations, Blocs)		Temps CPU	
				FSIC+ESB	FSIC+RDI	FSIC+ESB	FSIC+RDI
13	100	8175	10000	**84** (7,7)	**84** (7,7)	54"	3'08"
		24404	20000	84 (7,7)	84 (7,7)	1'49"	6'18"
		48470	30000	84 (7,7)	84 7,7)	2'43"	9'24"
		80767	40000	84 (7,7)	84 7,7)	3'41"	12'25"
		106421	50000	84 (7,7)	**78 (5,9)**	4'42"	15'31"
		136301	60000	84 (7,7)	78 (5,9)	5'45"	18'48"
14	120	99312	10000	**54 (4,7)**	**54 (4,7)**	5'45"	2'54"
		224214	20000	54 (4,7)	54 (4,7)	10'51"	5'49"
		366796	30000	54 (4,7)	54 (4,7)	17'36"	8'39"
		610963	40000	54 (4,7)	54 (4,7)	22'56"	11'28"
		916319	50000	54 (4,7)	54 (4,7)	28'59"	14'27"
		1282705	60000	54 (4,7)	54 (4,7)	38'02"	16'58"
15	150	129298	10000	**92 (6,16)**	**88** (6,14)	21'54"	9'44"
		386703	20000	92 (6,16)	88 (6,14)	33'06"	19'29"
		772430	30000	**88** (6,14)	88 (6,14)	35'32"	29'12"
		1284527	40000	88 (6,14)	88 (6,14)	46'58"	39'24"
		1928926	50000	88 (6,14)	88 (6,14)	58'05"	49'47"
		2701563	60000	**78 (5,14)**	88 (6,14)	1h59'17"	59'25"

Tableau 4-5 : Comparaison de FSIC +ESB et FSIC+RDI

La méthode FSIC+ESB est plus rapide que FSIC+RDI pour des petits exemples, mais elle est beaucoup plus lente que FSIC+RDI pour des grands exemples (voir Tableau 4-5), parce qu'il y a beaucoup plus d'essais : tous les blocs possibles pour une itération sont testés.

Il n'y a que dans l'exemple à 150 opérations que nous arrivons à gagner une station en appliquant FSIC+ESB par rapport à FSIC+RDI. La conclusion peut être donc qu'il est possible d'améliorer les solutions pour des grands exemples en appliquant ESB, mais que la technique de son application (choix des blocs à tester) doit encore être améliorée surtout pour des exemples avec un nombre d'opérations supérieur à 100. Pour l'exemple 12, l'application de RDI donne de moins bons temps de calcul. C'est un cas particulier où l'affectation dans l'ordre des opérations est le meilleur choix. Pour les autres exemples étudiés, ce n'est pas le cas (voir le Tableau 4-5).

4.4.3. RAP avec utilisation de BAL

Nous allons maintenant présenter des résultats obtenus en utilisant la technique BAL avec l'algorithme RAP. Cette technique nécessite le réglage des paramètres décrits précédemment. Nous avons testé plusieurs jeux de paramètres pour de petits et de grands exemples.

Dans le Tableau 4-6, nous donnons les résultats de calcul pour quelques exemples allant jusqu'à 45 opérations. Dans ce tableau se trouvent : les valeurs des paramètres utilisés, la valeur du critère pour la solution obtenue et le temps CPU. Les valeurs des paramètres utilisés ont été au préalable optimisées de la manière suivante. Pour chaque exemple nous avons donné à chacun des paramètres 10 valeurs différentes (en fixant les autres) et nous avons choisi les valeurs des paramètres qui ont donné les meilleurs résultats. Cette étude est très sensible et pourra être affinée lors de futurs travaux.

La technique BAL permet d'améliorer franchement l'algorithme RAP. Nous constatons que pour les 5 premiers exemples pour lesquels nous avons la valeur de l'optimum, nous la retrouvons avec cette heuristique, mais avec un temps de calcul de 6 à 40 fois plus petit. Pour le cinquième exemple (ex 12), qui est assez difficile, et pour lequel nous n'avons pas atteint l'optimum avec RAP et FSIC (voir le Tableau 4-1), ici nous l'avons retrouvé en 11" (le meilleur modèle MIP a mis 35'23" pour l'obtenir).

ex	NO	Paramètres					Valeur du critère (stations, blocs)	Temps CPU
		1/CouvArb	*Coef1*	*Coef2*	*NdblInit*	*ProbInit*		
3	11	360	0,75	0,35	6	0,65	26 (2,3)	0,08"
4	13	660	0,75	0,35	6	0,65	36 (3,3)	0,13"
5	15	24024	0,7	0,5	6	0,5	36 (3,3)	1,17"
10	35	33390720	0,7	0,5	6	0,5	40 (3,5)	0,83"
12	45	130320960	0,7	0,4	9	0,7	42 (3,6)	11"

Tableau 4-6 : Paramètres et résultats pour RAP+BAL

Ce résultat se confirme dans les deux tableaux suivants. Dans le Tableau 4-7, nous présentons les résultats comparatifs pour les cas industriels introduits dans le Chapitre 3 (l'exemple i0 correspond à l'exemple illustratif utilisé tout au long du Chapitre 3). Pour ces cas, nous connaissons les solutions optimales qui ont été obtenues en utilisant le modèle MIP. Nous les avons retrouvées avec nos heuristiques.

ex	NO	Valeur du critère (stations, blocs)	Temps CPU en secondes		
			FSIC+ESB	FSIC+RDI	RAP+BAL
i0	19	56 (4,8)	54,32	34,13	0,31
i1	9	42 (3,6)	1,38	0,51	1,02
i2	17	30 (2,5)	2,71	0,06	0,4
i3	15	44 (3,7)	3,34	1,33	0,51

Tableau 4-7 : Comparaison des temps de calcul (cas industriels)

Sur ces cas industriels, l'emploi de BAL avec RAP permet d'atteindre rapidement la valeur optimale du critère. Dans un exemple sur deux c'est l'heuristique la plus rapide.

Enfin dans le Tableau 4-8, nous présentons les résultats pour un exemple de 100 opérations (ex 13). Les algorithmes ont tourné de 10 000 à 60 000 itérations avec un pas de 10 000 itérations. La première colonne donne le nombre d'itérations, puis les colonnes suivantes présentent les critères obtenus par tous les algorithmes, suivis par les temps CPU.

Nombre	Critère					Temps CPU				
Itérations	RAP	FSIC	FSIC +ESB	FSIC +RDI	RAP +BAL	RAP	FSIC	FSIC +ESB	FSIC +RDI	RAP +BAL
10000	**82**	**84**	**82**	**84**	**68**	2'9"	3'6"	54"	3'8"	**16"**
20000	82	84	82	84	68	2'28"	3'15"	1'49"	6'18"	**22"**
30000	82	84	82	84	68	2'57"	3'27"	2'43"	9'24"	**36"**
40000	82	84	82	84	68	3'30"	3'42"	3'41"	12'25"	**41"**
50000	82	84	**78**	84	68	4'	4'	4'42"	15'31"	**56"**
60000	82	84	78	84	68	4'38"	4'21"	5'45"	18'48"	**1'9"**

Tableau 4-8 : Comparaison des heuristiques pour l'exemple 13

Il est important de noter que grâce à l'emploi de BAL, l'algorithme RAP a montré les meilleures performances. Le critère a été amélioré, une station a été gagnée par rapport à FSIC+ESB, une station et deux blocs par rapport à RAP tout court et une station et trois blocs par rapport à FSIC tout court et FSIC+RDI. En plus de cela, le temps de calcul a été réduit de 2 à 13 fois (en fonction du nombre d'itérations).

4.5. *Conclusions*

Dans ce chapitre, nous avons présenté une approche heuristique pour TLBP. Cette approche permet de traiter les problèmes de grand taille dans un temps limité. Elle se base sur les techniques COMSOAL qui sont développées et adaptées au problème considéré. Nous avons donc présenté d'abord l'algorithme COMSOAL de base. Puis, nous avons montré pourquoi cet algorithme ne peut pas être utilisé directement pour le problème considéré. Ensuite nous avons proposé deux algorithmes utilisant chacun une technique particulière pour adapter COMSOAL à TLBP.

Le premier algorithme se base sur une application directe des idées de COMSOAL en les complétant pour pouvoir tenir compte des nouvelles contraintes. Cet algorithme, appelé dans ce chapitre RAP ne limite pas la liste des opérations candidates *a priori*. Les contraintes complémentaires du problème sont traitées au moment de l'affectation des opérations. Par

conséquent, cet algorithme est susceptible de fournir souvent des itérations non concluantes, c'est-à-dire celles qui ne donnent aucune solution complète à cause de conflits entre les contraintes pendant la phase d'affectation des opérations. Le deuxième algorithme, appelé FSIC, a été conçu pour palier à l'insuffisance du premier. Il établit une sélection et une règle complémentaire dès la phase de la formation de la liste des opérations candidates pour l'affectation. Ces règles se basent sur la spécificité des contraintes du problème. Grâce à cela, le deuxième algorithme doit se solder a priori par moins d'itérations non concluantes, mais ajoute un risque complémentaire de blocage des solutions dans un optimum local. Nous avons implémenté ces deux algorithmes et nous les avons testés sur quelques cas industriels et des exemples générés aléatoirement. Les résultats des tests montrent un léger avantage à l'algorithme FSIC par rapport à RAP. La différence n'est pas grande, elle est constatée dans un petit nombre d'exemples. Ce qui ne permet pas de partager définitivement les deux algorithmes. Apparemment, les avantages de chaque algorithme sont compensés par ses inconvénients. Mais intuitivement nous donnons préférence à l'algorithme FSIC.

Ensuite, nous avons montré comment les deux algorithmes peuvent être améliorés. Nous avons proposé plusieurs techniques complémentaires relativement efficaces. Tous d'abord, nous avons constaté que l'algorithme FSIC pouvait être amélioré si une séquence fixe d'affectation définitive des opérations était remplacée par un choix aléatoire. Ce deuxième choix aléatoire, en plus de celui inspiré de COMSOAL, permet souvent d'éviter les itérations non concluantes (un seul des exemples nous donne des résultats contraires). Nous appelons cette technique RDI. Ensuite, nous avons décidé de renoncer au principe de COMSOAL qui consiste à ne jamais revoir des décisions partielles prises au fur et à mesure de l'avancement de l'algorithme. En s'inspirant des méthodes stochastiques de la recherche dans l'arbre d'énumération, nous avons proposé plusieurs techniques permettant de revenir en arrière et de recommencer la construction d'une solution. La principale technique, celle que nous appelons BAL (*Backtracking And Learning*), s'appuie sur l'utilisation de la remontée dans l'arbre d'énumération avec "apprentissage". Plus les solutions obtenues sur la branche courante sont bonnes, moins la probabilité de remonter est grande et inversement.

Nous avons montré que l'algorithme FSIC complété par les techniques RDI et ESB améliore sensiblement ses performances. La méthode FSIC+ESB est plus rapide que

FSIC+RDI pour de petits exemples, mais elle est beaucoup plus lente que FSIC+RDI pour de grands exemples. Il n'y a que dans l'exemple à 150 opérations que nous arrivons à gagner une station en appliquant FSIC+ESB par rapport à FSIC+RDI. La conclusion peut donc être qu'il est possible d'améliorer les solutions pour des grands exemples en appliquant ESB, mais que la technique de son application (choix des blocs à tester) doit encore être améliorée surtout pour des exemples avec un grand nombre d'opérations.

Mais les résultats les plus intéressants sont obtenus avec l'utilisation de BAL. Il est important de noter que grâce à l'emploi de BAL, l'algorithme RAP a montré les meilleures performances. Pour un exemple illustratif que nous avons présenté à la fin de ce chapitre le critère a été sensiblement amélioré, en plus de cela, le temps de calcul a été réduit de plusieurs fois.

Il est intéressant maintenant de voir comment il est possible de tenir compte des avantages des méthodes exactes et méthodes approchées en même temps. Une tentative de construire un algorithme hybride qui se base sur un couplage intelligent du modèle MIP et des heuristiques FSIC et RAP est présentée dans le chapitre suivant.

Chapitre 5

Couplage MIP - Heuristiques

5. Couplage MIP - Heuristiques

Dans le Chapitre 3, nous avons proposé des méthodes d'optimisation exactes basées sur la programmation linéaire en variables mixtes (MIP) et dans le Chapitre 4, des méthodes heuristiques basées sur COMSOAL et sur une recherche aléatoire dans l'arbre de décision. Dans ce chapitre, nous donnons quelques voies possibles permettant de combiner ces méthodes.

5.1. Utilisation des Bornes supérieures

La première approche consiste simplement à utiliser une heuristique parmi celles que nous avons proposées pour obtenir une borne supérieure de la fonction objectif et ensuite transmettre cette borne à Cplex.

En effet, comme nous l'avons déjà mentionné, les solvers de type Cplex font appel aux PSE pour trouver des valeurs optimales des variables entières et binaires (Cplex, 2001). Une PSE construit un arbre d'énumération, sur lequel chaque nœud représente un sous-problème. La branche de l'arbre est arrêtée (voir le Chapitre 2), si la borne inférieure locale (*LLB*) pour la solution courante est moins bonne qu'une borne supérieure (*UB*).

Cplex cherche lui-même les bornes. Mais lorsque nous donnons une borne supérieure du critère, Cplex fait comme s'il avait lui-même trouvé une solution entière, et utilise une stratégie de parcours de l'arbre différente de celle qu'il utilise avant d'avoir une première solution. C'est-à-dire qu'en donnant à Cplex une première borne supérieure (*UB*) explicitement, cela lui permet de ne pas avoir à trouver de première solution et d'abandonner rapidement certaines branches. Si la borne est bonne cela accélère les calculs.

Dans les exemples ci-dessous, nous avons utilisé FSIC. La démarche est la suivante (voir la Figure 5-1) :

- nous lançons l'heuristique FSIC ;

- la procédure est arrêtée dès qu'une solution faisable est trouvée ;
- nous lançons MIP en transmettant à Cplex la valeur trouvée par FSIC en qualité de borne supérieure du critère.

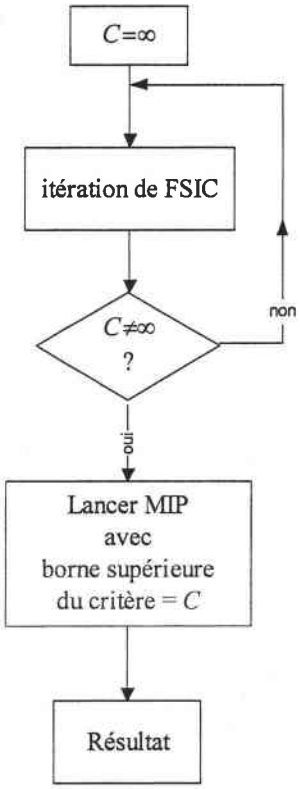

Figure 5-1 : Diagramme de l'approche

Une modification de cet algorithme est liée avec la possibilité de faire tourner plus longtemps l'heuristique. Dans ce cas, la borne *UB* peut être de meilleure qualité et cela peut encore accélérer les calculs de MIP. Le seul problème est de trouver un bon *compromis* entre le gain de temps pour MIP et le temps de calcul de l'heuristique.

Le Tableau 5-1 présente les résultats de l'emploi par Cplex d'une borne supérieure obtenue par FSIC. Ces calculs ont été effectués en utilisant le modèle MBI (modèle de base avec les intervalles d'indices de blocs et stations) présentée dans le Chapitre 3. La première colonne donne le numéro de l'exemple, la deuxième colonne le nombre d'opérations dans l'exemple, la troisième le temps total : le temps de l'heuristique plus le temps de MIP. La quatrième colonne donne pour rappel le temps de MBI utilisé tout seul, la cinquième colonne celui du meilleur modèle MIP du Chapitre 3.

ex	*NO*	**MBI avec Borne Sup**	**MBI**	**Meilleure méthode du Chapitre 3**
1	6	0,1"	0,3"	0,05"
2	9	0,1"	5'52"	1"
3	11	3"	26"	1"
6	18	3"	1'2"	1"
7	23	2'8"	1'38"	3"
8	25	30"	9"	9"
9	30	1'17"	52"	7"
10	35	5'40"	2h51'	4'35"
11	38	32'21"	>5h	35'23"

Tableau 5-1 : Utilisation d'une borne supérieure du critère pour MIP

Le fait de donner une borne supérieure à MIP de manière générale diminue le temps de calcul. Mais il y a des exemples où cet effet n'a pas eu lieu. Cela peut être expliqué par une mauvaise qualité de la borne si on ne fait qu'une seule itération de l'heuristique.

En ajoutant une borne supérieure obtenue après une seule itération de l'heuristique FSIC au modèle MIP, nous n'avons pas réussi à obtenir une solution optimale avec temps de calcul inférieur à 5h pour les exemples avec un grand nombre d'opérations (supérieure à 80).

Etant donné que $C_1m+C_2n \leq UB$, une borne supérieure du nombre de stations $\overline{m}$ peut également être calculée selon la formule :

$\overline{m}=(UB-\underline{n}C_2)/C_1$, où $\underline{n}$ est obtenu en utilisant (3-17).

Dans le Tableau 5-2, nous montrons le comportement de MIP si en plus d'une borne supérieure du critère, nous remplaçons la valeur de m_0 par la valeur de sa borne supérieure $\overline{m}$. La première colonne donne le numéro de l'exemple, la deuxième le nombre d'opérations, la troisième et la quatrième rappellent les valeurs de m_0 et de n_0, la colonne suivante présente la borne supérieure de la fonction objectif obtenue par l'heuristique, les 2 colonnes suivantes représentent les valeurs de la borne inférieure du nombre de blocs $\underline{n}$ et de la borne supérieure du nombre de stations $\overline{m}$ et enfin la dernière colonne donne le temps CPU de MIP. Les calculs sont faits en utilisant MBI.

ex	*NO*	m_0	n_0	*UB*	$\underline{n}$	$\overline{m}$	**Temps CPU**
7	23	6	2	38	4	3	2,21"
8	25	6	6	36	3	3	3"
9	30	5	5	50	5	4	5"
10	35	7	8	84	7	7	4'26"
11	38	7	8	86	8	7	5'40"

Tableau 5-2 : Utilisation des bornes supérieure du nombre de stations et du critère

Rappelons que C_1=10 et C_2=2. Le temps CPU obtenu par l'emploi de ces bornes (Tableau 5-2) montre des améliorations.

5.2. *Utilisation de la décomposition*

5.2.1. Approche générale

En présence de problèmes de grande taille, une solution peut être d'utiliser une décomposition en partitionnant l'ensemble d'opérations initial **N** en plusieurs (w) sous-ensembles (Dolgui *et al.*, 2000). L'approche MIP est appliquée de manière séquentielle en suivant la décomposition $\mathbf{N}^1$, $\mathbf{N}^2$, ..., $\mathbf{N}^w$. Le résultat global pour le problème de départ est donc obtenu à la fin de l'algorithme suite à la résolution de w problèmes MIP.

L'application de cette approche est liée au partitionnement de l'ensemble **N** en sous-ensembles. Le partitionnement doit se faire de telle manière qu'à chaque étape *toutes les contraintes soient respectées*. Nous notons $\mathbf{N}^u$ le sous-ensemble courant, c'est-à-dire, le sous-ensemble d'opérations que nous avons obtenu par une extraction des opérations de l'ensemble **N**, auquel à l'étape courante u nous appliquons un modèle MIP. L'optimisation à cette étape concerne uniquement les opérations de $\mathbf{N}^u$.

Pour respecter les contraintes de précédence, il faut que, lorsque nous traitons un sous-ensemble d'opérations $\mathbf{N}^u$ par MIP, toutes les opérations prédécesseurs des opérations de ce

sous-ensemble $\mathbf{N}^u$ soient dans le sous-ensemble $\mathbf{N}^u$ ou déjà assignées (lors du traitement d'un sous-ensemble antérieur) :

$$Pred(j) \subseteq \mathbf{N}^u \cup \breve{\mathbf{N}}^{u-1}, \text{ quel que soit } j \in \mathbf{N}^u, \tag{5-1}$$

où $\breve{\mathbf{N}}^{u-1} = \cup_{r=1}^{u-1} \mathbf{N}^r$.

Si une opération appartenant à $\mathbf{N}^u$ est également dans un sous-ensemble $e \in ES$, alors toutes les opérations de e doivent être dans $\mathbf{N}^u$, c.-à-d. :

$$e \cap \mathbf{N}^u = \varnothing \text{ ou } e \subseteq \mathbf{N}^u, \text{ quel que soit } e \in ES. \tag{5-2}$$

Enfin, une contrainte d'exclusion représentée par un sous ensemble $e \in \overline{EB}$ (ou $e \in \overline{ES}$) est maintenue uniquement dans le cas où toutes les opérations de e appartiennent à $\mathbf{N}^u$; dans le cas contraire, la contrainte donnée par e n'est pas à prendre en compte pour $\mathbf{N}^u$.

Pour *la décomposition*, nous appliquons la démarche suivante. Nous effectuons le partitionnement de l'ensemble **N** par une découpe du graphe de précédence.

D'abord, nous choisissons la taille désirée de chaque sous-ensemble en tenant compte des études du comportement des modèles MIP pour ce type de problème (voir le Chapitre 3). Ce choix dépend de la taille du problème et du budget de temps dont nous disposons pour trouver une solution. Plus *les sous-ensembles sont petits* plus c'est facile d'avoir un temps de calcul total raisonnable, mais en même temps plus nous avons de chances de nous éloigner de l'optimum global à une distance non négligeable.

Nous présentons l'approche générale dans l'algorithme suivant :

Algorithme 5-1 :

***Etape* 0.** w=1.

***Etape* 1.** Lancer la procédure *Décomposer* pour **N** et obtenir le sous ensemble $\mathbf{N}^w$ (opérations avec toutes les contraintes mises à jour).

***Etape* 2.** Trouver l'affectation optimale pour $\mathbf{N}^w$ en utilisant le modèle MIP.

***Etape* 3.** Si toutes les opérations de **N** sont assignées, arrêter l'algorithme, sinon continuer.

***Etape* 4.** Lancer la procédure *Modifier* pour **N**, w=w+1, aller à *Etape* 1.

Supposons que nous venons de terminer l'optimisation pour $\mathbf{N}^v$, il y a deux manières à procéder pour tenir compte de ses résultats et de modifier l'ensemble **N** (par la suite nous appelons les procédures correspondantes *Modifier*1 et *Modifier*2) :

1) transformer les blocs de la solution obtenue pour $\mathbf{N}^v$ en macro-opérations qui seront prises en compte dans **N** à la place des opérations qu'elles représentent ;
2) conserver la solution obtenue et continuer à travailler uniquement avec des opérations restantes, c'est-à-dire à la fin du traitement de chaque sous ensemble $\mathbf{N}^v$ de faire $\mathbf{N}=\mathbf{N}\setminus\mathbf{N}^v$.

La réalisation de la première approche est relativement simple, il suffit de remplacer les blocs obtenus par des macro-opérations et de modifier les contraintes en conséquence. Chaque macro-opération contenant l'ensemble des opérations du bloc, le temps d'une macro-opération étant le temps de la plus longue opération du bloc. Nous recréons un tout (un nouveau graphe de précédence et de nouvelles contraintes de compatibilité) avec ces nouvelles macro-opérations et les opérations choisies parmi celles qui restaient en attente (qui n'ont pas été traitées). Nous réajustons toutes les contraintes de précédence et de compatibilité (exclusion et inclusion) en tenant compte des macro-opérations, à savoir :

- les contraintes impliquant les seules opérations regroupées dans une macro-opération sont supprimées ;
- les contraintes impliquant une (ou plusieurs) opérations d'une macro-opération et une ou plusieurs opérations d'une autre macro-opération ou d'une opération seule sont ajustées.

La procédure *Modifier*1 réalise une découpe à base de cette approche.

***Modifier*1**

En partant d'une solution partielle trouvée.

***Etape* 1. pour** chaque bloc q trouvé :

Regrouper les opérations du bloc en une macro-opération

Allouer à cette macro-opération un temps correspondant au maximum des temps des opérations du bloc.

Supprimer les contraintes qui n'affectent que des opérations de ce bloc.

finpour

***Etape* 2.** Rajouter à ces macro-opérations les opérations de la découpe suivante parmi celles qui n'ont pas été choisies antérieurement.

***Etape* 3.** Réajuster toutes les contraintes (précédence, inclusion et exclusion).

***finModifier*1**

Pour cette approche (*Modifier*1) plus nous approchons de la fin de l'algorithme, plus MIP travaille « inutilement » (car les macro-opérations sont déjà placées) et le temps de calcul total peut augmenter à cause de cela.

Pour la deuxième approche, nous obtenons des modèles MIP avec un nombre de variables plus petit et un nombre de contraintes plus petit, les sous ensembles $\mathbf{N}^2$, $\mathbf{N}^3$, ..., $\mathbf{N}^w$ sont plus homogènes, mais nous ne tenons pas compte de la possibilité d'ajouter dans les stations et blocs déjà construits les opérations qui n'étaient pas traitées au moment de la formation de ces blocs et stations. L'importance de cette possibilité dépend de la qualité des découpes et des contraintes.

Nous pouvons introduire un algorithme mélangeant ces deux approches (voir *Modifier*2). Pour former $\mathbf{N}^{v+1}$, nous gardons les blocs de la solution obtenue pour $\mathbf{N}^v$ et nous les considérons comme des macro-opérations (comme dans *Modifier*1). Nous compléterons ces macro-opérations par des opérations rajoutées en faisant une nouvelle découpe dans la partie restante du graphe. S'il y a des contraintes ne reliant que la macro-opération i à une autre macro-opération et qu'il n'y a aucune contrainte reliant cette macro-opération i à une opération "rajoutée", *nous mettons de côté cette macro-opération i* (elle ne jouera plus sur la

suite de la construction de la solution). Nous faisons, bien sûr, la soustraction du temps de cette macro opération du temps de cycle de la station correspondante dans le modèle MIP pour $\mathbf{N}^{v+1}$. Une fois un découpage et un réajustement faits pour $\mathbf{N}^{v+1}$, nous lançons MIP, et ainsi de suite, tant que nous n'arrivons pas à assigner toutes les opérations de **N**.

Modifier2

En partant d'une solution partielle trouvée :

***Etape* 1. pour** chaque bloc *q* trouvé :

Regrouper les opérations du bloc en une macro-opération.

Allouer à cette macro-opération un temps correspondant au maximum des temps des opérations du bloc.

Supprimer les contraintes qui n'affectent que des opérations de ce bloc.

finpour

***Etape* 2.** Rajouter aux macro-opérations obtenues les opérations de la découpe suivante faite parmi les opérations qui n'ont pas encore été choisies.

***Etape* 3.** **pour** chaque macro-opération :

si cette macro-opération n'est liée par des contraintes qu'avec d'autres macro-opérations, elle est mise de côté et ne fait plus partie de l'ensemble courant d'opérations.

finsi

finpour

***Etape* 4.** Réajuster toutes les contraintes (précédence, inclusion et exclusion).

finModifier2

A la fin de l'algorithme, si nous utilisons *Modifier2*, la solution est obtenue en mettant ensemble toutes stations et blocs construits le long de toutes les étapes. Nous pouvons également faire un modèle MIP final à partir de tous les blocs d'une telle solution et essayer de l'améliorer.

5.3. Tests de faisabilité de l'approche

Pour vérifier la faisabilité de l'approche nous avons effectué un certain nombre de tests en découpant de manière arbitraire (au hasard) les graphes de précédence des exemples que

nous avons déjà traités (voir les Chapitres 3 et 4). Dans les tableaux de résultats qui suivent nous employons les notations complémentaires suivantes :

- *nbop* est le nombre d'opérations choisies lors d'une découpe ;
- *Mop* représente le nombre de macro-operations issues d'une étape de la découpe et qu'il faut garder à l'étape suivante.

Le Tableau 5-3 montre, pour quelques exemples individuels, les résultats d'une décomposition faite une seule fois : le graphe est décomposé uniquement en deux parties. La fonction *Modifie*2 est utilisée. Nous comparons ces résultats avec la solution exacte obtenue par le meilleur modèle MIP du Chapitre 3.

		Sans découpage			**Découpage en deux parties**							
					Premier MIP		**Deuxième MIP**			**Résultat final**		
ex	***NO***	***Blocs***	***Stations***	***Temps CPU***	***nbop***	***T(S)***	***nbop***	***MoP***	***Temps CPU***	***Blocs***	***Stations***	***Temps CPU***
4	13	3	3	0,08"	8	0,5"	5	2	2"	3	3	2,5"
5	15	2	2	1"	12	0,1"	3	3	0,1"	2	2	0,2"
8	25	4	2	9"	18	52"	7	4	14"	4	2	1'6"
8	25	4	2	9"	15	32"	10	3	6"	5	2	38"
10	35	5	3	4'35"	14	1'7"	21	3	2'23"	5	3	3'30"
11	38	5	3	17'45"	21	3'10"	17	3	1'13"	5	4	4'23"
12	45	6	3	35'23"	28	2'36"	17	2	1'53"	6	3	4'29"

Tableau 5-3 : Résultats avec et sans décomposition pour quelques exemples individuels

En utilisant ce découpage, nous avons obtenu l'optimum 5 fois sur 7. Une fois seulement, le temps de calcul avec découpage a dépassé le temps de calcul sans ce dernier. L'exemple numéro 8, a été lancé deux fois avec deux découpages différents. La première fois, le temps de calcul total est plus grand ; la deuxième fois, le temps est meilleur mais cette fois-

ci le critère n'est pas optimal. Les exemples avec 38 et 45 opérations montrent que le temps de calcul sans découpage commence à augmenter de manière exponentielle tandis que l'augmentation du temps de calcul entre ces deux exemples avec le découpage n'est que légère.

Le Tableau 5-4 présente le même type de tests pour quelques séries.

ex	*NO*	*T(S)* de MIP sans décomposition	Premier MIP	Deuxième MIP		Temps *T(S)*
			nbop	*nbop*	*MoP*	
s8	23	1h23'20"	12	11	3	32'58"
s9	25	5h04'55"	16	9	7	1'10"
s10	25	3h17'50"	14	11	4	2'45"
s11	29	24'31"	15	14	1	2'38"
s12	40	36'03"	18	22	1	8'05"

Tableau 5-4 : Résultats avec et sans décomposition pour des séries

Dans les trois tableaux suivants nous donnons quelques résultats de la décomposition en plus grand nombre d'étapes. Les exemples qui sont pris vont de 100 à 150 opérations. Une colonne supplémentaire appelée temps total a été rajoutée. En effet dans ce cas, la préparation du découpage prend un peu plus de temps. La différence entre la colonne temps total et la colonne « Durée de MIP » donne ce temps.

Chaque ligne des tableaux donne les données et le résultat d'une découpe. La dernière ligne donne les valeurs qui correspondent au nombre total des opérations, variables et contraintes traitées ainsi qu'au résultat final en termes de nombre de blocs et stations et au temps de calcul. Ces exemples n'ont pas pu être traités en utilisant MIP sans décomposition, car ils sont trop grands.

	NO	*MoP*	*CTNV*	*CNC*	Non Zero	Stations	Blocs	Durée de MIP	Temps Total
découpe 1	14		270	804	4620	2	3	1'9"	1'9"
découpe 2	9	3	249	766	953	2	4	37"	37"
découpe 3	17	3	417	1141	19716	3	4	12'28"	12'30"
découpe 4	29	4	711	2090	33995	2	3	7'44'	7'45"
découpe 5	31	4	430	1412	27598	3	4	4'17"	4'18"
Total	100	14	2077	6213	86882	7	11	26'15"	26'19"

Tableau 5-5 : Décomposition en 5 étapes pour l'exemple de 100 opérations

	MoP	*CTNV*	*CNC*	Non Zero	Stations	Blocs	Durée de MIP	Temps Total
découpe 1	27		564	1550	26594	2	4	26'54"	26'55"
découpe 2	6	1	564	487	5027	3	5	5"	5"
découpe 3	9	4	270	796	9987	3	4	12"	12"
découpe 4	12	3	312	975	9718	2	3	10"	11"
découpe 5	35	4	816	2609	41512	2	3	53'20"	53'28"
découpe 6	31	2	472	1565	27277	3	3	14'08"	14'59"
Total	120	14	2998	7982	120115	4	7	1h34'49"	1h35'50"

Tableau 5-6 : Décomposition en 6 étapes pour l'exemple de 120 opérations

L'emploi de la décomposition pour des exemples de taille importante permet d'obtenir des résultats. Par contre, les temps de calcul restent souvent relativement grand et nous ne sommes pas sûrs d'être à l'optimum. Une maîtrise plus évoluée de la taille des découpes et une méthode plus avancée de décomposition pourraient palier à ces défauts.

	NO	*MoP*	*CNC*	*CTNV*	Non Zero	Stations	Blocs	Durée de MIP	Temps Total
découpe 1	31		2263	748	32552	2	3	2h37'50"	2h38'11"
découpe 2	5	2	523	172	5352	2	3	6"	10"
découpe 3	3	1	296	100	5352	2	3	2"	5"
découpe 4	15	7	1615	532	26414	2	3	6'44"	6'48"
découpe 5	21	5	1912	628	35128	2	2	6'	6'4"
découpe 6	12	6	1261	436	22678	2	3	1'53"	1'54"
découpe 7	18	7	1836	604	31646	2	3	3'43"	3'43"
découpe 8	11	3	956	340	13371	2	3	28"	28"
découpe 9	34	8	2069	624	32333	3	5	51'28"	51'32"
Total	150	39	12731	4652	204826	8	12	3h49'14"	3h49'55"

Tableau 5-7 : Décomposition en 9 étapes pour l'exemple de 150 opérations

Nous avons vu que les résultats de cette approche dépendent de la manière dont nous décomposons le graphe de précédence. Pour la procédure *Décomposer*, nous pouvons donc proposer deux approches :

- la première est une découpe en w sous-ensembles de manière déterministe ;

- la deuxième est basée sur nos heuristiques de type COMSOAL (voir le Chapitre 4), elle utilise des choix aléatoires et doit être faite un nombre suffisamment grand de fois.

Dans les deux cas, les résultats des heuristiques peuvent également être utilisés pour avoir les bornes supérieures : celle du critère UB^v pour le solver MIP, celle du nombre de stations $\overline{m}^v$ pour le modèle MIP lui-même, v=1,2, ...w.

5.4. Décomposition déterministe

5.4.1. Un algorithme de décomposition déterministe

Introduisons quelques notations supplémentaires :

NbPred(i), (*NbSucc*(i)) est le nombre de prédécesseurs (successeurs) directs de i ;

Cont(i) est le nombre d'opérations auxquelles l'opération i est liée par les contraintes de précédence (successeurs) et de compatibilité (*ES*) ;

MinOp et *MaxOp* est le nombre minimum et le nombre maximum d'opérations autorisées dans un sous-ensemble (le nombre d'opérations doit être entre *MinOp* et *MaxOp*).

L'idée de l'heuristique de décomposition que nous proposons ici se base sur celle de Kilbridge et Wester (1961), mais en tenant compte de la spécificité du problème que nous traitons et des objectifs de la décomposition.

D'abord, nous calculons le *Rang* de chaque opération dans le graphe de précédence. Rappelons que le rang des opérations qui n'ont pas de prédécesseurs directs est égal à 1. Le rang de l'opération i est égal à

$$Rang(i) = \max(Rang(j), j \in Pred(i)) + 1,$$

où $Pred(i)$ est l'ensemble de prédécesseurs directs de l'opération i.

Puis, nous examinons les opérations de chaque rang une par une en commençant par le rang le plus petit et nous les ajoutons dans $\mathbf{N}^1$. Pour chaque opération ajoutée, nous vérifions que toutes les contraintes d'inclusion *ES* soient respectées. S'il y a des contraintes non respectées, nous ajoutons les opérations correspondantes même si elles sont d'un autre rang (avec leurs prédécesseurs non affectés, s'ils existent). Quand toutes opérations d'un rang sont traitées, nous passons au rang suivant. Tant que le nombre d'opérations dans $\mathbf{N}^1$ est inférieur à *MinOp* nous continuons cette procédure.

Une fois la procédure *Décomposer* arrêtée, nous ajustons la taille (réduction de l'ensemble $\mathbf{N}^1$) en tenant compte du paramètre *MaxOp* et nous formons les contraintes pour le modèle MIP correspondant (pour $\mathbf{N}^1$). Si *MaxOp* est dépassé, nous proposons un algorithme simple nommé ci-dessous *Réduire* pour réduire l'ensemble $\mathbf{N}^1$.

Ensuite, nous appliquons MIP sur ce premier sous-ensemble $\mathbf{N}^1$, et ainsi de suite pour $\mathbf{N}^2, \ldots, \mathbf{N}^w$.

Pour accélérer les calcul, nous pouvons lancer une heuristique avant d'utiliser chaque modèle MIP. La solution heuristique nous donnera une borne supérieure du critère UB^u et une borne supérieure du nombre de stations $\overline{m}^u$.

Décomposer

pour chaque $i \in \mathbf{N}$ calculer son rang $Rang(i)$ dans le graphe de précédence ;

$MaxRang$=max($Rang(i)$), $i \in \mathbf{N}$;

Initialiser u=1, $NbOpNu$ = 0, $\mathbf{N}^u = \{\varnothing\}$; r=1, $STOP$=0.

répéter

prendre une opération j telle que $j \in R_r = \{j \mid Rang(j)=r\}$ et $j \notin \mathbf{N}^u$;

faire $\mathbf{N}^u = \mathbf{N}^u + \{j\}$, $R_r = R_r - \{j\}$;

si $j \in e \in ES$ **alors** pour tout $i \in e$, $i \neq j$,

faire $\mathbf{N}^u = \mathbf{N}^u + \{i\}$, $R_r = R_r - \{i\}$;

finsi

si $R_r = \{\varnothing\}$, **alors**

si $r < MaxRang$, $r=r+1$, **sinon** $STOP$=1 ; **finsi**

finsi

si $|\mathbf{N}^u| \geq MinOp$, **alors** $STOP$=1 ;

jusqu'à ce que STOP=1

si $|\mathbf{N}^u| \geq MaxOp$, **alors** appeler la procédure <u>*Reduire*</u> ; **finsi**

Mettre à jour les contraintes à prendre en compte pour les opérations dans $\mathbf{N}^u$.

finDécomposer

Réduire

pour chaque $i \in \mathbf{N}^u$

Calculer le nombre *Cont*(i) d'opérations de $\mathbf{N}^u$ avec lesquelles l'opération i est liée par les contraintes de précédence (successeurs) et de compatibilité (*ES*) ;

finpour

Ranger les opérations de $\mathbf{N}^u$ dans l'ordre croissant des *Cont*(i), $i \in \mathbf{N}^u$;

Prendre les opérations dans cet ordre une par une :

tant que ($|\ \mathbf{N}^u\ | - MaxOp$) > *Cont*($i$)

Supprimer i de $\mathbf{N}^u$ avec toutes les opérations successeurs et celles qui sont liées avec i par des contraintes d'inclusion (*ES*) ;

fintantque

finRéduire

5.4.2. Un exemple de décomposition

Nous prenons l'exemple présenté dans la Figure 5-2 et nous montrons sa décomposition, les valeurs *MinOp* =6 et *MaxOp* =8. Le Tableau 5-8 donne les contraintes de compatibilité. Les rangs des opérations sont calculés dans le Tableau 5-9 :

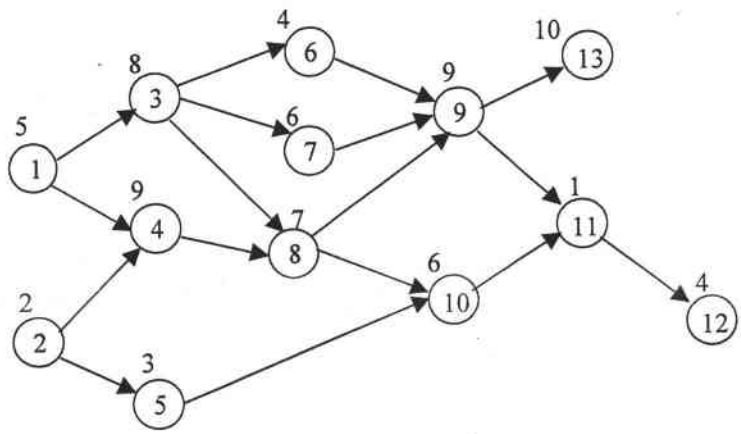

Figure 5-2 : Graphe de précédence

$\overline{EB}$	$\overline{ES}$	ES
{1, 7}	{2, 5}	{3, 4, 8}
{2, 8}	{8, 9}	{11, 13}
{7, 9, 13}	{3, 10}	

Tableau 5-8 : Contraintes de compatibilité

Opération *j*	1	2	3	4	5	6	7	8	9	10	11	12	13
NbPred(j)	0	0	1	2	1	1	1	2	3	2	2	1	1
NbSucc (j)	2	2	3	1	1	1	1	2	2	1	1	0	0
Rang(j)	1	1	2	2	2	3	3	3	4	4	5	6	5

Tableau 5-9 : Rangs des opérations

Nous commençons par mettre dans l'ensemble $\mathbf{N}^1$, les opérations de rang 1 : $\mathbf{N}^1 = \{1, 2\}$, pour ces opérations, il n'y a aucune contrainte de type *ES*. Le nombre d'opérations dans $\mathbf{N}^1$ est :

$| \mathbf{N}^1 | = 2$; $| \mathbf{N}^1 | < MinOp$

Nous prenons le rang 2. Nous rajoutons essayons l'opération 3 du rang 2.

$\mathbf{N}^1 = \{1, 2, 3\}$, $| \mathbf{N}^1 | = 3$; $| \mathbf{N}^1 | < MinOp$

Nous devons maintenant rajouter dans $\mathbf{N}^1$ les opérations 4 et 8 qui doivent être sur la même station que l'opération 3 car il existe $e = \{3, 4, 8\}$, $e \in ES$. Ces opérations ont tous leurs prédécesseurs dans $\mathbf{N}^1$.

$\mathbf{N}^1 = \{1, 2, 3, 4, 8\}$, $| \mathbf{N}^1 | = 5$, $| \mathbf{N}^1 | <$ MinOp.

Ensuite, nous rajoutons l'opération 5

$\mathbf{N}^1 = \{1, 2, 3, 4, 5, 8\}$, $| \mathbf{N}^1 | = 6$, $| \mathbf{N}^1 | = MinOp$.

Nous arrêtons la découpe. Toutes les contraintes de précédence sont automatiquement respectées (voir la figure suivante).

Les contraintes de compatibilité à prendre en compte pour $\mathbf{N}^1$ sont dans le Tableau 5-10.

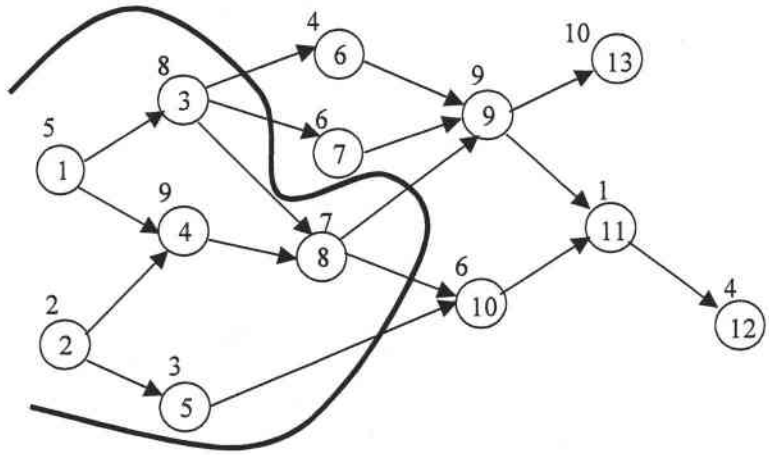

Figure 5-3 : Résultat de la décomposition

$\overline{EB}$	$\overline{ES}$	ES
{2, 8}	{2, 5}	{3, 4, 8}

Tableau 5-10 : Contraintes d'inclusion et d'exclusion

5.4.3. Quelques tests numériques

Dans le Tableau 5-11 se trouvent quelques exemples d'utilisation de l'algorithme de la décomposition déterministe proposé dans le paragraphe précédent. La première colonne donne le nombre total d'opérations de l'exemple, puis, pour chaque étape, le nombre d'opérations, le nombre de stations et de blocs trouvés et le temps d'exécution. La dernière colonne présente le temps total de la méthode. Le nombre de blocs et de stations de la deuxième étape représente le nombre de stations et de blocs finaux. Nous avons utilisé ici la procédure *Modifier*1.

Des tests plus approfondis devront être effectués par la suite.

	Premier étape				Deuxième étape				Temps total
NO	*NO*1	*m*	*n*	Temps	*NO*2	*m*	*n*	Temps	
23	12	2	3	2"	14	3	5	1"	3"
35	18	2	3	12"	20	3	5	7"	19"
38	20	2	4	18"	22	4	7	35"	53"
45	24	2	4	19"	25	4	7	42"	1'01"

Tableau 5-11 : Utilisation de la décomposition déterministe

5.5. Décomposition stochastique

Pour tracer les perspectives de notre recherche, nous avons voulu présenter une approche de décomposition itérative et stochastique qui doit être à notre avis étudiée attentivement et testée numériquement. Il s'agit d'une méthode qui utilisera nos heuristiques pour faire la décomposition. Sa description est présentée dans le paragraphe suivant.

L'algorithme FSIC ou RAP est lancé. Au bout d'un certain temps, nous l'arrêtons et obtenons une solution (stations et blocs). En utilisant cette solution, nous appliquons une découpe en choisissant un certain nombre de premières stations. Toutes les opérations affectées à ces stations et blocs sont mises dans $\mathbf{N}^1$. L'ensemble **N** est modifié par soustraction des opérations appartenant à $\mathbf{N}^1$. Ensuite, nous analysons la solution heuristique par rapport à $\mathbf{N}^1$ et décidons de passer ou non à l'utilisation de MIP pour $\mathbf{N}^1$ en fonction de la qualité de la solution heuristique obtenue pour $\mathbf{N}^1$.

Quand nous décidons de construire et de lancer le modèle MIP correspondant, si au bout d'un certain temps nous n'avons toujours pas atteint l'optimum, nous arrêtons MIP et la meilleure solution courante de MIP est comparée à la solution heuristique. La meilleure de deux est gardée pour $\mathbf{N}^1$. Et ainsi de suite pour $\mathbf{N}^2$, $\mathbf{N}^3$, ... tant que nous n'affectons pas toutes les opérations de **N**.

Introduisons des notations suivantes :

T_{elm} le temps de calcul alloué à un modèle partiel de MIP (pour un ensemble $\mathbf{N}^u$) ;

T_{tot} est le temps total alloué pour l'optimisation.

Algorithme 5-2 :

***Etape* 1.** Lancer FSIC (ou RAP) pour **N** et trouver une solution ; $w = 1$.

***Etape* 2.** Prendre cette solution et décider où la couper ce qui revient à décider le nombre de stations à prendre : $\mathbf{N}^w$ est égal à l'ensemble des opérations de ces stations ; $\mathbf{N} = \mathbf{N} \setminus \mathbf{N}^w$.

***Etape* 3.** Analyser la solution heuristique pour $\mathbf{N}^w$:

si il faut utiliser MIP pour $\mathbf{N}^w$, aller à l'*Etape* 4 ;

sinon aller à l'*Etape* 5 **finsi**

***Etape* 4.** Lancer MIP

si MIP ne donne pas de solution au bout de T_{elm} , MIP est stoppé : la solution courante de MIP est comparée à la solution partielle de l'heuristique ; la meilleure solution pour $\mathbf{N}^w$ est gardée.

sinon la solution de MIP est gardée **finsi**

***Etape* 5.** **si** il reste des opérations non affectées, alors $w = w + 1$, aller à l'*Etape* 2.

sinon

si le temps de calcul total < T_{tot}

aller à l'*Etape* 1;

sinon Fin de l'algorithme **finsi**

finsi

5.6. Conclusions

Nous avons montré dans ce chapitre qu'il était possible de coupler les méthodes exactes présentées au Chapitre 3 avec des méthodes heuristiques présentées au Chapitre 4 pour pouvoir aborder des problèmes de relativement grande taille. Ce couplage, s'il est bien fait, permet de tirer profit des avantages de deux approches.

Nous avons pensé à plusieurs approches de ce type en commençant par la plus simple et évidente : une méthode heuristique donne une borne supérieure à MIP et ainsi permet aux calculs d'être accélérés. En perspective, il serait intéressant de voir si en affinant la borne par des calculs heuristiques plus longs, les résultats peuvent être encore meilleurs.

Pour résoudre des problèmes de taille plus importante, nous avons proposé de faire appel à une décomposition. Nous avons développé deux approches de décomposition : la première est déterministe et par conséquent rapide, la deuxième est stochastique et, à cause de cela, il faut exécuter un grand nombre d'itérations pour pouvoir avoir un bon résultat. L'approche déterministe a montré des résultats encourageants. Mais nous sommes conscients qu'elle est sensible à la structure des contraintes et peut donner des solutions éloignées de l'optimum.

Une perspective intéressante est la décomposition stochastique basée sur l'utilisation de nos heuristiques. Nous avons montré comment ce couplage de deux approches peut-être fait en tenant compte du temps de calcul limité d'un coté et de la nécessité d'exécuter un nombre d'itérations relativement grand, de l'autre. Dans nos travaux futurs, nous envisageons de tester et de développer davantage cette approche.

Conclusion générale

Conclusion Générale

Nous avons étudié des lignes d'usinage où plusieurs opérations peuvent être exécutées en même temps par la même tête d'usinage. Chaque station contient une ou plusieurs têtes d'usinage, les têtes de chaque station sont activées séquentiellement, les stations sont synchronisées par un convoyeur commun. Le problème consiste à repartir les opérations par tête d'usinage et par station en minimisant le coût de la ligne et en tenant compte des contraintes de compatibilité des opérations et des têtes d'usinage ainsi que d'autres contraintes liées avec la spécificité du processus d'usinage dans ce type de lignes. Le critère d'optimisation est le coût de la ligne mesuré par la somme pondérée du nombre de stations et du nombre de têtes d'usinage.

Nous avons proposé un ensemble de modèles et d'algorithmes nouveaux et efficaces pour ce problème difficile. Nos méthodes se basent sur deux principales approches : la programmation linéaire en variables mixtes et la recherche stochastique dans un arbre d'énumération. La première approche nous a permis de développer des modèles MIP fournissant des solutions optimales pour le problème considéré, la deuxième approche s'est soldée par des algorithmes qui ne garantissent pas l'optimalité des solutions mais qui sont plus rapides. Nous avons également combiné les deux approches pour obtenir des algorithmes heuristiques intégrant les modèles exacts (en les utilisant pour des sous-problèmes). Ce couplage s'est basé sur une approche de décomposition du problème initial.

Les modèles MIP proposés ont été programmés avec ILOG Cplex, des tests ont été faits pour des cas industriels et des cas générés aléatoirement. Les résultats sont encourageants. Pour des problèmes industriels allant jusqu'à 30 opérations nous pouvons les utiliser avec un temps de calcul ne dépassant pas quelques minutes. Si le budget temps est plus grand (ce qui est souvent le cas lors de la conception des lignes qui dure plusieurs mois), nous pouvons espérer avoir des décisions optimales pour des problèmes de taille plus grande (disons jusqu'à

80 opérations et même avec des moyens de calcul plus puissants et disponibles allant jusqu'à 100 opérations dans certains cas).

Les perspectives liées avec les modèles MIP que nous avons proposés concernent l'amélioration des structures de données utilisées, l'optimisation des appels Cplex et les techniques plus avancées d'utilisation de Cplex liées au contrôle de branchements et de coupes, à l'intégration de bornes supplémentaires, ... Nous devons également essayer d'obtenir de meilleures bornes et une meilleure estimation des paramètres du modèle comme par exemple n_0 et m_0.

Les méthodes heuristiques basées sur la recherche aléatoire dans l'arbre d'énumération (et plus particulièrement sur les techniques de COMSOAL) sont également assez performantes. Nous avons pu constater que, dans de nombreux cas, elles donnent des solutions optimales ou proches de l'optimum. Le temps de calcul est relativement petit. L'approche la plus prometteuse est liée avec nos techniques de retour en arrière dans l'arbre d'énumération (BAL). Une étude plus approfondie sur l'apprentissage (choix des paramètres) de la méthode est nécessaire.

Il nous paraît intéressant de continuer la voie de la décomposition spatiale. Nous avons proposé plusieurs algorithmes et avons testé un certains nombres d'entre eux. Les résultats demandent de l'approfondissement mais sont déjà intéressants. L'approche liée avec la décomposition dynamique en utilisant des heuristiques, avec l'appel ou non du modèle MIP en fonction des évaluations faites sur des sous problèmes obtenus et du budget de temps, sera testée et améliorée très prochainement.

Les problèmes réels de ce type dépassent rarement 150 opérations. Dans certains cas ce nombre peut être facilement réduit en regroupant les opérations en macro-opérations. Comme nous l'avons montré dans la thèse, certains exemples industriels sont dimensionnés à environ 30 opérations (même moins). Nous pouvons donc dire que nos algorithmes sont déjà partiellement opérationnels. Nous l'avons démontré en traitant avec succès 4 cas industriels qui nous ont été fournis par nos partenaires de MZAL.

Une autre perspective concerne la généralisation de cette approche à d'autres types de lignes, par exemple, à des lignes sur lesquelles toutes les têtes d'usinage sont activées simultanément ou alors qui présentent deux possibilités d'activation (séquentielle et en série). Nous devons également intégrer d'autres contraintes, comme celles liées aux modes d'usinage, admissibles et optimisées (optimisation de la vitesse de coupe, par exemple). Le développement d'un prototype informatique convivial intégrant tous nos programmes et algorithmes est également à l'ordre du jour.

Les méthodes qui ont été développées dans cette thèse concernent des lignes d'usinage en production de masse et en grande série. Elles prennent en compte un ensemble de gammes issues d'une gamme-mère pour une famille de produits très proches : tous les produits passent sur toutes les machines de la ligne mais peuvent "sauter" une opération. C'est déjà une classe assez large des lignes d'usinage, mais il serait intéressant de généraliser les modèles proposés au cas des lignes d'usinage flexibles capables de travailler des familles de pièces différentes avec des gammes diverses. Pour ces lignes flexibles de nouvelles contraintes et de nouveaux composants de coûts doivent être pris en considération.

BIBLIOGRAPHIE

Amen, M., (2000a), An exact method for cost-oriented assembly line balancing, *International Journal of Production Economics*, 68 (1-3), p. 187-95.

Amen, M., (2000b), Heuristic methods for cost-oriented assembly line balancing: a survey, *International Journal of Production Economics*, 68, p. 1-14.

Amen, M., (2001), Heuristic methods for cost-oriented assembly line balancing: a comparison on solution quality and computing time, *International Journal of Production Economics*, 69, p. 255-264.

Andradottir, S. (1998), Simulation optimization. *Handbook on Simulation*, J. Banks (ed.), John Wiley & Sons, Inc., New York, p. 307-333.

Arcus, A.L., (1966), COMSOAL: a computer method of sequencing operations for assembly lines, *International Journal of Production Research*, 4, p. 259-277.

Arnold, H., (2001), *The recent history of the machine tool industry and the effects of technological change*, LMU, University of Munich, Institute for Innovation research and Technology management, 56 pages.

Askin, R.G. and Standridge, C.R., (1993), *Modeling and analysis of manufacturing systems*, John Wiley & Sons, Inc.

Baptiste, P., Le Pape C., Nuijten W. (2003) *Constraint Based Scheduling*: *Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers.

Baybars, I., (1986), A survey of exact algorithms for the simple assembly line balancing problem, *Management Science*, 32, p. 909–932.

Becker, C. and Scholl, A., (2004), A survey on problems and methods in generalized assembly line balancing, *European Journal of Operational Research*, 26 pages, (in Press).

Bernard, A., (sous la direction de Martin, P.), (2003), *IC2 Productique – Information – Commande – Communication,* FAO – Hermès Science Publications.

Bloch, C., Portmann, M., Vignier, A., (2003), Application des algorithmes génétiques à l'ordonnancement de la production, *Résolution de problèmes de RO par les métaheuristiques*, M. Pirlot, J. Teghem (eds), Hermès, Lavoisier.

Bohez, E.L.J., (2002), Five-axis milling machine tool kinematic chain design and analysis. *International Journal of Machine Tools and Manufacture*, 42, p. 505-520.

Boudhar, M. and Finke, G., (2001), Scheduling on batch processing machines with constraints of compatibility between jobs, *Management and Control of Production and Logistics: A Proceedings Volume of the IFAC Conference*, Z. Binder (ed.), Elsevier Science, vol. 3, p.703-708.

Boutevin, C., (2003), *Problèmes d'ordonnancement et d'affectation avec contraintes de ressources de type RCPSP et line balancing*, Thèse de Doctorat, Université Blaise Pascal, Clermont-Ferrand II, 2003, 306 pages.

Bowman, F.H., (1960), Assembly-line balancing by linear programming, *Operations Research*, 8 (3), p. 385-389.

Buffa, E.S. and Taubert, W.H., (1972) *Production-Inventory Systems: Planning and Control*, Irwin.

Bukchin, J., Dar-El, E.M. and Rubinovitz, J., (2002), Mixed-model assembly line design in make -to -order environment, *Computers & Industrial Engineering*, 41, p. 405-421.

Buzacott J.A. and Hanifin, L.E., (1978), Models of automatic transfer lines with inventory banks: a review and comparison, *AIIE Transactions*, 10 (2), p. 197-207.

Carlier, J., (1982), The one-machine sequencing problem, *European Journal of Operations Research*, 11 (2), p. 42-47.

Carlier, J., et Chretienne, P., (1988), *Problèmes d'ordonnancement ; modélisation / complexité / algorithmes*, Editions Masson.

Carlier, J. and Pinson, E., (1989), An algorithm for solving the job shop problem, *Management Science*, 35 (2), p. 164-176.

Caux, C., Pierreval, H., et Portmann, M. C, Les algorithmes génétiques et leur application aux problèmes d'ordonnancement. *Automatique Productique Informatique Industrielle (RAIRO APII)*, 29 (4-5), p. 409-443.

Cplex, (2001), Ilog Cplex 7.1 : *User's Manual*, ILOG, 360 pages.

Dallery, Y., David R. and Xie, X.L., (1988), An efficient algorithm for analysis of transfer lines with unreliable machines and finite buffers, *IIE Transactions*, 20 (9), p. 280-283.

Danzig, G. B., (1990) Origins of the simplex method, *An history of scientific computing*, S. G. Nash (ed.), p. 141-151.

Dar-El Mansoor, E.M., (1964), Assembly Line Balancing: an improvement on the ranked positional technique, *Journal of Industrial Engineering*, 15 (2), p. 73-77.

Demeulemeester, E. and Herroelen, W., (1992), A branch and bound procedure for multiple resource-constrained project scheduling problem, *Management Science*, 38, p. 1803-1818.

Dincbas, M., Callaos, N., Hernandez-Encinas, L. and Yetim, F., (2002), Application of constraint programming in business process optimisation, *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics.* Orlando, FL, Int. Inst. Inf. & Syst, vol. 10, p. 272-273.

Dolgui, A., Ereemev, A., Kolokolov, A. and Sigaev, V., (2002a), Buffer Allocation in Production Line with Unreliable Machines, *Journal of Mathematical Modeling and Algorithms*, 2, p. 89-104.

Dolgui, A., and Finel., B., (2002), Load balancing for transfer lines, *Proceedings of the International Conference on Theory of Conflict and its Applications*, Voronezh, Russia, 2002, p. 134-139.

Dolgui, A., Finel, B., Guschinsky, N., Levin, G. and Vernadat, F., (2002b), Load balancing for transfer lines: new algorithms and some comparisons, *Proceedings of the International Conference on Production System Design, Supply Chain Planning and Optimization*, 23-25 October, 2002, Szczecin, Poland. Informa, 2002, vol. 1, p. 13-21.

Dolgui, A., Finel, B., Guschinsky, N., Levin, G. and Vernadat, F., (2002c), A Transfer Line Design and Balancing Approach, *Proceedings of the 1st CIRP (UK) International Seminar on 'Digital Enterprise Technology'*, Durham, UK, September 16-19, 2002, CD-ROM, 4 pages.

Dolgui, A., Finel, B., Guschinsky, N., Levin, G. and Vernadat, F., (2003), Some Optimization Approaches for Transfer Lines with Blocks of Parallel Operations. *Preprints of the 7th IFAC Symposium on Intelligent Manufacturing Systems*, L. Monostori, B. Kadar, G. Morel (Eds.), 6-8 April 2003, Budapest, Hungary, p. 261-266.

Dolgui A. , Finel, B., Guschinsky, N., Levin, G. and Vernadat, F., (2005), An Heuristic Approach for Transfer Lines Balancing. *Journal of Intelligent Manufacturing*, 16 (2), p. 159-171.

Dolgui, A., Finel, B., Guschinsky, N., Levin, G. and Vernadat. F., MIP approach to balancing transfer lines with blocks of parallel operations, *IIE Transactions* (in revision).

Dolgui, A., Guschinski, N., Levin, G., (1999a), Optimal Design of Transfer Lines and Multi-position Machines, *Proceedings of the 7th IEEE Mediterranean Conference on Control and Automation (MED'99)*. Haifa, Israel, June 28-30, CD-ROM, p. 1962-1973.

Dolgui, A., Guschinski, N. and Levin, G., (1999b), On problem of optimal design of transfer lines with parallel and sequential operations. *Proceedings of the 7th International Conference on Emerging Technologies and Factory Automation* (ETFA'99), IEEE, Spain, (J.M.Fuertes Ed.), vol. 1, p. 329-334.

Dolgui A., Guschinsky, N. and Levin, G., (2000), *Approaches to balancing of transfer lines with blocks of parallel operations*; Institute of Engineering Cybernetics/University of Technology of Troyes, Minsk Academy of Sciences, Preprint N°8, 42 pages.

Dolgui, A., Guschinski, N. and Levin, G., (2001a), Decomposition Methods to Optimize Transfer Line with Parallel and Sequential Machining, *Management and Control of Production and Logistics: A Proceedings Volume of the IFAC Conference*, Z. Binder (ed), Elsevier Science, vol. 3, p. 983-988.

Dolgui, A., Guschinski, N., Levin, G., (2004), A Special Case of Transfer Lines Balancing by Graph Approach. *European Journal of Operational Research*, (in Press).

Dolgui, A., Guschinski, N., Levin, G., Harrath, Y., (2002d), Un modèle de programmation linaire pour l'équilibrage des charges des lignes de transfert. *Journal Européen des Systèmes Automatisés* (JESA), 36 (1), p. 11-31.

Dolgui, A., Guschinski, N., Levin, G., Harrath, Y., (2001b). Optimal design of class transfer lines with blocks of parallel operations. *Manufacturing, Modeling, Management and Control: A Proceedings Volume of the IFAC Symposium*, P. Groumpos (ed.), Elsevier Sciences, p. 36-41.

Dolgui, A., Guschinski, N., Levin, G. and Proth, J.M. (2004), Graph approach for balancing a class of transfer lines. *European Journal of Operational Research*, (submitted).

Dolgui, A. and Ofitserov, D., (1997), A Stochastic Method for Discrete and Continuous Optimization in Manufacturing Systems, *Journal of Intelligent Manufacturing*, 8 (5), p. 405-413.

Dolgui, A., Pashkevich, A., (2001), *Manufacturing Systems Design*, Université de Technologie de Troyes, 304 pages.

Dolgui, A., et Thirion, C., (1999), Utilisation des plans d'expériences pour le paramétrage des heuristiques d'ordonnancement dans un atelier job-shop, *Actes de la deuxième conférence francophone de Modélisation et Simulation (MOSIM'99)*, G. Habchi, A. Haurat (eds.), SCS Publication, p. 351-356.

Erel, E. and Sarin, S.C., (1998), A survey of the assembly line balancing procedures, *Production Planning and Control*, 9 (5), p. 414-434.

Eurostat (2004), *Statistiques en bref : Industries Commerces et Services*, thème 4-18.

Felder, E., (1997), Procédés d'usinage : présentation, *Techniques de l'Ingénieur*, B7 000, 15 pages.

Feuvrier, (1971), *La simulation des systèmes*, Dunod.

Finel B., (1991) *La programmation par contraintes*, Examen probatoire en Informatique, Camos, CNAM.

Finel, B., Dolgui, A., et Vernadat, F., (2003), Structuration des lignes de transfert à l'étape de leur conception préliminaire, *Proceedings of the 5th International Industrial Engineering Conference "Industrial Engineering and the New Global Challenges"*, D. Ait-Kadi, S. D'Amours (eds.), 26 au 29 octobre 2003, Québec, Canada, CD-ROM, 10 pages.

François, P., (1995), *Modélisation des machines de production*, rapport de DEA Production Automatisée, ENS Cachan.

Ghosh, S. and Gagnon, R., (1989), A comprehensive literature review and analysis of the design, balancing and scheduling of assembly lines, *International Journal of Production Research*, 27 (4), p. 637-670.

Giard, V., (2003), *Gestion de la production et des flux*, Economica.

Gill, P.E., Murray, W. and Wright, M.H., (1981), *Practical Optimization,* Academic Press, New York.

Glover, F. and Laguna, M., (1997), *Tabu search*, Kluwer Academic Publishers..

Goldberg, D.E., (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley.

Gondran, M., et Minoux, M., (1979), *Graphes et Algorithmes*, Editions Eyrolles, Paris.

Hackman, S.T., Magazine, M.J. and Wee, T.S., (1989), Fast effective algorithms for simple assembly line balancing problems, *Operations Research*, 37 (6), p. 916-924.

Harrath, Y. (2000), *Méthodes d'optimisation combinatoire pour la conception préliminaire des lignes de transfert en production mécanique*, Mémoire DEA, Université de technologie de Troyes, 42 pages.

Helgerson, W.P. and Birnie, D.P., (1961), Assembly line balancing method using the ranked positional weight technique, *Journal of Industrial Engineering*, 12 (6), p. 394-398.

Hertz, A., Widmer, M., (1995), La méthode Tabou appliquée aux problèmes d'ordonnancement, *Automatique, Productique, Informatique industrielle* (*RAIRO APII*), 29 (4-5), p. 253-378.

Hitomi, K., (1995), *Manufacturing Systems Engineering*, Taylor & Francis.

Hoffmann, T.R., (1992), Eureka: A hybrid system for assembly line balancing, *Management Science*, 38, p. 39-47.

Ignall, E.J., (1965), A review of assembly line balancing, *Journal of Industrial Engineering*, 16, p. 244-254.

Jackson, J.R., (1956), A computing procedure for the line balancing problem, *Management Science*, 2, p. 261-271.

John, G., (2003), *Rethinking Your CPLEX Parameter Settings*, ILOG CPLEX Development Team, 3 pages.

Johnson, L.A., Montgomery, D.C., (1974) *Operations research in production planning, scheduling and inventory control*, John Wiley & Sons, New York.

Johnson, R.V., (1988), Optimally Balancing Large Assembly Lines With FABLE, *Management Science*, 34 (2), February 1988, p. 240-253.

Jones, D.R. and Beltramo, M.A., (1991), Solving partitioning problems with genetic algorithms, *Proceedings of ICGA91*, San Diego, USA, p. 442-449.

Kilbridge, M.D. and Wester, L., (1961), A heuristic method of assembly line balancing, *Journal of Industrial Engineering*, 12 (4), p. 292-298.

Kim, J.Y., Kim, Y. and Kim, Y.K., (2001), An endosymbiotic evolutionary algorithm for optimization, *Applied Intelligence*, 15 (2), p. 117-130.

Kirkpatrick, C.D., Gelatt, Jr., Vecchi, M.P., (1983), Optimization by Simulated Annealing, *Science*, 220 (4598), p. 671-680.

Klein, R. and Scholl, A., (1996), Maximizing the production rate in simple assembly line balancing-a branch and bound procedure, *European Journal of Operational Research*, 91 (2), p. 367-380.

Lapierre, S.D., Ruiz, A., Soriano, D., (2004), Balancing assembly lines with Tabu search, *European Journal of Operational Research*, (in Press).

Lecomte, C., (1993), *Un système à base de règles d'aide à l'ordonnancement d'un atelier travaillant à la commande*, Thèse de doctorat, Ecole Centrale de Paris.

Levy, M.L., (1996), *Méthodes par décomposition temporelle et problèmes d'ordonnancement*, Thèse de doctorat, Institut National de Polytechnique de Toulouse, 1996.

Malakooti, B.B., (1994), Assembly line balancing with buffers by multiple criteria optimization, *International Journal of Production Research*, 32 (9), p. 2159-78.

Martin, P. and D'Acunto, A., (2003), Design of a production system: an application of integration product-process, *International Journal of Computer Integrated Manufacturing*, 16 (7-8), p. 509-516.

McDonnel Feit, E. and Wu, S.D., (2000), Transfer Line Design with Uncertain Machine Performance Information, *IEEE Transactions on Robotics and Automation*, 16 (5), p. 581-587.

McMullen, P.R. and Frazier, G.V., (1998), Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel stations, *International Journal of Production Research*, 36, p. 2717-2741.

McMullen, P.R. and Tarasewich, P., (2003), Using ant techniques to solve the assembly line balancing problem, *IIE Transactions*, 35 (7), p. 605-617.

Minzu, V. and Henrioud, J.M., (1997a), Tasks-to-workstations assignment in assembly systems using a stochastic algorithm, *Control of Industrial Systems. Control for the Future of the Youth*: *Proceedings Volume of the IFAC Conference*, vol. 3, p. 1403-1408.

Minzu, V. and Henrioud, J.M., (1997b), Stochastic algorithm for tasks assignment in single or mixed-model assembly lines, *Journal Européen des Systèmes Automatisés (APII-JESA)*, 32 (7-8), p. 831-851.

Moodie, C.L. and Young, H.H., (1995), A heuristic method of assembly line balancing for assumptions of constant or variable work element times, *Journal of Industrial Engineering*, 16 (1), p. 23-29.

Nemhauser, G.L. and Wolsey, L.A., (1988), *Integer and Combinatorial Optimization*, John Wiley & Sons, New York.

Nkasu, M.M. and Leung, K.H., (1995), A Stochastic Approach to Assembly Line Balancing, *International Journal of Production Research*, 33 (4), p. 975-991.

Nof, S.Y., Wilhelm, W.E. and Warnecke, H.J., (1997), *Industrial Assembly*, Chapman & Hall.

Oumara, A., and Finke, G., (2001), Flow shop problems with two batch processing machines, *International Journal of Mathematical Algorithms*, 2, p. 269-287.

Patterson, J.H. and Albracht, J.J., (1975), Assembly-Line Balancing: Zero-One Programming with Fibonacci Search, *Operations Research*, 23, p. 166-172.

Pierreval, H., Caux, C., Paris, J.L. and Viguier, F., (2003), Evolutionary approaches to design and optimisation of manufacturing systems. *Computer & Industrial Engineering*, 44, p. 339-364.

Ponnambalam, S.G., Aravindan, P. and Mogileeswar Naidu, G., (1999), A comparative evaluation of assembly line balancing heuristics, *International Journal of Advanced Manufacturing Technology*, 15 (8), p. 577-86.

Portmann, M.C., (1988), Méthodes de décompositions spatiale et temporelle en ordonnancement de la production, *Automatique, Productique, Informatique Industrielle* (*RAIRO-APII*), 22 (5), p. 439-451.

Portmann, M.C., (1996), Genetic algorithm and scheduling: a state of the art and some propositions, *Proceedings of the Workshop on Production Planning and Control*, Mons, Belgium, p. I-XIV.

Proth, J.M., (1992), *Conception et gestion des systèmes de production*, Paris, PUF.

Pruvost, F., (1993), *Conception et calcul des machines-outils*, vol. 1, Presses Polytechniques et Universitaires Romanes.

Pruvost, F., (1997), Machine-outil : Présentation, *Techniques de l'ingénieur*, B7 120-1, 10 pages.

Quantinet, B., (2004), *Un prototype de système d'aide à la décision pour la conception des machines et des lignes d'usinage*, Rapport de stage technique de 2ème année, Ecole des Mines de Saint-Etienne, 38 pages.

Rekiek, B., (2000), *Assembly Line Design (multiple objective grouping genetic algorithm and the balancing of mixed-model hybrid assembly line)*, PhD Thesis, Université Libre de Bruxelles, CAD/CAM Department, Brussels, Belgium.

Rekiek, B., De Lit, P., and Delchambre, A., (2000), Designing mixed-product assembly lines, *IEEE Transactions on Robotics and Automation*, 16 (3), p. 268-280.

Rekiek, B. and Delchambre, A., (2001), Application de l'approche "Equal Piles" à l'équilibrage des lignes d'assemblage, *Actes de la 3e Conférence de Modélisation et SIMulation "Conception, Analyse et Gestion des Systèmes Industriels" (MOSIM'01)*, Troyes (France), 25-27 avril 2001, SCS Pulishing House, 2001, vol. 1, p. 347 – 352.

Rekiek, B., Dolgui, A., Delchambre, A. and Bratcu, A., (2002), State of the art of assembly lines design optimization. *Annual Reviews in Control*, 26/2, p. 163-174.

Rubinovitz, J. and Levitin, G., (1995), Genetic algorithm for assembly line balancing, *International Journal of Production Economics*, 41 (1-3), p. 343-354.

Scholl, A., (1999), *Balancing and sequencing of assembly lines*, Heidelberg Physica.

Scholl, A. and Becker, C., (2003), *A note on "An exact method for cost-oriented assembly line balancing"*. Friedrich-Schiller-Universität Jena, D-07743.

Scholl, A. and Becker, C., (2004), State-of-art exact and heuristic solution procedures for simple assembly line balancing, *European Journal of Operational Research*, 35 pages, (in Press).

Scholl, A. and Klein, R., (1997), SALOME: A bi-directional branch and bound procedure for assembly line balancing. *INFORMS Journal on Computing,* 9, p. 319-334.

Scholl, A. and Klein, R., (1999), Balancing Assembly lines effectively - A computational Comparison, *European Journal of Operational Research*, 114, p. 50-58.

Schrage, L. and Baker, K., (1978), Dynamic programming solution of sequencing problems with precedence constraints, *Operations Research*, 26, p. 444-449.

Sessi, (2004), *Les Quatre pages de statistiques industrielles*, DiGITIP, Ministère de l'Économie, des Finances et de l'Industrie, N° 188, avril 2004.

Soriano, P. and Gendreau, M., (1997), Fondements et applications des méthodes de recherche avec Tabou, *RAIRO Operations Research*, 31, p. 133-159.

Sotskov, Yu., Dolgui, A. and Portmann, M.C., (2004), Stability analysis of an optimal balance for an assembly line with fixed cycle time, *European Journal of Operational Research*, (in Press).

Sprecher, A., (1999), A competitive branch-and-bound algorithm for the simple assembly line balancing problem, *International Journal of Production Research*, 8, p. 1787-1816.

Surech, G. and Sahu, S., (1994), Stochastic Assembly Lines Balancing Using Simulated Annealing, *International Journal of Production Research*, 32 (8), p. 1801-1810.

Swisher, J.R., Hyden, P.D., Jacobson, S.H. and Schruben, L. W., (2000), A Survey of Simulation Optimization Techniques and Procedures. *Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick (eds.), p. 119 –128.

Talbot, F. B. and Patterson, J.H., (1984), An Integer Programming Algorithm with Network Cuts for Solving the Assembly Line Balancing Problem, *Management Science*, 30, p. 85-99.

Talbot, F. B., Patterson, J.H. and Gehrlein, W., (1986), A comparative evaluation of heuristic line balancing techniques, *Management Science*, 32, p. 430-454.

Van Laarhoven, P.J.M., Aarts, E.H.L. and Lenstra, J.K., (1992), Job-shop scheduling by simulated annealing, *Operations Research*, 40, p. 113-125.

Van Zante-de Fokkert, J.I. and De Kok, T.G., (1997), The mixed and multi model line balancing problem: comparison, *European Journal of Operational Research*, 100 (3), p. 399-412.

Vernadat, F., (1999), *Technique de Modélisation en Entreprise : Applications aux Processus Opérationnels*, Economica, 129 pages.

Vrat, P. and Virani, A., (1976), A Cost Model for Optimal Mix of Balanced Stochastic Assembly Lines and the Modular Assembly Systems for a Customer Oriented Production Systems, *International Journal of Production Research*, 14 (4), p. 445-463.

Wen-Chyuan, C., (1998), The application of a Tabu search metaheuristic to the assembly line balancing problem, *Annals of Operations Research*, 77, p. 209-27.

Williams, H.P., (1993), *Model Building in Mathematical Programming*, John Wiley & Sons, New York.

Zeigler, B.P., (1984), *Multifacetted Modelling and Discrete Event Simulation*. London, Academic Press.

Zeigler, B.P., (2000), *The Theory of Modeling and Simulation*, John Wiley & Sons, New York.