Algorithmes approchés pour des problèmes d'ordonnancement multicritères de type job shop flexible et job shop multiressource

Geoffrey Vilcot

UNIVERSITÉ FRANÇOIS RABELAIS
TOURS

École Doctorale : Santé, Sciences
et Technologies

Année Universitaire : 2006-2007

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE TOURS

Discipline : Informatique

présentée et soutenue publiquement

par :

Geoffrey VILCOT

le 19 novembre 2007

Algorithmes approchés pour des problèmes d'ordonnancement multicritères de type job shop flexible et job shop multiressource

---

Directeur de thèse : Jean-Charles Billaut

JURY :

Nom	**Qualité**	**Lieu d'exercice**
BILLAUT Jean-Charles	Professeur des universités	Ecole Polytechnique de l'université de Tours
DAUZÈRE-PÉRÈS Stéphane	Professeur	Ecole des Mines de St-Etienne
LOPEZ Pierre	Chercheur	LAAS-CNRS, Toulouse
PORTMANN Marie-Claude	Professeur des universités	Ecole des Mines de Nancy
T'KINDT Vincent	Maître de Conférences, HDR	Ecole Polytechnique de l'université de Tours
TEGHEM Jacques	Professeur	Faculté Polytechnique de Mons

Membre invité : BULÉON Karl, Responsable produit, Volume Software, Tours

Avant-propos

It's been a long road, getting from there to here.
It's been a long time, but my time is finally near.
(Diane Warren - *Where My Heart Will Take Me (Faith of the Heart)*)

Le travail présenté dans ce document à été réalisé au sein du Laboratoire d'Informatique de l'Université François-Rabelais de Tours ainsi que dans l'entreprise Volume Software dont le siège est basé à Tours.

J'aimerais remercier Jean-Charles Billaut pour avoir été mon encadrant de thèse pendant ces 3 années ainsi que Karl Buléon qui est le responsable de l'équipe *nouveaux produits* à Volume Software et qui m'a accueilli dans son équipe.

Ensuite j'aimerais remercier toute l'équipe « Ordonnancement et Conduite » du laboratoire, Armand de Garsignies PDG de Volume Software, Jean Carton qui m'a prodigué de nombreux conseils quant à la programmation sous VB.Net. Enfin j'aimerais remercier Yannick Kergosien et Julien Janvier, deux étudiants du cycle d'ingénieur qui m'ont aidé dans mes recherches.

J'adresse aussi mes remerciements à Marie-Claude Portmann et Stéphane Dauzère-Pérès d'avoir acceptés d'être rapporteur de ma thèse, ainsi qu'à Jacques Teghem, Pierre Lopez et Vincent T'Kindt pour avoir acceptés d'être dans mon jury.

Table des matières

Avant-propos **3**

Glossaire **9**

Introduction **11**

1 Présentation du contexte industriel et scientifique **13**

1.1 Contexte industriel 13

1.2 Modélisation du problème 14

1.3 Optimisation multicritère 16

1.3.1 Définition de l'optimalité 16

1.3.2 Détermination des optima de Pareto 16

1.4 Notions sur la théorie de la complexité 18

1.5 Généralités sur les méthodes de résolution 19

1.5.1 Recherche Tabou 20

1.5.2 Algorithme Génétique 21

1.6 Présentation du document 22

2 État de l'art **25**

2.1 Job shop flexible 25

2.2 Job shop multiressource 34

2.3 Job shop multicritère 36

3 Job shop flexible multicritère **39**

3.1 Définition formelle du problème 39

3.2 Modèles de PLNE 40

3.3 Approche Tabou pour le job shop flexible 43

3.3.1 Les points communs . . . 43
3.3.2 Approche Epsilon-contrainte, $TS\epsilon$ . . . 48
3.3.3 Approche par combinaison linéaire des critères, $TS\ell c$ . . . 49
3.4 Approche par algorithme génétique pour le job shop flexible . . . 51
3.4.1 Un algorithme génétique à codage indirect, AG_{ind} . . . 51
3.4.2 Un algorithme génétique avec un codage direct, AG_{dir} . . . 53
3.4.3 Algorithme Mémétique, Mem . . . 58
3.5 Expérimentations . . . 60
3.5.1 Protocole expérimental . . . 60
3.5.2 Indicateurs . . . 62
3.5.3 Expérimentations préliminaires . . . 63
3.5.4 Comparaison des méthodes . . . 64
3.5.5 Comparaison avec un front de référence . . . 74
3.5.6 Comparaison avec la recherche Tabou de Dauzère-Pérès et Paulli . 77
3.5.7 Expérimentation sur les critères du *makespan* et de la somme des retards . . . 83
3.6 Conclusion sur le problème de job shop flexible . . . 86

4 Job shop multiressource multicritère **87**
4.1 Définition formelle du problème . . . 87
4.2 Modélisation par un PLNE . . . 88
4.3 Modélisations par un graphe . . . 89
4.3.1 Graphe avec un sommet par opération . . . 89
4.3.2 Graphe avec un sommet par sous-opération . . . 90
4.4 Approche Tabou pour le job shop multiressource, TS_{MPT} . . . 93
4.5 Approche par algorithme génétique pour le job shop multiressource, AG_{MPT} 97
4.6 Expérimentations . . . 99
4.6.1 Les instances . . . 99
4.6.2 Expérimentations préliminaires . . . 100
4.6.3 Comparaison des méthodes . . . 100
4.6.4 Comparaison avec CPLEX . . . 102
4.7 Conclusion sur le job shop multiressource . . . 102

5 Applications et mise en oeuvre **105**

5.1 Présentation générale de DirectPlanning 105
5.2 Interface manuelle 106
5.2.1 Prise en compte des calendriers 107
5.2.2 Gestion des opérations verrouillées 108
5.2.3 Temps de montage dépendant de la séquence 108
5.2.4 Le non chevauchement des opérations 108
5.2.5 L'aide à la planification 110
5.2.6 Le module d'ordonnancement 112
5.3 Conclusion sur DirectPlanning 116

Conclusion **117**

A Expérimentations préliminaires **119**
A.1 Paramétrage des algorithmes pour le job shop flexible 119
A.1.1 Paramétrage des recherches Tabou 119
A.1.2 Paramétrage de l'algorithme génétique à codage indirect 122
A.1.3 Paramétrage de l'algorithme génétique à codage direct 124
A.1.4 Paramétrage de l'approche mémétique 125
A.2 Paramétrage des algorithmes pour le job shop multiressource 127
A.2.1 Paramétrage de la recherche Tabou 127
A.2.2 Paramétrage de l'algorithme génétique 128

Glossaire

- J : ensemble des travaux.
- n : nombre de travaux ($|J| = n$).
- n_i : nombre d'opérations dans le travail i.
- $\mathcal{R}$: ensemble des ressources.
- m : nombre de ressources ($|\mathcal{R}| = m$).
- R_k : la ressource k.
- $O_{i,j}$: opération j du travail i.
- O_{i,n_i} : dernière opération du travail i.
- $O_{i,j,k}$: sous-opération, s'exécutant sur R_k, de l'opération j du travail i.
- $\Gamma_{i,j}$: ensemble des successeurs directs de l'opération $O_{i,j}$.
- $\Gamma^{-1}_{i,j}$: ensemble des prédécesseurs directs de l'opération $O_{i,j}$.
- $t_{i,j}$: date de début réelle de l'opération $O_{i,j}$.
- $C_{i,j}$: date de fin réelle de l'opération $O_{i,j}$.
- C_i : date de fin réelle du travail i.
- $r_{i,j}$: date de début au plus tôt de l'opération $O_{i,j}$.
- r_i : date de début au plus tôt du travail i.
- $t_{i,j}$: date de début réel de l'opération $O_{i,j}$.
- $d_{i,j}$: date de fin au plus tard souhaitée de l'opération $O_{i,j}$.

- d_i : date de fin au plus tard souhaitée du travail i.
- $R_{i,j}$: ensemble des ressources pouvant exécuter l'opération $O_{i,j}$ (cas flexible).
- $R^m_{i,j}$: ensemble des ressources devant exécuter l'opération $O_{i,j}$ (cas multiressource).
- $p_{i,j,k}$: durée opératoire de l'opération $O_{i,j}$ sur la ressource R_k.
- $a(O_{i,j})$: ressource qui exécute l'opération $O_{i,j}$ dans la solution considérée.
- C_{max} : date de fin de la dernière opération ou *Makespan* ; $C_{max} = \max_{i=1..n} C_i$.
- ΣC : somme des dates de fin ; $\Sigma C = \sum_{i=1..n}(C_i)$.
- L_{max} : plus grand retard algébrique ; $L_{max} = \max_{i=1..n}(C_i - d_i)$.
- ΣT : somme des retards absolus ; $\Sigma T = \sum_{i=1..n}(\max(0; C_i - d_i))$.
- ΣU : nombre de travaux en retard.

Introduction

Les problèmes abordés dans ce document s'inscrivent dans le contexte d'un partenariat, sous forme d'un contrat CIFRE, entre la société Volume Software d'une part et le Laboratoire d'Informatique de l'Université de Tours d'autre part. Ce partenariat porte sur le développement du logiciel *DirectPlanning*. Ce logiciel est destiné à la planification et à l'ordonnancement dans des ateliers de production. Il se présente sous la forme d'un planning mural interactif. De plus il est très flexible et peut s'adapter facilement aux différents métiers des utilisateurs. Il possède aussi une interface vers les ERP et peut ainsi s'intégrer dans le système d'informations de l'entreprise. Par ailleurs, son module d'ordonnancement permet de proposer rapidement de très bonnes solutions à l'utilisateur.

La première version du logiciel a été, en partie, développée lors d'une thèse CIFRE précédente réalisée par Philippe Mauguière [Mauguière, 2004]. Grâce à son travail, *DirectPlanning* peut prendre en compte des problèmes de job shop avec temps de montage dépendants de la séquence, des calendriers et des tâches verrouillées.

Cependant, d'autres besoins industriels ont été remontés. Le premier de ces besoins est la notion de machines compatibles. En effet, il n'est pas rare qu'une machine se trouve en plusieurs exemplaires dans l'atelier de fabrication ou alors qu'une machine puisse réaliser différents types d'opérations. Alors que jusqu'à présent, c'était à l'utilisateur de choisir, a priori, l'affectation des opérations sur les ressources, il devenait primordial de pouvoir laisser le logiciel trouver de lui-même la ou les meilleures affectations possibles. Il en résulte l'étude du problème de job shop flexible présentée au chapitre 3.

En second lieu, les industriels nous ont demandé de pouvoir créer des tâches qui nécessiteraient plus d'une ressource pour être accomplies. Cela arrive, par exemple, lorsqu'une tâche a besoin à la fois d'une machine et d'un ou plusieurs opérateurs humains. Nous avons donc étudié dans le chapitre 4 les problèmes d'ordonnancement multiressources.

Au-delà de ces considérations, *DirectPlanning* s'adresse à un vaste ensemble hétérogène d'industries. Dans ces conditions, il est difficile, pour ne pas dire impossible, de déterminer quels seront les critères à optimiser pour chaque client. Nous avons donc opté pour une approche multicritère qui cherche à déterminer le front de Pareto. Au vu de la taille des instances industrielles que *DirectPlanning* est amené à résoudre, une approche exacte ne peut être envisagée. Nous nous sommes donc naturellement dirigés vers des méta-heuristiques.

La première méthode que nous utilisons est une recherche Tabou. Une recherche Tabou est une méthode de voisinage : elle part d'une solution existante et elle va définir des

solutions « proches », appelées des *voisins*, ces solutions différant peu de la précédente. Dans le cas d'un problème d'ordonnancement, un voisin peut, par exemple être défini par une simple interversion de deux opérations. Le meilleur voisin devient la solution courante et le processus recommence. Un mécanisme appelé *la liste Tabou* empêche de revenir sur une solution déjà explorée.

La seconde méthode que nous utilisons est un algorithme génétique, inspirée par la théorie de l'évolution de Darwin. Un tel algorithme manipule plusieurs solutions simultanément, que l'on appelle population d'individus cet ensemble. A une itération donnée, on va choisir de façon probabiliste des individus que l'on va croiser deux à deux, les meilleurs ayant plus de chances d'être choisis. De ce mélange, on obtient de nouveaux individus qui possèdent des caractéristiques de leurs deux parents. Un mécanisme de sélection est là pour éliminer les individus les moins bons. Ainsi, au fil des itérations, la population va évoluer vers de meilleurs individus. Ce genre d'algorithmes est particulièrement bien adapté pour déterminer une approximation du front de Pareto puisqu'ils manipulent un ensemble de solutions.

Une recherche Tabou et un algorithme génétique pour le problème de job shop flexible multicritère sont implémentés dans la future version de *DirectPlanning*.

Chapitre 1

Présentation du contexte industriel et scientifique

1.1 Contexte industriel

Ce travail de recherche s'inscrit dans le cadre d'une collaboration avec la société Volume Software. Cette société est spécialisée dans le développement d'outils informatiques dédiés aux industries de l'imprimerie et du cartonnage. Volume Software commercialise notamment les ERP (*Enterprise Resource Planning*) *VoluPack* et *VoluPrim*, dédiés à ces métiers.

Les industries de l'imprimerie et du cartonnage, comme la plupart des secteurs industriels, doivent faire face à des délais clients toujours plus contraignants et pouvoir gérer journalièrement de nombreuses modifications de commandes. Dans ce contexte, les problématiques d'ordonnancement d'ateliers prennent tout leur sens.

Les processus industriels de ces secteurs d'activités peuvent nécessiter des gammes simples ou complexes (opérations en parallèle). Par exemple, pour éditer un livre on doit faire face à des problématiques de mise en page, d'impression, de pliage, d'encartage, de découpe et de reliure. La diversité des ressources (machines et hommes) et des contraintes (attente d'outils nécessaires à la fabrication, arrêts machine, ...) conduit à des problèmes complexes mais concrets.

De la collaboration entre le Laboratoire d'Informatique de l'Université de Tours et de la société Volume Software est né le logiciel *DirectPlanning* (www.directplanning.com). Ce logiciel se présente comme un planning mural interactif. Bien que Volume Software soit spécialisée dans l'imprimerie et le cartonnage, *DirectPlanning* peut facilement être personnalisé pour n'importe quel type de métier. L'apport principal de la collaboration porte sur le module d'ordonnancement de *DirectPlanning*.

Les études précédemment menées ([Mauguière, 2004]) ont principalement porté sur le problème de job shop avec prise en compte des temps de préparation dépendants de la séquence et des calendriers [Mauguière *et al.*, 2005].

Toutefois, plusieurs clients de Volume Software ont fait remonter des besoins en termes

de fonctionnalités, qu'ils auraient aimés voir apparaître dans *DirectPlanning*.

Le premier de ces besoins a porté sur la possibilité de pouvoir changer dynamiquement l'affectation des tâches sur leurs ressources compatibles (une imprimante quatre couleurs peut effectuer des impressions en trois couleurs, certaines machines peuvent faire de l'impression et/ou de la découpe, etc.). Cela a mis en évidence l'existence de plusieurs ressources possibles pour effectuer une tâche, autrement dit un problème d'affectation, qui pouvait être résolu par le module d'ordonnancement.

Le second de ces besoins était de gérer finement la disponibilité de certains opérateurs à compétences particulières (caleurs), qui sont nécessaires au calage des impressions et de celles d'autres opérateurs (rouleurs par exemple), qui sont nécessaires en cours d'opération pour s'assurer du bon déroulement (remplissage de papier, déchargement des impressions au fur et à mesure, ...). Cela a mis en évidence le besoin de plusieurs ressources simultanément pour effectuer une tâche, autrement dit un problème multiressource, qui devait être pris en compte au niveau du module d'ordonnancement.

D'autre part, *DirectPlanning* est commercialisé dans un grand nombre d'entreprises, et pas uniquement dans le domaine de l'imprimerie et du cartonnage. Certains clients ont des problèmes pour tenir leurs délais, d'autres ont des problèmes de gestion des en-cours de production, etc. Les critères à optimiser dépendent donc des clients. De plus, avant qu'une solution ne soit retournée par le logiciel, certains clients souhaitaient avoir le choix entre plusieurs solutions, afin de pouvoir appliquer des critères subjectifs.

Il était donc naturel d'aborder le problème sous un angle multicritère, permettant ainsi au décideur de choisir le ou les critères à optimiser, de lui offrir une garantie sur ces critères, et de diversifier le nombre de solutions qui lui sont proposées.

Tous ces besoins sont à l'origine des problèmes abordés dans la suite du document.

1.2 Modélisation du problème

Les problèmes d'ateliers que nous considérons sont appelés dans la littérature des problèmes de *job shop*. Dans ces problèmes, on a des travaux composés d'opérations à ordonnancer sur des ressources, et l'objectif est d'optimiser un ou plusieurs critères. Dans la littérature au lieu du terme opération, on parle aussi de tâche, d'activité. Les ressources (renouvelables) peuvent être de nature différentes : machines, outils, opérateurs... Dans la suite de ce document, sauf mention contraire, nous utiliserons les termes génériques de travaux, d'opérations et de ressources.

Notations

Job shop Formellement un problème d'atelier « à cheminements multiples », ou problème de *job shop*, est défini ainsi : on considère un ensemble J composé de n travaux. Ces travaux doivent être ordonnancés sur m ressources disjonctives, l'ensemble des ressources est noté $\mathcal{R}$. On note R_k la $k^{ème}$ ressource. Chaque travail i est composé de n_i opérations. L'opération j du travail i est notée $O_{i,j}$. L'opération $O_{i,j}$ s'exécute sur la ressource

$R_{O_{i,j}}$ et a une durée notée $p_{i,j}$. Dans les problèmes que nous considérons, les gammes ne sont pas nécessairement linéaires, c'est-à-dire qu'une opération peut avoir plusieurs successeurs (l'ensemble $\Gamma_{i,j}$) et plusieurs prédécesseurs (l'ensemble $\Gamma_{i,j}^{-1}$). On suppose que chaque travail n'a qu'une seule opération finale notée O_{i,n_i}. Pour une opération $O_{i,j}$, sa date de début réel est notée $t_{i,j}$ et sa date de fin $C_{i,j}$. A chaque travail i est associée une date de début au plus tôt r_i et une date de fin souhaitée d_i. La date de fin d'un travail est notée C_i : $C_i = C_{i,n_i}$.

Job shop flexible Le job shop flexible est une extension du problème de job shop. Formellement, chaque opération est exécutée sur une ressource à choisir parmi plusieurs. On note $R_{i,j}$ l'ensemble des ressources pouvant exécuter l'opération $O_{i,j}$. On note la durée opératoire de l'opération $O_{i,j}$ sur la ressource R_k par $p_{i,j,k}$. On note $a(O_{i,j}) \in R_{i,j}$ la ressource qui exécute finalement $O_{i,j}$ dans la solution considérée.

Job shop multiressource Le job shop multiressource est une extension du problème de job shop. Formellement, chaque opération doit être exécutée sur plusieurs ressources. Ce problème se trouve dans la littérature sous l'appellation « *job shop with multiprocessor tasks* ». Généralement on considère que toutes les ressources sont utilisées par l'opération pendant la même période de temps. On appellera « sous-opération » l'exécution d'une opération sur une des ressources nécessaires. On note $R_{i,j}^m$ l'ensemble des ressources devant exécuter l'opération $O_{i,j}$, et $p_{i,j,k}$ la durée opératoire de la sous-opération $O_{i,j,k}$ sur la ressource R_k. On ne considère pas a priori que $p_{i,j,k} = p_{i,j} \forall R_k \in R_{i,j}^m$.

Critères d'évaluation Pour évaluer la qualité d'un ordonnancement on utilise des mesures ou critères. Les critères que nous considérons sont les suivants :

- *Makespan* : la date de fin de la dernière opération, elle est notée $C_{max} = \max_{i=1..n} C_i$.
- *La somme des dates de fin des travaux*, notée $\Sigma C = \sum_{i=1..n}(C_i)$.
- *La date de fin moyenne des travaux*, $\overline{C} = \frac{1}{n}\Sigma C$. Optimiser $\overline{C}$ est équivalent à optimiser ΣC.
- *Les en-cours*, $\overline{F} = \frac{1}{n}\sum_{i=1..n}(C_i - r_i)$.
- *Le plus grand retard*, noté $L_{max} = \max_{i=1..n}(C_i - d_i)$.
- *La somme des retards*, notée $\Sigma T_i = \sum_{i=1..n}(\max(0; C_i - d_i))$.
- *Le retard moyen*, $\overline{T} = \frac{1}{n}\Sigma T$. Optimiser $\overline{T}$ est équivalent à optimiser ΣT.
- *Le nombre de travaux en retard*, noté ΣU.

Classification des problèmes d'ordonnancement Il existe une notation pour référencer les problèmes d'ordonnancement, initialement proposée par [Graham *et al.*, 1979]. Cette notation est divisée en trois champs : $\alpha|\beta|\gamma$.

- Le champ α représente la typologie du problème : J pour un problème de job shop, FJ pour le job shop flexible, F pour un flow shop, MPT (*Multi-Processor Task*) pour des problèmes multiressources.
- Le champ β donne les contraintes du problème : d_i s'il y a des dates de fin souhaitées, r_i s'il y a des dates de début au plus tôt, Ssd pour indiquer la présence de temps

de montage dépendants de la séquence, $\tilde{d}_i$ s'il y a des dates de fin impératives pour les travaux, $\tilde{d}_{i,j}$ s'il y a des dates de fin impératives sur les opérations, *recrc* si un travail doit passer plusieurs fois sur la même ressource, $unavail_j$ s'il y a des périodes d'inactivités, etc.
- Le champ γ contient le ou les critères à optimiser : $C_{max}, L_{max}, \overline{C}, \overline{T}\ldots$ (On renvoie à [T'kindt et Billaut, 2006] pour une notation plus précise du champ γ).

1.3 Optimisation multicritère

Beaucoup de travaux de recherche en ordonnancement n'optimisent qu'un seul critère, alors que les véritables problématiques industrielles sont de nature multicritère. Ainsi, un décideur industriel peut être intéressé par minimiser les en-cours ($\overline{F}$), tout en minimisant le retard moyen de livraison au client ($\overline{T}$), sans oublier le plus grand retard (L_{max}). Ces critères sont de nature conflictuelle, c'est-à-dire qu'il n'existe pas de solution optimale pour tout les critères simultanément. Ainsi la notion d'optima de Pareto est fondamentale en optimisation multicritère.

On renvoie à [T'kindt et Billaut, 2006] pour plus de détails concernant l'optimisation multicritère en ordonnancement. Nous rappelons ci-après les notions de base utiles à la compréhension du reste du document.

1.3.1 Définition de l'optimalité

Soit $\mathcal{S}$ l'ensemble des solutions et $\mathcal{Z}$ l'image de $\mathcal{S} \subset \mathbb{R}^K$ dans l'espace des critères, avec K critères Z_i.

Définition : $x \in \mathcal{S}$ est un optimum de Pareto Faible si et seulement si $\nexists y \in \mathcal{S}$ tel que $\forall i = 1, .., K; Z_i(y) < Z_i(x)$. On note WE l'ensemble des Pareto faibles de $\mathcal{S}$.

Définition : $x \in \mathcal{S}$ est un optimum de Pareto strict si et seulement si $\nexists y \in \mathcal{S}$ tel que $\forall i = 1, .., K; Z_i(y) \leq Z_i(x)$ avec au moins une inégalité stricte. On note E l'ensemble des Pareto stricts de $\mathcal{S}$ et $E \subseteq WE$.

La figure 1.1 illustre les notions de Pareto faible et Pareto strict dans le cas de deux critères.

Par la suite nous nous intéressons uniquement à la détermination de Pareto stricts.

1.3.2 Détermination des optima de Pareto

Il existe trois grandes approches pour le décideur :

1. Le décideur souhaite que le système lui retourne une solution. Dans ce cas, le décideur donne sa préférence entre les critères, sous forme de contraintes (bornes) ou sous forme de poids associés aux critères, par exemple. C'est une approche *a priori*.

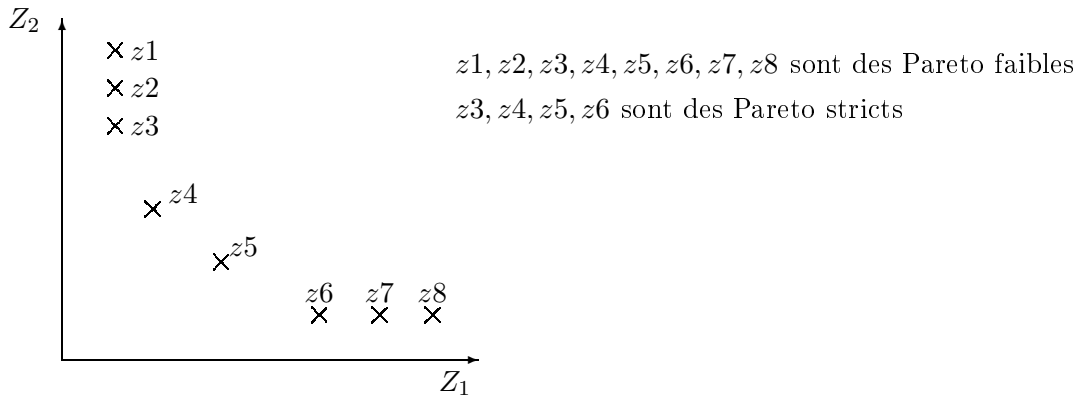

FIG. 1.1 – Illustration des notions Pareto faible et Pareto strict

2. Le décideur souhaite interagir avec le système de résolution. A chaque itération, le décideur précise la direction de recherche qu'il préfère. C'est une approche *interactive*.
3. Le décideur souhaite choisir parmi un ensemble de solutions de Pareto. C'est une approche *a posteriori*. Dans ce cas, le système de résolution doit déterminer l'ensemble des optima de Pareto.

Pour trouver des optima de Pareto, il existe différentes familles de méthodes de résolution. Cette liste n'est pas exhaustive et on renvoie à [T'kindt et Billaut, 2006] pour plus de détails.

Combinaison linéaire des critères Avec cette approche, on agrège les valeurs des différents critères pour obtenir au final un seul indicateur. Des coefficients sur chaque critère permettent de préciser la direction de la recherche. Dans ce cas le champ γ prend comme valeur : $F_\ell(Z_1, \ldots, Z_K)$.

Approche ϵ-contrainte Avec cette approche, un seul critère Z_u est optimisé, sous la contrainte que tous les autres critères sont bornés par une valeur. Dans ce cas le champ γ prend comme valeur : $\epsilon(Z_u|Z_1, \ldots, Z_{u-1}, Z_{u+1}, \ldots, Z_K)$.

Objectif à atteindre Avec cette approche le problème n'est pas de trouver un optimum de Pareto, mais de trouver une solution qui satisfait les objectifs sur chaque critère. Dans ce cas le champ γ prend comme valeur : $GP(Z_1, \ldots, Z_K)$ (*Goal Programming*).

Approche lexicographique Les critères sont optimisés les uns après les autres. Une fois que le premier critère Z_1 est optimisé, on optimise le second critère en s'interdisant de dégrader le premier, etc. Dans ce cas le champ γ prend comme valeur : $Lex(Z_1, \ldots, Z_K)$.

1.4 Notions sur la théorie de la complexité

La théorie de la complexité permet de déterminer la difficulté théorique intrinsèque d'un problème donné, ou d'un problème par rapport à un autre. Concrètement on cherche à savoir si le problème étudié est plutôt « facile » ou plutôt « difficile » à résoudre. La théorie de la complexité permet de classer les problèmes selon leur difficulté.

On étudie habituellement deux types de problèmes : *les problèmes de décision* et *les problèmes de recherche*. Les problèmes de décision sont des problèmes pour lesquels on cherche à répondre à une question par OUI ou par NON. Par exemple, « dans un graphe orienté G, existe-t-il un chemin entre le sommet a et le sommet z de longueur $\leq l$? ». Dans les problèmes de recherche, on cherche à trouver une solution à un problème. Par exemple, « dans un graphe orienté G trouver un chemin entre le sommet a et le sommet z de longueur inférieure ou égale à l ». Enfin, les problèmes d'optimisation forment un sous-groupe des problèmes de recherche. Dans ce cas, la solution doit minimiser ou maximiser une fonction objectif. Par exemple, « dans un graphe orienté G trouver le plus court chemin entre le sommet a et le sommet z ».

En ordonnancement, on rencontre à la fois des problèmes de décision et d'optimisation. Les problèmes de décision sont surtout utilisés pour résoudre le problème d'optimisation qui lui est associé, ou pour aider à déterminer sa classe de complexité.

Pour un algorithme, il existe deux types de complexité : la complexité temporelle qui renseigne sur le temps de calcul nécessaire pour obtenir une solution optimale et la complexité spatiale qui renseigne sur l'espace mémoire nécessaire. Par la suite, nous nous intéressons uniquement à la complexité temporelle.

La complexité d'un algorithme va mesurer le nombre d'instructions élémentaires nécessaires à l'algorithme pour résoudre le problème considéré. Comme pour un ordinateur donné, le temps d'exécution dépend du nombre d'instructions, on utilise le terme de *temps d'exécution*. On dit souvent que le temps d'exécution dépend de la taille de l'instance du problème. En effet, il est plus long de résoudre un problème d'ordonnancement de 10 000 opérations que le même problème composé de seulement 10 opérations. On définit la taille de l'instance d'un problème comme étant le nombre de symboles, à une constante multiplicative près, nécessaires au codage de l'instance. Un algorithme est dit *polynomial* si son temps d'exécution est borné par une fonction polynomiale de la taille de l'instance. On dit qu'il a une complexité en $O(p(n))$, où p est le polynôme et n la taille de l'instance. On dit qu'un algorithme est pseudo-polynomial si le polynôme bornant le temps d'exécution dépend de la taille de l'instance et de l'amplitude du plus grand entier. Sinon, la complexité est dite *exponentielle*.

Les problèmes appartenant à la classe $\mathcal{P}$ sont des problèmes pouvant être résolus en temps polynomial. Les problèmes polynomiaux sont considérés comme étant des problèmes « faciles ». La classe $\mathcal{NP}$ contient les problèmes pour lesquels il existe un algorithme non déterministe polynomial pour les résoudre. On dit aussi que la classe $\mathcal{NP}$ est la classe des problèmes pour lesquels on peut vérifier une réponse OUI en temps polynomial. Un algorithme non déterministe polynomial peut résoudre un problème en temps polynomial à condition de disposer d'un module divinatoire qui permet de faire le

bon choix à chaque itération. Si l'algorithme n'a pas de module divinatoire, alors il doit énumérer toutes les possibilités à chaque itération, et dans ce cas le temps d'exécution devient exponentiel. C'est pourquoi on dit qu'une réponse positive à un problème de $\mathcal{NP}$ peut être vérifiée en temps polynomial. Au jour d'aujourd'hui, on ne sait pas réaliser un module divinatoire.

La classe $\mathcal{NP}$-complet est une sous-classe de $\mathcal{NP}$. Elle est constituée des problèmes « les plus difficiles » de $\mathcal{NP}$. Pour définir la classe $\mathcal{NP}$-complet, il est nécessaire d'introduire la notion de *réduction polynomiale* d'un problème P_1 vers un problème P_2. On dit que P_1 se réduit polynomialement à P_2 si et seulement si il existe un algorithme polynomial construisant à partir des données d'une instance de P_1 les données d'une instance de P_2, de sorte que la réponse à P_1 est OUI si et seulement si la réponse à P_2 est OUI. On note $P_1 \propto P_2$ cette réduction polynomiale. Un problème est $\mathcal{NP}$-complet si tous les problèmes de $\mathcal{NP}$ se réduisent polynomialement à lui.

Les problèmes $\mathcal{NP}$-difficiles sont les problèmes de recherche pour lesquels les problèmes de décision associés sont $\mathcal{NP}$-complets. On trouvera une définition plus rigoureuse d'un problème $\mathcal{NP}$-difficile dans [Garey et Johnson, 1979].

La grande question de la théorie de la complexité est $\mathcal{P} = \mathcal{NP}$? On sait que $\mathcal{P} \subseteq \mathcal{NP}$ (si on peut trouver une solution en temps polynomial, alors on peut vérifier une réponse OUI en temps polynomial).

Les problèmes abordés dans cette thèse sont tous $\mathcal{NP}$-difficiles. Ils ne peuvent donc pas être résolus de façon optimale en temps polynomial, sauf si $\mathcal{P} = \mathcal{NP}$.

1.5 Généralités sur les méthodes de résolution

Il existe deux grandes familles de méthodes de résolution : les méthodes exactes et les méthodes approchées. Les méthodes exactes cherchent à trouver la ou les solutions optimales. Selon le type de problème considéré (notamment les problèmes NP-difficiles), le temps de calcul peut être prohibitif. On peut citer comme méthodes : la procédure par séparation et évaluation (PSE) ([Carlier et Pinson, 1989]), les algorithmes de programmation dynamique, la programmation linéaire en nombres entiers ou les algorithmes polynomiaux pour les problèmes de la classe $\mathcal{P}$. Hormis les algorithmes polynomiaux pour les problèmes polynomiaux, ces méthodes sont généralement limitées à la résolution de petites instances en raison de leur complexité algorithmique. Toutefois, les progrès réalisés récemment dans les solvers commerciaux (*CPLEX, XPRESS-MP,...*) sont tels, qu'il n'est parfois pas superflu de tester quelques modèles de programmation linéaire sur des instances de grande taille.

Les méthodes approchées trouvent de bonnes solutions (c'est-à-dire non nécessairement optimales) en un temps raisonnable. On peut citer comme méthodes les algorithmes gloutons, les méthodes d'explorations de voisinage [Kirkpatrick *et al.*, 1983], les algorithmes évolutionnaires [Holland, 1975], les colonies de fourmis [Colorni *et al.*, 1991], etc.

Dans ce manuscrit on s'intéresse principalement à deux méthodes : la recherche Tabou

et les Algorithmes Génétiques.

1.5.1 Recherche Tabou

Fred Glover a été le premier à proposer la recherche Tabou comme méthode de résolution ([Glover, 1989]). Une recherche Tabou est une méthode d'exploration de voisinage.

On part d'une solution initiale s_{init}. Cette solution devient la solution courante s et à partir de cette solution on évalue un ensemble de solutions $V(s)$ qui lui sont « proches », ces solutions sont appelées les voisins. Le meilleur voisin v_{best} devient la nouvelle solution courante s, ce changement de solution courante s'appelle le mouvement. Le processus recommence jusqu'à ce qu'une condition d'arrêt soit satisfaite. Pour éviter de boucler parmi les solutions explorées, une liste, appelée liste *tabou* $T\ell$, contient les solutions déjà explorées et on s'interdit de choisir une solution de cette liste. On peut noter que le meilleur voisin n'est pas forcément de meilleure qualité que la solution courante, ainsi la méthode permet de sortir des extrema locaux. La solution retournée est la meilleure solution s_{best} trouvée au cours de la recherche.

La génération de la solution initiale dépend du problème considéré. Dans le cas d'un problème d'ordonnancement d'atelier, on utilise le plus souvent un algorithme glouton.

La construction d'une solution voisine dépend également du problème et pour un même problème, plusieurs méthodes peuvent être envisagées. Dans le cas d'un problème de job shop, une méthode de construction de voisins simple consiste par exemple à intervertir deux opérations consécutives sur une ressource.

Une fois que la méthode de construction d'un voisin est définie, il faut savoir quels voisins seront explorés, c'est ce qu'on appelle le voisinage. Pour qu'un voisinage soit efficace il faut qu'il ait deux propriétés. La première est que la taille du voisinage ne soit pas exponentielle avec la taille de l'instance du problème. La seconde est que le voisinage soit « connexe ». On dit qu'un voisinage est connexe s'il est possible d'atteindre n'importe quelle solution en partant de n'importe quelle autre solution, et ce en un nombre fini de mouvements. On peut noter qu'une condition suffisante est de montrer qu'une solution optimale est atteignable.

L'évaluation de la qualité de tous les voisins de façon exacte peut demander un temps de calcul important. Pour améliorer la vitesse d'exécution de la recherche Tabou, on peut évaluer les voisins de façon approchée, en utilisant une borne inférieure par exemple.

On peut gérer la liste tabou de différentes façons. La plus simple est de considérer l'enregistrement de l'intégralité de la solution dans la liste tabou. Cette méthode présente l'avantage de certifier qu'on ne peut pas explorer une solution déjà visitée, le principal défaut vient du temps de calcul nécessaire pour déterminer si le voisin considéré est tabou ou non. Une autre façon est de n'enregistrer que les caractéristiques du mouvement. Si on reprend notre exemple d'une permutation d'opérations, on pourrait enregistrer seulement dans la liste tabou les deux opérations interverties et on interdirait d'intervertir à nouveau ces opérations.

Dans ce cas où la liste tabou ne contient que la description du mouvement, on limite

la taille de la liste : les éléments les plus anciens sont remplacés par les plus récents. Ainsi, après un certain nombre d'itérations on peut à nouveau faire un mouvement qui avait déjà était fait par le passé, mais a priori dans ce cas la solution courante est suffisamment différente pour ne pas boucler.

Certaines recherches Tabou utilisent un critère d'aspiration. Dans ce cas, on s'autorise à choisir un voisin tabou s'il est strictement améliorant. Le critère d'aspiration n'est utilisé que lorsque la liste tabou ne contient que la description du mouvement.

Une autre technique pour améliorer l'efficacité de la méthode est l'utilisation de points de redémarrage. Au cours de la recherche, on enregistre de bons voisins qui n'ont pas été choisis car ils n'étaient pas les meilleurs. A la fin de la recherche, on relance la méthode en partant d'un de ces points de redémarrage.

L'algorithme 1 décrit le schéma général d'une recherche Tabou.

Algorithme 1 Principe général d'une recherche Tabou

Pré-conditions: Qual(x) : Qualité (à maximiser) de la solution x

```
   /* Initialisation */
 1 Génération de la solution initiale s_init
 2 s = s_best = s_init ;
 3 Tℓ = ∅
   /* Boucle principale */
 4 Tant que Condition d'arrêt non satisfaite faire
 5     v_best = ∅
       /* Recherche du meilleur voisin non-tabou */
 6     Construire l'ensemble des voisins V(s) de la solution s
 7     Pour chaque v ∈ V(s) faire
 8         Si v ∉ Tℓ et Qual(v) > Qual(v_best) alors
 9             v_best = v
10         fin Si
11     fin Pour
12     s = v_best
13     Tℓ = Tℓ ∪ v_best /* Mise à jour de la liste tabou */
14     Si Qual(s_best) < Qual(s) alors
15         s_best = s /* Mise à jour de la meilleure solution trouvée */
16     fin Si
17 fin Tant que
18 retourner s_best
```

1.5.2 Algorithme Génétique

John H. Holland ([Holland, 1975]) à été le premier à proposer d'adapter la théorie de l'évolution de Darwin pour la résolution de problèmes. L'idée générale est que dans la nature grâce à la sélection naturelle, une population va s'améliorer au fil des générations.

L'algorithme manipule une population d'individus et chaque individu représente une

solution du problème considéré. Un individu est représenté par son chromosome. A partir du chromosome, il est possible de construire une solution au problème. La qualité d'un individu est donnée par sa *force* qui dépend de ce qu'on cherche à optimiser.

A chaque génération, on va *croiser* deux individus pour générer un ou plusieurs individus *enfants* qui auront des caractéristiques des deux parents. La probabilité pour un individu d'être choisi pour engendrer des enfants est proportionnelle à sa force. En plus du croisement, un individu enfant a une certaine probabilité (habituellement faible) de muter, c'est-à-dire que le chromosome va être modifié de façon aléatoire.

On applique un opérateur de sélection pour éliminer les individus les moins bons. Cet opérateur peut être stochastique, c'est-à-dire qu'un mauvais individu aura une plus grande probabilité d'être éliminé qu'un individu meilleur.

L'algorithme 2 donne le schéma général d'un algorithme génétique.

Algorithme 2 Principe général d'un algorithme génétique

1 Génération de la population initiale P_0
2 $t = 0$
3 **Tant que** Critère d'arrêt non satisfait **faire**
4 Génération des enfants E_t à partir des parents P_t
5 Mutation éventuelle des enfants E_t
6 Sélection des meilleurs individus de $P_t \cup E_t$ pour former P_{t+1}
7 $t = t + 1$
8 **fin Tant que**
9 **retourner** le ou les meilleurs individus de P_t

Nous conseillons la lecture de [Portmann et Vignier, 2001] où une présentation des algorithmes génétiques appliqués à l'ordonnancement est proposée.

1.6 Présentation du document

Le chapitre 2 présente des états de l'art sur les problèmes qui seront abordés ultérieurement. Tout d'abord le problème du job shop flexible, qui est celui pour lequel la littérature est certainement la plus abondante ; ensuite le problème du job shop multiressource, pour lequel les études sont moins nombreuses ; enfin le problème du job shop multicritère qui a donné lieu à très peu d'études.

Le chapitre 3 porte sur l'étude du job shop flexible multicritère (sans multiressource) et le chapitre 4 porte sur l'étude du job shop multiressource multicritère (sans problème d'affectation).

Dans le chapitre 3, après avoir rappelé deux modélisations du problème par un programme linéaire en nombres entiers, nous proposons cinq algorithmes de résolution :

- deux algorithmes de recherche Tabou - méthode connue pour son efficacité pour le problème du job shop - qui diffèrent par leur approche du problème multicritère,
- deux algorithmes génétiques - méthode connue pour son efficacité pour l'approxi-

mation du front de Pareto - qui diffèrent à la base par leur codage d'une solution,
- un algorithme mémétique, c'est-à-dire une hybridation entre un algorithme génétique et une recherche locale.

Ces méthodes de résolution sont évaluées sur des instances de la littérature adaptées au problème avec dates de fin souhaitées.

Dans le chapitre 4, après avoir proposé une modélisation du problème par un programme linéaire en nombres entiers, nous proposons deux algorithmes de résolution, inspirés des algorithmes proposés au chapitre 3 :

- un algorithme Tabou,
- un algorithme génétique.

Ces deux algorithmes plus le programme linéaire sont comparés sur des instances adaptées de celles de la littérature et sur de nouvelles instances générées aléatoirement.

Le chapitre 5 présente le logiciel DirectPlanning commercialisé par la Société Volume Software et quelques algorithmes implémentés pour tenir compte de certaines contraintes liées à la problématique industrielle (tâches verrouillées, temps de préparations dépendant de la séquence et calendriers).

Chapitre 2

État de l'art

Les problèmes d'ordonnancement d'ateliers ont été très étudiés dans la littérature depuis plus de 50 ans. Nous ne faisons pas ici un état de l'art exhaustif mais nous nous intéressons principalement aux problèmes d'ordonnancement d'ateliers de type job shop avec affectation et/ou multiressource.

2.1 Job shop flexible

Brucker et Schlie 1990

Les problèmes de job shop flexible ont été très étudiés dans la littérature. Le premier article traitant de ce type de problème est [Brucker et Schlie, 1990]. Les auteurs proposent un algorithme polynomial pour le problème de job shop avec machines à usages multiples (*multi-purpose machine*), avec comme objectif le *makespan* dans le cas où on a deux travaux.

Dans la problématique *multi-purpose machine*, chaque opération a besoin d'un outil pour pouvoir être réalisée. Une machine peut avoir plusieurs outils installés. Chaque opération peut donc être exécutée sur un ensemble de machines selon les outils installés. Dans le cas où chaque opération nécessite un outil différent, on retrouve un problème de job shop flexible tel que nous l'avons défini.

Jurisch 1992

Dans sa thèse Jurisch ([Jurisch, 1992]) propose une procédure par séparation et évaluation (PSE) pour la résolution du problème de job shop avec machines à capacités multiples. En plus de la PSE, l'auteur donne quelques heuristiques ainsi que des bornes inférieures au problème.

L'auteur s'appuie beaucoup sur la notion de *bloc*. Un *bloc* est une succession (au moins deux) d'opérations critiques sur une même ressource. Une propriété intéressante est que si on a deux solutions y et y' avec $C_{max}(y') < C_{max}(y)$, alors au moins une des propriétés suivantes est vérifiée :

- Dans y' au moins une opération d'un bloc B de y est exécutée sur une autre ressource que dans y,
- Dans y' au moins une opération d'un bloc B de y, différente de la première opération de B est exécutée avant la première opération de B,
- Dans y' au moins une opération d'un bloc B de y, différente de la dernière opération de B est exécutée après la dernière opération de B.

Brandimarte 1993

Brandimarte ([Brandimarte, 1993]) a proposé une approche hiérarchique pour résoudre le problème de job shop flexible. Dans une première étape, le problème d'affectation est résolu et on résout dans une seconde étape le problème de job shop restant. L'auteur propose plusieurs méthodes qui utilisent toutes une recherche Tabou. L'auteur considère indépendamment deux critères, le *makespan* et la somme pondérée des retards.

Dans la première approche, un algorithme à règles de priorité affecte les opérations sur les ressources. Ensuite, la Recherche Tabou résout le problème de job shop. Une fois le résultat obtenu, le processus peut recommencer avec une règle de priorité différente.

Dans la seconde approche, les deux premières étapes sont identiques. Mais au lieu de relancer un algorithme de liste, seules les opérations critiques sont susceptibles d'être réaffectées. Une liste tabou est maintenue pour éviter de revenir à une affectation déjà évaluée.

La figure 2.1 donne le schéma général de chaque approche.

L'auteur propose quatre voisinages différents. Dans tous les cas, l'auteur ne considère que des permutations entre des opérations successives sur une ressource. Dans le premier voisinage, on choisit aléatoirement un ensemble d'échanges possible d'opérations, la taille de l'ensemble étant un paramètre de la méthode. Dans le second voisinage, on choisit aléatoirement un travail. Chaque opération de ce travail est échangée avec les opérations adjacentes sur chaque ressource. Dans le troisième voisinage, une ressource est choisie aléatoirement et toutes les combinaisons de la séquence des opérations sont évaluées. Dans le quatrième voisinage, seules les opérations critiques sont considérées.

Hurink, Jurisch et Thole 1994

[Hurink *et al.*, 1994] proposent de résoudre un problème de job shop avec machines à capacités multiples. L'objectif est de minimiser le *makespan*. Les auteurs proposent deux voisinages qui sont basés sur la notion de *bloc* ([Jurisch, 1992]).

Le premier voisinage $N1$ consiste à changer l'affectation d'une opération d'un bloc B ou à déplacer une opération d'un bloc B, différente de la première (resp. la dernière), avant (resp. après) toutes les autres opérations de B. Les auteurs conjecturent que ce

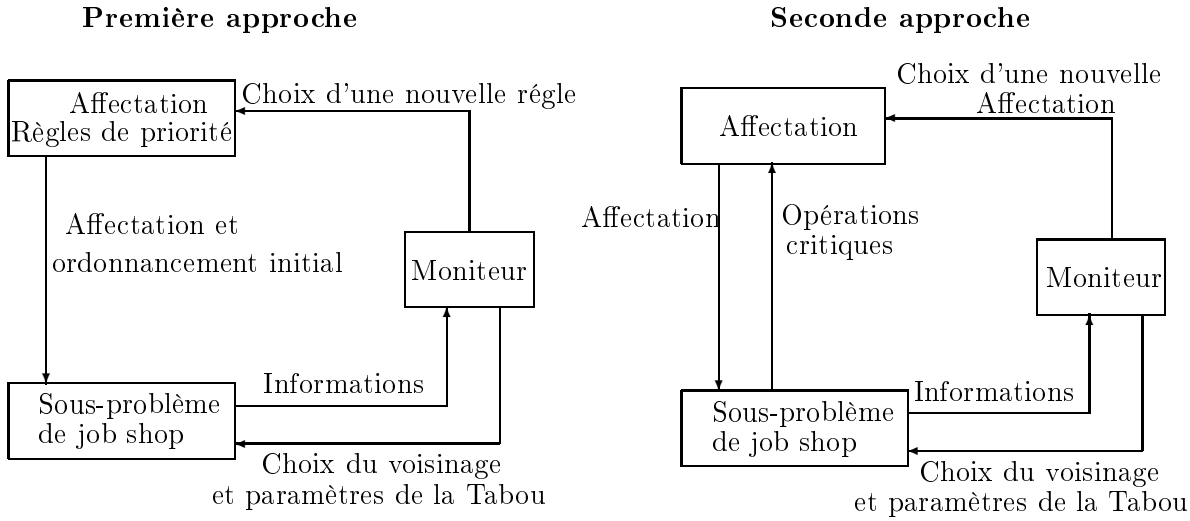

FIG. 2.1 – Les deux approches proposées par [Brandimarte, 1993]

voisinage est connexe.

Le second voisinage $N2$ consiste à changer l'affectation d'une opération j d'un bloc B ou à déplacer une opération j d'un bloc B, différente de la première (resp. la dernière), avant (resp. après) toutes les autres opérations de B. Si le déplacement de j provoque un circuit, alors j est déplacée dans le bloc B à la position la plus à gauche (resp. à droite) telle que l'ordonnancement soit faisable. Les auteurs prouvent que ce voisinage est connexe. On peut remarquer que $N1 \subseteq N2$.

Les auteurs proposent aussi une heuristique qui construit un ordonnancement. Cette méthode est basée sur les techniques d'insertion.

Paulli 1995

Paulli ([Paulli, 1995]) s'intéresse à la problématique des ateliers flexibles (*Flexible Manufacturing System* - FMS), qui sont un cas particulier du job shop flexible. L'auteur prend en compte une contrainte supplémentaire : le nombre de travaux présents simultanément dans l'atelier est limité à la valeur P. Ainsi si l'atelier est plein, un nouveau travail ne pourra commencer que lorsqu'un des travaux en cours dans l'atelier sera terminé. L'objectif est de minimiser le *makespan*.

L'auteur part des constatations suivantes :

1. Si l'affectation des opérations sur les ressources est fixée, alors le problème devient un problème de job shop avec la contrainte supplémentaire de n'avoir qu'un nombre limité de travaux simultanément en cours d'exécution,
2. Si on fixe l'ordre dans lequel les travaux entrent et sortent de l'atelier, alors la

première opération d'un travail entrant est un successeur de la dernière opération du travail sortant.

Si ces deux conditions sont réalisées, alors le problème résultant est un problème de job shop classique.

L'auteur propose une méthode hiérarchique en deux temps pour résoudre le problème :

1. Affecter les opérations aux ressources et lier les travaux entre eux.
2. Résoudre le problème de job shop résultant.

Pour améliorer la méthode, l'auteur utilise une boucle de retour entre l'étape 2 et l'étape 1 pour affiner l'affectation et les liens entre les travaux. L'algorithme 3 résume la méthode proposée par Paulli.

Algorithme 3 Algorithme proposé par [Paulli, 1995] pour le problème de FMS

```
1 Affecter les opérations aux ressources
2 Lier les travaux
3 Tant que Condition d'arrêt non vérifiée faire
4     Résoudre le problème de job shop
5     Ré-affecter et/ou re-lier les travaux selon les informations données par la résolution du job shop
6 fin Tant que
```

La première affectation (ligne 1) et les premiers liens (ligne 2) sont réalisés par des règles de priorités. Le problème de job shop (ligne 4) est résolu par une recherche Tabou. Le déplacement d'une opération sur une autre ressource, à une position faisable, assure la ré-affection. Similairement, on supprime un lien entre deux travaux et on crée un nouveau lien pour changer les liens entre les travaux (ligne 5).

Perregaard 1995

Perregaard ([Perregaard, 1995]) considère dans sa thèse les problèmes de job shop hybride et de flow shop hybride. La différence de ce type de problème avec le job shop classique est que les ressources sont rassemblées en étages de machines parallèles. Une opération pourra donc s'exécuter sur n'importe quelle ressource de l'étage considéré. C'est un cas particulier du job shop flexible.

L'auteur propose une procédure par séparation et évaluation pour résoudre le problème.

Chambers et Barnes 1996 et 1998

Chambers et Wesley Barnes ([Chambers et Barnes, 1996]) ont proposé en 1996 une recherche Tabou pour résoudre le problème de job shop flexible en minimisant le *makespan*. Les auteurs considèrent deux types de voisins, le premier consiste en un changement dans la séquence des opérations sur une ressource, le second modifie l'affectation d'une opération.

Le voisinage est défini ainsi : chaque ressource est considérée successivement et pour chacune d'elle, on identifie les suites d'opérations critiques. Pour chaque suite, la première (resp. la dernière) opération est intervertie avec l'opération précédente (resp. suivante) sur la ressource. De plus, pour chaque opération critique on essaye de la réaffecter sur toutes les ressources compatibles.

Dans [Chambers et Barnes, 1998], les auteurs ont étendu leur méthode en implémentant une gestion particulière de la liste tabou.

Dauzère-Pérès et Paulli 1997

Dauzère-Pérès et Paulli ([Dauzère-Pérès et Paulli, 1997]) proposent une modélisation et une recherche Tabou pour le problème de job shop flexible (qu'ils nomment *general multiprocessor job-shop*). Cette méthode étant importante pour la lecture de la suite de ce document, nous la présentons plus en détail.

Un nouveau modèle de graphe Les auteurs commencent par proposer une extension au graphe conjonctif-disjonctif classique, originalement proposé par Roy et Sussman ([Roy et Sussmann, 1964]). Le graphe classique est défini ainsi : $G = (V, A, E^a)$, l'ensemble des sommets est V avec un sommet par opération plus un sommet source s et un sommet puits t. A est l'ensemble des arcs conjonctifs : il existe un arc conjonctif entre deux opérations successives d'un même travail. De plus, on a des arcs conjonctifs entre s et chaque première opération d'un travail, ainsi qu'entre chaque dernière opération d'un travail et t. E_k^a est l'ensemble des arcs disjonctifs entre chaque paire d'opérations affectées à la ressource R_k. On pose $E^a = \bigcup_{R_k \in \mathcal{R}} E_k^a$. La longueur d'un arc est égale à la durée opératoire de l'opération correspondant au sommet de départ de l'arc. On obtient une *sélection* S_k en fixant le sens de chaque arc dans E_k^a. On définit une *sélection complète* S telle que $S = \bigcup_{R_k \in \mathcal{R}} S_k$. On note $f(x)$ (resp. $p(x)$) l'opération successeur (resp. prédécesseur) de x dans le travail.

Si nous considérons la version étendue où l'affectation n'est pas encore connue, il existe un arc disjonctif dans E_k^a entre chaque paire d'opérations pour lesquelles elles peuvent s'exécuter sur la même ressource. De plus, comme les durées opératoires d'une opération dépendent de la ressource, les arcs disjonctifs entre deux mêmes opérations peuvent avoir des longueurs différentes.

Trouver une solution faisable pour le problème de job shop flexible signifie déterminer une sélection complète de la façon suivante :

Étape 1 Pour chaque opération x, fixer le sens d'un ou deux arcs disjonctifs de E^a attachés à x, retirer tous les autres. S doit être tel que :

1. Au plus un arc conjonctif se termine sur une opération et au plus un arc conjonctif part d'une opération.
2. Si une opération a un arc entrant et un arc sortant, alors les deux arcs sont dans le même ensemble S_k.
3. Dans chaque ensemble $S_k \neq \emptyset$, il doit exister une et seulement une opération sans arc entrant et une et seulement une opération sans arc sortant.

Étape 2 Pour chaque ensemble $S_k \neq \emptyset$, $S_k = S_k \cup \{(s,i);(j,t)\}$, où les opérations i et j sont respectivement la première et la dernière opération ordonnancées sur la ressource R_k.

Si la sélection S définit un graphe sans circuit, alors la solution au problème de job shop flexible est réalisable.

La recherche Tabou proposée On obtient un voisin par le déplacement d'une opération. Cette technique permet à la fois la modification de l'affectation et la modification de la séquence. Dans le graphe conjonctif, déplacer une opération x entre les opérations y et z signifie :

- effacer l'arc (v,x) et (x,w), où v (respectivement w) est le prédécesseur (respectivement le successeur) de x sur la ressource qui l'exécute,
- ajouter un arc (v,w),
- effacer l'arc (y,z),
- ajouter les arcs (y,x) et (x,z).

On note ce mouvement par le triplet $\{x,y,z\}$.

Les auteurs ont démontré qu'aucun circuit n'est créé en déplaçant une opération x, si x est ordonnancée entre les opérations y ($y \neq f(x)$) et z ($z \neq p(x)$), et si :

$$r_y < r_{f(x)} + p_{f(x),a(f(x))}$$
$$\text{et} \quad r_z + p_{z,a(z)} > r_{p(x)}$$

Le voisinage proposé par les auteurs consiste à explorer tous les triplets $\{x,y,z\}$ qui satisfont la condition précédente. Les voisins sont évalués grâce à une borne inférieure, de plus les voisins sont classés en cinq catégories selon leur « qualité » espérée.

Les auteurs se sont intéressés à trois façons différentes de gérer la liste tabou. Soit x l'opération déplacée entre les opérations y et z ; v et w sont les opérations ordonnancées avant et après x sur la ressource sur laquelle x est actuellement affectée, on note cette ressource $a(x)$.

1. On ajoute (v,w) à la liste tabou. Un déplacement (x',y',z') est tabou si (y',z') appartient à la liste.
2. On ajoute $(x,a(x))$ à la liste tabou. Un déplacement (x',y',z') est tabou si $(x',a(y'))$ appartient à la liste.
3. On ajoute (x,y) à la liste tabou. Un déplacement (x',y',z') est tabou si (x',y') ou (z',x') appartient à la liste.

Lors de leurs expérimentations, les auteurs ont constaté que la troisième façon de gérer la liste tabou donnait les meilleurs résultats.

Brucker, Jurisch et Krämer 1997

Brucker, Jurisch et Krämer ([Brucker *et al.*, 1997]) ont étudié la complexité de plusieurs problèmes d'ordonnancement où il est question d'affectation. La première partie est consacrée aux problèmes à machines parallèles. La seconde partie s'intéresse aux problématiques d'atelier : flow shop, job shop et open shop.

Mesghouni 1999

Mesghouni ([Mesghouni, 1999]) a étudié lors de sa thèse le problème de job shop flexible. Il a proposé deux approche évolutionnistes pour le résoudre.

Mastrolilli et Gambardella 2000

Mastrolilli et Gambardella ([Mastrolilli et Gambardella, 2000]) ont proposé deux structures de voisinage ($Nopt1$ et $Nopt2$) pour le problème de job shop flexible. Leurs voisinages sont basés sur le déplacement d'une opération dans le graphe disjonctif. Les auteurs ont démontré que si une solution faisable S n'a pas de voisin selon le premier voisinage, i.e. $Nopt1(S) = \emptyset$, alors S est un ordonnancement optimal pour le *makespan*. $Nopt2$ est une extension de $Nopt1$ où $Nopt2$ conserve la propriété d'optimalité en cas d'absence de voisin. Les auteurs ont démontré la connexité de $Nopt2$. D'après leurs expérimentations, malgré l'absence de la connexité du voisinage $Nopt1$, ce dernier donne de meilleurs résultats que $Nopt2$ grâce à sa plus grande vitesse d'exécution.

Kacem, Hammadi et Borne 2002

Kacem, Hammadi et Borne ([Kacem *et al.*, 2002]) ont proposé un algorithme génétique multicritère pour le problème de job shop flexible, utilisant de la logique floue pour l'évaluation des individus. Les critères qu'ils considèrent sont le *makespan*, la charge maximale des ressources et la somme des charges des ressources.

Kis 2003

Kis ([Kis, 2003]) étudie un problème d'atelier où chaque travail possède une gamme complexe : certaines séquences d'opérations doivent être exécutées en parallèles, d'autres séquences peuvent s'exécuter sur des ressources différentes. Au final chaque travail est décrit par un graphe qui contient des chemins *'et'* et des chemins *'ou'*. Ce problème est plus général que le job shop flexible. L'auteur propose deux méthodes pour résoudre le problème : une recherche Tabou et un algorithme génétique. L'objectif est la minimisation du *makespan*.

Le voisinage de la recherche Tabou est basé sur des techniques d'insertion de séquences d'opérations dans un ordonnancement partiel. L'algorithme génétique utilise uniquement un opérateur de croisement. Cet opérateur considère deux individus : un maître et un apprenti. L'idée est d'utiliser un sous-ensemble du maître pour améliorer l'apprenti et ainsi générer un nouvel individu enfant. Les expérimentations montrent la supériorité de la recherche Tabou sur l'algorithme génétique.

Mati et Xie 2004

Mati et Xie ([Mati et Xie, 2004]) ont étudié la complexité des problèmes de job shop flexible où seulement deux travaux sont considérés.

Le premier cas étudié est celui du problème de job shop à deux travaux avec des machines à capacités multiples non reliées. Les auteurs ont montré que ce problème est NP-Difficile pour les critères du *makespan* et pour la somme des dates de fin des travaux. Dans le cas où la préemption est autorisée, le problème est NP-Difficile pour les critères $L_{max}, \Sigma T, \Sigma wT, \Sigma wC, \Sigma U$ et ΣwU.

Le second problème considéré est le problème où un travail est flexible et le second n'est pas flexible (i.e. une seule ressource par opération). Dans ce cas les auteurs ont montré que le problème est polynomial pour la minimisation du *makespan*. Ils ont proposé un algorithme polynomial basé sur une approche géométrique.

Dupas 2004

Dupas dans son mémoire de HDR ([Dupas, 2004]) considère les problèmes d'ordonnancement d'ateliers de nature cyclique et flexible. Son étude s'intéresse à la résolution de ces problèmes via des algorithmes évolutionnistes. En ce qui concerne le cas du job shop flexible, l'auteur reprend l'algorithme proposé par [Mesghouni, 1999] avec une approche multicritère basée sur NSGA-II ([Deb *et al.*, 2002]).

Nait Tahar, Yalaoui, Amodeo et Chu 2004

Nait Tahar, Yalaoui, Amodeo et Chu ([Tahar *et al.*, 2004]) se sont intéressés à un problème d'ordonnancement dans un atelier de fabrication de cartons. Dans leur problématique, certaines ressources sont en plusieurs exemplaires (machines parallèles identiques). De plus, ils considèrent des temps de montage dépendants de la séquence entre les opérations. Le critère minimisé est la somme des retards absolus. Leur problème est un job shop hybride. Les auteurs proposent un algorithme génétique.

Un individu est représenté sous la forme d'une matrice de m lignes et de n colonnes, ici m est le nombre de ressources et n est le nombre maximal d'opérations sur une ressource. Sur chaque ligne on a la séquence des opérations sur la ressource. Ce codage est proche de celui que nous proposons dans la section 3.4.2. Le croisement proposé est une adaptation du croisement à un point.

Alvarez-Valdes, Fuertes, Tamarit, Giménez et Ramos 2005

Alvarez-Valdes, Fuertes, Tamarit, Giménez et Ramos ([Alvarez-Valdes *et al.*, 2005]) ont étudié un problème d'ordonnancement pour l'industrie du verre. Le problème est à la base un job shop flexible, mais il comporte un certain nombre de particularités.

Les travaux peuvent aboutir soit à des produits finis soit à des produits semi-finis. Ainsi il peut exister des contraintes de précédence entre certains travaux, un produit fini peut nécessiter un ou plusieurs produits semi-finis. De plus, comme un travail peut fournir plusieurs unités de produits semi-finis, un produit fini peut ne nécessiter qu'une partie d'un travail produisant des produits semi-finis. Un travail de produits finis peut avoir une date de fin souhaitée, une date de fin au plus tôt et une date de fin au plus

tard. Les travaux possèdent un poids reflétant l'importance du respect de leur date de fin souhaitée.

Les ressources peuvent être aussi bien des machines que des équipes de travailleurs. Chaque ressource peut avoir des périodes d'inactivités selon un calendrier qui lui est propre. Certaines ressources ne peuvent exécuter qu'une opération à la fois, d'autres peuvent en traiter plusieurs simultanément. Dans le cas d'une équipe, le nombre de travailleurs peut varier selon la période du jour (équipe réduite la nuit par exemple), dans ce cas on considère que la ressource a une vitesse variable selon la date.

Les travaux sont composés de plusieurs opérations. Chaque opération peut s'exécuter sur plusieurs ressources. Certaines opérations d'un travail peuvent se chevaucher selon un coefficient. Ce coefficient représente la proportion qu'une opération doit être traitée avant que l'opération suivante puisse commencer.

Dans le cas où un travail i nécessite un autre travail j (cas d'un produit semi-fini), la première opération de i peut éventuellement commencer avant la fin de la dernière opération de j, cela modélise un chevauchement entre travaux.

Certaines séquences d'opérations doivent s'exécuter sans délai entre elles. C'est le cas lorsque les opérations manipulent du verre en fusion.

La fonction objectif est une fonction linéaire par morceaux autour de la date de fin souhaitée.

Les auteurs proposent une méthode de résolution en deux étapes : le première étape construit une solution réalisable, la seconde étape améliore cette solution grâce à une recherche locale.

Zhang et Gen 2005

Zhang et Gen ([Zhang et Gen, 2005]) ont proposé un algorithme génétique pour le problème de job shop flexible multicritère. Ils considèrent les même critères que [Kacem *et al.*, 2002] c'est-à-dire le *makespan*, la charge maximale et la somme des charges des ressources. D'après les expérimentations présentées par les auteurs, cet algorithme est meilleur que celui de [Kacem *et al.*, 2002].

Ho, Tay et Lai 2007

Ho, Tay et Lai ([Ho *et al.*, 2007]) ont proposé un algorithme génétique pour le job shop flexible. L'originalité de leur approche vient de l'ajout d'un module d'apprentissage. Au fil des générations, ce module apprend à reconnaître certaines bonnes caractéristiques des chromosomes.

On représente un individu par un chromosome composé de deux parties : la première partie représente l'ordre des opérations entre elles, la seconde partie est l'affectation des opérations sur les ressources. Ce codage n'étant pas direct, les auteurs ont proposé un algorithme pour générer un ordonnancement actif à partir d'un chromosome. La méthode de croisement est du type « à deux points » pour chacune des deux parties du chromosome.

Le module d'apprentissage contient m chromosomes a priori de bonne qualité. On compare les n meilleurs individu de la population courante avec ceux contenus dans la mémoire du module d'apprentissage, l'objectif est de trouver les k plus proches individus de la mémoire. Ces k individus remplacent les k plus mauvais de la population courante. La mise à jour du module intervient toutes les q itérations de l'algorithme génétique.

2.2 Job shop multiressource

Dauzère, Roux et Lasserre 1998

Dauzère-Pérès, Roux et Lasserre ([Dauzère-Pérès *et al.*, 1998]) ont étendu le modèle et la recherche tabou de [Dauzère-Pérès et Paulli, 1997]. Dans leur article, en plus de la flexibilité, ils considèrent qu'une opération peut nécessiter plusieurs ressources simultanément (*multiressource*) et une opération peut avoir plus d'un successeur et plus d'un prédécesseur.

Les auteurs ont démontré que le voisinage de la recherche Tabou proposée est connexe. L'évaluation d'un voisin est réalisée grâce à une borne inférieure.

Chen et Lee 1999

Chen et Lee ([Chen et Lee, 1999]) ont étudié un problème d'atelier dans lequel une opération nécessite un ensemble de ressources. Cet ensemble de ressources doit être choisi parmi un ensemble d'ensembles de ressources qui dépend de l'opération, les sous-ensembles n'étant pas forcément disjoints. Par exemple, une opération peut être exécutée soit sur $\{R_1, R_2, R_3\}$, $\{R_1, R_2\}$, $\{R_1, R_3\}$, $\{R_2, R_3\}$ ou $\{R_2\}$. La durée de l'opération dépend de l'ensemble des ressources choisi. L'opération doit être exécutée simultanément sur toutes les ressources de l'ensemble considéré.

Les auteurs proposent une approche en deux temps : affectation puis ordonnancement. Ils proposent un algorithme pseudo-polynomial pour résoudre optimalement l'affectation dans le cas où il n'y a que deux ressources et dans un cas particulier du problème à trois ressources. Une heuristique à garantie de performance est proposée pour la phase d'ordonnancement.

Cheng, Wang et Sriskandarajah 1999

Cheng, Wang et Sriskandarajah ([Cheng *et al.*, 1999]) ont étudié le problème de flow shop à deux machines dans lequel un opérateur est chargé de préparer et de finaliser (démontage) les travaux sur les machines. Deux cas sont considérés : séparable et non séparable. Dans le cas séparable, la phase de montage peut commencer avant que l'opération soit disponible sur la ressource. De même on considère que l'opération n'est plus présente sur la ressource lors du démontage. Dans le cas non séparable, l'opération doit être sur la ressource pour que le montage puisse commencer, idem pour le démontage. Les

auteurs ne considèrent que le cas de mouvements cycliques, c'est-à-dire que l'opérateur doit se déplacer entre les deux machines selon un certain cycle.

Les auteurs ont démontré que le problème est NP-complet au sens fort aussi bien dans le cas séparable que non séparable. Ils ont aussi proposé des heuristiques pour résoudre le problème.

Shachnai et Turek 1999

Shachnai et Turek ([Shachnai et Turek, 1999]) ont étudié un problème d'ordonnancement d'opérations au sein d'un système multi-ressource, plus spécifiquement un système informatique. Dans ce problème, une opération nécessite plusieurs ressources simultanément, ainsi qu'une certaine quantité de chaque ressource. L'objectif est de minimiser les temps de réponses. Les auteurs proposent une heuristique pour résoudre le problème.

Artigues et Roubellat 2000

Artigues et Roubellat ([Artigues et Roubellat, 2000]) ont étudié le problème de gestion de projet à contraintes de ressource (RCPSP) dans le cas multi-mode. Dans ce problème, il faut ordonnancer des travaux composés d'opérations sur des ressources qui peuvent être cumulatives. Les auteurs proposent une extension du modèle de graphe pour prendre en compte les ressources cumulatives.

Les auteurs proposent un algorithme pour insérer des travaux dans un ordonnancement existant, avec comme objectif de minimiser l'impact sur le plus grand retard.

Dauzère-Pérès et Pavageau 2001

Dauzère-Pérès et Pavageau ([Dauzère-Pérès et Pavageau, 2001]) ont proposé une modélisation d'un problème de job shop flexible et multiressource qui est une extension du modèle de [Dauzère-Pérès *et al.*, 1998]. Dans ce nouveau modèle, une opération peut nécessiter plusieurs ressources, mais la durée opératoire peut être différente selon la ressource. Le modèle est restreint à deux cas : soit les dates de début de chaque sous-opération sont identiques, soit ce sont les dates de fin qui sont identiques.

Un autre apport de ce papier est l'extension de la *condition de Dauzère* ([Dauzère-Pérès et Paulli, 1997]) au cas multiressource.

Oğuz, Ercan, Cheng et Fung 2003

Oğuz, Fikret Ercan, Cheng et Fung ([Oğuz *et al.*, 2003]) ont étudié le problème de flow shop hybride à deux étages avec des opérations multi-ressources. Dans ce problème, un travail doit passer en premier sur l'étage 1 et en second sur l'étage 2. La particularité de ce problème est qu'une opération peut nécessiter simultanément plusieurs ressources d'un même étage. Les auteurs proposent un algorithme basé sur des règles de priorités avec comme objectif la minimisation du *makespan*.

Şerifoğlu et Ulusoy 2004

Sivrikaya Şerifoğlu et Ulusoy ([Serifoğlu et Ulusoy, 2004]) ont proposé un algorithme génétique pour le problème de flow shop hybride multi-étage avec des opérations multi-ressource. C'est une généralisation du problème étudié par [Oğuz *et al.*, 2003].

2.3 Job shop multicritère

Huckert, Rhode et Weber 1980

Huckert, Rhode, Roglin et Weber ([Huckert *et al.*, 1980]) ont proposé une méthode interactive pour résoudre le problème de job shop multicritère, avec dates de fin souhaitées. Les critères considérés sont : le *makespan*, la date de fin moyenne des travaux, le temps d'inactivité des ressources, le plus grand retard et le nombre de travaux en retards.

Les auteurs proposent un algorithme inspiré de la méthode STEM ([Benayoun *et al.*, 1971]). Cette méthode est en deux étapes. Dans la première étape, le décideur donne un objectif à atteindre, i.e. une solution idéale. La seconde étape utilise un algorithme glouton pour trouver une solution selon cet objectif. Une solution est proposée et le décideur peut soit arrêter, soit préciser un nouvel objectif.

Deckro, Hebert et Winkofsky 1982

Deckro, Hebert et Winkofsky ([Deckro *et al.*, 1982]) ont étudié le problème de job shop avec dates de fin souhaitées. Ils proposent un programme linéaire en nombres entiers pour le résoudre. Ils considèrent les critères du *makespan*, la date de fin moyenne des travaux et la somme des avances et des retards. L'approche que les auteurs proposent est basée sur la programmation par objectifs.

Esquivel, Ferrero, Gallar, Salto, Alfonso et Schütz 2002

Esquivle, Ferrero, Gallar, Salto, Alfonso et Schütz ([Esquivel *et al.*, 2002]) ont appliqué plusieurs algorithmes génétiques pour la résolution du problème de job shop. Ils ont étudié le problème sous l'angle monocritère et multicritère. Leurs algorithmes utilisent des principes pour améliorer l'efficacité de la méthode. Le premier principe est celui de générer plusieurs enfants à partir des deux mêmes parents. Le second principe est d'empêcher le croisement de deux individus de la « même famille », c'est-à-dire deux individus qui ont un ancêtre commun trop proche au niveau du nombre de générations. Pour la résolution du problème multicritère, les auteurs considèrent deux approches : la combinaison linéaire des critères et la détermination du front de Pareto.

Pour la combinaison linéaire, les auteurs optimisent les critères du *makespan*, de l'avance globale et la somme pondérée des fins des travaux. Ils proposent de manipuler trois populations indépendantes. Chaque population optimise un seul critère. Une fois que les trois populations ont convergé, elles fusionnent pour ne former plus qu'une

seule population. Cette unique population optimise la combinaison linéaire des critères. A l'issue de cette étape, il est possible de découper la population en trois en classant les individus selon leur qualité respective pour chacun des trois critères. L'algorithme 4 décrit cette méthode.

Algorithme 4 Algorithme génétique multicritère proposé par [Esquivel *et al.*, 2002]

Initialiser trois populations de taille S (une par objectif)
Répéter
    Faire évoluer chaque population indépendamment jusqu'à ce que la condition d'arrêt θ_1 soit vérifiée
    Fusionner les trois populations en une seule de taille $3 \times S$
    Faire évoluer la population unifiée jusqu'à ce que la condition d'arrêt θ_1 soit vérifiée
    Si La Condition d'arrêt θ_2 n'est pas vérifiée **alors**
        Créer une population de taille S avec les meilleurs individus selon le premier critère
        Créer une population de taille S avec les meilleurs individus selon le second critère
        Créer une population de taille S avec les meilleurs individus selon le troisième critère
    fin Si
jusqu'à La Condition d'arrêt θ_2 est vérifiée

Pour la recherche du front de Pareto, les auteurs considèrent les critères du *makespan* et l'avance-retard moyen des travaux autour d'une date de fin souhaitée commune. L'algorithme 5 décrit comment est gérée la population à une génération donnée.

Algorithme 5 Gestion de la population dans [Esquivel *et al.*, 2002]

Pré-conditions: n_1 : nombre de croisements
Générer l'ensemble des enfants O ($|O| = 2 \times n_1$)
Soit O_{nond} l'ensemble des enfants non-dominés par le front de Pareto courant.
Si $O_{nond} \neq \emptyset$ **alors**
    Insérer O_{nond} dans la population
Sinon
    Insérer n_1 meilleurs individus de O dans la population
fin Si

Suresh et Mohanasundaram 2006

Suresh et Mohanasundaram ([Suresh et Mohanasundaram, 2006]) ont proposé de déterminer une approximation du front de Pareto pour le problème de job shop multicritère grâce à un recuit simulé. Ils considèrent les critères du *makespan* et la date de fin moyenne des travaux.

L'algorithme proposé optimise une combinaison linéaire des critères, les coefficients

sont notés w_1 et w_2. Lors du premier appel, $w_1 = 1$ et $w_2 = 0$, c'est-à-dire qu'un seul critère est optimisé. La solution initiale est générée aléatoirement. A l'issue du premier appel, la meilleure solution sert de solution initiale pour une nouvelle recherche où les coefficients sont modifiés ($w_1 = w_1 - 0.2$; $w_2 = 1 - w_1$). Dans leur approche, une archive externe sert pour mémoriser les solutions non-dominées trouvées lors du déroulement de l'algorithme.

Chapitre 3

Job shop flexible multicritère

Dans ce chapitre nous nous intéressons à des problèmes d'ordonnancement d'atelier dans lesquels des ressources peuvent être équivalentes entre elles. Nous résolvons ces problèmes grâce à plusieurs méthodes : deux recherches Tabou, deux algorithmes génétiques et un algorithme « mémétique ».

3.1 Définition formelle du problème

Le problème que nous considérons est le job shop flexible (FJSP - *Flexible job shop Scheduling Problem*) qui est une généralisation du problème de job shop classique. Nous rappelons ici les notations.

On considère un ensemble J de n travaux. Ces travaux doivent être ordonnancés sur m ressources disjonctives, l'ensemble des ressources est noté $\mathcal{R}$. On note R_k la $k^{ème}$ ressource. Chaque travail i est composé de n_i opérations liées entres elles par des contraintes de gamme. L'opération j du travail i est notée $O_{i,j}$. Les gammes ne sont pas nécessairement linéaires, c'est-à-dire qu'une opération peut avoir plusieurs successeurs (l'ensemble $\Gamma_{i,j}$) et plusieurs prédécesseurs (l'ensemble $\Gamma_{i,j}^{-1}$). On suppose que chaque travail n'a qu'une seule opération finale notée O_{i,n_i}. Chaque opération peut être exécutée sur plus d'une ressource. On note $R_{i,j}$ l'ensemble des ressources pouvant exécuter l'opération $O_{i,j}$. On note la durée opératoire de l'opération $O_{i,j}$ sur la ressource R_k par $p_{i,j,k}$. On note $a(O_{i,j})$ la ressource qui exécute $O_{i,j}$ dans la solution considérée. A chaque travail i est associée une date due d_i. Il est possible de propager cette date due à toutes les opérations du travail i (algorithme 6).

Algorithme 6 Propagation des dates dues

$\mathcal{O} = \emptyset$
Pour $i = 1$ à n **faire**
    Pour $j = 1$ à $n_i - 1$ **faire**
        $d_{i,j} = \infty$
        $nbSucc_{i,j} = |\Gamma_{i,j}|$
    fin Pour
    $d_{i,n_i} = d_i$
    Ajouter O_{i,n_i} à $\mathcal{O}$
fin Pour
Tant que $\mathcal{O} \neq \emptyset$ **faire**
    Retirer une opération $O_{i,j}$ de $\mathcal{O}$
    $p_{max} = \max_{R_k \in R_{i,j}} p_{i,j,k}$ /* *L'affectation de l'opération n'étant pas connue, on prend sa plus grande durée possible.* */
    Pour chaque $O_{k,l} \in \Gamma_{i,j}^{-1}$ **faire**
        $d_{k,l} = \min(d_{k,l}, d_{i,j} - p_{max})$
        $nbSucc_{k,l} = nbSucc_{k,l} - 1$
        Si $nbSucc_{k,l} = 0$ **alors**
            Ajouter $O_{k,l}$ à $\mathcal{O}$
        fin Si
    fin Pour
fin Tant que

3.2 Modèles de PLNE

H.M. Wagner ([Wagner, 1959]) a proposé une modélisation du problème de job-shop flexible sous la forme d'un programme linéaire en nombres entiers. Ce modèle ne prend en compte que des gammes linéaires. Les variables du modèle sont les suivantes :

$$x_{i,j,k}^{(l)} = \begin{cases} 1 & \text{si } O_{i,j} \text{ est la } l^{ème} \text{ opération sur } R_k \\ 0 & \text{sinon} \end{cases} \quad (3.1)$$

$t_k^{(l)}$: est la date de début de la $l^{ème}$ opération sur R_k (3.2)

$s_k^{(l)}$: est la durée entre la fin de la $l^{ème}$ opération et le début de la $(l+1)^{ème}$ sur R_k (3.3)

s_k^0 : est la durée entre 0 et la première opération sur R_k (3.4)

m_k : est le nombre maximum d'opérations pouvant être exécutées sur R_k (3.5)

$Tx_k^{(l)}$: est le temps opératoire de la $l^{ème}$ opération sur R_k (3.6)

Dans ce modèle, une opération $O_{i,j}$ peut être exécutée par les ressources $R_{k1}, R_{k2}, \ldots, R_{kq}$. Les contraintes 3.7 expriment qu'une opération $O_{i,j}$ n'est affectée qu'à une seule ressource

et qu'à une seule position.

$$\sum_{l=1}^{l=m_{k1}} x_{i,j,k1}^{(l)} + \sum_{l=1}^{l=m_{k2}} x_{i,j,k2}^{(l)} + \ldots + \sum_{l=1}^{l=m_{kq}} x_{i,j,kq}^{(l)} = 1 \quad \forall i = 1,\ldots,n; \forall j = 1,\ldots,n_i \tag{3.7}$$

Les contraintes 3.8 expriment qu'à une position donnée sur R_k, au plus une opération est présente.

$$\sum_{O_{i,j}|R_k \in R_{i,j}} x_{i,j,k}^{(l)} \leq 1 \quad \forall k = 1,\ldots,m; \forall l = 1,\ldots,m_k \tag{3.8}$$

Les contraintes 3.9 déterminent le temps opératoire de la $l^{ème}$ opération exécutée sur R_k.

$$Tx_k^{(l)} = \sum_{O_{i,j}|R_k \in R_{i,j}} p_{i,j,k} \times x_{i,j,k}^{(l)} \quad \forall k = 1,\ldots,m; \forall l = 1,\ldots,m_k \tag{3.9}$$

Les contraintes 3.10 expriment les contraintes de gamme opératoire pour deux opération $O_{i,j}$ et $O_{i,j+1}$ sur les ressources R_{k_1} et R_{k_2}, respectivement.

$$t_{k1}^{(l1)} + p_{i,j,k1} \times x_{i,j,k1}^{(l1)} \leq t_{k2}^{(l2)} + HV \times (2 - x_{i,j,k1}^{(l1)} - x_{i,j+1,k2}^{(l2)}) \quad \forall l1 = 1,\ldots,m_{k1}; \forall l2 = 1,\ldots,m_{k2} \tag{3.10}$$

Le calcul des dates de début au plus tôt est obtenu par les contraintes 3.11 et 3.12.

$$t_k^{(1)} = s_k^0 \qquad \forall k = 1,\ldots,m \tag{3.11}$$

$$t_k^{(l)} = s_k^0 + \sum_{r=1}^{r=l-1} (Ts_k^{(r)} + s_k^{(r)}) \quad \forall k = 1,\ldots,m; \forall l = 2,\ldots,m_k \tag{3.12}$$

Le C_{max} est calculé par le dernier jeu de contraintes.

$$t_k^{m_k} + T_{x_k}^{m_k} \leq C_{max} \quad \forall k = 1,\ldots,m \tag{3.13}$$

Le modèle comporte $\#ope(m+3)+m$ variables et $\#ope^2 + 4\#ope + m$ contraintes, avec $\#ope = \sum_{i=1}^{n} n_i$ le nombre total d'opérations.

B. Penz ([Penz, 1994]) a proposé une autre modélisation (simplifiée) du problème.

$$x_{i,j,k} = \begin{cases} 1 & \text{si } O_{i,j} \text{ est affectée sur } R_k \\ 0 & \text{sinon} \end{cases} \tag{3.14}$$

$$y_{c,d}^{a,b} = \begin{cases} 1 & \text{si } O_{a,b} \text{ est effectuée avant } O_{c,d} \\ 0 & \text{sinon} \end{cases} \tag{3.15}$$

Les contraintes 3.16 forcent qu'une opération soit affectée à une seule machine.

$$\sum_{R_k \in R_{i,j}} x_{i,j,k} = 1 \quad \forall i = 1, \ldots, n; \forall j = 1, \ldots, n_i \tag{3.16}$$

Les contraintes de gammes et de date de fin sont représentées par les équations 3.17 et 3.18.

$$t_{i,j+1} \geq t_{i,j} + \sum_{R_k \in R_{i,j}} x_{i,j,k} \times p_{i,j,k} \quad \forall i = 1, \ldots, n; \forall j = 1, \ldots, n_i - 1 \tag{3.17}$$

$$C_{max} \geq t_{i,n_i} + \sum_{R_k \in R_{i,n_i}} x_{i,n_i,k} \times p_{i,n_i,k} \qquad \forall i = 1, \ldots, n \tag{3.18}$$

Les équations 3.19 à 3.22 permettent d'arbitrer les disjonctions entre les opérations sur une même ressource. Si pour les opérations $O_{a,b}$ et $O_{c,d}$ l'intersection $R_{a,b} \cap R_{c,d}$ n'est pas vide, alors les contraintes s'écrivent pour les ressources $R_k \in R_{a,b} \cap R_{c,d}$:

$$y_{c,d}^{a,b} + y_{a,b}^{c,d} = 1 \tag{3.19}$$

$$\begin{aligned} t_{c,d} \quad \geq \quad & t_{a,b} + y_{c,d}^{a,b} \times p_{a,b,k} - HV \times y_{a,b}^{c,d} \\ & -HV \times \sum_{R_r \in R_{a,b} \text{ et } R_r \neq R_k} x_{a,b,r} \\ & -HV \times \sum_{R_r \in R_{c,d} \text{ et } R_r \neq R_k} x_{c,d,r} \end{aligned} \tag{3.20}$$

$$\begin{aligned} t_{a,b} \quad \geq \quad & t_{c,d} + y_{a,b}^{c,d} \times p_{c,d,k} - HV \times y_{c,d}^{a,b} \\ & -HV \times \sum_{R_r \in R_{a,b} \text{ et } R_r \neq R_k} x_{a,b,r} \\ & -HV \times \sum_{R_r \in R_{c,d} \text{ et } R_r \neq R_k} x_{c,d,r} \end{aligned} \tag{3.21}$$

Ce modèle comporte $\#ope(\#ope + m)$ variables et $\mathcal{O}(\#ope \times m)$ contraintes.

Ces modèles permettent de formaliser le problème. Toutefois, pour des instances de taille moyenne (d'après nos expérimentations : supérieure à 6 ressources et 6 travaux) le nombre de variables et de contraintes est déjà trop important pour envisager une résolution exacte par un solver commercial comme CPLEX ou XPRESS-MP.

3.3 Approche Tabou pour le job shop flexible

Les algorithmes de recherche Tabou (cf. section 1.5.1) ont démontré leur efficacité pour résoudre des problèmes d'ordonnancement et plus particulièrement les problèmes de job-shop ([Nowicki et Smutnicki, 1993], [Dauzère-Pérès et Paulli, 1997]).

Dans le cadre d'une approche multicritère, on peut appliquer différentes techniques pour la détermination d'optima de Pareto (voir la section 1.3). Nous nous sommes intéressés à deux méthodes pour la détermination de solutions non-dominées : une combinaison linéaire des critères (notée par la suite $TS\ell c$) et une approche epsilon-contrainte (notée par la suite $TS\epsilon$). Notons que l'utilisation d'une recherche Tabou pour aborder une approche epsilon-contrainte est originale. A notre connaissance, cette méthode n'avait pas été utilisée auparavant pour ce genre d'approche. Ces deux versions de la recherche Tabou (inspirées par [Dauzère-Pérès et Paulli, 1997]) partagent des points communs.

Lors d'études préliminaires, nous avons développé d'autres versions de la recherche Tabou utilisant un voisinage différent ([Vilcot *et al.*, 2006b] et [Vilcot *et al.*, 2006c]). Les recherches Tabou présentées ici sont des versions plus performantes ([Vilcot et Billaut, 2007a] et [Vilcot et Billaut, 2007c]).

3.3.1 Les points communs

Les deux versions de la recherche Tabou ont les parties suivantes en commun : le codage d'une solution, la structure de voisinage et la gestion de la liste tabou.

Le codage d'une solution Une solution est représentée par le graphe conjonctif classique $G = (V, E)$. L'ensemble des sommets est $V = \{O_{i,j}, 1 \leq i \leq n, 1 \leq j \leq n_i\} \cup \{s, t\}$ avec s le sommet source et t le sommet puits. On a un arc entre deux sommets s'il existe une contrainte de précédence entre les opérations correspondantes. Il y a un arc entre O_{i,n_i} et t de longueur $p_{i,n_i,k}$, où R_k est la ressource sur laquelle O_{i,n_i} s'exécute. Il y a un arc entre s et $\{O_{i,j} | \Gamma_{i,j}^{-1} = \emptyset\}$ de longueur r_i. Pour les autres arcs, la longueur est égale à la durée opératoire de l'opération sur la ressource exécutée à l'origine de l'arc. Ce graphe ne doit pas contenir de circuit, sinon l'ordonnancement est infaisable et tous les arcs ont une longueur positive ou nulle. La figure 3.1 donne un exemple d'un tel graphe (arcs pleins représentant les contraintes de gammes et les arcs en pointillés les contraintes de ressources) et la figure 3.2 le diagramme de Gantt correspondant.

En faisant une recherche du plus long chemin entre s et t dans ce graphe on obtient la valeur du *makespan*. Le plus grand retard est calculé en comparant la date de fin de la dernière opération de chaque travail à la date due du travail.

La solution initiale La solution initiale est obtenue par un algorithme glouton fonctionnant en deux étapes. La première étape consiste à résoudre a priori le problème d'affectation pour chaque opération. Ensuite il reste à résoudre un problème de job shop classique, ce qui constitue la seconde étape.

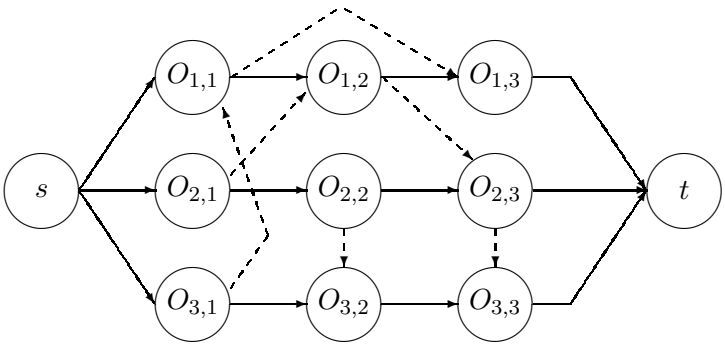

FIG. 3.1 – Exemple d'une représentation d'une solution sous forme de graphe

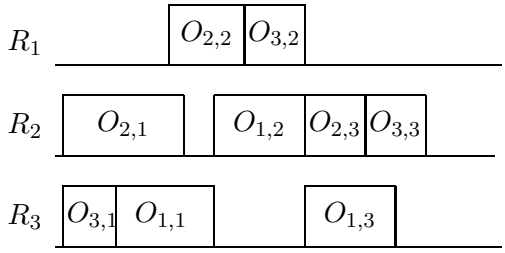

FIG. 3.2 – Exemple d'un diagramme de Gantt

Première étape - Les opérations sont triées, dans une liste $\mathcal{L}$, dans l'ordre croissant des $|R_{i,j}|$ et dans l'ordre décroissant des durées opératoires moyennes en cas d'égalité. On définit la charge d'une ressource par la somme des durées opératoires des opérations affectées à cette ressource. On trie les ressources dans l'ordre croissant de leur charge, on appelle cette liste $\mathcal{R}'$. On prend la première opération $O_{i,j}$ de $\mathcal{L}$, on affecte $O_{i,j}$ sur la première ressource R_k de $\mathcal{R}'$ qui appartient à $R_{i,j}$. La charge de la ressource choisie est mise à jour ainsi que le tri des ressources. Ce processus est itéré tant que $\mathcal{G}$ n'est pas vide. Cette première étape est reprise de [Dauzère-Pérès et Paulli, 1997].

Seconde étape - Le problème de job shop résultant est résolu par un algorithme glouton basé sur la règle SLACK. On note $\mathcal{O}$ l'ensemble des opérations candidates. Au commencement, $\mathcal{O}$ contient toutes les opérations sans prédécesseur, i.e. $\Gamma_{i,j}^{-1} = \emptyset$. La date de début au plus tôt ($r_{i,j}$) d'une opération $O_{i,j}$ est le maximum entre les dates de fin de chaque opération précédente dans la gamme et la date de disponibilité de la ressource R_k qui doit exécuter $O_{i,j}$.

Soit $C^* = \min_{O_{i,j} \in \mathcal{O}}(C_{i,j})$ où $C_{i,j}$ est la date de fin de l'opération $O_{i,j}$ si $O_{i,j}$ est ordonnancée à la date actuelle. Soit R_k^* la ressource exécutant l'opération $O_{i,j}$ telle que $C_{i,j} = C^*$.

Ensuite, parmi $\mathcal{C} = \{O_{i,j} \in \mathcal{O} | r_{i,j} < C^* \text{ et } a(O_{i,j}) = R_k^*\}$, on ordonnance l'opération qui à la plus petite marge (SLACK) $S_{i,j} = d_{i,j} - (r_{i,j} + p_{i,j,k})$: $O_{i^*,j^*} = O_{i^*,j^*} | S_{i^*,j^*} = \min_{O_{i,j}} S_{i,j}$

O_{i^*,j^*} est retirée de $\mathcal{O}$. Les successeurs de O_{i^*,j^*} sont ajoutés à $\mathcal{O}$ si tous leurs prédécesseurs ont été ordonnancés. Les dates de début au plus tôt sont mises à jour et le processus est itéré tant qu'il reste des opérations à ordonnancer.

L'algorithme 7 décrit la seconde étape.

Définition d'un voisin La définition d'un voisin est la même que dans [Dauzère-Pérès et Paulli, 1997] (voir section 2.1 page 29), elle est rappelée ici pour faciliter la compréhension.

On obtient un voisin par le déplacement d'une opération. Cette technique permet à la fois la modification de l'affectation et la modification de la séquence. Dans le graphe conjonctif, déplacer une opération x entre les opérations y et z signifie :

- effacer l'arc (v, x) et (x, w), où v (respectivement w) est le prédécesseur (respectivement le successeur) de x sur la ressource exécutant x,
- ajouter un arc (v, w),
- effacer l'arc (y, z),
- ajouter les arcs (y, x) et (x, z).

On note ce mouvement par $\{x, y, z\}$. La figure 3.3 illustre un tel mouvement.

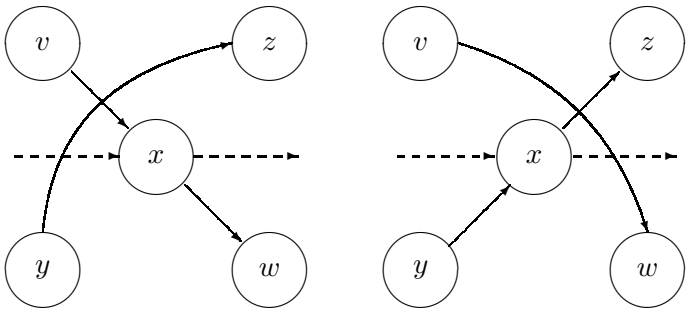

FIG. 3.3 – Déplacer l'opération x entre y et z

La liste Tabou On utilise la liste Tabou $T\ell$ pour empêcher la recherche de boucler entre des solutions. La liste Tabou a une taille fixe et quand cette liste est pleine, l'élément le plus ancien est remplacé par le nouvel élément (règle du premier rentré, premier sorti).

Dans [Dauzère-Pérès et Paulli, 1997] les auteurs proposent trois types de liste Tabou. Nous utilisons celle qu'ils ont considéré comme étant la plus performante. La liste fonctionne ainsi : après le déplacement de x entre y et z, on ajoute les couples (x, y) et (z, x) à la liste Tabou. Un mouvement $\{x', y', z'\}$ est interdit si $(x', y') \in T\ell$ ou si $(z', x') \in T\ell$.

La structure de voisinage On définit le voisinage V ainsi : chaque opération $O_{i,j}$ est échangée avec l'opération immédiatement précédente sur la ressource, si ce n'est pas la première opération. Pour pouvoir changer l'affectation, pour chaque ressource dans $R_{i,j}$ (sauf pour la ressource courante $a(O_{i,j})$), $O_{i,j}$ est déplacée à la première position

Algorithme 7 Algorithme glouton pour le job shop

$\mathcal{O} = \emptyset$
Pour $i = 0$ à $i = m$ **Faire** $C_m[i] = 0$ **fin Pour**
Pour $i = 1$ à n **faire**
    Pour $j = 1$ à n_i **faire**
        $r_{i,j} = 0$
        Si $\Gamma_{i,j}^{-1} = \emptyset$ **Alors** Ajouter $O_{i,j}$ à $\mathcal{O}$ **fin Si**
    fin Pour
fin Pour
Tant que $\mathcal{O} \neq \emptyset$ **faire**
    $C^* = \infty$
    /* *Recherche de l'opération se terminant le plus tôt* */
    Pour chaque $O_{i,j} \in \mathcal{O}$ **faire**
        $R_k = a(O_{i,j})$
        $dateFin = \max(r_{i,j}, C_m[k]) + p_{i,j,k}$
        Si $C^* > dateFin$ **alors**
            $C^* = dateFin$
            $R_k^* = R_k$
        fin Si
    fin Pour
    /* *Recherche de l'opération la plus prioritaire* */
    $minSlack = \infty$
    Pour chaque $O_{i,j} \in \mathcal{O}$ **faire**
        $R_k = a(O_{i,j})$
        Si $R_k = R_k^*$ et $r_{i,j} < C^*$ et $minSlack > d_{i,j} - (r_{i,j} + p_{i,j,k})$ **alors**
            $minSlack = d_{i,j} - (r_{i,j} + p_{i,j,k})$
            $O_{i^*,j^*} = O_{i,j}$
        fin Si
    fin Pour
    /* *Ordonnancer* O_{i^*,j^*} *au plus tôt* */
    $r_{i^*,j^*} = \max(r_{i^*,j^*}, C_m[R_k^*])$
    $C_{i^*,j^*} = r_{i^*,j^*} + p_{i^*,j^*,k^*}$
    $C_m[k^*] = C_{i^*,j^*}$
    /* *Mise à jour des listes et des indicateurs* */
    Retirer O_{i^*,j^*} de $\mathcal{O}$
    Pour chaque $O_{i,j} \in \Gamma_{i^*,j^*}$ **faire**
        $r_{i,j} = \max(r_{i,j}, C_{i^*,j^*})$
        Si Tous les prédécesseurs d'$O_{i,j}$ ont été ordonnancés **alors**
            Ajouter $O_{i,j}$ à $\mathcal{O}$
        fin Si
    fin Pour
fin Tant que

d'insertion possible qui commence au même moment ou juste après la date de début actuelle de l'opération. La figure 3.4 montre un déplacement qui change l'affectation, où x commence dès que y se termine (les lignes pointillées représentent les contraintes de gamme et elles ne sont pas modifiées). Par rapport à [Dauzère-Pérès et Paulli, 1997], notre voisinage est plus restreint.

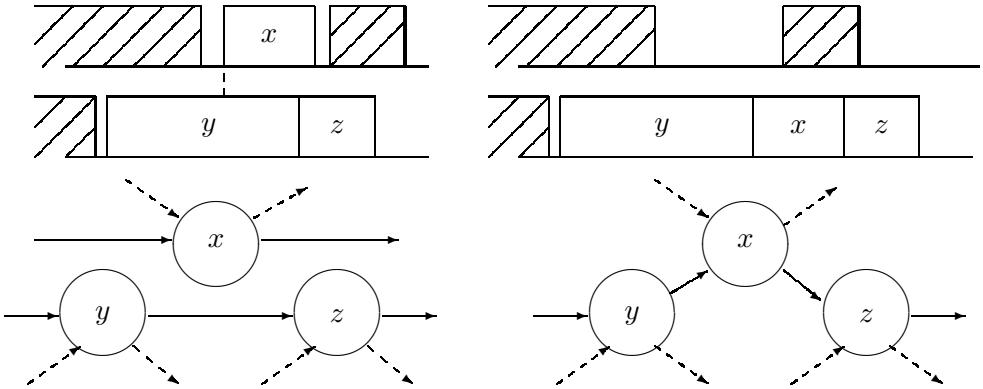

FIG. 3.4 – La modification de l'affectation pour x

L'algorithme 8 décrit la procédure d'exploration du voisinage. Seq_k est l'ensemble des opérations ordonnancées sur R_k et $Seq_{k,l}$ indique l'opération en position l sur la ressource R_k. Cet algorithme commence par essayer tous les échanges d'opérations sur une même ressource, avec l'opération précédente, avant d'essayer de changer les affectations. La procédure $RechercheInsertion(r, o)$ trouve la position d'insertion de l'opération o sur la ressource R_r. La procédure *Évaluer_le_voisin* évalue la qualité du voisin selon l'approche choisie (ϵ-contrainte où combinaison linéaire) et enregistre la solution si c'est la meilleure trouvée jusqu'à présent.

Proposition 1 : Le voisinage V est connexe.

Preuve : On considère une solution initiale S^i et une solution désirée S^t. On doit démontrer qu'il est toujours possible d'obtenir S^t depuis S^i en un nombre fini de mouvements. Nous considérons les graphes conjonctifs correspondant à ces ordonnancements et nous les notons G^i et G^t respectivement. Depuis S^i il est toujours possible de changer l'affectation des opérations pour que l'affectation soit identique à celle de S^t. De par la définition du mouvement qui réalise le changement d'affectation, il est évident qu'un tel ensemble de mouvements ne peut pas créer de circuit dans le graphe conjonctif. La notation S^i est maintenant utilisée pour indiquer l'ordonnancement courant dans l'étape de modification. Nous pouvons maintenant supposer que l'affectation dans S^i est la même que dans S^t.

Supposons que G^t est trié dans l'ordre topologique. On commence par considérer les opérations du premier rang dans G^t. Comme ces opérations n'ont pas de prédécesseur, il est toujours possible - en utilisant le mouvement d'échange - de les mettre au premier rang dans G^i.

Algorithme 8 L'exploration du voisinage

```
1  Pour chaque R_k ∈ R faire
2      Pour l = 1 jusqu'à |Seq_k| faire
3          Si l ≥ 2 et (Seq_{k,l}, Seq_{k,l-2}) ∉ Tℓ et (Seq_{k,l-1}, Seq_{k,l}) ∉ Tℓ alors
4              Déplacer Seq_{k,l} entre Seq_{k,l-2} et Seq_{k,l-1} ; Évaluer_le_voisin ; Annuler
               le déplacement
5          fin Si
6          Pour chaque R_{k'} ∈ R_{Seq_{k,l}} faire
7              l' ← RechercheInsertion(k', Seq_{k,l})
8              Si R_{k'} ≠ R_k et (Seq_{k,l}, Seq_{k',l'-1}) ∉ Tℓ et (Seq_{k',l'}, Seq_{k,l}) ∉ Tℓ alors
9                  Déplacer Seq_{k,l} entre Seq_{k',l'-1} et Seq_{k',l'} ; Évaluer_le_voisin ; An-
                   nuler le déplacement
10             fin Si
11         fin Pour
12     fin Pour
13 fin Pour
```

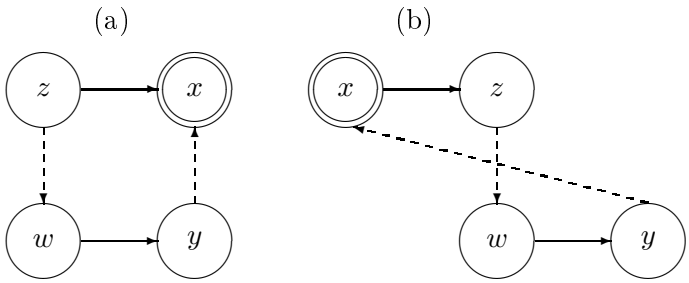

FIG. 3.5 – Génération d'un circuit après une modification de la séquence

Considérons maintenant une opération x de rang r dans G^t et supposons que les opérations de rang $r' < r$ dans G^t sont sur les mêmes rangs dans G^i. La figure 3.5 illustre le *seul cas* où un échange de x avec son prédécesseur immédiat sur la ressource génère un circuit (les lignes pointillées représentent des contraintes de gamme). Notons que le chemin de w à y peut être composé par plus d'un arc. Le mouvement de x dans G^i ne peut pas générer le circuit de la figure 3.5(b) car les opérations z, w et y, qui précédent x dans la figure 3.5(b), sont de rang $r' < r$ et elles sont donc déjà correctement placées dans G^i. Il est donc toujours possible d'échanger x avec sont prédécesseur jusqu'à ce que son rang soit le même dans G^i que dans G^t. Par le déplacement des opérations de G^i dans l'ordre topologique de G^t, on obtient S^t depuis S^i après un nombre fini de déplacements. □

3.3.2 Approche Epsilon-contrainte, $TS\epsilon$

On suppose qu'on cherche à optimiser le C_{max} sous la contrainte $L_{max} < \epsilon$.

Évaluation d'un voisin L'originalité de la recherche Tabou utilisant une approche ϵ-contrainte réside dans la sélection du meilleur voisin. On distingue trois cas :

1. Il existe au moins un voisin qui respecte la borne ϵ. On considère alors que le meilleur voisin est celui qui minimise C_{max} parmi ceux qui vérifient $L_{max} < \epsilon$.
2. Aucun voisin ne respecte la borne ϵ et il en existe au moins un qui a un C_{max} meilleur que celui de la solution courante. Alors on considère que le meilleur voisin est celui qui minimise le L_{max} tout en étant au moins aussi bon sur le C_{max} que la solution courante.
3. Aucun voisin ne respecte la borne ϵ et aucun n'améliore le C_{max} de la solution courante. Alors on considère que le meilleur voisin est celui qui minimise le L_{max}.

La figure 3.6 illustre les trois cas possibles pour le choix du meilleur voisin (entouré).

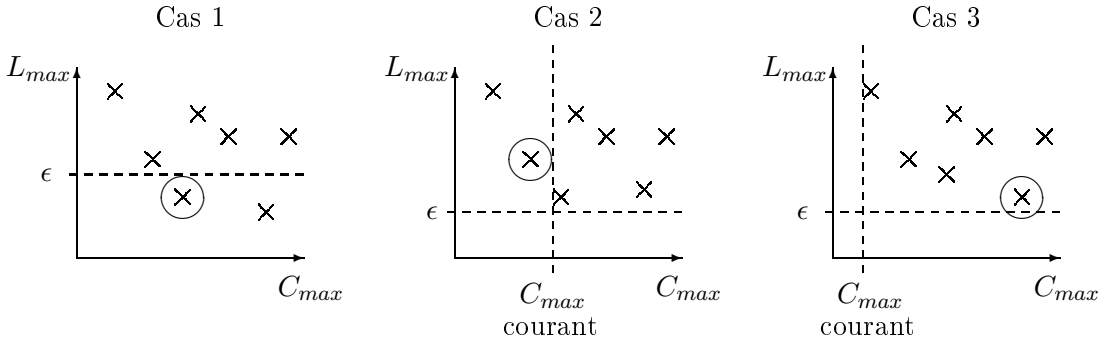

FIG. 3.6 – Choix du meilleur voisin dans le cas d'une approche ϵ-contrainte

Avec cette sélection, on cherche d'abord à obtenir une solution respectant la borne. Puis on améliore le *makespan* en respectant au maximum la borne. On s'autorise quand même à explorer, temporairement, des solutions non satisfaisantes. La solution retournée par $TS\epsilon$ est la meilleure solution respectant la borne trouvée au cours de la recherche. Si aucune solution respectant cette borne n'a été trouvée, alors la recherche retourne $\emptyset$.

Détermination d'un front de Pareto En appelant successivement $TS\epsilon$, on peut déterminer une approximation du front de Pareto. Le principe est décrit par l'algorithme 9.

3.3.3 Approche par combinaison linéaire des critères, $TS\ell c$

Évaluation d'un voisin On évalue une solution par une combinaison linéaire des critères. On considère les critères du *makespan* (C_{max}) et du plus grand retard (L_{max}).

Un des problèmes posés par une telle approche est l'ordre de grandeur des critères. Par exemple, on peut envisager une solution S où $C_{max} = 1000$ et $L_{max} = 10$. Dans ce cas le *makespan* occulte le plus grand retard. De plus, il peut être impossible d'estimer *a priori* l'ordre de grandeur de chaque critère. Pour surmonter cette difficulté, les coefficients de la combinaison linéaire sont normalisés dynamiquement. L'évaluation d'une solution est

Algorithme 9 Détermination d'un front de Pareto avec l'approche epsilon-contrainte

Pré-conditions: $TS\epsilon(val, s)$: lance la recherche Tabou epsilon-contrainte avec comme objectif $L_{max} < val$ et en partant de la solution s. Si $s = \emptyset$, alors la solution initiale est générée par l'algorithme glouton (cf. page 46).

```
1  ε = ∞
2  S ← TSε(ε, ∅)
3  Tant que S ≠ ∅ faire
4      Mettre à jour l'ensemble des solutions non-dominées avec S
5      ε = L_max(S)
6      S ← TSε(ε, S)
7      Si S = ∅ alors
8          S ← TSε(ε, ∅)
9      fin Si
10 fin Tant que
```

donnée par la formule suivante :

$$Z(S) = \frac{\alpha \times C_{max}(S)}{\max(1, \max(C_{max}) - \min(C_{max}))} + \frac{\beta \times L_{max}(S)}{\max(1, \max(L_{max}) - \min(L_{max}))}$$

Avec
- $\{\alpha, \beta\}$ les coefficients choisis par le décideur,
- $C_{max}(S)$ et $L_{max}(S)$ le *makespan* et le plus grand retard de la solution S,

On note S^i_{best} le voisin choisi à l'itération i, et k le numéro de l'itération courante de $TS\ell c$. On calcule les valeurs maximales et minimales des critères par la formule :

$$Y = \{S^i_{best}, \forall i, i < k\} \ ; \ \max(C_{max}) = \max_{y \in Y} C_{max}(y) \ ; \ \min(C_{max}) = \min_{y \in Y} C_{max}(y)$$

$$\max(L_{max}) = \max_{y \in Y} L_{max}(y) \ ; \ \min(L_{max}) = \min_{y \in Y} L_{max}(y)$$

Cette normalisation est inspirée par [Deb *et al.*, 2002] pour le calcul de la distance de *crowding*.

Diversification Parfois la recherche Tabou tombe dans un optimum local et est incapable de s'en sortir. Dans ce cas de figure, on peut utiliser un opérateur de diversification, inspiré par le principe de l'oscillation stratégique (*strategic oscillation* en anglais). L'oscillation stratégique consiste à autoriser l'exploration de solutions non faisables pendant une courte période de temps. Un exemple peut être trouvé dans [Downsland, 1998].

La diversification que nous avons implémentée fonctionne selon le principe suivant. Dans $TS\ell c$, après un certain nombre d'itérations sans amélioration, la recherche recommence avec la meilleure solution connue S^+ et on tire aléatoirement de nouveaux coefficients α' et β' pour la combinaison linéaire. En utilisant cette technique, certaines solutions qui n'auraient jamais étaient explorées avec les coefficients initiaux peuvent être explorées. La recherche continue avec ces nouveaux coefficients jusqu'à ce qu'une

meilleure solution que S^+ (avec les coefficients corrects α et β) soit trouvée ou jusqu'à ce que le critère d'arrêt soit atteint. Dans le premier cas, $TS\ell c$ continue avec la nouvelle solution et les coefficients corrects, sinon c'est la fin de la recherche Tabou.

3.4 Approche par algorithme génétique pour le job shop flexible

Dans le cadre de la détermination d'une approximation d'un front de Pareto, les algorithmes génétiques sont particulièrement bien adaptés. En effet, comme ils manipulent un ensemble de solutions, ils peuvent fournir une approximation du front de Pareto en une seule exécution. Nous proposons deux algorithmes génétiques. Le premier a un codage indirect ([Vilcot et Billaut, 2006]), le second a un codage direct ([Vilcot *et al.*, 2006d], [Vilcot *et al.*, 2006a], [Vilcot et Billaut, 2007d]). Ensuite nous proposons un algorithme « memetique » ([Vilcot et Billaut, 2007b]).

3.4.1 Un algorithme génétique à codage indirect, AG_{ind}

Le codage d'un individu Un individu est codé par deux vecteurs, qui ont une taille égale au nombre d'opérations dans le problème. Le premier vecteur correspond à l'affectation des opérations sur les ressources. Le second vecteur est la priorité de chaque opération.

Ce codage est indirect, c'est-à-dire que le chromosome est insuffisant pour représenter une solution sans ambiguïté et donc pour évaluer la qualité de l'individu ; on a besoin d'un générateur d'ordonnancements pour obtenir une solution. Le générateur commence par affecter les opérations sur les ressources selon le vecteur d'affectation et on résout le problème de job-shop résultant par un procédé similaire à l'algorithme 7 (page 46) sauf qu'au lieu de choisir l'opération qui a la plus petite marge SLACK, on choisit l'opération la plus prioritaire selon le vecteur des priorités.

Le tableau 3.1 donne un exemple d'un tel codage.

Opérations	1	2	3	4	5	6	7
Affectation	R1	R2	R2	R3	R1	R2	R1
Priorité	15	33	6	42	73	99	40

TAB. 3.1 – Exemple de codage d'une solution

Le croisement Le croisement choisi est un croisement dit *à un point* : à partir des deux parents $P1$ et $P2$ on va choisir aléatoirement un point de coupure pour chacun des deux vecteurs. Dans le nouvel individu enfant E, chaque vecteur sera composé des éléments de

$P1$ jusqu'au point de coupure et des éléments de $P2$ après le point de coupure. La figure 3.7 illustre le croisement de deux individus.

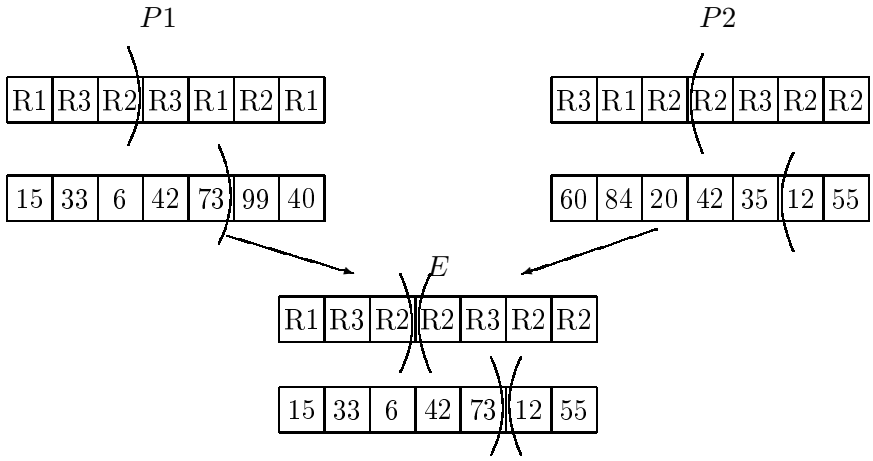

FIG. 3.7 – Croisement des parents $P1$ et $P2$ pour former un enfant E

La mutation L'opérateur de mutation consiste à changer aléatoirement la machine affectée à une opération ou à modifier aléatoirement la priorité d'une opération.

L'évaluation des individus Notre objectif étant d'avoir le plus grand nombre de solutions non dominées, le calcul de la force des individus doit tenir compte du nombre de solutions que domine l'individu mais aussi de la diversité du front de Pareto, afin d'éviter que toutes les solutions soient dans une plage de valeurs trop proches.

Pour cela, on définit une grille dans l'espace des critères (cf. figure 3.8). Tous les individus contenus dans une même case ont la même force. Cette force est calculée en comptant le nombre d'individus se trouvant dans les cases totalement dominées et en soustrayant le nombre d'individus dans la case elle-même (inspirée par [Zitzler *et al.*, 2004]). Ceci a pour but de diversifier la répartition de la population. Par exemple pour la case (C,3), les cases totalement dominées sont {(D,4), (D,5), (D,6), (E,4), (E,5), (E,6)}, qui contiennent en tout huit individus, et le nombre d'éléments dans la case (C,3) est 2. La force des individus de la case (C,3) est donc $8 - 2 = 6$.

L'opérateur de sélection On utilise le tournoi binaire comme opérateur de sélection. Le principe de cet opérateur est le suivant : on tire au hasard deux individus dans la population et on garde le meilleur.

La population La population est de taille fixe. A chaque génération elle est partiellement renouvelée. La population initiale est générée aléatoirement : pour chaque opération de chaque individu on tire aléatoirement une affectation compatible et la priorité est tirée aléatoirement.

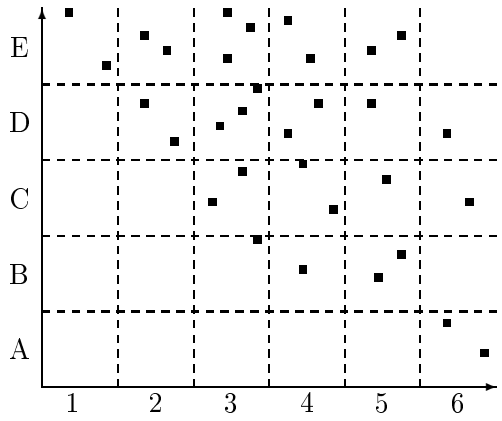

FIG. 3.8 – Calcul de la force des individus

Au cours de l'exécution, les meilleurs individus sont sauvegardés dans une archive externe.

3.4.2 Un algorithme génétique avec un codage direct, AG_{dir}

Un défaut des codages indirects est qu'il n'existe pas une bijection entre un individu et une solution. En effet, plusieurs codages différents peuvent donner la même solution. Telle est la raison qui nous a poussé à développer un algorithme génétique utilisant un codage direct, c'est-à-dire tel que pour un individu donné, il n'existe qu'une seule solution au problème et réciproquement.

Le codage d'un individu On représente un individu par une matrice $\mathcal{C}$ composée de m lignes (une par ressource) où $\mathcal{C}_{k,\ell}$ indique l'opération en position ℓ sur la ressource R_k ($1 \leq k \leq m$), ℓ est compris entre 1 et le nombre maximum d'opérations affectées sur la ressource. La matrice suivante représente le codage de l'individu correspondant au diagramme de Gantt de la figure 3.2 (page 44).

$$\mathcal{C} = \begin{pmatrix} O_{2,2} & O_{3,2} & - & - \\ O_{2,1} & O_{1,2} & O_{2,3} & O_{3,3} \\ O_{3,1} & O_{1,1} & O_{1,3} & - \end{pmatrix}$$

On dit que la matrice a un « profil », qui correspond au nombre de positions valides sur chaque ligne (dépend du nombre d'opérations affectées sur la ressource).

Le croisement Le croisement fonctionne ainsi : premièrement on sélectionne deux parents a et b selon l'opérateur de sélection. Ensuite on génère aléatoirement une matrice $\mathcal{B}$ de booléens, avec le même profil que a. Chaque élément a une équiprobabilité d'être '1' ou '0'. Un '1' signifie que l'opération correspondante dans a sera à la même position dans l'enfant (première étape dans la figure 3.9). Les opérations de a qui sont maintenant dans

l'enfant sont effacées dans b et on obtient pour chaque ligne un ensemble des opérations restantes qui viennent de b. On remplit les positions vides dans l'enfant avec les opérations restantes dans l'ordre où elles apparaissent dans b. Dans le cas où aucune position n'est présente pour placer l'opération, on place cette opération à la fin de la séquence sur la même ressource que dans b. Enfin, s'il reste au final des positions vides sur une ressource, les sous-séquences sont calées à gauche.

La figure 3.9 donne un exemple de croisement

A ce point, il est possible que l'enfant ne soit pas réalisable, c'est-à-dire qu'il y ait un circuit dans le graphe correspondant. Dans ce cas, on identifie une opération qui fait partie du circuit et qui correspond à un '0' dans la matrice $\mathcal{B}$ (il existe au moins une telle opération, sinon le circuit serait aussi dans a). On force la valeur de cet élément à '1' dans la matrice $\mathcal{B}$ et un nouvel enfant est généré. Dans le pire des cas, toutes les valeurs de la matrice $\mathcal{B}$ sont à '1' et l'enfant est un clone de a. Ce cas ne s'est jamais présenté durant nos expérimentations, et généralement cette réparation converge en une ou deux itérations.

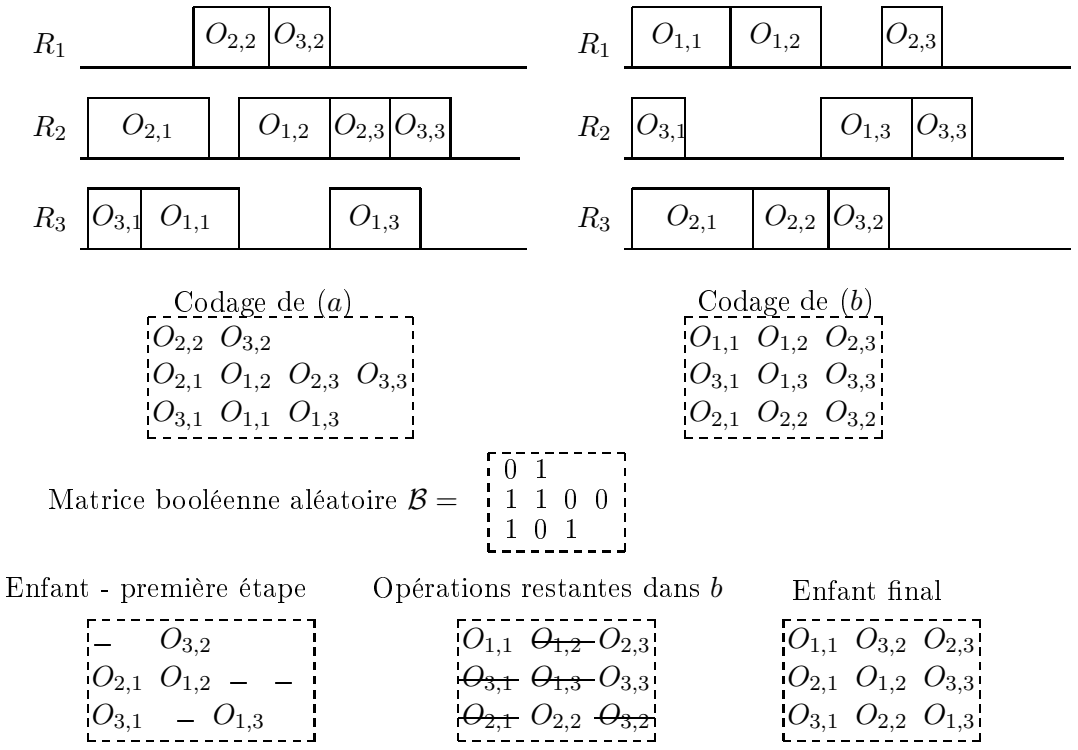

FIG. 3.9 – Croisement

La mutation Dans cet algorithme, la mutation ne s'applique pas à seulement un gène, mais à l'individu en entier. L'opérateur de mutation est une adaptation de la définition d'un voisin dans la recherche Tabou de [Dauzère-Pérès et Paulli, 1997]. Le principe est le suivant : on choisit aléatoirement une opération $O_{i,j}$, cette opération est déplacée à une

position aléatoire sur une des machines compatibles de $R_{i,j}$. On s'assure que le déplacement ne crée pas de circuit dans le graphe. On calcule dynamiquement la probabilité de mutation (Tm) à chaque génération avec la formule suivante :

$$Tm = TM_{max} \times \frac{\text{Nombre de générations sans amélioration}}{\text{Nombre maximum de générations sans amélioration}}$$

Avec TM_{max} la probabilité de mutation maximale définie par le décideur.

La population On peut générer la population initiale de deux façons, soit elle est partiellement initialisée par des solutions issues d'une recherche Tabou, soit elle est totalement aléatoire. Pour générer aléatoirement une solution faisable, on commence par affecter aléatoirement les opérations sur les ressources. Ensuite on utilise l'algorithme glouton (algorithme 7 page 46) sauf qu'au lieu de choisir l'opération suivante qui a la plus petite marge SLACK, on choisit l'opération suivante de façon aléatoire. La population est de taille N.

Par la suite, on note « $AG_{dir}r$ » la version de l'algorithme sans initialisation par $TS\ell c$.

NSGA-II L'algorithme génétique utilise le schéma d'algorithme NSGA-II ([Deb *et al.*, 2002]). Le but de NSGA-II est d'être rapide et élitiste. Il fonctionne de la façon suivante.

Tous les individus de la population initiale (notée P_0 ; $|P_0| = N$) sont évalués selon la méthode des *niveaux non-dominés*. La figure 3.10 illustre le concept des niveaux non-dominés. Le premier niveau contient tous les individus dominants de la population. Ensuite, si on ne tient plus compte de ces individus, le nouvel ensemble d'individus non dominés constitue le second niveau. Ce processus est itéré tant que tous les individus de la population ne sont pas classés. Les niveaux forment la force (à minimiser) des individus de la population initiale. L'algorithme 10 décrit cette procédure. $p \prec q$ signifie que la solution p domine strictement la solution q.

On utilise un tournoi binaire pour la sélection des parents. On utilise l'opérateur de croisement et de mutation pour créer le premier ensemble d'enfants Q_0 ($|Q_0| = N$).

A une génération donnée t, on définit $R_t = P_t \cup Q_t$. On recherche les niveaux non-dominés sur R_t. On trie les individus de R_t selon leur niveau (algorithme 10). On définit par le front $\mathcal{F}_f$ les solutions non-dominées du niveau f.

Les individus dans la population parente P_{t+1} à la génération $t+1$ sont les solutions des fronts $\mathcal{F}_1$ à $\mathcal{F}_\lambda$ avec λ tel que $\sum_{i=1}^{\lambda} |\mathcal{F}_i| \leq N$ et $\sum_{i=1}^{\lambda+1} |\mathcal{F}_i| > N$ plus les $N - \sum_{i=1}^{\lambda} |\mathcal{F}_i|$ premières solutions de $\mathcal{F}_{\lambda+1}$ selon l'ordre décroissant de leur distance de *crowding*. Les solutions restantes sont éliminées. Au final, la taille de P_{t+1} est N. On utilise les opérateurs de croisement et de mutation sur les individus de P_{t+1} pour former une nouvelle population enfante Q_{t+1}. La figure 3.11 illustre la création de la population P_{t+1}.

La distance de *crowding* pour un individu est définie ainsi ([Deb *et al.*, 2002]) : c'est « une estimation du périmètre formé par le pavé utilisant les plus proches voisins comme sommets ». On assure la diversité grâce à la distance de crowding. Pour un individu, la

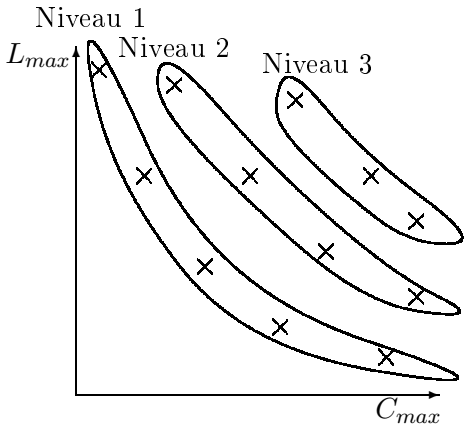

FIG. 3.10 – Niveaux non-dominés

Algorithme 10 Fast non dominated sort [Deb *et al.*, 2002]

Pour chaque $p \in P$ **faire**
  $S_p = \emptyset$; $n_p = 0$
  Pour chaque $q \in P$ **faire**
    Si $p \prec q$ **alors**
      /* *si p domine q, ajouter q à l'ensemble des solutions dominées par p* */
      $S_p = S_p \cup \{q\}$
    Sinon Si $q \prec p$ **alors**
      /* *si q domine p, incrémenter le compteur de dominance de p* */
      $n_p = n_p + 1$
    fin Si
  fin Pour
  Si $n_p = 0$ **alors**
    $p_{rank} = 1$; $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$ /* *p appartient au premier front* */
  fin Si
fin Pour
$i = 1$ /* *Initialise le compteur de front* */
Tant que $\mathcal{F}_i \neq \emptyset$ **faire**
  $Q = \emptyset$ /* *Utilisé pour stocker les membres du prochain front* */
  Pour chaque $p \in \mathcal{F}_i$ **faire**
    Pour chaque $q \in S_p$ **faire**
      $n_q = n_q - 1$
      Si $n_q = 0$ **alors**
        $q_{rank} = i + 1$; $Q = Q \cup \{q\}$ /* *q appartient au prochain front* */
      fin Si
    fin Pour
  fin Pour
  $i = i + 1$; $\mathcal{F}_i = Q$
fin Tant que

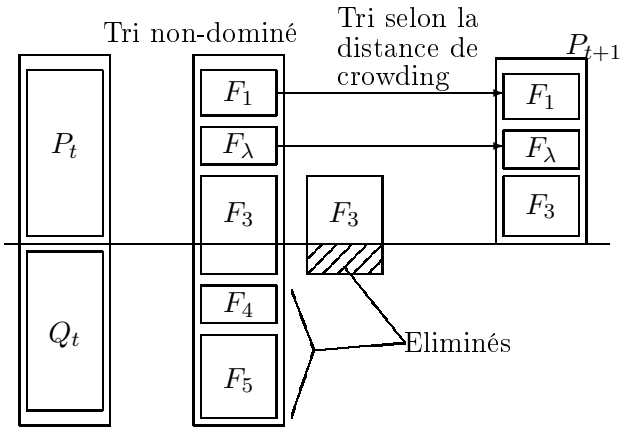

FIG. 3.11 – Construction de la population P_{t+1}

distance de crowding est la somme des distances normalisées (dans l'espace des critères) entre les voisins *droit* et *gauche* pour chaque objectif considéré. Les solutions extrêmes ont une distance de crowding égale à l'infini. L'algorithme 11 décrit le calcul des distances de crowding, la figure 3.12 illustre le concept dans deux dimensions.

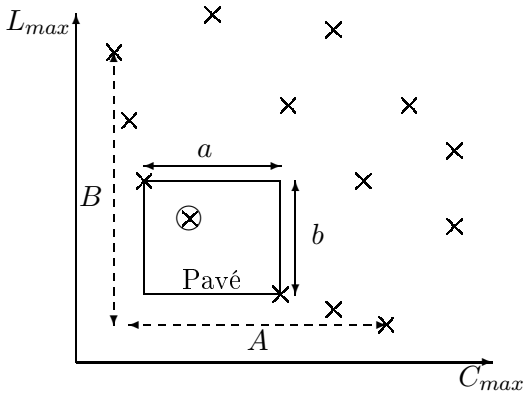

FIG. 3.12 – Calcul de la distance de *crowding*

Dans notre exemple à deux dimensions, la distance de crowding d_c de l'individu vaut :

$$d_c = \frac{a}{A} + \frac{b}{B}$$

L'opérateur de sélection est un tournoi binaire : entre deux individus, on sélectionne celui avec le plus faible niveau. Si les deux individus sont sur le même niveau, le meilleur individu est celui avec la plus grande distance de crowding.

L'algorithme s'arrête après un certain nombre de générations sans amélioration ou après un certain temps de calcul, tout deux définit par l'utilisateur.

Algorithme 11 Calcul des distances de *crowding* des individus d'un front

Pré-conditions: I : ensemble des individus du front considéré
Pré-conditions: tri(J, c) : fonction qui trie les individus de l'ensemble J selon leur valeur du critère c.
Pré-conditions: f_c^{max} : plus grande valeur du critère c dans le front.
Pré-conditions: f_c^{min} : plus petite valeur du critère c dans le front.
Pré-conditions: $I[x].c$: valeur du critère c du $x^{\text{ième}}$ individu dans l'ensemble I.

```
1  l = |I| /* Nombre de solutions dans I */
2  Pour chaque i faire
3      I[i]_distance = 0 /* Initialisation des distances */
4  fin Pour
5  Pour chaque fonction objectif m faire
6      I = tri(I, m)
7      I[1]_distance = I[l]_distance = ∞ /* Les individus aux extrémités ont une distance infinie */
8      Pour i=2 à l − 1 faire
9          I[i]_distance = I[i]_distance + (I[i+1].m − I[i−1].m) / (f_m^max − f_m^min)
10     fin Pour
11 fin Pour
```

Le code de l'algorithme de Deb est disponible sur le web :
http ://www.iitk.ac.in/kangal/deb_research.shtml

Toutefois, pour des raisons d'efficacité, il nous a semblé préférable de le recoder entièrement.

3.4.3 Algorithme Mémétique, Mem

Un algorithme Mémétique est une hybridation entre un algorithme génétique et une recherche locale. Dans ce type d'algorithme, une méthode de recherche locale, ici une recherche Tabou, remplace la mutation dans l'algorithme génétique. Nous proposons d'hybrider notre algorithme génétique à codage direct (AG_{dir}) avec nos deux variantes de notre recherche Tabou.

Le principe est le suivant : premièrement on lance la recherche Tabou basée sur la combinaison linéaire des critères ($TS\ell c$) pour trouver quelques bonnes solutions au problème. Ensuite ces solutions sont injectées dans la population initiale de l'algorithme génétique. La recherche Tabou basée sur l'approche ϵ-contrainte ($TS\epsilon$) remplace la mutation. Par la suite on continuera à utiliser le terme « mutation » pour faciliter la lecture.

Pour l'initialisation, $TS\ell c$ est lancée avec comme valeurs de coefficients $(\alpha, \beta) = \{(0,1); (0.25, 0.75); (0.5, 0.5); (0.75, 0.25); (1,0)\}$.

Insertion de la recherche Tabou dans l'algorithme génétique On peut envisager plusieurs techniques pour remplacer la mutation classique par la recherche Tabou. La plus simple étant qu'à chaque fois qu'un individu doit muter, on lance la recherche Tabou avec

cet individu comme point de départ. Dans ce cas, le mutant est la solution renvoyée par la recherche Tabou.

Cette première approche naïve peut être inefficace avec NSGA-II. En effet, NSGA-II est un algorithme fortement élitiste et beaucoup de solutions sont éliminées à chaque génération. Ainsi, si la solution fournie par la recherche Tabou n'est pas de qualité suffisante elle sera éliminée sans jamais être utilisée lors d'un croisement. Dans ce cas le temps de calcul utilisé lors de la recherche Tabou est complètement perdu.

La première technique que nous avons utilisée pour prévenir ce phénomène est de faire en sorte qu'un individu issu de la mutation ne puisse pas être éliminé tant qu'il ne s'est pas reproduit. Le problème avec cette méthode est que l'enfant ainsi obtenu n'a aucune garantie de ne pas être éliminé. On repousse le problème « d'une génération ».

La seconde technique consiste à ne faire muter que des individus de bonne qualité. Pour cela on ne s'autorise à lancer la recherche Tabou que sur des solutions appartenant au premier front.

Plus spécifiquement concernant l'appel à $TS\epsilon$, le paramètre ϵ est tiré aléatoirement entre $0.7 \times L_{max}(S)$ et $1.2 \times L_{max}(S)$, où S est la solution à muter.

Une remarque qu'on peut faire avec les méthodes de voisinage comme une recherche Tabou est que parfois il vaut mieux partir d'une solution mauvaise S' pour arriver à une bonne plutôt que de partir d'une solution déjà bonne S. En effet en partant d'une bonne solution, on risque d'être bloqué dans l'extremum local de cette solution. La figure 3.13 illustre ce genre de difficultés.

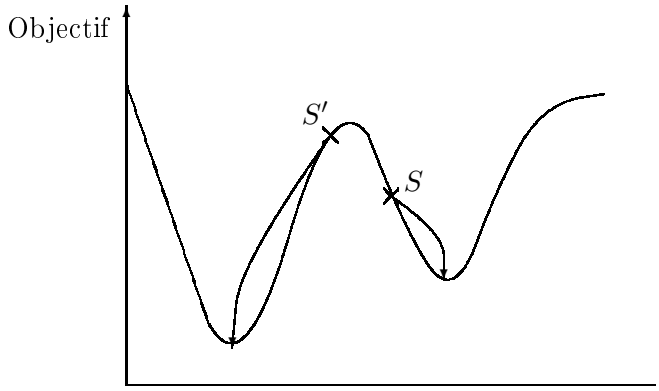

FIG. 3.13 – Efficacité d'un algorithme selon la solution initiale

Dans notre algorithme, à chaque mutation la probabilité de lancer $TS\epsilon$ avec comme solution de départ l'individu à muter est de 50%, dans les autres cas, $TS\epsilon$ utilise comme solution de départ celle générée par l'algorithme glouton.

L'algorithme 12 décrit le fonctionnement de la méthode. La recherche Tabou basée sur la combinaison linéaire des critères génère le premier ensemble de solutions initiales S_{init} à la ligne 1. A la ligne 2, on ajoute à S_{init} des solutions aléatoires S_{rand} pour que $|S_{init} \cup S_{rand}| = N$, on obtient ainsi la première population parent $P_0 = S_{init} \cup S_{rand}$.

La première population enfant Q_0 est obtenue à partir de P_0 à la ligne 3. On initialise le compteur de génération t à la ligne 4.

De la ligne 5 à la ligne 26, on a la boucle principale de l'algorithme. A la ligne 5, on classe les individus de Q_t selon leur niveau avec l'algorithme *Fast non-dominated sort* (cf. algorithme 10). On obtient $\mathcal{F}$ qui est l'ensemble des niveaux non-dominés, on note $\mathcal{F}_1$ le premier niveau, $\mathcal{F}_2$ le deuxième niveau, etc.

La boucle de la ligne 7 à la ligne 14 correspond à la mutation. A la ligne 8 on vérifie que l'individu appartient au premier front et que la probabilité de mutation est vérifiée. La ligne 9 calcule la valeur d'epsilon : on tire selon une loi aléatoire uniforme ϵ entre 70% et 120% de la valeur du L_{max} de l'individu à muter. A la ligne 10, on lance la recherche Tabou ϵ-contrainte sur l'individu considéré.

A la ligne 13, on réunit la population de parents P_t et d'enfants Q_t dans la population R_t. A la ligne 14, on classe les individus de R_t selon leur niveau avec l'algorithme *Fast non-dominated sort* (cf. algorithme 10). La future population de parents P_{t+1} est initialisée à la ligne 15. De la ligne 16 à la ligne 24, on remplit la population de parents pour la génération suivante. La boucle de la ligne 17 à 21 permet de remplir P_{t+1} avec les individus des premiers niveaux, tel que $|P_{t+1}| \leq N$. A la ligne 22, la fonction *tri(F)* trie les individus du niveau F selon leur distance de crowding décroissante. La ligne 24 finit de remplir la population P_{t+1} pour que $|P_{t+1}| = N$. Enfin à la ligne 25, on crée la population d'enfants de la génération suivante.

3.5 Expérimentations

Afin de pouvoir évaluer correctement nos algorithmes, nous avons choisi de les tester sur des instances issues de la littérature.

Toutes les expérimentations ont été réalisées sur un PC équipé d'un Pentium IV cadencé à 2.8GHz et ayant 512Mo de mémoire vive.

3.5.1 Protocole expérimental

On réalise les expérimentations sur les instances d'Hurink ([Hurink *et al.*, 1994]), elles-mêmes proviennent de [Adams *et al.*, 1988]. Ce sont des instances pour le problème de job shop flexible. La durée opératoire de chaque opération ne dépend pas de la ressource sur laquelle elle est affectée. Nous avons ajouté des dates de fin souhaitées aux travaux pour pouvoir faire des évaluations sur le plus grand retard. Nous nous sommes inspirés pour cela de [Demirkol *et al.*, 1998]. On génère, selon une loi uniforme, la date de fin souhaitée d_i d'un travail i grâce à la formule :

$$d_i \hookrightarrow \mathcal{U}\left[\mu_i \times \left(1 - \frac{R}{2}\right); \mu_i \times \left(1 + \frac{R}{2}\right)\right]$$

Algorithme 12 Algorithme memetic

Pré-conditions: $TS\ell c$: Recherche Tabou basée sur la combinaison linéaire
Pré-conditions: $TS\epsilon$: Recherche Tabou basée sur l'approche ϵ-contrainte
Pré-conditions: Croisement(Z) : lance les opérations de croisement sur la population Z
Pré-conditions: Fast-Non-Dominated-Sort(Z) : trie les individus selon leur niveau
Pré-conditions: Crowding-distance(F) : calcule la distance de crowding des individus du niveau F
Pré-conditions: tri(F, $\prec_n$) : trie les individus du niveau F selon l'ordre $\prec_n$

Lancer $TS\ell c$, soit S_{init} l'ensemble des solutions renvoyées par $TS\ell c$
$P_0 = S_{init} + S_{rand}$, avec S_{rand} des solutions aléatoire. $|P_0| = N$
$Q_0 =$ Croisement(P_0)
$t = 0$
Tant que Critère d'arrêt non atteint **faire**
    $\mathcal{F} =$ Fast-Non-Dominated-Sort(Q_t)
    Pour $i = 1$ à N **faire**
        Si $Q_t[i] \in \mathcal{F}_1$ **et** la probabilité de mutation est vérifiée **alors**
            $\epsilon =$ rand$(0.7 \times L_{max}(Q_t[i]), 1.2 \times L_{max}(Q_t[i]))$
            $Q_t[i] =$TSϵ($Q_t[i]$, ϵ)
        fin Si
    fin Pour
    $R_t = P_t \cup Q_t$
    $\mathcal{F} =$ Fast-Non-Dominated-Sort(R_t)
    $P_{t+1} = \emptyset$
    $i = 1$
    Tant que $|P_{t+1}| + |\mathcal{F}_i| \leq N$ **faire**
        Crowding-distance($\mathcal{F}_i$)
        $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
        $i = i + 1$
    fin Tant que
    tri($\mathcal{F}_i$, $\prec_n$)
    Crowding-distance($\mathcal{F}_i$)
    $P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$
    $Q_{t+1} =$ croisement(P_{t+1})
    $t = t + 1$
fin Tant que

avec

$$\mu_i = \left(1 + \frac{T \times n}{m}\right) \times \sum_{j=1}^{n_i} p_{i,j}$$

où T (importance de la date de fin souhaité) et R (amplitude de variation) sont deux paramètres fixés à $T = 0.3$ et $R = 0.5$.

Les instances sont découpées en quatre sous-ensembles :
- `sdata` : instances de Lawrence (une ressource possible par opération), pas de problème d'affectation,
- `edata` : instances avec très peu de flexibilité (ressources possibles par opération), très proches du problème de job-shop classique,
- `rdata` : instances avec en moyenne deux machines possibles par opération,
- `vdata` : instances avec en moyenne $m/2$ machines possibles par opération.

Les instances sont découpées par classe selon leur taille :

Instances	m	n	Instances	m	n
la01 à la05	5	10	la21 à la25	10	15
la06 à la10	5	15	la26 à la30	10	20
la11 à la15	5	20	la31 à la35	10	30
la16 à la20	10	10	la36 à la40	15	15

3.5.2 Indicateurs

Afin de comparer les fronts de Pareto, nous avons utilisé les indicateurs de l'optimisation multicritère ([Jaszkiewicz, 2004]).

Faible dominance d'un ensemble - *Weak outperformance* Un ensemble de solutions non-dominées $S_\times$ domine faiblement un ensemble $S_\circ$ si aucune solution dans $S_\times$ n'est dominée par une solution dans $S_\circ$ et si au moins une solution dans $S_\times$ domine une solution dans $S_\circ$. La figure 3.14 illustre la notion de dominance faible : l'ensemble $S_\times$ (les solutions représentées par une croix) domine faiblement l'ensemble $S_\circ$ (les solutions représentées par un cercle).

$Ow(S_\times, S_\circ)$ est égal à 1 si $S_\times$ domine faiblement $S_\circ$, 0 sinon. Notons que $Ow(S_\times, S_\circ) + Ow(S_\circ, S_\times)$ peut être égal à 0 ou 1, mais pas à 2.

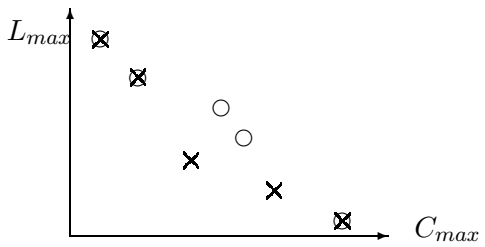

FIG. 3.14 – Notion de dominance faible

Soit exp_i et exp_j deux séries d'expérimentations et S_{exp_i} (resp. S_{exp_j}) l'ensemble des ensembles des solutions non-dominées pour la série exp_i (resp. exp_j). Pour chaque $j \neq i$, $SOw(exp_i, exp_j)$ est la somme pour chaque instance considérée des $Ow(S_{exp_i}, S_{exp_j})$.

Plus cet indicateur est grand, meilleur est S_{exp_i}.

La contribution au front idéal - *Net Front Contribution* Considérons deux ensembles de solutions non-dominées $S_\times$ et $S_\circ$, $S_\star$ représente l'ensemble des solutions non-dominées dans $S_\times \cup S_\circ$. $NFC(S_\times, S_\circ)$ est la proportion de solutions de $S_\times$ dans $S_\star$. $\overline{NFC}(exp_i, exp_j)$ dénote la moyenne de $NFC(S_{exp_i}, S_{exp_j})$ pour toutes les instances considérées.

L'indicateur $\overline{NFC}$ correspond à la moyenne des contributions au front idéal. Plus cet indicateur est grand, meilleur est S_{exp_i}.

Il faut noter que cet indicateur n'est pas symétrique, c'est à dire qu'on peut avoir $NFC(S_{exp_i}, S_{exp_j}) + NFC(S_{exp_j}, S_{exp_i}) \neq 100\%$. Cela est notamment vrai si les deux fronts sont identiques.

Distance à un front de référence Soit F_r un front de référence, c'est-à-dire un ensemble de solutions non-dominées à la fois entre-elles mais aussi par n'importe quelle autre solution connue. Soit $S_\times$ un ensemble de solutions qu'on souhaite évaluer. On définit la distance $d_{fr}(S_\times)$ de notre ensemble au front de référence par :

$$d_{fr}(S_\times) = \frac{1}{|F_r|} \sum_{r \in F_r} \min_{z \in S_\times} d(z, r)$$

Où $d(z, r)$ est la distance euclidienne dans l'espace des critères. Cet indicateur a été originellement proposé par [Czyak et Jaszkiewicz, 1996].

Diversité Soit S un ensemble de solutions non-dominées. On définit $div(S)$ comme étant la moyenne des distances de *crowding* des solutions de S (en excluant les solutions extrêmes qui ont leur distance égale à infini).

3.5.3 Expérimentations préliminaires

Des expérimentations préliminaires ont été réalisées sur quelques instances pour trouver les meilleurs paramètres pour nos algorithmes. Les conclusions sont données ci-dessous.

Pour les deux versions de la recherche Tabou, $TS\epsilon$ et $TS\ell c$, le nombre d'itérations maximum sans amélioration $\#Iter$ est fixé à 500 et la taille de la liste Tabou $|T\ell|$ à 100.

Pour AG_{ind}, la taille de la population N est 500, le nombre de générations maximum sans amélioration $\#MaxGen$ est 2000, le taux de renouvellement de la population tx est 0.7 et le taux de mutation Mut est 0.01.

Pour AG_{dir}, la taille de la population N est fixée à 500 et la probabilité maximale de mutation TM_{max} à 0.5.

Enfin pour l'algorithme mémétique, la taille de la population N est fixée à 100, la probabilité maximale de mutation TM_{max} à 0.8, le nombre d'itérations maximum sans amélioration des recherches Tabou $\#Iter$ est fixé à 300 et la taille des listes Tabou $|T\ell|$ à 200.

On renvoie à l'annexe A.1 pour le détail des expérimentations.

3.5.4 Comparaison des méthodes

Dans cette section, nous comparons les méthodes entre elles pour déterminer la ou les plus performantes.

3.5.4.1 Comparaison de $TS\ell c$ et de $TS\epsilon$

Nous commençons par comparer les deux versions de la recherche Tabou. Cette comparaison est biaisée dans le sens où $TS\epsilon$ permet de déterminer une approximation du front de Pareto grâce à l'algorithme 9, alors que $TS\ell c$ ne détermine que des solutions supportées. Pour avoir un ensemble de solutions à comparer, nous avons lancé onze fois $TS\ell c$ avec $\alpha \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}$ (avec $\beta = 1 - \alpha$ pour chaque valeur de α).

Le tableau 3.2 représente les performances de $TS\epsilon$ vis-à-vis de $TS\ell c$ et le tableau 3.3 représente les performances de $TS\ell c$ vis-à-vis de $TS\epsilon$.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	2	**85%**	1	45%	0	56%	0	62%
5 x 15	2	**87%**	1	55%	1	64%	1	**70%**
5 x 20	2	**78%**	1	**77%**	0	66%	0	67%
10 x 10	1	52%	1	63%	0	52%	0	48%
10 x 15	0	58%	0	36%	1	57%	0	**75%**
10 x 20	1	53%	1	28%	1	**72%**	0	63%
10 x 30	2	40%	1	35%	0	8%	0	19%
15 x 15	0	33%	2	50%	0	51%	0	32%

TAB. 3.2 – Comparaison de $TS\epsilon$ contre $TS\ell c$

On peut constater que pour les petites instances ($m = 5$ et $n \leq 20$) $TS\epsilon$ obtient de bonnes performances. A l'inverse, lorsque la taille de l'instance augmente, $TS\ell c$ domine $TS\epsilon$ alors que $TS\ell c$ n'est pas faite pour déterminer le front de Pareto. On peut tout de même noter qu'à aucun moment une méthode domine totalement l'autre, c'est-à-dire que nous n'avons jamais $SOw = 5$ et $\overline{NFC} = 100\%$.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	2	68%	3	73%	0	47%	0	38%
5 x 15	1	62%	1	56%	0	36%	0	30%
5 x 20	0	38%	0	37%	0	34%	0	34%
10 x 10	0	53%	0	42%	1	48%	0	52%
10 x 15	1	52%	2	64%	1	43%	0	25%
10 x 20	1	47%	3	**72%**	0	28%	0	37%
10 x 30	3	**70%**	3	65%	**4**	**92%**	3	**81%**
15 x 15	3	**74%**	2	50%	0	49%	2	68%

TAB. 3.3 – Comparaison de $TS\ell c$ contre $TS\epsilon$

3.5.4.2 Comparaison de AG_{ind} et de AG_{dir}

Dans cette section nous comparons les deux algorithmes génétiques. Pour ces expérimentations, AG_{dir} a sa population initialisée par $TS\ell c$. Les tableaux 3.4 et 3.5 représentent les performances entre les deux algorithmes et le tableau 3.6 donne les temps de calcul. Pour ce dernier tableau, le temps de calcul de AG_{dir} comprend aussi le temps de calcul nécessaire à $TS\ell c$ pour trouver les solutions initiales de la population. Les temps de calculs ont été limités à 10 minutes pour les approches génétiques, mais pas pour $TS\ell c$, ce qui explique pourquoi dans certains cas $CPU(AG_{dir}) > 600s$.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	2	**82%**	0	47%	0	0%	0	25%
5 x 15	0	34%	0	36%	0	6%	0	17%
5 x 20	1	71%	0	30%	0	12%	0	13%
10 x 10	0	48%	0	35%	0	10%	0	48%
10 x 15	0	20%	0	7%	0	8%	0	17%
10 x 20	0	60%	0	51%	0	15%	0	13%
10 x 30	1	51%	0	53%	0	14%	0	0%
15 x 15	1	49%	0	57%	1	45%	0	0%

TAB. 3.4 – Comparaison de AG_{ind} contre AG_{dir}

On constate que dans peu de cas AG_{ind} domine faiblement AG_{dir}. Plus exactement, on a 6 instances sur 160 où $Ow(AG_{ind}, AG_{dir}) = 1$. Si on s'intéresse au tableau 3.5, on constate que AG_{dir} domine faiblement AG_{ind} dans 65 instances sur 160. Ce qui signifie que pour 89 instances $(160 - 65 - 6)$, les fronts obtenus par les deux algorithmes se croisent. Si on regarde l'indicateur $\overline{NFC}$, on constate que AG_{ind} devient de moins en moins performant avec l'augmentation de la flexibilité : 52% (en moyenne) pour `sdata`, 40% pour `edata`, 14% pour `rdata` et 17% pour `vdata` (cf. tableau 3.4). A l'inverse, le *Net Front Contribution* de AG_{dir} s'améliore avec la flexibilité.

En ce qui concerne les temps de calcul, on constate que jusqu'aux instances de taille $m = 10$ et $n = 15$, les deux algorithmes atteignent leur nombre maximum de générations

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	0	23%	1	64%	**5**	**100%**	1	75%
5 x 15	3	74%	2	72%	4	94%	2	83%
5 x 20	1	45%	1	70%	1	88%	0	87%
10 x 10	0	54%	1	65%	4	90%	0	52%
10 x 15	3	81%	4	93%	4	92%	3	83%
10 x 20	0	40%	0	49%	3	85%	3	87%
10 x 30	1	49%	1	47%	3	86%	**5**	**100%**
15 x 15	2	51%	0	43%	2	55%	**5**	**100%**

TAB. 3.5 – Comparaison de AG_{dir} contre AG_{ind}

	sdata		edata		rdata		vdata	
CPU	AG_{ind}	AG_{dir}	AG_{ind}	AG_{dir}	AG_{ind}	AG_{dir}	AG_{ind}	AG_{dir}
5 x 10	69	94	95	106	105	495	155	216
5 x 15	183	110	204	225	263	538	387	546
5 x 20	307	174	517	416	526	645	556	665
10 x 10	298	280	223	333	272	583	353	292
10 x 15	569	311	508	298	438	448	599	949
10 x 20	600	514	592	515	600	1057	599	1534
10 x 30	600	864	600	1067	600	2053	600	4097
15 x 15	600	488	567	660	600	1022	600	2902

TAB. 3.6 – Temps de calcul moyen (en secondes) pour AG_{dir} et AG_{ind}

sans amélioration, excepté pour `vdata`. Pour les grande instances, $TS\ell c$ pèse fortement sur la vitesse de AG_{dir}.

On peut dire globalement que AG_{dir} est meilleur que AG_{ind}.

3.5.4.3 Comparaison de AG_{dir} avec ou sans initialisation par $TS\ell c$

Dans cette partie on cherche à voir quelle est l'importance de l'initialisation par $TS\ell c$ dans AG_{dir}. Ici $AG_{dir}r$ a été relancé avec comme durée maximale d'exécution 10 minutes, augmentée du temps de calcul nécessaire pour la recherche Tabou. Ainsi le temps d'exécution maximal de chaque méthode est équivalent. Les tableaux 3.7 et 3.8 représentent les performances entre les deux algorithmes et le tableau 3.9 donne les temps de calcul.

	`sdata`		`edata`		`rdata`		`vdata`	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	1	74%	0	30%	0	32%	0	53%
5 x 15	0	31%	0	17%	0	6%	0	6%
5 x 20	0	37%	0	24%	0	23%	0	9%
10 x 10	0	18%	0	13%	0	9%	0	15%
10 x 15	0	10%	0	6%	0	10%	0	11%
10 x 20	0	33%	0	27%	0	9%	0	24%
10 x 30	0	5%	0	7%	0	20%	0	29%
15 x 15	0	21%	0	20%	0	0%	0	0%

TAB. 3.7 – Comparaison de $AG_{dir}r$ sans initialisation contre AG_{dir} avec initialisation

	`sdata`		`edata`		`rdata`		`vdata`	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	0	44%	3	82%	0	68%	0	47%
5 x 15	2	76%	3	84%	3	94%	3	94%
5 x 20	2	79%	2	76%	0	77%	0	91%
10 x 10	2	82%	4	87%	4	91%	3	85%
10 x 15	3	90%	4	94%	4	90%	3	89%
10 x 20	1	67%	2	73%	3	91%	2	76%
10 x 30	4	95%	4	93%	1	80%	2	71%
15 x 15	2	79%	3	80%	**5**	**100%**	**5**	**100%**

TAB. 3.8 – Comparaison de AG_{dir} avec initialisation contre $AG_{ind}r$ sans initialisation

On constate que $AG_{dir}r$ ne domine AG_{dir} que pour une seule instance sur les 160. Quant à la contribution au front idéal, $AG_{dir}r$ ne dépasse les 50% que pour deux classes d'instances. Réciproquement, AG_{dir} avec initialisation par $TS\ell c$ est nettement plus performant que la version sans initialisation, surtout pour l'indicateur $\overline{NFC}$.

Les temps de calcul sont évidemment du même ordre de grandeur, même si dans quelques cas $AG_{dir}r$ prend deux fois plus de temps (ex : `sdata` $m = 10; n = 10$). On constate que pour les petites instances, la limitation à 10 minutes n'est pas le facteur

	sdata		edata		rdata		vdata	
CPU	Sans init	Avec init	Sans init	Avec init	Sans init	Avec init	Sans init	Avec init
5 x 10	75	94	120	106	372	495	545	216
5 x 15	223	110	247	225	593	538	504	546
5 x 20	141	174	493	416	645	645	665	665
10 x 10	401	280	366	333	669	583	436	292
10 x 15	593	311	528	298	694	448	1034	949
10 x 20	594	514	690	561	953	1057	1605	1534
10 x 30	927	864	1068	1067	2054	2053	4097	4097
15 x 15	712	488	756	660	1080	1022	2275	2902

TAB. 3.9 – Temps de calcul moyen (en secondes) pour AG_{dir} avec et sans initialisation

limitant, à l'inverse le temps de calcul augmente fortement pour les grosses instances avec beaucoup de flexibilité.

On peut donc conclure que l'apport de $TS\ell c$ dans AG_{dir} est significatif et qu'il permet d'obtenir de meilleures solutions dans un temps équivalent.

3.5.4.4 Comparaison de $TS\epsilon$ et de AG_{dir}

Nous nous intéressons maintenant aux performances de $TS\epsilon$ par rapport à AG_{dir} avec initialisation par $TS\ell c$. Les tableaux 3.10 et 3.11 représentent les performances entre les deux algorithmes et le tableau 3.12 donne les temps de calcul.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	0	34%	0	21%	0	10%	0	36%
5 x 15	0	62%	0	22%	0	26%	0	34%
5 x 20	0	37%	0	18%	0	27%	0	16%
10 x 10	0	7%	0	23%	0	24%	0	9%
10 x 15	0	22%	0	22%	1	45%	0	62%
10 x 20	0	29%	0	8%	1	46%	0	60%
10 x 30	0	10%	0	0%	0	8%	0	13%
15 x 15	0	21%	0	23%	0	28%	0	27%

TAB. 3.10 – Comparaison de $TS\epsilon$ contre AG_{dir}

On constate que $TS\epsilon$ domine AG_{dir} pour seulement 2 instances sur 160. De plus, la contribution moyenne au front idéal est toujours inférieure à 62% et est supérieure à 50% pour 3 classes d'instances seulement. Si on regarde plus spécifiquement le tableau 3.11, on se rend compte que AG_{dir} est nettement plus performant que $TS\epsilon$ pour les instances avec peu de flexibilité (`sdata` et `edata`). Pour les instances avec plus de flexibilité (`rdata` et `vdata`), AG_{dir} domine moins $TS\epsilon$, mais l'indicateur $\overline{NFC}$ reste fort. Ce qui indique que AG_{dir} trouve beaucoup de solutions appartenant au front idéal, mais que $TS\epsilon$ arrive tout de même à trouver quelques solutions qui dominent au moins une trouvée par AG_{dir}.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	4	**100%**	4	95%	3	90%	0	64%
5 x 15	2	88%	4	95%	1	74%	0	66%
5 x 20	3	95%	**5**	**100%**	2	74%	0	84%
10 x 10	4	97%	3	77%	1	76%	3	91%
10 x 15	3	88%	3	78%	2	55%	0	38%
10 x 20	2	71%	4	92%	1	54%	0	40%
10 x 30	4	95%	**5**	**100%**	4	92%	3	87%
15 x 15	3	79%	3	80%	3	72%	2	73%

TAB. 3.11 – Comparaison de AG_{dir} contre $TS\epsilon$

	sdata		edata		rdata		vdata	
CPU	$TS\epsilon$	AG_{dir}	$TS\epsilon$	AG_{dir}	$TS\epsilon$	AG_{dir}	$TS\epsilon$	AG_{dir}
5 x 10	17	94	16	106	51	495	65	216
5 x 15	27	110	39	225	119	538	184	546
5 x 20	40	174	94	416	180	645	246	665
10 x 10	27	280	50	333	204	583	57	292
10 x 15	99	311	131	298	200	448	715	949
10 x 20	157	514	114	534	643	1057	1542	1534
10 x 30	193	864	228	1067	421	2053	3024	4097
15 x 15	147	488	216	660	697	1022	1817	2902

TAB. 3.12 – Temps de calcul moyen (en secondes) pour $TS\epsilon$ et AG_{dir}

Pour les temps de calcul, on constate que AG_{dir} est toujours plus lent que $TS\epsilon$. On peut tout de même remarquer que les temps de calcul restent dans des ordres de grandeur similaires.

3.5.4.5 Comparaison de $TS\epsilon$ et de Mem

Dans cette partie on compare $TS\epsilon$ avec l'algorithme Mémétique Mem. Les tableaux 3.13 et 3.14 représentent les performances entre les deux algorithmes et le tableau 3.15 donne les temps de calcul.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	0	50%	0	15%	0	29%	0	11%
5 x 15	1	53%	0	22%	0	19%	0	24%
5 x 20	0	34%	0	19%	0	22%	0	35%
10 x 10	0	15%	0	28%	2	55%	1	32%
10 x 15	2	52%	2	49%	2	63%	0	72%
10 x 20	0	10%	2	46%	3	84%	2	75%
10 x 30	0	10%	1	20%	0	9%	0	23%
15 x 15	1	23%	2	43%	3	82%	1	54%

TAB. 3.13 – Comparaison de $TS\epsilon$ contre Mem

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	3	100%	4	95%	0	71%	2	89%
5 x 15	3	88%	4	93%	1	81%	1	76%
5 x 20	4	100%	3	88%	1	78%	0	65%
10 x 10	4	95%	2	75%	1	45%	3	75%
10 x 15	2	58%	1	51%	0	37%	0	28%
10 x 20	4	90%	2	54%	0	16%	0	25%
10 x 30	4	90%	4	80%	3	91%	2	77%
15 x 15	3	77%	2	57%	0	18%	0	46%

TAB. 3.14 – Comparaison de Mem contre $TS\epsilon$

Mem est globalement plus performant que $TS\epsilon$, ce qui n'est pas surprenant dans la mesure où Mem utilise $TS\epsilon$ en tant que mutation. Ce qui est plus surprenant, c'est que $TS\epsilon$ arrive pour 25 instances à dominer Mem. Cela nous fait dire que dans certains cas, Mem passe beaucoup de temps à diversifier les solutions et aussi que $TS\epsilon$ peut rester dans un extremum local lorsque la méthode est appelée depuis une solution a priori déjà bonne.

Au niveau des temps de calcul, on constate que Mem atteint toujours sa durée maximale. Si pour les grandes instances, $CPU(Mem) > 600s$, c'est qu'une mutation a été appelée avant la limite des 10 minutes et que dans ce cas, il faut attendre la fin de $TS\epsilon$.

On peut donc en conclure qu'en règle générale, Mem est plus performant que $TS\epsilon$.

	sdata		edata		rdata		vdata	
CPU	$TS\epsilon$	Mem	$TS\epsilon$	Mem	$TS\epsilon$	Mem	$TS\epsilon$	Mem
5 x 10	17	602	16	602	51	603	65	603
5 x 15	27	606	39	603	119	605	184	608
5 x 20	40	613	94	608	180	615	246	618
10 x 10	27	606	50	604	204	640	57	607
10 x 15	99	614	131	614	200	651	715	721
10 x 20	157	637	114	673	643	725	1542	1012
10 x 30	193	679	228	734	421	941	3024	1086
15 x 15	147	638	216	648	697	709	1817	1007

TAB. 3.15 – Temps de calcul moyen (en secondes) pour $TS\epsilon$ et Mem

3.5.4.6 Comparaison de $AG_{dir}r$ et de Mem

Dans cette partie on compare $AG_{dir}r$ sans initialisation par la Tabou avec Mem. Les tableaux 3.16 et 3.17 représentent les performances entre les deux algorithmes et le tableau 3.18 donne les temps de calcul.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	1	74%	0	53%	0	53%	0	20%
5 x 15	0	30%	1	37%	0	5%	0	17%
5 x 20	0	39%	0	35%	0	13%	0	12%
10 x 10	1	38%	2	54%	1	33%	1	48%
10 x 15	0	17%	0	22%	0	0%	0	0%
10 x 20	0	53%	0	25%	0	8%	0	0%
10 x 30	0	45%	0	10%	0	0%	0	0%
15 x 15	0	24%	0	40%	0	0%	0	0%

TAB. 3.16 – Comparaison de $AG_{dir}r$ contre Mem

On constate une forte dominance de Mem sur $AG_{dir}r$, surtout sur les grandes instances avec une forte flexibilité (`rdata` et `vdata`). Pour les instances avec une faible flexibilité (`sdata` et `edata`), on peut remarquer que les méthodes sont globalement plus équilibrées. On peut en conclure que l'apport de la recherche Tabou permet de changer l'affectation de façon plus efficace.

En ce qui concerne les temps de calcul, on remarque que Mem atteint toujours la limite des 10 minutes et que $AG_{dir}r$ atteint cette limite dès que la taille du problème devient importante.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	2	65%	1	78%	0	47%	3	80%
5 x 15	3	87%	1	63%	4	95%	2	83%
5 x 20	1	75%	0	65%	0	87%	1	88%
10 x 10	1	62%	1	46%	2	68%	0	52%
10 x 15	3	83%	2	78%	**5**	**100%**	**5**	**100%**
10 x 20	0	47%	2	75%	4	92%	**5**	**100%**
10 x 30	1	55%	4	90%	**5**	**100%**	**5**	**100%**
15 x 15	2	76%	2	60%	**5**	**100%**	**5**	**100%**

TAB. 3.17 – Comparaison de Mem contre $AG_{dir}r$

	sdata		edata		rdata		vdata	
CPU	$AG_{dir}r$	Mem	$AG_{dir}r$	Mem	$AG_{dir}r$	Mem	$AG_{dir}r$	Mem
5 x 10	70	602	216	602	370	603	434	603
5 x 15	116	606	269	603	525	605	524	608
5 x 20	325	613	459	608	533	615	600	618
10 x 10	224	606	425	604	601	640	288	607
10 x 15	508	614	600	614	561	651	576	721
10 x 20	581	637	603	653	603	725	602	1012
10 x 30	607	679	607	734	607	941	607	1086
15 x 15	604	638	604	648	603	709	603	1007

TAB. 3.18 – Temps de calcul moyen (en secondes) pour $AG_{dir}r$ et Mem

3.5.4.7 Comparaison de AG_{dir} et de Mem

Dans cette partie on compare AG_{dir} avec initialisation par $TS\ell c$ avec Mem, cela nous permet de voir quel est l'impact de $TS\epsilon$ à la place de la mutation. Les tableaux 3.19 et 3.20 représentent les performances entre les deux algorithmes et le tableau 3.21 donne les temps de calcul.

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	1	64%	1	82%	1	68%	0	25%
5 x 15	2	78%	2	81%	0	45%	0	33%
5 x 20	1	72%	1	71%	0	61%	0	80%
10 x 10	1	82%	3	69%	3	88%	2	87%
10 x 15	2	84%	4	80%	1	60%	1	70%
10 x 20	2	53%	2	71%	2	85%	2	84%
10 x 30	2	60%	3	83%	1	50%	0	58%
15 x 15	1	35%	3	83%	5	100%	3	90%

TAB. 3.19 – Comparaison de AG_{dir} contre Mem

	sdata		edata		rdata		vdata	
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$
5 x 10	3	80%	0	79%	0	32%	0	75%
5 x 15	2	65%	0	46%	1	55%	0	67%
5 x 20	1	83%	1	59%	0	44%	0	20%
10 x 10	0	53%	1	35%	0	14%	0	40%
10 x 15	0	36%	1	20%	0	40%	0	30%
10 x 20	1	47%	0	33%	0	15%	0	16%
10 x 30	1	40%	0	17%	0	50%	0	42%
15 x 15	2	66%	0	27%	0	0%	0	10%

TAB. 3.20 – Comparaison de Mem contre AG_{dir}

Si on regarde l'indicateur SOw des tableaux 3.19 et 3.20, on se rend compte que pour beaucoup d'instance les fronts se croisent. A l'inverse, si on s'intéresse à $\overline{NFC}$, on constate un net avantage pour AG_{dir}.

En ce qui concerne les temps de calcul, pour les petites instances AG_{dir} trouve bien plus rapidement ses solutions. Par contre sur les grandes instances, le temps nécessaire à $TS\ell c$ pour initialiser la population pèse sur les performances.

3.5.4.8 Conclusion sur les comparaisons des méthodes entre-elles

De ces expérimentations, on peut dire qu'aucune méthode ne domine strictement toutes les autres. On peut cependant remarquer que l'algorithme génétique à codage direct AG_{dir} avec initialisation par la recherche Tabou $TS\ell c$ donne en règle générale les meilleurs résultats. En deuxième position vient l'algorithme Mem, puis $TS\ell c$.

	sdata		edata		rdata		vdata	
CPU	AG_{dir}	Mem	AG_{dir}	Mem	AG_{dir}	Mem	AG_{dir}	Mem
5 x 10	94	602	106	602	495	603	216	603
5 x 15	110	606	225	603	538	605	546	608
5 x 20	174	613	416	608	645	615	665	618
10 x 10	280	606	333	604	583	640	292	607
10 x 15	311	614	298	614	448	651	949	721
10 x 20	514	637	515	653	1057	725	1534	1012
10 x 30	864	679	1067	734	2053	941	4097	1086
15 x 15	488	638	660	648	1022	709	2902	1007

TAB. 3.21 – Temps de calcul moyen (en secondes) pour AG_{dir} et Mem

3.5.5 Comparaison avec un front de référence

Après avoir comparé les méthodes deux-à-deux, nous comparons nos algorithmes à un front de référence. Pour une instance donnée, on construit le front de référence F_r en agrégeant dans un ensemble toutes les solutions trouvées par les cinq méthodes ($TS\ell c$, $TS\epsilon$, AG_{ind}, AG_{dir} et Mem), puis de cet ensemble, on ne garde que les solutions non-dominées. Ainsi, F_r contient les meilleures solutions trouvées par nos méthodes.

Les tableau 3.22 à 3.26 montrent le détail des performances de nos algorithmes par rapport au front de référence.

	d_{fr}		$\overline{NFC}$	d_{fr}		$\overline{NFC}$
	Moyenne	Ecart type		Moyenne	Ecart type	
	sdata			edata		
5 x 10	27.0	33.6	12%	29.3	19.1	21%
5 x 15	24.9	31.6	39%	19.4	10.7	6%
5 x 20	35.2	21.4	12%	13.1	5.9	7%
10 x 10	35.2	40.7	0%	30.5	19.4	0%
10 x 15	34.5	52.8	23%	21.7	16.5	11%
10 x 20	16.5	13.0	4%	6.8	4.3	12%
10 x 30	36.0	46.5	0%	11.3	3.6	0%
15 x 15	28.2	13.7	0%	30.6	29.8	3%
	rdata			vdata		
5 x 10	18.0	17.3	3%	9.3	5.4	0%
5 x 15	13.2	12.7	4%	8.0	5.8	5%
5 x 20	4.7	3.3	5%	4.1	2.8	6%
10 x 10	26.5	24.4	29%	31.2	41.6	3%
10 x 15	22.3	23.3	21%	10.6	9.8	19%
10 x 20	22.6	7.0	2%	18.6	11.1	30%
10 x 30	17.6	8.7	19%	13.2	11.7	31%
15 x 15	12.4	13.7	24%	5.7	3.9	34%

TAB. 3.22 – Comparaison de $TS\ell c$ au front de référence

	d_{fr}		$\overline{NFC}$	d_{fr}		$\overline{NFC}$
	Moyenne	Ecart type		Moyenne	Ecart type	
	sdata			edata		
5 x 10	19.8	15.7	17%	9.6	6.6	14%
5 x 15	7.5	8.5	**46%**	9.9	11.7	12%
5 x 20	21.9	18.4	13%	11.9	7.6	17%
10 x 10	22.5	16.2	5%	17.9	16.7	21%
10 x 15	18.9	27.1	18%	21.7	28.6	13%
10 x 20	29.5	36.3	4%	13.1	18.7	4%
10 x 30	31.1	47.8	0%	17.8	10.7	0%
15 x 15	31.8	30.3	19%	25.3	21.9	17%
	rdata			vdata		
5 x 10	5.8	3.8	5%	10.2	4.3	3%
5 x 15	15.5	16.0	13%	10.6	12.4	19%
5 x 20	4.0	1.8	16%	7.2	2.5	12%
10 x 10	18.8	15.1	16%	22.3	16.5	8%
10 x 15	9.8	17.3	**45%**	10.9	8.1	**55%**
10 x 20	2.1	3.0	37%	6.1	7.9	**58%**
10 x 30	17.6	9.4	3%	10.7	7.0	12%
15 x 15	35.9	31.3	10%	2.5	3.8	27%

TAB. 3.23 – Comparaison de $TS\epsilon$ au front de référence

	d_{fr}		$\overline{NFC}$	d_{fr}		$\overline{NFC}$
	Moyenne	Ecart type		Moyenne	Ecart type	
	sdata			edata		
5 x 10	6.8	9.5	**73%**	12.9	10.3	**42%**
5 x 15	31.1	33.1	34%	15.5	11.8	33%
5 x 20	16.2	28.6	**66%**	36.8	37.2	27%
10 x 10	14.3	15.6	**44%**	26.9	28.5	33%
10 x 15	42.1	36.5	13%	69.6	57.4	0%
10 x 20	19.3	21.2	**45%**	30.0	32.1	**43%**
10 x 30	14.9	12.4	**43%**	40.6	90.7	**54%**
15 x 15	21.3	26.5	30%	7.7	13.8	**42%**
	rdata			vdata		
5 x 10	30.9	23.4	0%	13.1	8.8	9%
5 x 15	36.7	20.3	0%	18.5	11.3	2%
5 x 20	16.4	15.6	6%	14.4	10.4	9%
10 x 10	23.6	17.7	10%	14.8	15.6	**40%**
10 x 15	66.8	38.4	0%	14.4	10.7	5%
10 x 20	28.6	26.7	12%	116.4	60.4	0%
10 x 30	173.8	107.2	4%	201.8	68.2	0%
15 x 15	22.9	28.5	**41%**	126.6	42.5	0%

TAB. 3.24 – Comparaison de AG_{ind} au front de référence

	d_{fr}		$\overline{NFC}$	d_{fr}		$\overline{NFC}$
	Moyenne	Ecart type		Moyenne	Ecart type	
	sdata			edata		
5 x 10	15.9	24.3	21%	5.5	12.3	**62%**
5 x 15	10.2	14.3	**62%**	6.0	12.4	**65%**
5 x 20	37.0	54.9	**41%**	8.2	10.5	**53%**
10 x 10	10.9	13.2	**46%**	16.8	22.0	33%
10 x 15	0.5	1.2	**63%**	1.2	1.7	**71%**
10 x 20	14.3	15.6	22%	1.1	2.2	**40%**
10 x 30	53.1	110.1	39%	10.2	13.7	31%
15 x 15	16.8	17.3	24%	31.0	28.3	38%
	rdata			vdata		
5 x 10	3.5	5.2	**68%**	5.9	2.9	18%
5 x 15	4.8	5.3	38%	6.9	3.7	23%
5 x 20	8.6	11.5	**44%**	8.2	13.7	**64%**
10 x 10	4.5	6.9	**68%**	10.9	13.2	**45%**
10 x 15	16.7	19.3	**40%**	2.3	3.0	33%
10 x 20	8.8	9.9	**43%**	17.7	20.1	35%
10 x 30	9.8	11.2	**48%**	40.4	41.2	**51%**
15 x 15	10.6	21.5	**49%**	3.3	7.4	**70%**

TAB. 3.25 – Comparaison de AG_{dir} au front de référence

	d_{fr}		$\overline{NFC}$	d_{fr}		$\overline{NFC}$
	Moyenne	Ecart type		Moyenne	Ecart type	
	sdata			edata		
5 x 10	9.3	17.6	**47%**	2.8	3.3	**55%**
5 x 15	6.5	11.4	**59%**	3.4	2.1	39%
5 x 20	13.2	15.6	**44%**	5.0	7.8	**46%**
10 x 10	11.6	14.0	27%	15.1	11.9	16%
10 x 15	8.9	7.1	36%	30.7	23.8	16%
10 x 20	22.1	32.8	29%	16.7	20.0	15%
10 x 30	54.2	102.6	18%	21.2	14.5	14%
15 x 15	21.4	31.0	28%	30.9	59.4	15%
	rdata			vdata		
5 x 10	6.4	1.8	26%	1.4	1.9	**70%**
5 x 15	5.8	6.7	**49%**	2.1	2.7	**56%**
5 x 20	4.5	3.5	38%	6.6	2.5	16%
10 x 10	32.0	20.9	8%	14.7	13.5	18%
10 x 15	36.8	25.5	15%	26.8	8.1	7%
10 x 20	28.6	16.0	8%	35.1	32.7	7%
10 x 30	15.7	17.8	**45%**	11.5	9.8	37%
15 x 15	69.4	44.6	0%	56.4	33.4	3%

TAB. 3.26 – Comparaison de Mem au front de référence

Quelle que soit la méthode, on se rend compte que l'écart type de l'indicateur d_{fr} est de l'ordre de grandeur de sa moyenne, ce qui signifie que la distance au front de référence est très variable. Si on regarde le tableau 3.22, on peut remarquer que $TS\ell c$ est relativement performante sur les grosses instances (`vdata`, $m \geq 10; n \geq 15$). On peut donc en déduire que $TS\ell c$ arrive à trouver de bonnes solutions quand l'espace des solutions est important.

$TS\epsilon$ présente des résultats assez homogènes (cf. tableau 3.23). AG_{ind} présente de très bonnes performances pour les instances avec pas ou peu de flexibilité (`sdata` et `edata`). Notons que AG_{ind} participe à hauteur de 73% aux fronts de référence pour les instances `sdata` de taille $m = 5; n = 10$. A l'inverse, sur les instances de `rdata` et `vdata`, ses performances sont très mauvaises avec l'indicateur $\overline{NFC}$ valant souvent zéro et une distance au front de référence importante ($\overline{d_{fr}} > 100$).

Le tableau 3.25 montre que AG_{dir} présente des performance homogènes au regard de la taille et de la flexibilité de l'instance. De plus, ces performances sont très bonnes avec une distance moyenne au front de référence souvent inférieure à 10 et pour 19 classes d'instances on a $\overline{NFC} \geq 40\%$. Notons tout de même que AG_{dir} semble moins efficace pour les instances de job shop classique (`sdata`).

Le tableau 3.26 présente les performances de Mem par rapport au front de référence. On peut constater que Mem est plus efficace sur les petites instances ($m = 5$). L'augmentation de la flexibilité impacte peu sur les performances de Mem.

Le tableau 3.27 donne un récapitulatif des cinq méthodes par rapport au front de référence.

	d_{fr}		$\overline{NFC}$		
	Moyenne	Ecart type	min	moy	max
$TS\ell c$	20.7	22.9	0%	12%	39%
$TS\epsilon$	16.1	**19.4**	0%	17%	58%
AG_{ind}	42.5	58.1	0%	23%	73%
AG_{dir}	**12.2**	26.5	18%	**46%**	70%
Mem	21.1	31.4	0%	28%	70%

TAB. 3.27 – Récapitulatif au front de référence

On constate que la meilleure méthode est AG_{dir} tant au niveau de la distance moyenne avec le front de référence qu'à sa contribution à ce même front. Notons tout de même que $TS\epsilon$ est la méthode la plus stable : $\overline{d_{fr}} = 16.1$ et écart type = 19.4.

3.5.6 Comparaison avec la recherche Tabou de Dauzère-Pérès et Paulli

Dans cette partie nous comparons nos méthodes avec la recherche Tabou proposée par [Dauzère-Pérès et Paulli, 1997], que nous nommerons par la suite DPP. Cette méthode est mono-critère et n'optimise que le *makespan*. Pour nous comparer, nous extrayons des résultats de nos méthodes les meilleures valeurs du *makespan*. Pour $TS\ell c$ cela revient à considérer $\alpha = 1$ et $\beta = 0$. Pour $TS\epsilon$, cela revient à fixer $\epsilon = \infty$.

Les tableaux 3.30 à 3.33 donnent les valeurs du *makespan* trouvées par les différentes méthodes, la colonne Δ_r correspond à l'écart relatif entre la meilleure de nos cinq méthodes et DPP :

$$\Delta_r = \frac{\min(C_{max}(TS\ell c); C_{max}(TS\epsilon); C_{max}(AG_{ind}); C_{max}(AG_{dir}); C_{max}(Mem)) - C_{max}(DPP)}{\min(C_{max}(TS\ell c); C_{max}(TS\epsilon); C_{max}(AG_{ind}); C_{max}(AG_{dir}); C_{max}(Mem))}$$

Les valeurs en gras indiquent que l'algorithme correspondant a trouvé le meilleur C_{max} parmi nos cinq méthodes. Les valeurs sont en gras souligné lorsque la valeur est aussi celle retournée par DPP.

On peut remarquer que dans dans le pire des cas, l'écart relatif avec DPP est égal à 15.5%. Pour 21 instances (sur les 160), on trouve la même valeur que DPP. L'écart type moyen avec DPP est de 4.02% pour `sdata`, 4.97% pour `edata`, 6.24% pour `rdata` et 4.24% pour `vdata`.

Le tableau 3.28 donne le nombre de fois où chaque méthode a trouvé la meilleure valeur (valeur en gras dans les tableaux 3.30 à 3.33).

	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem
`sdata`	5	5	28	18	9
`edata`	1	4	19	17	6
`rdata`	1	10	7	18	7
`vdata`	1	19	6	9	11
Total	8	38	60	62	33

TAB. 3.28 – Nombre de fois où algorithme trouve la meilleure valeur pour le C_{max} seul

Le tableau 3.29 donne le nombre de fois où chaque méthode a trouvé la même valeur que DPP (valeur en souligné dans les tableaux 3.30 à 3.33).

	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem
`sdata`	3	4	9	9	7
`edata`	1	1	1	6	3
`rdata`	0	0	0	0	0
`vdata`	1	2	2	3	2
Total	5	7	12	18	12

TAB. 3.29 – Nombre de fois où algorithme trouve la même valeur que DPP pour le C_{max} seul

On peut constater que AG_{dir} présente le plus grand nombre de meilleures valeurs et aussi une bonne homogénéité. Cependant, pour les instances avec peu de flexibilité, AG_{ind} présente de meilleures performances. Pour les instances de `rdata`, c'est AG_{dir} qui présente la meilleure performance et pour `vdata`, c'est $TS\epsilon$ qui donne les meilleurs résultats.

	DPP	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem	Δ_r
la01	666	715	785	**677**	715	698	1.6%
la02	655	688	678	677	**655**	**655**	0.0%
la03	603	654	694	**625**	654	654	3.5%
la04	590	673	673	**595**	651	673	0.8%
la05	593	**593**	**593**	**593**	**593**	**593**	0.0%
la06	926	**926**	**926**	**926**	**926**	**926**	0.0%
la07	890	967	967	**892**	949	965	0.2%
la08	863	894	881	**863**	**863**	906	0.0%
la09	951	**951**	**951**	**951**	**951**	**951**	0.0%
la10	958	980	**958**	**958**	**958**	**958**	0.0%
la11	1222	1288	1288	**1222**	**1222**	1227	0.0%
la12	1039	1054	1054	**1039**	**1039**	1054	0.0%
la13	1150	1182	1182	**1163**	1182	1182	1.1%
la14	1292	1351	1351	**1292**	**1292**	**1292**	0.0%
la15	1207	1450	1435	**1207**	1354	**1207**	0.0%
la16	959	1007	1007	1008	**1004**	1007	4.5%
la17	785	871	895	834	**828**	875	5.2%
la18	861	942	942	**877**	942	942	1.8%
la19	850	908	926	901	874	**869**	2.2%
la20	902	950	998	963	**907**	954	0.6%
la21	1057	1185	**1123**	1149	1158	1213	5.9%
la22	948	1123	1195	1089	**1020**	1036	7.1%
la23	1032	**1040**	1055	1076	**1040**	**1040**	0.8%
la24	946	1106	1097	1078	**1073**	1079	11.8%
la25	992	1128	1117	**1089**	1107	1131	8.9%
la26	1218	1603	1620	**1370**	1445	1487	11.1%
la27	1269	1603	1585	**1423**	1589	1615	10.8%
la28	1237	1450	1429	**1345**	1361	1400	8.0%
la29	1215	1514	1520	**1314**	1441	1402	7.5%
la30	1355	1619	1611	**1468**	1522	1546	7.7%
la31	1784	2056	2189	**1825**	1972	2150	2.2%
la32	1850	1997	2007	1950	**1906**	1994	2.9%
la33	1719	1983	1981	**1889**	1919	1929	9.0%
la34	1721	2024	2046	**1863**	1923	1914	7.6%
la35	1888	2380	2380	**1919**	2145	2355	1.6%
la36	1313	1427	1590	1408	**1382**	1512	5.0%
la37	1456	1737	1742	**1553**	1737	1736	6.2%
la38	1254	**1357**	1362	1370	1357	1380	7.6%
la39	1245	1512	1515	**1427**	1510	1499	12.8%
la40	1242	1400	1400	**1303**	1383	1326	4.7%

TAB. 3.30 – Comparaison avec DPP pour `sdata`

	DPP	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem	Δ_r
la01	609	676	707	624	**610**	617	0.2%
la02	655	684	684	656	**655**	**655**	0.0%
la03	554	595	609	580	**569**	570	2.6%
la04	568	606	644	591	**590**	602	3.7%
la05	503	517	543	513	**512**	**512**	1.8%
la06	833	875	857	835	857	**833**	0.0%
la07	765	808	814	800	**765**	803	0.0%
la08	845	912	899	**860**	881	912	1.7%
la09	878	888	897	912	**878**	**878**	0.0%
la10	866	885	897	**866**	**866**	881	0.0%
la11	1103	1235	1242	**1106**	1151	1171	0.3%
la12	960	**960**	**960**	982	**960**	980	0.0%
la13	1053	1123	1138	**1066**	1090	1073	1.2%
la14	1123	1319	1179	**1137**	**1137**	1163	1.2%
la15	1111	1257	1251	**1153**	1175	1202	3.6%
la16	915	926	954	949	**915**	974	0.0%
la17	707	744	770	741	717	**715**	1.1%
la18	843	922	**871**	925	895	894	3.2%
la19	796	930	906	**830**	862	890	4.1%
la20	864	919	978	917	**886**	949	2.5%
la21	1046	1133	1112	1151	**1102**	1123	5.1%
la22	890	1085	**957**	1006	983	1036	7.0%
la23	953	1067	1076	1076	**1052**	1068	9.4%
la24	918	1040	1045	1055	**1023**	1091	10.3%
la25	955	1024	1042	1055	1020	**1006**	5.1%
la26	1138	1319	1337	**1292**	1297	1373	11.9%
la27	1215	1369	1451	**1344**	1360	1380	9.6%
la28	1169	1527	1555	**1291**	1527	1437	9.5%
la29	1157	1537	1448	**1243**	1537	1388	6.9%
la30	1225	1528	1522	**1326**	1380	1555	7.6%
la31	1556	1806	1850	1729	**1694**	1758	8.1%
la32	1698	1921	1985	**1850**	1862	1933	8.2%
la33	1547	1849	1847	**1692**	1767	1811	8.6%
la34	1623	1977	1965	**1784**	1905	1977	9.0%
la35	1736	2265	2282	**1872**	2073	2278	7.3%
la36	1171	1378	1356	**1281**	1305	1452	8.6%
la37	1418	1798	1817	**1512**	1745	1789	6.2%
la38	1172	1316	**1301**	1306	1316	1304	9.9%
la39	1207	1497	1416	**1381**	1492	1492	12.6%
la40	1150	1337	1340	1293	**1284**	1313	10.4%

TAB. 3.31 – Comparaison avec DPP pour `edata`

	DPP	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem	Δ_r
la01	574	586	586	610	**580**	608	1.0%
la02	532	549	559	555	**542**	564	1.8%
la03	479	499	502	508	**497**	500	3.6%
la04	504	528	534	527	518	**507**	0.6%
la05	458	473	471	480	**463**	477	1.1%
la06	800	827	818	819	**804**	809	0.5%
la07	750	**752**	764	761	**752**	754	0.3%
la08	767	777	771	782	**769**	770	0.3%
la09	854	888	**855**	865	863	885	0.1%
la10	805	827	**806**	816	808	**806**	0.1%
la11	1072	1095	1085	1081	**1076**	1081	0.4%
la12	936	971	948	949	947	**944**	0.8%
la13	1038	1092	1095	1051	**1042**	1053	0.4%
la14	1070	1101	1092	**1075**	1083	1083	0.5%
la15	1090	1153	1153	**1105**	1114	1115	1.4%
la16	717	810	822	791	**779**	802	8.0%
la17	646	725	762	703	**670**	719	3.6%
la18	669	711	**710**	712	711	732	5.8%
la19	703	769	756	779	**739**	803	4.9%
la20	756	857	833	**795**	806	854	4.9%
la21	846	934	956	997	934	**911**	7.1%
la22	772	913	**897**	936	899	937	13.9%
la23	853	928	**910**	960	925	914	6.3%
la24	820	924	895	975	**891**	906	8.0%
la25	802	941	**932**	948	941	958	13.9%
la26	1070	1196	1233	1218	1194	**1157**	7.5%
la27	1100	1276	**1193**	1234	1216	1255	7.8%
la28	1085	1261	1192	**1190**	1218	1222	8.8%
la29	1004	1245	1179	**1132**	1211	1200	11.3%
la30	1089	1280	1290	1292	**1247**	1292	12.7%
la31	1528	1761	**1693**	1730	1745	1696	9.7%
la32	1660	1787	1827	1870	1772	**1765**	5.9%
la33	1501	1649	1685	1705	1643	**1617**	7.2%
la34	1539	1786	1905	1755	**1753**	1791	12.2%
la35	1555	1735	1752	1761	**1712**	1723	9.2%
la36	1030	1234	1230	**1177**	1214	1249	12.5%
la37	1082	1384	**1281**	1322	1360	1388	15.5%
la38	989	1202	1165	**1151**	1181	1200	14.1%
la39	1024	1208	**1185**	1217	**1185**	1229	13.6%
la40	980	1135	1129	1154	**1118**	1173	12.3%

TAB. 3.32 – Comparaison avec DPP pour `rdata`

	DPP	$TS\ell c$	$TS\epsilon$	AG_{ind}	AG_{dir}	Mem	Δ_r
la01	572	580	581	606	576	**572**	0.0%
la02	529	545	549	549	542	**540**	2.0%
la03	479	501	488	502	488	**481**	0.4%
la04	503	519	508	521	510	**506**	0.6%
la05	460	472	470	481	**469**	473	1.9%
la06	800	831	817	825	**809**	829	1.1%
la07	750	759	763	767	759	**754**	0.5%
la08	766	779	770	**768**	772	770	0.3%
la09	853	861	**855**	864	859	856	0.2%
la10	805	815	**808**	813	809	831	0.4%
la11	1071	1114	1101	**1079**	1081	1091	0.7%
la12	936	951	946	**939**	941	945	0.3%
la13	1038	1084	1123	1041	**1040**	1041	0.2%
la14	1070	1108	**1071**	1075	1074	1084	0.1%
la15	1089	1121	1096	**1093**	1094	1104	0.4%
la16	717	734	**720**	767	734	743	0.4%
la17	646	659	**646**	663	**646**	652	0.0%
la18	663	667	670	**663**	**663**	676	0.0%
la19	617	677	680	675	**667**	679	7.5%
la20	756	**756**	**756**	**756**	**756**	**756**	0.0%
la21	814	952	**878**	930	922	930	7.3%
la22	744	872	828	843	840	**817**	8.9%
la23	818	896	**875**	941	881	908	6.5%
la24	784	921	**854**	874	920	894	8.2%
la25	757	866	**850**	851	859	875	10.9%
la26	1056	1173	**1162**	1166	1173	1181	9.1%
la27	1087	1170	**1119**	1213	1152	1151	2.9%
la28	1072	1171	**1133**	1214	1161	1145	5.4%
la29	997	1165	**1108**	1121	1124	1165	10.0%
la30	1071	1141	**1116**	1261	1133	1162	4.0%
la31	1521	1655	1658	1765	1641	**1605**	5.2%
la32	1659	1744	1761	1883	**1721**	1726	3.6%
la33	1498	1672	1805	1726	1656	**1627**	7.9%
la34	1537	1711	**1654**	1752	1708	1723	7.1%
la35	1551	1665	1664	1788	1649	**1643**	5.6%
la36	948	1048	1057	1156	1033	**1027**	7.7%
la37	986	1156	**1131**	1261	1156	1193	12.8%
la38	943	1073	**1028**	1104	1043	1062	8.3%
la39	922	1124	**1067**	1161	1095	1092	13.6%
la40	955	1036	1069	1161	**1031**	1130	7.4%

TAB. 3.33 – Comparaison avec DPP pour `vdata`

3.5.7 Expérimentation sur les critères du *makespan* et de la somme des retards

Selon les jeux de données, la conflictualité des critères du *makespan* et du plus grand retard est parfois mise en doute. En effet, si les dates de fin souhaitées des travaux sont trop proches les unes des autres, voire identiques dans le cas extrême, alors optimiser le L_{max} revient presque à optimiser le C_{max}.

Afin de tester nos algorithmes sur des critères plus conflictuels, nous avons relancé des expérimentations sur les critères C_{max} et ΣT. Les tests ont été réalisés avec les algorithmes $TS\epsilon$, $TS\ell c$, AG_{dir}, et Mem. Les méthodes AG_{ind} et $AG_{dir}r$ n'ont pas été testées en raison de leurs faibles performances lors des expérimentations précédentes.

Pour évaluer la différence de conflictualité entre les combinaisons C_{max}/L_{max} et $C_{max}/\Sigma T$, nous comparons le nombre de solutions trouvées et la diversité des fronts entre les résultats obtenus par les algorithmes C_{max}/L_{max} et $C_{max}/\Sigma T$.

Les tableaux 3.34 à 3.37 montrent la différence moyenne en terme de nombre de solutions trouvées. La colonne $\overline{\Delta ndSol}$ correspond à la moyenne de la différence du nombre de solutions trouvées entre l'optimisation de $C_{max}/\Sigma T$ et l'optimisation de C_{max}/L_{max}. Plus formellement :

$$\overline{\Delta ndSol} = \frac{1}{5}\sum_{i=1}^{5}\Big(nbSol_i(C_{max}/\Sigma T) - nbSol_i(C_{max}/L_{max})\Big)$$

Avec $nbSol_i(C_{max}/\Sigma T)$ (resp. $nbSol_i(C_{max}/L_{max})$) le nombre de solutions non-dominées trouvées lors de l'optimisation des critères $C_{max}/\Sigma T$ (resp. C_{max}/L_{max}) pour la $i^{\text{éme}}$ instance. La colonne $\sigma(\Delta ndSol)$ correspond à l'écart type.

Les tableaux 3.38 à 3.41 montrent la diversité moyenne obtenue par nos quatre algorithmes entre l'optimisation de $C_{max}/\Sigma T$ ($\overline{divA}$) et l'optimisation de C_{max}/L_{max} ($\overline{divB}$).

Le tableau 3.42 récapitule la valeur des indicateurs pour chaque algorithme.

	sdata		edata		rdata		vdata	
	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$
5 x 10	1.2	1.3	-0.4	1.7	0.2	2.6	-0.8	1.1
5 x 15	1.0	2.5	-0.4	0.9	0	2.9	-1.0	2.0
5 x 20	**3.4**	2.3	1.8	1.3	1.8	0.8	1.2	2.5
10 x 10	-1.4	3.8	-0.8	3.2	-0.8	1.3	-0.6	3.5
10 x 15	-0.2	1.8	0.4	2.7	1.2	2.8	1.4	3.1
10 x 20	0	1.2	1.0	1.7	0.4	2.2	0.8	3.0
10 x 30	0	1.2	0.8	1.6	-1.2	3.1	0.6	1.9
15 x 15	0.4	2.3	0.4	1.1	0.4	0.9	0.6	1.5

TAB. 3.34 – Différence du nombre de solutions trouvées pour $TS\ell c$

	sdata		edata		rdata		vdata	
	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$
5 x 10	1.6	0.9	1.6	2.2	0.4	4.1	-0.4	3.6
5 x 15	0.4	2.2	0.4	1.5	-0.8	5.6	**-7.6**	5.8
5 x 20	1.0	1.6	-0.2	3.8	-3.2	4.1	**-4.4**	6.7
10 x 10	-1.4	0.9	0.2	1.8	0.6	2.1	-1.6	1.1
10 x 15	0.6	3.3	-0.8	1.1	0.8	2.0	1.2	4.6
10 x 20	0	1.2	**2.4**	2.4	1.2	5.0	**2.6**	2.6
10 x 30	1.6	2.7	**2.2**	1.3	0.2	1.3	1.4	1.3
15 x 15	1.2	1.3	0.8	3.7	0.4	3.4	1.0	1.4

TAB. 3.35 – Différence du nombre de solutions trouvées pour $TS\epsilon$

	sdata		edata		rdata		vdata	
	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$
5 x 10	1.6	0.5	**5.0**	6.4	**-2.0**	5.4	2.2	5.3
5 x 15	1.0	3.8	**3.4**	3.8	-1.0	6.9	-4.2	9.3
5 x 20	**6.0**	5.1	**7.2**	7.3	**-7.0**	8.7	-6.2	4.7
10 x 10	-0.6	2.7	**2.4**	5.4	-1.0	1.2	0.2	3.1
10 x 15	**2.0**	3.1	0.8	2.9	0.8	3.3	1.2	3.3
10 x 20	**2.8**	4.9	**3.0**	3.8	1.8	3.4	2.0	2.0
10 x 30	1.4	2.1	**3.8**	3.2	**3.6**	3.2	0.8	3.5
15 x 15	**-2.0**	2.5	0.8	3.9	0.4	2.7	0.2	1.3

TAB. 3.36 – Différence du nombre de solutions trouvées pour AG_{dir}

	sdata		edata		rdata		vdata	
	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$	$\overline{\Delta ndSol}$	$\sigma(\Delta ndSol)$
5 x 10	**3.4**	3.6	**7.0**	2.6	1.0	2.6	1.0	8.5
5 x 15	0.4	1.8	**3.4**	3.3	**-2.6**	5.8	**-4.4**	5.0
5 x 20	**3.0**	2.2	**2.2**	2.3	**-6.0**	7.5	1.2	6.3
10 x 10	**-2.4**	5.2	**2.2**	3.0	0	2.7	-0.6	5.6
10 x 15	0.4	2.6	-1.6	3.2	1.0	3.2	-1.0	3.4
10 x 20	1.8	4.1	1.2	2.4	**2.4**	2.4	**2.0**	2.9
10 x 30	1.0	1.2	-0.6	2.4	-1.8	2.6	0.2	1.9
15 x 15	-0.4	3.4	1.8	1.5	**2.0**	3.1	-1.8	1.3

TAB. 3.37 – Différence du nombre de solutions trouvées pour Mem

	sdata		edata		rdata		vdata	
	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$
5 x 10	1.0	1.5	1.2	1.3	0.9	0.8	0.7	0.6
5 x 15	1.2	0.9	1.1	1.2	1.1	0.9	0.7	0.6
5 x 20	0.7	1.4	0.8	1.4	0.4	0.7	0.5	0.7
10 x 10	0.8	1.2	1.3	1.2	1.0	0.8	0.6	1.2
10 x 15	0.7	1.2	1.4	1.2	1.4	1.3	1.0	0.9
10 x 20	1.2	1.0	1.0	0.6	0.9	0.6	0.8	0.9
10 x 30	1.2	0.7	0.4	0.8	1.2	1.0	1.0	0.8
15 x 15	0.8	1.2	1.3	0.9	1.0	1.4	0.9	1.2

TAB. 3.38 – Distance de *crowding* moyenne pour $TS\ell c$

	sdata		edata		rdata		vdata	
	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$
5 x 10	1.0	1.3	1.2	1.0	1.0	0.6	0.7	0.7
5 x 15	1.0	0.9	1.4	0.8	0.6	0.6	0.9	0.4
5 x 20	1.3	1.7	0.6	1.1	0.6	0.4	0.5	0.4
10 x 10	0.9	1.6	1.6	1.4	1.1	1.1	1.1	1.5
10 x 15	1.1	0.9	1.2	1.4	1.2	1.2	0.6	0.6
10 x 20	1.2	1.0	0.7	0.4	1.2	1.2	0.8	1.0
10 x 30	0.9	0.4	1.1	0.7	0.5	0.6	0.9	0.6
15 x 15	1.0	1.6	0.7	1.4	1.1	0.7	1.3	0.6

TAB. 3.39 – Distance de *crowding* moyenne pour $TS\epsilon$

	sdata		edata		rdata		vdata	
	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$
5 x 10	1.3	1.3	0.7	1.5	0.5	0.5	0.5	0.7
5 x 15	0.9	1.3	0.6	1.1	0.7	0.4	0.4	0.3
5 x 20	0.6	1.2	0.4	0.8	0.4	0.2	0.3	0.2
10 x 10	0.6	1.0	1.0	1.4	1.2	1.2	0.6	1.0
10 x 15	0.5	1.2	1.2	1.4	1.1	1.3	0.9	1.2
10 x 20	1.1	1.0	1.2	1.0	0.5	0.6	0.7	1.3
10 x 30	0.8	0.8	0.8	1.4	0.6	1.1	0.9	0.8
15 x 15	1.4	1.2	1.2	0.9	1.2	1.3	1.6	1.2

TAB. 3.40 – Distance de *crowding* moyenne pour AG_{dir}

	sdata		edata		rdata		vdata	
	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$	$\overline{divA}$	$\overline{divB}$
5 x 10	0.8	1.0	0.5	1.1	0.6	0.6	0.6	0.4
5 x 15	1.2	1.4	0.6	0.8	0.6	0.5	0.4	0.3
5 x 20	0.6	1.0	0.6	0.8	0.3	0.2	0.3	0.3
10 x 10	0.9	1.4	0.6	1.4	0.7	1.1	0.8	1.7
10 x 15	0.7	1.3	1.6	1.1	0.8	1.1	0.8	0.6
10 x 20	0.7	0.9	1.3	1.2	0.6	0.9	0.6	1.0
10 x 30	1.5	1.4	1.5	1.1	1.1	1.0	1.1	1.0
15 x 15	1.4	0.8	1.4	0.7	0.9	1.1	0.8	1.3

TAB. 3.41 – Distance de *crowding* moyenne pour Mem

	$TS\ell c$	$TS\epsilon$	AG_{dir}	Mem
$\overline{\Delta ndSol}$	0.4	0.1	1.0	0.5
$\overline{divA}$	1.0	1.0	0.8	0.8
$\overline{divB}$	1.0	0.9	1.0	1.0

TAB. 3.42 – Récapitulatif pour la comparaison entre C_{max}/L_{max} et $C_{max}/\Sigma T$

On constate qu'en moyenne la différence du nombre de solutions trouvées est faible. A l'inverse l'écart type est souvent assez élevé, voire supérieur à la moyenne. En moyenne, on ne trouve pas plus d'une solution en plus entre C_{max}/L_{max} et $C_{max}/\Sigma T$. Quant à la diversité, que ce soit pour C_{max}/L_{max} ou $C_{max}/\Sigma T$, la distance de *crowding* moyenne se situe aux alentours de 1.

De ces informations, on peut en conclure que le degré de conflictualité entre l'optimisation de C_{max}/L_{max} et l'optimisation de $C_{max}/\Sigma T$ est du même ordre de grandeur.

3.6 Conclusion sur le problème de job shop flexible

Dans ce chapitre nous avons étudié le problème de job shop flexible multicritère. Si ce problème a déjà été étudié dans la littérature dans le cas monocritère, l'approche multicritère a été moins explorée. Pour résoudre ce problème nous avons proposé deux recherches Tabou, deux algorithmes génétiques et un algorithme mémétique.

Nos expérimentations nous ont permis de comparer nos algorithmes entre eux. Il en résulte que la meilleure méthode est notre algorithme génétique à codage direct avec une population partiellement initialisée par la recherche Tabou basée sur la combinaison linéaire des critères. Néanmoins, dans certains cas cette méthode est dominée par l'algorithme mémétique. Nous avons testé nos algorithmes sur deux combinaisons de critères, la première est l'optimisation du couple C_{max}/L_{max}, la seconde est l'optimisation de $C_{max}/\Sigma T$.

Chapitre 4

Job shop multiressource multicritère

Ce chapitre est consacré à l'étude de problèmes d'atelier pour lesquels une opération nécessite plusieurs ressources simultanément. Pour résoudre ce problème, nous avons adapté la recherche Tabou et l'algorithme génétique à codage direct.

4.1 Définition formelle du problème

Bien souvent dans les industries, notamment dans l'imprimerie, l'exécution d'une opération peut nécessiter plusieurs ressources simultanément. Par exemple, une opération d'impression nécessite une imprimante rotative et un opérateur humain. De plus, l'opération peut ne pas avoir besoin de toutes les ressources tout le temps. Ainsi dans notre exemple, l'opérateur humain n'est nécessaire qu'au début de l'opération pour « caler » l'imprimante. Une fois le calage effectué, l'opérateur peut aller s'occuper d'une autre machine.

On appelle ce type de problème un job shop multiressource. Nous rappelons ici les notations.

On considère un ensemble J composé de n travaux. Ces travaux doivent être ordonnancés sur m ressources disjonctives, l'ensemble des ressources est noté $\mathcal{R}$. On note R_k la $k^{ème}$ ressource. Chaque travail i est composé de n_i opérations. L'opération j du travail i est notée $O_{i,j}$. Les gammes ne sont pas nécessairement linéaires, c'est-à-dire qu'une opération peut avoir plusieurs successeurs (l'ensemble $\Gamma_{i,j}$) et plusieurs prédécesseurs (l'ensemble $\Gamma_{i,j}^{-1}$). On suppose que chaque travail n'a qu'une seule opération finale notée O_{i,n_i}. Chaque opération nécessite plusieurs ressources pour son exécution. On note $R_{i,j}^m$ l'ensemble des ressources devant exécuter l'opération $O_{i,j}$. On note $O_{i,j,k}$ la sous-opération de $O_{i,j}$ s'exécutant sur la ressource R_k et $p_{i,j,k}$ sa durée opératoire. On suppose donc que toutes les sous-opérations d'une opération n'ont pas nécessairement la même

durée. En revanche, on suppose que toutes les sous-opérations de $O_{i,j}$ débutent à la même date $t_{i,j}$. L'opération suivante dans la gamme ne peut commencer que quand la dernière sous-opération de l'opération est terminée. On note $\Gamma^{-1}_{i,j,k}$ l'ensemble des sous-opérations précédentes dans la gamme. A chaque travail i est associée une date due d_i. Il est facile de propager cette date due à toutes les opérations du travail i (cf. l'algorithme 6 page 40).

La figure 4.1 donne un exemple d'ordonnancement multiressource.

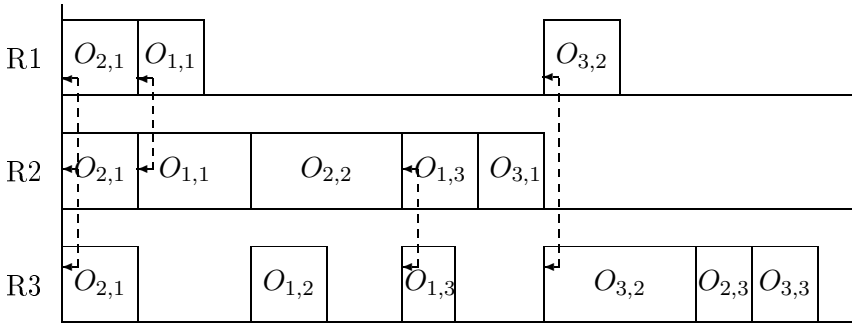

FIG. 4.1 – Exemple d'ordonnancement multiressource

4.2 Modélisation par un PLNE

A notre connaissance, il n'existe pas de modèle de programmation linéaire en nombres entiers pour le problème de job shop multiressource tel que nous l'avons défini. Nous proposons d'adapter le modèle de Manne ([Manne, 1960]).

Les variables de notre modèle sont :

$$t_{i,j} \quad : \quad \text{La date de début de } O_{i,j} \tag{4.1}$$

$$p_{i,j,k} \quad : \quad \text{La durée opératoire de } O_{i,j} \text{ sur } R_k \tag{4.2}$$

$$y^{a,b,k}_{c,d} \quad = \quad \begin{cases} 0 & \text{si } O_{a,b} \text{ est effectuée avant } O_{c,d} \text{ sur } R_k \\ 1 & \text{sinon} \end{cases} \tag{4.3}$$

Les contraintes sont :

$$t_{i,j} \geq 0 \qquad \forall i = 1, \ldots, n; \forall j = 1, \ldots, n_i \tag{4.4}$$

$$t_{i,j+1} \geq t_{i,j} + p_{i,j,k} \qquad \forall R_k \in R^m_{i,j}; \quad \forall i = 1, \ldots, n; \forall j = 1, \ldots, n_i - 1 \tag{4.5}$$

$$t_{c,d} \geq t_{a,b} + p_{a,b,k} - HV \times y^{a,b,k}_{c,d} \qquad \forall R_k \in R^m_{a,b} \cap R^m_{c,d}; \quad \forall a = 1, \ldots, n; \forall b = 1, \ldots, n_a; \quad \forall c = 1, \ldots, n; \forall d = 1, \ldots, n_c \tag{4.6}$$

$$t_{a,b} \geq t_{c,d} + p_{c,d,k} - HV \times (1 - y^{a,b,k}_{c,d}) \qquad \forall R_k \in R^m_{a,b} \cap R^m_{c,d}; \quad \forall a = 1, \ldots, n; \forall b = 1, \ldots, n_a; \quad \forall c = 1, \ldots, n; \forall d = 1, \ldots, n_c \tag{4.7}$$

Les contraintes 4.4 indiquent que les travaux ne peuvent pas commencer avant la date zéro.

L'équation 4.5 représente les contraintes de gammes opératoires. Pour qu'une opération puisse commencer, il faut que toutes les sous-opérations de l'opération précédente soient achevées.

Les contraintes 4.6 et 4.7 permettent d'arbitrer les disjonctions sur les ressources. Dans l'équation 4.6, si $O_{a,b}$ s'exécute avant $O_{c,d}$ sur la ressource R_k alors $O_{c,d}$ doit commencer après la fin de la sous-opération de $O_{a,b}$ sur R_k. Si $O_{a,b}$ s'exécute après $O_{c,d}$ sur R_k, alors la partie droite de l'équation est négative ce qui rend vraie l'inégalité. L'équation 4.7 utilise le même principe.

Nous avons $\#ope \times (2 + \#ope \times m)$ variables et $\#ope + (\#ope - n) \times m + 2 \times (\#ope^2 \times m)$ contraintes. De la même façon que pour le job shop flexible, il n'est pas possible de résoudre des instances réelles à l'aide d'un solver commercial. Toutefois, comme nous le verrons ultérieurement, certaines instances générées aléatoirement ont pu être résolues par CPLEX.

4.3 Modélisations par un graphe

On peut adapter le graphe conjonctif pour modéliser une solution au problème de job shop multiressource. Deux type de graphes peuvent être définis. Selon le modèle utilisé, le nombre de sommets par opération diffère.

4.3.1 Graphe avec un sommet par opération

Dans ce modèle de graphe, on a un sommet par opération. Les contraintes multiressources sont modélisées par plusieurs arcs entrant et plusieurs arcs sortant sur les sommets pour traduire les contraintes de précédence sur les ressources.

Formellement, soit le graphe $G = (V, E)$. L'ensemble des sommets est $V = \{O_{i,j}, 1 \leq i \leq n, 1 \leq j \leq n_i\} \cup \{s, t\}$ avec s le sommet source et t le sommet puits. On a un

arc entre deux sommets s'il existe une contrainte de précédence entre les opérations correspondantes. Il y a un arc entre O_{i,n_i} et t de longueur $p_{i,n_i,k}$, où R_k est la ressource sur laquelle O_{i,n_i} s'exécute. Il y a un arc entre s et $\{O_{i,j}|\Gamma_{i,j}^{-1} = \emptyset\}$ de longueur r_i. Pour une opération $O_{i,j}$, les arcs correspondants aux contraintes de gamme ont comme longueur $\max_k p_{i,j,k}$. La longueur d'un arc entre deux opérations successives $O_{i,j}$ et $O_{i',j'}$ sur la ressource R_k vaut $p_{i,j,k}$.

Ce graphe ne doit pas contenir de circuit, sinon l'ordonnancement est infaisable, et tous les arcs ont une longueur positive ou nulle. La figure 4.2 donne le graphe correspondant au diagramme de Gantt de la figure 4.1. Les arcs pleins représentent les contraintes de gamme et les arcs en pointillés représentent les contraintes de ressource.

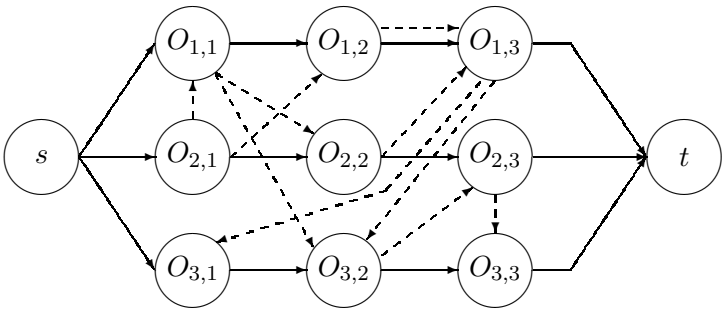

FIG. 4.2 – Représentation sous forme d'un graphe simple d'un ordonnancement multiressource

Ce modèle de graphe permet de prendre en compte des durées opératoires différentes pour chaque sous-opération. De plus, ce graphe est sans circuit, on peut donc utiliser des algorithmes simples de recherche de plus long chemin, par exemple un parcours en ordre topologique.

4.3.2 Graphe avec un sommet par sous-opération

Dans ce modèle de graphe, on représente une opération par plusieurs sommets : un par sous-opération.

Formellement, soit le graphe $G = (V, E)$. L'ensemble des sommets est $V = \{O_{i,j,k}, 1 \leq i \leq n, 1 \leq j \leq n_i\, R_k \in R_{i,j}^m\} \cup \{s, t\}$ avec s le sommet source et t le sommet puits. On a un arc entre deux sommets s'il existe une contrainte de précédence entre les opérations correspondantes. Il y a un arc entre O_{i,n_i} et t de longueur $p_{i,n_i,k}$, où R_k est la ressource sur laquelle $O_{i,n_i,k}$ s'exécute. Il y a un arc entre s et $\{O_{i,j,k}|\Gamma_{i,j}^{-1} = \emptyset\}$ de longueur r_i. Pour les autres arcs, la longueur est égale à la durée opératoire de la sous-opération à l'origine de l'arc. Entre chaque sous-opération d'une même opération il y a des arcs de synchronisation bi-directionnels, leurs longueurs sont fixées à 0. Ce graphe ne doit pas contenir de circuit de longueur strictement positive, sinon l'ordonnancement est infaisable, et tous les arcs ont une longueur positive ou nulle. La figure 4.3 donne le graphe correspondant au diagramme de Gantt de la figure 4.1, les numéros des ressources ont été omis pour faciliter la lecture.

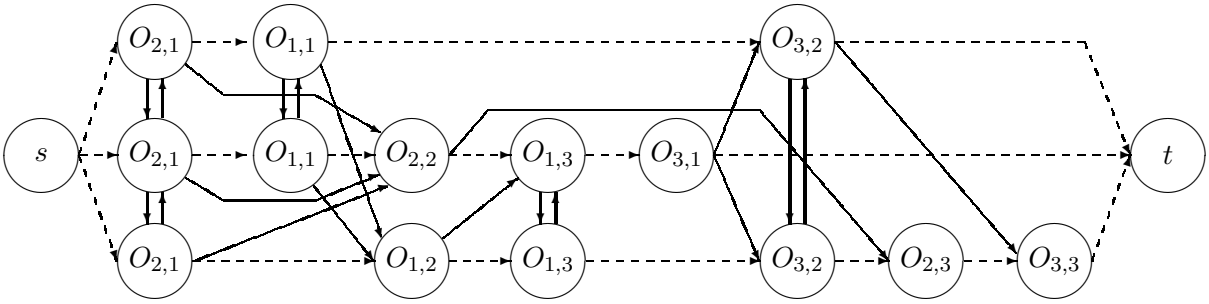

FIG. 4.3 – Représentation sous forme d'un graphe détaillé d'un ordonnancement multi-ressource

En modifiant la valuation des arcs de synchronisation, il est possible de décaler dans le temps les dates de début des sous-opérations. La figure 4.4 donne un exemple de décalage des dates de début des sous-opérations. Dans cette exemple, $O_{1,3,b}$ commence 2 unités de temps après $O_{1,3,a}$. Les avantages de cette méthode sont qu'on ne crée pas de circuit de longueur strictement positive et que les problèmes avec ou sans décalage sont gérés de la même façon. Nous considérons que les durées de décalage sont des données du problème. Par la suite, ce décalage fixe entre sous-opérations est égal à zéro. Notons également que quand ce décalage n'est pas nul, une solution réalisable peut comporter des circuits faisant intervenir plusieurs travaux.

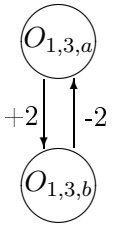

FIG. 4.4 – Modélisation d'un décalage dans les dates de début des sous-opérations

L'inconvénient principal de cette extension du modèle, par rapport au modèle de graphe simple, est la présence de circuit et d'arcs de longueur négative. On est donc obligé d'utiliser des algorithmes de recherche de plus long chemin prenant en compte ces particularités. Ici on peut utiliser, par exemple, l'algorithme de Bellman-Ford.

Cependant, l'utilisation de l'algorithme de Bellman-Ford peut augmenter fortement le temps de calcul. En effet, dans le cas de la prise en compte des périodes d'inactivités (cf. section 5.2.1 page 107), le calcul de la date de fin d'une opération à partir de sa date de début n'est plus une simple addition (cf. algorithmes 16 à 19, page 109). Nous avons donc cherché à adapter l'algorithme de parcours en ordre topologique à notre problème de graphe spécifique au problème de job shop multiressource.

L'algorithme 13 décrit la méthode pour calculer les dates de début des opérations. Dans cet algorithme on a les éléments suivants :

- $Liste$: liste contenant les (sous-)opérations pouvant être traitées. Les sous-opérations sont triées par opération.
- $Ajouter(Liste, O_{i,j,k})$: Ajoute la sous-opération $O_{i,j,k}$ à $Liste$.
- $CalculFin(p_{i,j,k}, t_{i,j,k})$: Calcule la date de fin d'exécution de l'opération $O_{i,j,k}$ à partir de sa durée et de sa date de début d'exécution.
- $PremierNMR(Liste)$: retourne la première opération unitaire (c'est-à-dire sans sous-opération) de $Liste$ (et la retire), retourne $\emptyset$ s'il y en a aucune.
- $PremierGroupeMR(Liste)$: retourne le premier ensemble de sous-opérations complet (toutes les sous-opérations de l'opération doivent y être) de la $Liste$ (et le retire), retourne $\emptyset$ s'il y en a aucun.
- $\tau_{i,j,k}$: la durée de décalage entre la sous-opération $O_{i,j,k}$ et la sous-opération commençant le plus tôt.

Algorithme 13 Algorithme de calcul des dates de début des opérations

```
1  Liste = ∅
2  Pour  i = 1 à n faire
3      Pour  j = 1 à n_i faire
4          Pour chaque R_k ∈ R^m_{i,j} faire
5              t_{i,j,k} = 0
6              Nb_pred(O_{i,j,k}) = |Γ^{-1}_{i,j,k}|
7              Si Nb_pred(O_{i,j,k}) = 0 alors
8                  Ajouter(Liste, O_{i,j,k})
9              fin Si
10         fin Pour
11     fin Pour
12 fin Pour
13 Tant que Liste ≠ ∅ faire
14     O = PremierNMR(Liste)
15     Si O = ∅ alors
16         O= PremierGroupeMR(Liste)
17     fin Si
18     /* calcul de la date de début de l'opération dans l'ordonnancement */
19     O_{i,j,k}|(t_{i,j,k} − τ_{i,j,k}) = max_{O_{u,v,l}∈O}(t_{u,v,l} − τ_{u,v,l})
20     O_{d,t,k'}|τ_{d,t,k'} = 0
21     t_{d,t,k'} = t_{i,j,k} − τ_{i,j,k}
22     Pour chaque  O_{i,j,k} ∈ O  faire
23         t_{i,j,k} = t_{d,t,k'} + τ_{i,j,k}
24         C_{i,j,k} = CalculFin(p_{i,j,k}, t_{i,j,k})
25         Pour chaque  O_{u,v,l} ∈ Γ_{i,j,k}  faire
26             t_{u,v,l} = max(t_{u,v,l}, C_{i,j,k}) /* date au plus tôt du début de O_{u,v,l}  */
27             Nb_pred(O_{u,v,l}) = Nb_pred(O_{u,v,l}) − 1
28             Si Nb_pred(O_{u,v,l}) = 0 alors
29                 Ajouter(Liste, O_{u,v,l}) /* O_{u,v} peut être traitée */
30             fin Si
31         fin Pour
32     fin Pour
33 fin Tant que
```

4.4 Approche Tabou pour le job shop multiressource, TS_{MPT}

Nous avons commencé par développer une recherche Tabou pour résoudre le problème. Cette recherche Tabou est inspirée de celle que nous avons développée pour le job shop flexible (cf. section 3.3 page 43).

Codage d'une solution Une solution est représentée par le graphe avec un sommet par sous-opération (cf. section 4.3.2).

Solution initiale La génération de la solution initiale est une adaptation de l'algorithme 7 (page 46) au cas multiressource. L'algorithme 14 décrit comment la solution initiale est générée. Ici, la fonction $choix(X)$ est une fonction qui choisit l'opération avec la plus petite marge (SLACK, voir page 44).

Définition du voisinage On obtient un nouveau voisin en déplaçant une opération. Cependant, si l'opération est composée de sous-opérations, la manipulation peut devenir plus complexe. Nous avons choisi comme mouvement l'échange d'une sous-opération avec l'opération précédente sur la ressource.

Formellement, soit x_a la sous-opération de x qu'on souhaite déplacer entre y et z, où z est l'opération précédente à x_a sur la ressource considérée. Soit r_z la date de début de l'opération z. Pour toutes les sous-opérations x_i de l'opération x ($x_i \neq x_a$), on cherche à déplacer x_i vers la gauche tant qu'elle n'aura pas de prédécesseur ressource ayant une date de début strictement inférieure à r_z (sous réserve de compatibilité avec la gamme).

La figure 4.5 illustre un tel mouvement.

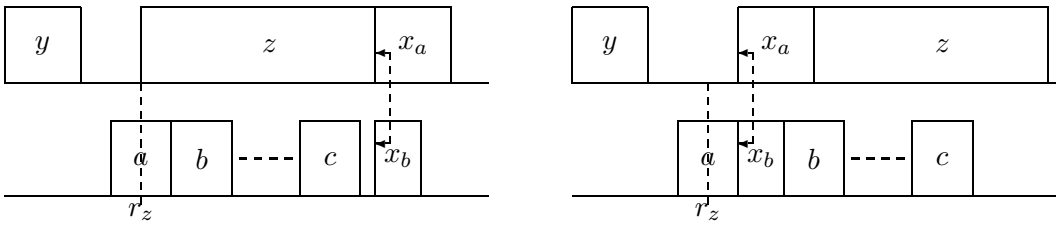

FIG. 4.5 – Déplacement de la sous-opération x_a avant z

Pour améliorer la convergence, on déplace une sous-opération tant que l'ordonnancement obtenu est réalisable et de meilleure qualité.

L'algorithme 15 décrit la procédure pour trouver un voisin. De façon similaire, on peut générer un voisin en déplaçant une sous-opération vers la droite.

Pour générer le voisinage, on explore un voisin pour chaque sous-opération du problème. L'évaluation se fait selon l'approche ϵ-contrainte développée dans la section 3.3.2 (page 48).

On réutilise aussi l'algorithme 9 (page 50) pour déterminer une approximation du front de Pareto.

Algorithme 14 Algorithme glouton pour générer une solution pour le job shop multiressource

Pré-conditions: $Choix(X)$: Choisit un élément de la liste X

$\mathcal{O} = \emptyset$
Pour $i = 0$ à $i = m$ **Faire** $C_m[i] = 0$ **fin Pour**
Pour $i = 1$ à n **faire**
    Pour $j = 1$ à n_i **faire**
        $r_{i,j} = 0$
        Si $\Gamma_{i,j}^{-1} = \emptyset$ **Alors** Ajouter $O_{i,j}$ à $\mathcal{O}$ **fin Si**
    fin Pour
fin Pour
Tant que $\mathcal{O} \neq \emptyset$ **faire**
    $C^* = \infty$
    /* *Recherche de l'opération se terminant le plus tôt* */
    Pour chaque $O_{i,j} \in \mathcal{O}$ **faire**
        $dateFin = -\infty$
        Pour chaque $R_k \in R_{i,j}^m$ **faire**
            $dateFin = \max\left(dateFin; \max(r_{i,j}, C_m[k]) + p_{i,j,k}\right)$
        fin Pour
        Si $C^* > dateFin$ **alors**
            $C^* = dateFin$
            $\mathcal{R}^* = R_{i,j}^m$
        fin Si
    fin Pour
    /* *Choix d'une opération concurrente* */
    $\mathcal{L} = \{O_{i,j} \in \mathcal{O} | r_{i,j} < C^* \text{ et } R_{i,j}^m \cap \mathcal{R}^* \neq \emptyset\}$
    $O_{i^*,j^*} = Choix(\mathcal{L})$
    /* *Ordonnancer* O_{i^*,j^*} *au plus tôt* */
    Pour chaque $R_k \in R_{i^*,j^*}^m$ **faire**
        $r_{i^*,j^*} = \max(r_{i^*,j^*}, C_m[R_k])$ /* *Trouver la date de début de l'opération* */
    fin Pour
    $C_{i^*,j^*} = r_{i^*,j^*} + \max_{R'_k \in R_{i^*,j^*}^m}(p_{i^*,j^*,k'})$
    Pour chaque $R_k \in R_{i^*,j^*}^m$ **faire**
        $C_m[k^*] = r_{i^*,j^*} + p_{i^*,j^*,k}$ /* *Mise à jour des dates de disponibilité des ressources* */
    fin Pour
    /* *Mise à jour des listes et des indicateurs* */
    Retirer O_{i^*,j^*} de $\mathcal{O}$
    Pour chaque $O_{i,j} \in \Gamma_{i^*,j^*}$ **faire**
        $r_{i,j} = \max(r_{i,j}, C_{i^*,j^*})$
        Si Tous les prédécesseurs d'$O_{i,j}$ ont été ordonnancés **alors**
            Ajouter $O_{i,j}$ à $\mathcal{O}$
        fin Si
    fin Pour
fin Tant que

Algorithme 15 Décalage à gauche d'une opération

Pré-conditions: : x_a : la (sous)-opération à déplacer.

```
1  bool = VRAI
2  Z_prec = −∞
3  Z = Z_prec
4  Si x_a est une opération unitaire alors
5      Tant que (bool = VRAI) et (Z_prec ≥ Z) faire
6          Décaler x_a d'une position vers la gauche
7          Z_prec = Z
8          Si cet ordonnancement est réalisable alors
9              Si cette solution obtenue n'est pas interdite par la liste Tabou alors
10                 Ajouter au voisinage cette solution
11                 Z = qualité de la solution
12             fin Si
13         Sinon
14             bool = FAUX
15         fin Si
16     fin Tant que
17 Sinon
18     Tant que (bool = VRAI) et (Z_prec < Z) faire
19         Z_prec = Z
20         r_z = la date de début du prédécesseur ressource de x_a
21         Pour chaque  x_i ∈ x\{x_a} faire
22             Tant que la date de début du prédécesseur machine de x_i ≥ r_V faire
23                 décaler x_i d'une position vers la gauche
24             fin Tant que
25         fin Pour
26         Décaler x d'une position vers la gauche
27         Si cet ordonnancement est réalisable alors
28             Si cette solution obtenue n'est pas interdite par la liste Tabou alors
29                 Ajouter au voisinage cette solution
30                 Z = qualité de la solution
31             fin Si
32         Sinon
33             bool = FAUX
34         fin Si
35     fin Tant que
36 fin Si
```

La liste Tabou Notre liste Tabou contient deux éléments : l'indice de l'opération ou la sous opération décalée, et le nombre de décalages avec un signe désignant le sens droite ou gauche, effectué pour obtenir la solution.

Une solution est interdite si pour l'obtenir à partir de la solution courante, il faut effectuer un mouvement qui se trouve dans la liste Tabou. C'est-à-dire qui a le même

nombre de décalages pour la même opération à déplacer, sans tenir compte du sens.

La taille de la liste est gérée dynamiquement. Cette gestion dynamique est basée sur un principe de détection de cycle. Ce dernier repose sur la sauvegarde d'un certain nombre des derniers mouvements effectués (deux fois le nombre d'opérations et de sous-opérations) dans une liste. Si a une itération donnée, deux mouvements consécutifs ont été effectués et que ces deux mouvements sont aussi dans la liste à la suite (en tenant compte du sens du déplacement), alors il y a une forte probabilité pour qu'un cycle se forme. C'est à ce moment là qu'il faut jouer sur la taille de la liste de Tabou de manière à sortir du cycle éventuel. Suite à des résultats expérimentaux, la gestion dynamique de la taille, de façon à optimiser le mieux possible la qualité des solutions, est la suivante :

- tout d'abord, la taille, à l'initialisation, est égale à un tiers du nombre d'opérations unitaires (sans les sous-opérations) plus le nombre d'opérations non-unitaires.
- La taille est augmentée de 10% du nombre d'opérations et de sous-opérations, lorsqu'un cycle éventuel est détecté et que la taille de la liste Tabou est inférieure ou égale à la moitié du nombre d'opérations et de sous-opérations.
- La taille est diminuée de 10% du nombre d'opérations et de sous-opérations, lorsqu'un cycle éventuel est détecté et que la taille de la liste Tabou est supérieure à la moitié du nombre d'opérations et de sous-opérations, ou encore lorsqu'aucun voisin n'est possible à partir d'une solution. De plus, on vide la liste Tabou de façon à répartir équitablement entre tous les mouvements.

Les conditions d'augmentation et de diminution de la taille de la liste Tabou permettent de faire osciller convenablement cette taille. Cette oscillation rend encore plus dynamique la recherche de solutions, et permet d'augmenter la qualité des solutions trouvées par cette recherche Tabou.

4.5 Approche par algorithme génétique pour le job shop multiressource, AG_{MPT}

Nous avons adapté l'algorithme génétique proposé à la section 3.4.2 à la problématique multiressource. Cette version de l'algorithme utilise aussi le modèle NSGA-II.

Le codage d'un individu On représente un individu par une matrice $\mathcal{C}$ avec m lignes (une par ressource) et $|rang|$ colonnes, où $|rang|$ est le nombre de rangs dans le graphe disjonctif « simple »(cf. section 4.3.1) associé à l'ordonnancement. A la ligne l et à la colonne c il y a l'opération qui doit être exécutée sur la ressource R_l au rang c dans le graphe. Un '_' signifie qu'il n'y a pas d'opération au rang correspondant dans le graphe sur la ressource considérée. La figure 4.6 donne le codage associé à la figure 4.1.

$$\mathcal{C} = \begin{pmatrix} O_{2,1} & O_{1,1} & \_ & \_ & \_ & O_{3,2} & \_ & \_ \\ O_{2,1} & O_{1,1} & O_{2,2} & O_{1,3} & O_{3,1} & \_ & \_ & \_ \\ O_{2,1} & \_ & O_{1,2} & O_{1,3} & \_ & O_{3,2} & O_{2,3} & O_{3,3} \end{pmatrix}$$

FIG. 4.6 – Codage d'un individu

Le croisement On génère un nouvel individu grâce à la méthode suivante : premièrement, on sélectionne deux individus a et b grâce à l'opérateur de sélection. Ensuite, on génère une matrice booléenne $\mathcal{B}$ de la même taille que a, les valeurs de cette matrice sont tirées aléatoirement avec une équiprobabilité. Notons que pour une opération donnée, les coefficients doivent être identiques pour chaque sous-opération donc lorsque la première sous-opération d'une opération est rencontrée, la valeur obtenue pour cette sous-opération est également attribuée aux autres sous-opérations. Un '1' signifie que l'opération correspondante dans a sera placée à la même position dans l'enfant. A la fin de cette étape, les opérations qui ne sont pas encore placées sont insérées dans les trous dans le même ordre que dans b.

S'il y a un circuit dans l'enfant, l'opération venant de b qui est impliquée dans le circuit est déplacée à une position qui ne crée pas de circuit, c'est-à-dire entre ses prédécesseurs et ses successeurs.

La figure 4.7 donne un exemple de croisement.

$$a = \begin{pmatrix} 1,1 & \_ & \_ & 2,1 & \_ & \_ & 3,2 & \_ \\ 1,1 & \_ & 1,3 & 2,1 & 2,2 & 3,1 & \_ & \_ \\ \_ & 1,2 & 1,3 & 2,1 & \_ & 2,3 & 3,2 & 3,3 \end{pmatrix}$$

$$b = \begin{pmatrix} 2,1 & 1,1 & \_ & \_ & \_ & 3,2 & \_ & \_ \\ 2,1 & 1,1 & 2,2 & 1,3 & 3,1 & \_ & \_ & \_ \\ 2,1 & \_ & 1,2 & 1,3 & \_ & 3,2 & 2,3 & 3,3 \end{pmatrix}$$

$$\mathcal{B} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\text{Enfant partiel} = \begin{pmatrix} 1,1 & \_ & \_ & \_ & \_ & \_ & 3,2 & \_ \\ 1,1 & \_ & 1,3 & \_ & 2,2 & \_ & \_ & \_ \\ \_ & \_ & 1,3 & \_ & \_ & \_ & 3,2 & \_ \end{pmatrix}$$

$$\text{Enfant final} = \begin{pmatrix} 1,1 & 2,1 & \_ & \_ & \_ & 3,2 & \_ & \_ \\ 1,1 & 2,1 & 1,3 & 2,2 & 3,1 & \_ & \_ & \_ \\ \_ & 2,1 & 1,3 & 1,2 & \_ & 3,2 & 2,3 & 3,3 \end{pmatrix}$$

FIG. 4.7 – Exemple d'un croisement

La mutation On applique l'opérateur de mutation sur un individu. Dans l'individu à muter, on sélectionne aléatoirement une opération x. Cette opération x est déplacée sur un rang pour lequel il n'y aura pas de circuit dans le graphe correspondant, c'est-à-dire entre le dernier prédécesseur et le premier successeur de x.

On calcule dynamiquement la probabilité de mutation (Tm) à chaque génération avec la formule suivante :

$$Tm = TM_{max} \times \frac{\text{Nombre de générations sans amélioration}}{\text{Nombre maximum de générations sans amélioration}}$$

Avec TM_{max} la probabilité de mutation maximale définie par le décideur.

La population La génération de la population initiale est une adaptation de l'algorithme 7 (page 46) au cas multiressource. L'algorithme 14 décrit comment un individu est généré aléatoirement. Dans ce cas, la fonction $choix(X)$ est une fonction qui choisit aléatoirement un élément de la liste X.

La population peut aussi être partiellement initialisée par TS_{MPT}. On nomme AG_{MPT} la version avec initialisation par la Tabou et $AG_{MPT}r$ la version avec initialisation totalement aléatoire.

4.6 Expérimentations

Des expérimentations ont été réalisées pour comparer nos algorithmes.

Toutes les expérimentations ont été réalisées sur un PC équipé d'un Pentium IV cadencé à 2.8GHz et ayant 512Mo de mémoire vive. Pour l'algorithme génétique, le temps de calcul a été limité à 10 minutes.

4.6.1 Les instances

Pour nos expérimentations, nous réutilisons les instances `sdata` de la section 3.5, ces instances possèdant une ressource par opération.

A notre connaissance il n'existe pas dans la littérature d'instances pour le problème de job shop multiressource. Nous avons donc généré nos propres instances. Nous considérons quatre paramètres : n le nombre de travaux, n_i le nombre d'opérations par travail, m le nombre de ressources et $\#sta$ le nombre d'étages. Chaque travail i contient n_i opérations. La durée opératoire d'une opération $O_{i,j}$ est tirée aléatoirement entre 1 et 130. Les dates dues sont générées avec la même méthode que celle décrite précédemment (cf. section 3.5.1). Les ressources sont réparties en étages (1 à $\#sta$). Chaque opération $O_{i,j}$ nécessite entre 1 et $\#sta$ ressources mais ne peut nécessiter plus d'une ressource d'un même étage.

Le tableau 4.1 donne les instances qui ont été générées par cette méthode. Chaque ensemble est constitué de cinq instances différentes.

Instances	n	n_j	m	$\#sta$	Instances	n	n_j	m	$\#sta$
GJ01 à GJ05	5	5	10	10	GJ21 à GJ25	10	10	20	4
GJ06 à GJ10	5	10	20	4	GJ26 à GJ30	10	5	20	10
GJ11 à GJ15	5	10	20	5	GJ31 à GJ35	10	10	20	10
GJ16 à GJ20	10	5	20	4					

TAB. 4.1 – Les instances générées

4.6.2 Expérimentations préliminaires

Des expérimentations préliminaires ont été réalisées sur quelques instances pour trouver les meilleurs paramètres pour nos algorithmes.

Pour la recherche Tabou TS_{MPT}, le meilleur nombre maximum d'itérations sans amélioration trouvé est $\#iter = 2000$.

Pour l'algorithme génétique AG_{MPT}, on trouve comme taille de population $N = 100$, nombre maximum de générations sans amélioration $\#MaxGen = 1000$ et la probabilité de mutation $Mut = 0.5$.

On renvoie à l'annexe A.2 pour le détail des expérimentations.

4.6.3 Comparaison des méthodes

Les premières expérimentations concernent la comparaison des deux méthodes pour la détermination d'un ensemble de solution non-dominées sur les critères C_{max}/L_{max}.

4.6.3.1 Comparaison de TS_{MPT} avec $AG_{MPT}r$

Dans cette partie, nous comparons les performances de la recherche Tabou TS_{MPT} avec celles de l'algorithme génétique $AG_{MPT}r$. Le tableau 4.2 donne les résultats de la comparaison entre les deux méthodes. Les temps de calcul moyens sont donnés en secondes.

	TS_{MPT} vs $AG_{MPT}r$		$AG_{MPT}r$ vs TS_{MPT}			
Instances	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	$\overline{CPU(TS_{MPT})}$	$\overline{CPU(AG_{MPT}r)}$
la01 à la05	3	85%	0	25%	33.59	242.45
la06 à la10	1	60%	1	55%	118.80	528.18
la11 à la15	2	73%	0	33%	266.31	584.29
la16 à la20	4	87%	0	13%	133.62	599.66
la21 à la25	**5**	**100%**	0	0%	540.58	600.40
la26 à la30	**5**	**100%**	0	0%	1518.12	600.69
la31 à la35	**5**	**100%**	0	0%	2338.61	601.50
la36 à la40	**5**	**100%**	0	0%	1356.68	600.50
GJ01 à GJ05	1	60%	3	93%	4.30	27.91
GJ06 à GJ10	4	87%	0	13%	41.06	134.90
GJ11 à GJ15	4	90%	0	10%	65.58	127.30
GJ16 à GJ20	2	80%	0	20%	140.51	283.42
GJ21 à GJ25	3	70%	1	30%	522.91	520.35
GJ26 à GJ30	4	90%	0	10%	279.03	303.16
GJ31 à GJ35	4	90%	0	10%	1071.29	599.82

TAB. 4.2 – Comparaison entre TS_{MPT} et $AG_{MPT}r$

On constate que la recherche Tabou est toujours meilleure que l'algorithme génétique aussi bien sur les instances `sdata` que sur nos instances générées. Par ailleurs, on constate

que sur les instances `sdata`, $AG_{MPT}r$ atteint souvent son temps de calcul maximum (10 minutes).

4.6.3.2 Comparaison de AG_{MPT} avec et sans initialisation par TS_{MPT}

Nous avons testé notre algorithme génétique avec et sans initialisation de la population par TS_{MPT}. Le tableau 4.3 donne les résultats, où $AG_{MPT}r$ est la « version sans initialisation » et AG_{MPT} la version « avec initialisation ». Les temps de calcul pour AG_{MPT} incluent le temps nécessaire pour la recherche Tabou.

	$AG_{MPT}r$ vs AG_{MPT}		AG_{MPT} vs $AG_{MPT}r$			
Instances	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	$\overline{CPU(AG_{MPT}r)}$	$\overline{CPU(AG_{MPT})}$
la01 à la05	0	25%	**5**	**100%**	242.45	**111.48**
la06 à la10	0	17%	4	90%	528.18	**227.60**
la11 à la15	0	15%	4	95%	584.29	632.50
la16 à la20	0	0%	**5**	**100%**	599.66	**433.17**
la21 à la25	0	0%	**5**	**100%**	600.40	830.61
la26 à la30	0	0%	**5**	**100%**	600.69	1772.00
la31 à la35	0	0%	**5**	**100%**	601.50	2471.48
la36 à la40	0	0%	**5**	**100%**	600.50	1610.40
GJ01 à GJ05	1	93%	1	90%	27.91	27.71
GJ06 à GJ10	0	10%	4	90%	134.90	**73.86**
GJ11 à GJ15	0	0%	**5**	**100%**	127.30	**116.26**
GJ16 à GJ20	0	10%	4	90%	283.42	**234.56**
GJ21 à GJ25	0	15%	4	85%	520.35	695.15
GJ26 à GJ30	0	0%	**5**	**100%**	303.16	315.75
GJ31 à GJ35	0	0%	**5**	**100%**	599.82	1180.33

TAB. 4.3 – Comparaison entre $AG_{MPT}r$ et AG_{MPT}

On constate une nette domination de la version avec initialisation. De plus dans plusieurs cas, le temps de calcul de TS_{MPT} plus celui de AG_{MPT} est inférieur à celui de $AG_{MPT}r$.

4.6.3.3 Comparaison de TS_{MPT} avec AG_{MPT}

Dans cette partie, nous comparons les performances de la recherche Tabou TS_{MPT} avec celles de l'algorithme génétique AG_{MPT}. Le tableau 4.4 donne les résultats de la comparaison entre les deux méthodes. Les temps de calcul moyens sont donnés en secondes. Le temps de calcul de AG_{MPT} n'inclut pas celui de TS_{MPT}.

On constate que AG_{MPT} donne de meilleurs résultats que TS_{MPT}, ce qui n'est pas surprenant dans la mesure où les solutions de TS_{MPT} sont injectées comme population initiale dans AG_{MPT}. On peut remarquer que pour huit instances, AG_{MPT} n'arrive pas à améliorer les solutions trouvées par la recherche Tabou.

Instances	TS_{MPT} vs AG_{MPT}		AG_{MPT} vs TS_{MPT}			
	SOw	$\overline{NFC}$	SOw	$\overline{NFC}$	$\overline{CPU(TS_{MPT})}$	$\overline{CPU(AG_{MPT})}$
la01 -> la05	0	62%	4	**100%**	33.59	92.40
la06 -> la10	0	30%	**5**	**100%**	118.80	157.91
la11 -> la15	0	28%	**5**	**100%**	266.31	458.64
la16 -> la20	0	41%	4	**100%**	133.62	345.33
la21 -> la25	0	48%	**5**	**100%**	540.58	440.93
la26 -> la30	0	24%	**5**	**100%**	1518.12	600.54
la31 -> la35	0	33%	**5**	**100%**	2338.61	601.42
la36 -> la40	0	32%	**5**	**100%**	1356.68	600.67
GJ01 -> GJ05	0	60%	3	**100%**	4.30	25.81
GJ06 -> GJ10	0	80%	2	**100%**	41.06	54.89
GJ11 -> GJ15	0	24%	**5**	**100%**	65.58	81.89
GJ16 -> GJ20	0	23%	**5**	**100%**	140.51	157.81
GJ21 -> GJ25	0	27%	**5**	**100%**	522.91	357.37
GJ26 -> GJ30	0	30%	4	**100%**	279.03	150.89
GJ31 -> GJ35	0	0%	**5**	**100%**	1071.29	503.57

TAB. 4.4 – Comparaison entre TS_{MPT} et AG_{MPT}

4.6.4 Comparaison avec CPLEX

Notre programme linéaire, testé avec CPLEX (version 8.0), arrive à résoudre optimalement les problèmes GJxx jusqu'à GJ20 (sauf pour GJ13 où le solver n'arrive pas à trouver l'optimum). Nous avons limité le temps de calcul à 12 heures. Le tableau 4.5 donne les valeurs du *makespan* obtenus par CPLEX et par nos algorithmes. La colonne 'CPU(CPLEX)' correspond au temps de calcul (en secondes) nécessaire à CPLEX pour trouver l'optimum sur le C_{max}. Les colonnes $\Delta(TS_{MPT})$ et $\Delta(AG_{MPT}r)$ correspondent à l'écart relatif entre nos méthodes et l'optimum.

Dans 6 cas sur 19, TS_{MPT} trouve une solution optimale et dans 4 cas pour $AG_{MPT}r$. On constate que dans le pire des cas, on est à 6.33% de l'optimum avec la recherche Tabou (15.59% pour l'algorithme génétique). Globalement on peut constater que TS_{MPT} est proche de l'optimum : $\overline{\Delta(TS_{MPT})} = 1.71\%$. Par contre $AG_{MPT}r$ est clairement moins performant : $\overline{\Delta(AG_{MPT}r)} = 6.9\%$

Remarque : nous n'avons pas présenté les résultats pour AG_{MPT} car les valeurs du *makespan* sont identiques à celles de TS_{MPT}, sauf pour l'instance GJ17 où $C_{max}(AG_{MPT}) = 664$.

4.7 Conclusion sur le job shop multiressource

Dans ce chapitre nous avons présenté nos travaux sur le problème de job shop multiressource multicritère. Nous avons proposé un modèle de programmation linéaire en nombres entiers, une recherche Tabou, un algorithme génétique ainsi qu'une hybridation de ces deux dernières méthodes. Nous avons modélisé le problème avec deux types de

Instance	CPLEX	CPU(CPLEX)	TS_{MPT}	$\Delta(TS_{MPT})$	$AG_{MPT}r$	$\Delta(AG_{MPT}r)$
GJ01	533	0.06	**533**	0.00%	544	2.02%
GJ02	418	0.05	**418**	0.00%	**418**	0.00%
GJ03	487	0.06	**487**	0.00%	**487**	0.00%
GJ04	383	0.03	**383**	0.00%	**383**	0.00%
GJ05	460	0.02	**460**	0.00%	**460**	0.00%
GJ06	894	2.52	898	0.45%	963	7.17%
GJ07	900	4.14	**900**	0.00%	918	1.96%
GJ08	763	45.39	794	3.90%	821	7.06%
GJ09	767	1.69	789	2.79%	892	14.01%
GJ10	998	36.03	1017	1.87%	1100	9.27%
GJ11	839	12.97	891	5.84%	977	14.12%
GJ12	1057	210.59	1100	3.91%	1194	11.47%
GJ13	#	#	1071	#	1162	#
GJ14	786	25.52	790	0.51%	815	3.56%
GJ15	930	1580.28	934	0.43%	1034	10.06%
GJ16	600	1925.77	614	2.28%	644	6.83%
GJ17	656	42110.38	669	1.94%	774	15.25%
GJ18	594	2590.08	603	1.49%	608	2.30%
GJ19	693	965.81	698	0.72%	821	15.59%
GJ20	681	13.34	727	6.33%	760	10.39%

TAB. 4.5 – Comparaison avec CPLEX

graphes différents, un premier type avec une modélisation simple, ce qui permet l'utilisation d'algorithmes de parcours de graphes rapides, le second est plus évolué et permet de modéliser des décalages entre sous-opérations d'une même opération.

Au niveau des expérimentations, nous avons constaté une supériorité de la recherche Tabou sur l'algorithme génétique. Par ailleurs, la recherche Tabou présente des résultats proches de l'optimal pour les instances qui ont pu êtres résolues par le programme linéaire.

Chapitre 5

Applications et mise en oeuvre

Dans ce chapitre nous allons aborder l'implémentation des algorithmes d'ordonnancement au sein du logiciel *DirectPlanning* (www.directplanning.com).

5.1 Présentation générale de DirectPlanning

DirectPlanning est le logiciel de planification édité par la société Volume Software. Ce logiciel se présente comme un planning mural interactif. Une de ses grandes forces est qu'il est entièrement paramétrable pour s'adapter à n'importe quel métier. Il est aussi simple d'utilisation, on peut notamment déplacer une tâche très simplement tout en respectant les contraintes de précédence. Parmi les fonctionnalités on a aussi la possibilité de verrouiller des tâches, d'appliquer des calendriers aux ressources, d'afficher des alertes en cas de retard, etc.

Historique La première version (estampillée 1.1) est sortie en 2003. Cette version a été en partie développée par Philippe Mauguière dans le cadre de sa thèse de doctorat ([Mauguière, 2004]). Le module d'ordonnancement de cette version utilise une *Shifting Bottleneck Procedure* inspirée de [Adams *et al.*, 1988].

La seconde version (estampillée 1.2) est sortie en 2005. Elle prend en compte en plus des temps de montage dépendants de la séquence. Dans cette version, le module d'ordonnancement a été entièrement refait. L'algorithme principal de l'ordonnancement est une recherche Tabou, mais on a aussi la possibilité de lancer une évolution de la *Shifting Bottleneck Procedure* de la version précédente.

La version en cours de développement (estampillée 2.0) devrait sortir à la fin de l'année 2007. Les fonctionnalités qui seront présentes dans la version commercialisée sont encore en discussion, par contre en ce qui concerne l'ordonnancement, on peut dire que l'ajout

principal est l'affectation automatique des opérations, c'est-à-dire la notion de job shop flexible développé dans la section 3.

Vue d'ensemble La figure 5.1 montre l'affichage principal de DirectPlanning. On y trouve un diagramme de Gantt représentant le planning selon le mode de projection choisi. DirectPlanning propose plusieurs modes de projection. Classiquement, on peut avoir une ligne par ressource sur laquelle les opérations s'exécutent, mais il est possible de choisir un autre mode de projection. Par exemple on peut projeter par client, dans ce cas toutes les opérations d'un même client seront sur la même ligne. On peut facilement personnaliser les modes d'affichage pour adapter la lecture du planning aux besoins du client.

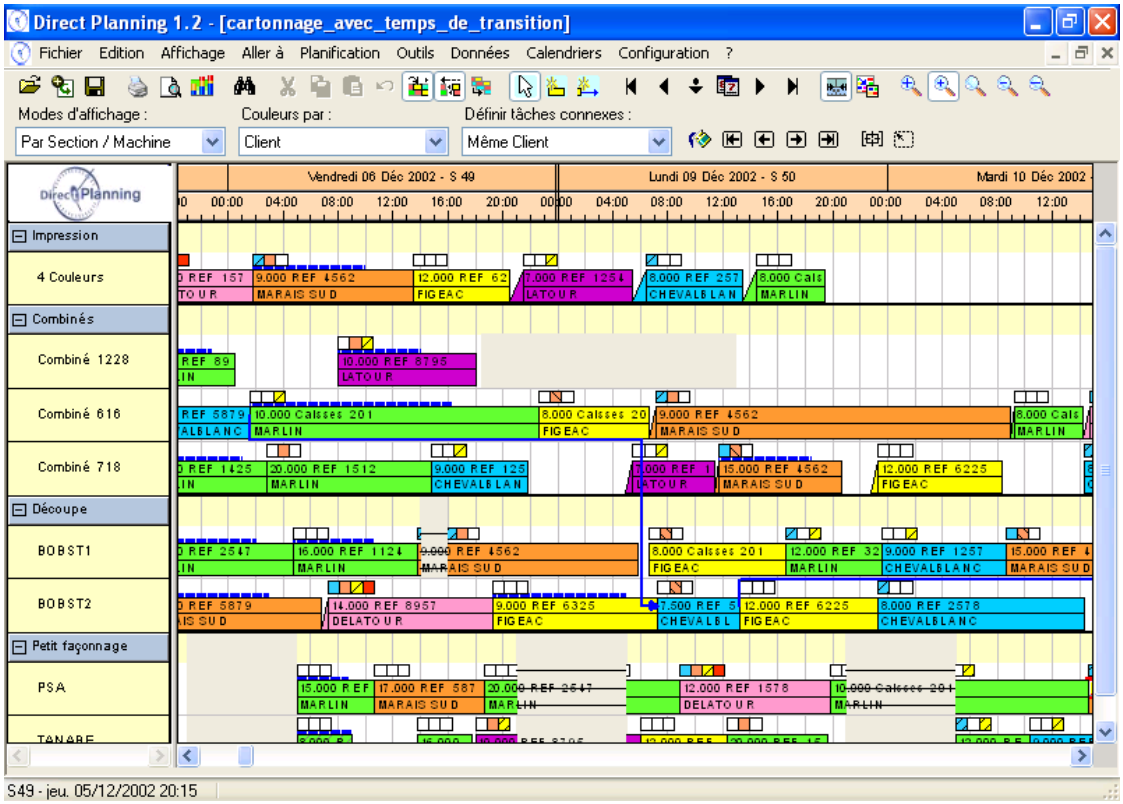

FIG. 5.1 – Ecran principal de DirectPlanning

5.2 Interface manuelle

DirectPlanning est un outil interactif, on peut déplacer très simplement une opération aussi bien dans le temps que d'une ressource à une autre. Trois modes de fonctionnement sont proposés :

– *Le non-chevauchement des tâches*, dans ce mode DirectPlanning vérifie uniquement que les opérations ne se chevauchent pas sur les ressources primaires.

- *L'aide à la planification*, dans ce mode les opérations respectent les contraintes de précédences, les dates de début au plus tôt, les opérations verrouillées et vérifie que l'ordonnancement soit réalisable (i.e. sans circuit). Dans le cas où une contrainte ne pourrait pas être respectée, DirectPlanning affiche un message d'erreur et l'aide à la planification se désactive.
- *Le calage à gauche*, ce mode est identique au précédent sauf que les opérations sont calées le plus tôt possible.

Dans cette partie, nous nous intéressons aux algorithmes utilisés pour l'interface manuelle. Ces algorithmes proviennent de [Mauguière, 2004]. Même s'ils n'ont rien de très original, il en existe peu de traces dans la littérature.

Notation Dans la suite de ce chapitre, les notations ont été allégées pour faciliter la lecture. Ainsi, i désigne la $i$éme opération du problème, p_i est la durée opératoire de cette opération sur la ressource sur laquelle elle est affectée.

Intégration du travail de Philippe Mauguière Comme indiqué en introduction, ce travail fait suite à celui réalisé par Philippe Mauguière lors de sa thèse ([Mauguière, 2004]). Ses travaux ont principalement porté sur la prise en compte des calendriers, des temps de montage dépendant de la séquence et des opérations verrouillées pour le problème de job shop. L'intérêt des méthodes qu'il propose est qu'il suffit d'ajouter ses algorithmes dans les fonctions d'évaluation de nos algorithmes pour pouvoir gérer ces contraintes.

5.2.1 Prise en compte des calendriers

La gestion des périodes d'inactivité est commune aux trois modes de fonctionnement. Cette gestion vient de [Mauguière *et al.*, 2005]. Dans cet article, les auteurs considèrent cinq types de périodes d'inactivités :

1. Les périodes d'inactivités sont traversables et les opérations peuvent être interrompues.
2. Les périodes d'inactivités sont non traversables et les opérations ne peuvent pas être interrompues.
3. Certaines périodes d'inactivités sont traversables et les opérations peuvent être interrompues.
4. Les périodes d'inactivités sont traversables et certaines opérations peuvent être interrompues.
5. Certaines périodes d'inactivités sont traversables et certaines opérations peuvent être interrompues.

DirectPlanning permet de gérer le cas 1. Lors de la création du logiciel, Philippe Mauguière a développé quatre algorithmes qui permettent de gérer les périodes d'inactivité. Le premier, $Ajuster\_t_j(t, R_k)$, permet d'ajuster la date de début t d'une opération en fonction des périodes d'indisponibilité présentes sur la ressource R_k. $Calculer\_C_j(t, p, R_k)$ permet de calculer la date de fin d'exécution d'une opération à partir de sa date de début t, de sa durée opératoire p et des périodes d'inactivités de la ressource R_k. De

la même façon, $Ajuster\_C_j(t, R_k)$ permet d'ajuster la date de fin d'une opération et $Calculer\_t_j(t, p, R_k)$ de calculer sa date de début en fonction de sa date de fin, de sa durée et des périodes d'inactivités. Les algorithmes 16 à 19 décrivent ces procédures. Dans ces algorithmes, une période d'inactivité est définie par le couple $(s_f; e_f)$. s_f est le début de la période d'inactivité f et e_f la fin de cette même période.

5.2.2 Gestion des opérations verrouillées

Une fonctionnalité très appréciée de DirectPlanning est le verrouillage d'opération. Verrouiller une opération signifie que cette dernière ne pourra en aucun cas être déplacée, ni par l'utilisateur, ni par les algorithmes d'aide à la planification, ni par le module d'ordonnancement.

Une opération verrouillée est vue comme une opération s'exécutant sur une ressource R_k avec une date de fin au plus tard impérative $\widetilde{d_i}$ et une date de début au plus tôt impérative r_i tel que :

$$Calculer\_C_j(r_i, p_i, R_k) = \widetilde{d_i}$$

On considère que la date importante dans une opération verrouillée est sa date de fin $\widetilde{d_i}$. Si une opération i est verrouillée, alors le booléen $v_i = vrai$.

5.2.3 Temps de montage dépendant de la séquence

Dans DirectPlanning, une opération peut nécessiter un temps de montage avant qu'elle ne commence réellement son exécution. Ces temps sont dépendant de la séquence, c'est-à-dire que la durée de montage dépend de l'opération qui vient de se terminer et de l'opération qui va commencer.

Si a et b sont deux opérations successives sur une même ressource, on note $s_{a,b}$ le temps de montage de l'opération b si a est l'opération précédente sur la ressource. Le temps de montage ne fait pas partie de l'opération, ainsi on peut commencer le montage d'une opération b avant la fin des opérations précédentes de b dans la gamme (on parle de *anticipatory setup time*).

La figure 5.2 illustre le concept de temps de montage. On a les opérations a puis b sur une même ressource. L'opération x est l'opération qui précède b dans la gamme.

5.2.4 Le non chevauchement des opérations

L'algorithme 20 décrit la procédure de non-chevauchement des opérations. Cette procédure prend en compte les opérations verrouillées, les périodes d'indisponibilité et les temps de montage dépendants de la séquence. Les paramètres p_i, t_i et C_i correspondent à la durée opératoire, à la date de début réelle et à la date de fin de l'opération i avant le traitement de l'algorithme, les variables p'_i, t'_i et C'_i sont calculées par la procédure.

Algorithme 16 $Ajuster\_t_j(t, R_k)$

Pré-conditions: t : date de début initiale
Pré-conditions: R_k : ressource sur laquelle s'exécute l'opération

```
1 Si ∃f|s_f ≤ t < e_f alors
2     retourner Ajuster_t_j(e_f, R_k)
3 Sinon
4     retourner t
5 fin Si
```

Algorithme 17 $Calculer\_C_j(t, p, R_k)$

Pré-conditions: t : date de début initiale
Pré-conditions: R_k : ressource sur laquelle s'exécute l'opération
Pré-conditions: p : durée opératoire de l'opération sur R_k

```
1 Si ∃f|t < s_f < t + p alors
2     retourner Calculer_C_j(e_f, p − (s_f − t), R_k)
3 Sinon
4     retourner t + p
5 fin Si
```

Algorithme 18 $Ajuster\_C_j(t, R_k)$

Pré-conditions: t : date de fin initiale
Pré-conditions: R_k : ressource sur laquelle s'exécute l'opération

```
1 Si ∃f|s_f < t ≤ e_f alors
2     retourner Ajuster_t_j(s_f, R_k)
3 Sinon
4     retourner t
5 fin Si
```

Algorithme 19 $Calculer\_t_j(t, p, R_k)$

Pré-conditions: t : date de fin initiale
Pré-conditions: R_k : ressource sur laquelle s'exécute l'opération
Pré-conditions: p : durée opératoire de l'opération sur R_k

```
1 Si ∃f|t − p < s_f < t alors
2     retourner Calculer_C_j(s_f, p − (t − e_f), R_k)
3 Sinon
4     retourner t − p
5 fin Si
```

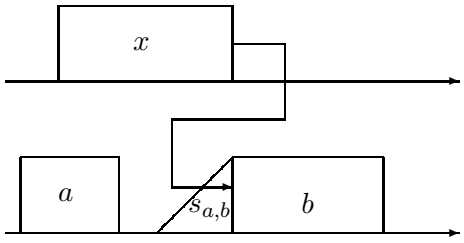

FIG. 5.2 – Illustration d'un temps de montage

L'algorithme séquence les opérations dans l'ordre de leur t_i croissant, sauf si une opération normale venait chevaucher une opération verrouillée. Dans ce cas l'opération normale sera séquencée après l'opération verrouillée. Les dates de début finales des opérations sont nécessairement supérieures ou égales à leurs dates de début initiales : $t'_i \geq t_i$.

De la ligne 1 à la ligne 6 on a la phase d'initialisation de l'algorithme et de la ligne 7 à la ligne 37 on a la boucle principale. Les principaux paramètres de l'algorithme sont :

- $\mathcal{O}$: l'ensemble de cardinalité p des opérations à traiter,
- N : l'ensemble de cardinalité q des opérations verrouillées sur la ressource,
- i : l'opération en cours de traitement,
- t : date de disponibilité de la ressource, c'est-à-dire la date de fin de la dernière opération placée,
- f : indice de la prochaine opération verrouillée à placer selon l'ordre de $\mathcal{O}$,
- g : indice de la prochaine opération verrouillée se situant après t.

Suite à l'initialisation, on traite toutes les opérations dans l'ordre de leur t_i croissant. Deux cas de figures peuvent se présenter, soit l'opération est verrouillée (lignes 9 à 17), soit elle ne l'est pas (ligne 18 à 36). Si l'opération est verrouillée, on fixe sa date de fin à sa valeur actuelle (ligne 10) puis on calcule sa date de début en tenant compte d'un éventuel temps de montage et des périodes d'inactivité (ligne 11 et 12). Les lignes 13 et 14 sont là pour incrémenter les compteurs.

Si l'opération n'est pas verrouillée, on cherche la première date de début d'exécution valide, qui est soit la date de disponibilité de la ressource t, soit la date de début de l'opération dans le cas où l'utilisateur a inséré un temps mort (lignes 16 et 17). De la ligne 18 à 21, on regarde la prochaine opération verrouillée, notée ici h. A la ligne 22, on calcule la date de fin C de h dans le cas où il n'y aurait pas de temps mort entre les deux opérations. Si cette date de fin est supérieure à la date de fin impérative de h alors il faut séquencer i après h (ligne 23 à 27). De la ligne 28 à 33, on refait le test avec la prochaine opération verrouillée. Enfin, à la ligne 34 on finalise le positionnement de i et on met à jour la date de disponibilité de la ressource à la ligne 35.

5.2.5 L'aide à la planification

L'algorithme 21 décrit la procédure d'aide à la planification. Cette procédure prend en compte les opérations verrouillées, les périodes d'indisponibilité et les temps de montage

Algorithme 20 Non chevauchement des opérations

Pré-conditions: R_k : ressource sur laquelle le non-chevauchement est lancé

$\mathcal{O} = \{i | a(i) = R_k\}$
Trier $\mathcal{O}$ selon les t_i croissant.
$N = \{i \in \mathcal{O} | v_i = vrai\}$ /* *Les opérations de N sont numérotées dans l'ordre croissant des* t_i */
$p = |\mathcal{O}|; q = |N|$
$pred(\mathcal{O}_{[1]}) = \emptyset; pred(\mathcal{O}_{[l]}) = \mathcal{O}_{[l-1]}, \forall l \in [2 \dots p]$
$f = 0; g = 0; t = 0$
Pour $index = 1$ à p **faire**
    $i = \mathcal{O}_{[index]}$
    Si $v_i = vrai$ **alors**
        $C'_i = C_i$
        $p'_i = s_{pred(i),i} + p_i$
        $t'_i = Calculer\_t_j(C'_i, p'_i, R_k)$
        $f = f + 1; g = \max(g, f)$
        $t = \max(t, Calculer\_t_j(C'_i, R_k)$
    Sinon
        $p'_i = s_{pred(i),i} + p_i$
        $t'_i = \max(t, Calculer\_t_j(C_i, p'_i, R_k))$
        $x = 0; d = \infty$
        Si $g < q$ **alors**
            $h = N_{[g]}; x = s_{i,h} + p_h; d = C_h$
        fin Si
        $C = Calculer\_C_j(t'_i, p'_i + x, R_k)$
        Tant que $C > d$ **faire**
            $t = d$
            $pred(h) = pred(i); pred(i) = h; g = g + 1$
            $p'_i = s_{pred(i),i} + p_i$
            $t'_i = Ajuster\_t_j(\max(t_i, t), R_k)$
            $x = 0; d = \infty$
            Si $g < q$ **alors**
                $h = N_{[g]}; x = s_{i,h} + p_h; d = C_h$
            fin Si
            $C = Calculer\_C_j(t'_i, p'_i + x, R_k)$
        fin Tant que
        $C'_i = Calculer\_C_j(t'j, p'j, R_k)$
        $t = Calculer\_t_j(C'_i, R_k)$
    fin Si
fin Pour

dépendants de la séquence. La procédure respecte la séquence des opérations donnée par l'utilisateur, sauf quand la séquence donnée mène à une infaisabilité à cause des opérations verrouillées. Dans ce dernier cas, l'algorithme s'autorise à échanger des opérations dans la séquence pour respecter ces contraintes fortes.

De la ligne 1 à la ligne 12 on a la phase d'initialisation. A la ligne 3 on propage les dates de fin impératives aux prédécesseurs des opérations verrouillées. A la fin de l'initialisation, la liste F contient toutes les opérations sans prédécesseur qui sont les premières sur leur ressource.

De la ligne 13 à la ligne 42 on a la boucle principale de l'algorithme. On extrait la première opération i de cette liste à la ligne 14. Deux cas se présentent, soit l'opération i possède une date de fin impérative, soit elle n'en possède pas. Dans le premier cas (ligne 16 à 20) on place cette opération le plus tôt possible et on vérifie qu'elle ne se termine pas après sa date de fin au plus tard, si elle se termine après cette date, alors on a détecté une infaisabilité et l'algorithme s'arrête (ligne 19). Dans le cas où i n'a pas de date de fin au plus tard (ligne 21 à 27), on essaye de placer i le plus tôt possible (ligne 23). Puis on vérifie que ce placement ne va pas entrer en conflit avec une opération verrouillée j (ligne 23). Si tel est le cas, on inverse les positions de i et de j (ligne 24 et 25). On recommence cette manipulation tant que des opérations verrouillées empêchent le placement de i.

De la ligne 29 à 34 on met à jour les successeurs directs j de i. Si tous les prédécesseurs de j ont été placés, alors on ajoute j à la liste F (ligne 32). On fait la même chose pour l'opération suivante sur R_k (ligne 35 à 41).

5.2.6 Le module d'ordonnancement

DirectPlanning est capable d'ordonnancer des problèmes de job shop flexible selon quatre critères : la date de fin de la dernière opération (C_{max}), le plus grand retard (L_{max}), la somme des retards (ΣT) et le nombre de travaux en retards (ΣU). Le module d'ordonnancement utilise la meilleure combinaison des algorithmes que nous avons développé, à savoir la recherche Tabou optimisant la combinaison linéaire des critères (cf. section 3.3 page 43) et l'algorithme génétique à codage direct (cf. section 3.4.2 page 53).

Une interface simple permet à l'utilisateur de configurer les paramètres de l'ordonnancement. La figure 5.3 montre la fenêtre de configuration standard. Dans cette fenêtre, on peut choisir parmi cinq niveaux de recherche pré-configurés, les critères à optimiser, le choix de lancer l'algorithme génétique et l'activation du mode affectation automatique. La figure 5.4 montre la fenêtre de paramétrage pour les « experts ». On peut régler de façon fine toutes les options de nos deux algorithmes.

Pendant le déroulement de l'algorithme, une fenêtre montre la progression de l'ordonnancement en temps réel (cf. figure 5.5). Cette fenêtre propose une interaction avec l'utilisateur, ainsi ce dernier peut choisir d'interrompre à tout moment l'ordonnancement.

A la fin de l'ordonnancement, les résultats sont proposés sous forme de tableaux (cf. figure 5.6) ainsi que sous forme graphique (cf. figure 5.7). L'utilisateur peut facilement naviguer entre les différentes solutions proposées.

Algorithme 21 Aide à la planification

Soit L_k la liste des opérations sur la ressource R_k triée dans l'ordre des t_i croissants.
Soit $pos(i)$ la position de l'opération i sur sa ressource.
Calculer les éventuelles dates de fin impératives des opérations.
Pour chaque opération i **faire**
    $nbPred(i) = |\Gamma_i^{-1}|$
    Si $pos(i) > 1$ **alors**
        $nbPred(i) = nbPred(i) + 1$
    fin Si
    Si $nbPred(i) = 0$ **alors**
        Ajouter i à F /* *F est la liste des opérations prêtes à être ordonnancées* */
    fin Si
fin Pour
Tant que F n'est pas vide **faire**
    Soit i la première opération de F, retirer i de F.
    $R_k = a(i)$ /* *R_k est la ressource qui exécute i* */
    Si i possède une date de fin impérative **alors**
        Placer i le plus tôt possible
        Si i se termine après sa date de fin impérative **alors**
            retourner *Erreur infaisabilité détectée*
        fin Si
    Sinon
        Placer i le plus tôt possible
        Tant que i est en conflit avec une opération verrouillée j **faire**
            $pos(i) = pos(i) + 1$ /* *On décale i après l'opération verrouillée* */
            $pos(j) = pos(j) - 1$
            Placer i le plus tôt possible
        fin Tant que
    fin Si
    /* *On met à jour les successeurs de i* */
    Pour chaque $j \in \Gamma_i$ **faire**
        $nbPred(j) = nbPred(j) - 1$
        Si $nbPred(j) = 0$ **alors**
            Ajouter j à F
        fin Si
    fin Pour
    Si $pos(i) < |L_k|$ **alors**
        $j = L_k[pos(i) + 1]$ /* *j est l'opération suivante sur la ressource* */
        $nbPred(j) = nbPred(j) - 1$
        Si $nbPred(j) = 0$ **alors**
            Ajouter j à F
        fin Si
    fin Si
fin Tant que

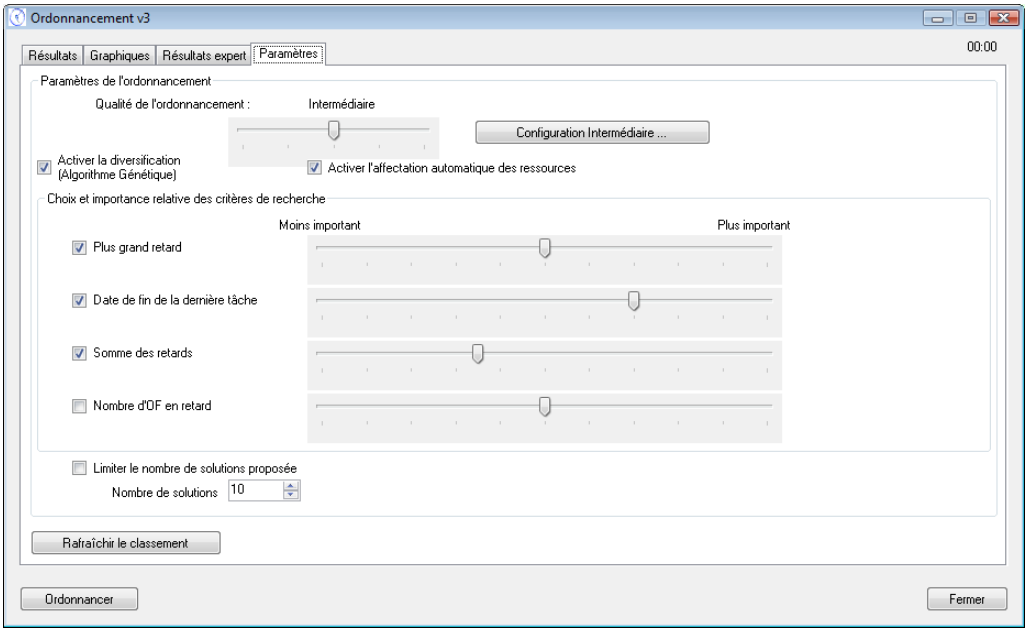

FIG. 5.3 – Paramétrage de l'ordonnancement

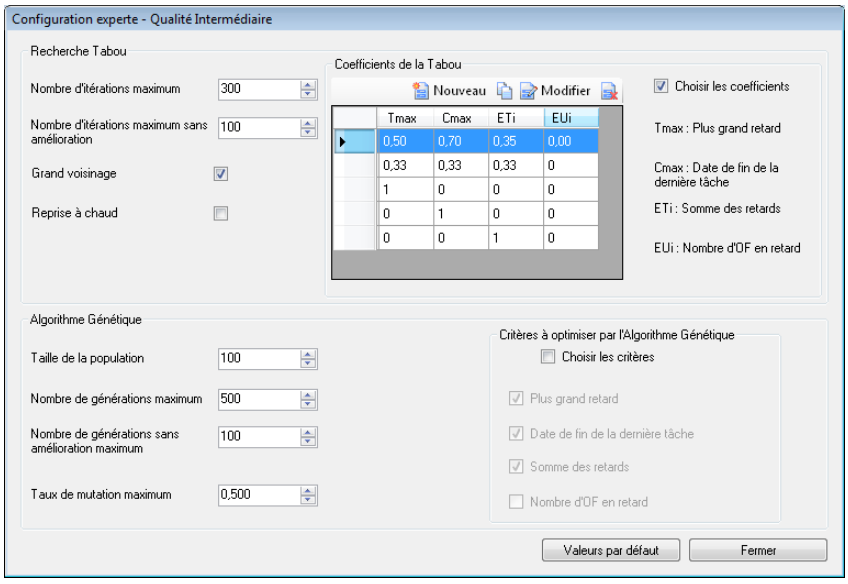

FIG. 5.4 – Paramétrage avancé de l'ordonnancement

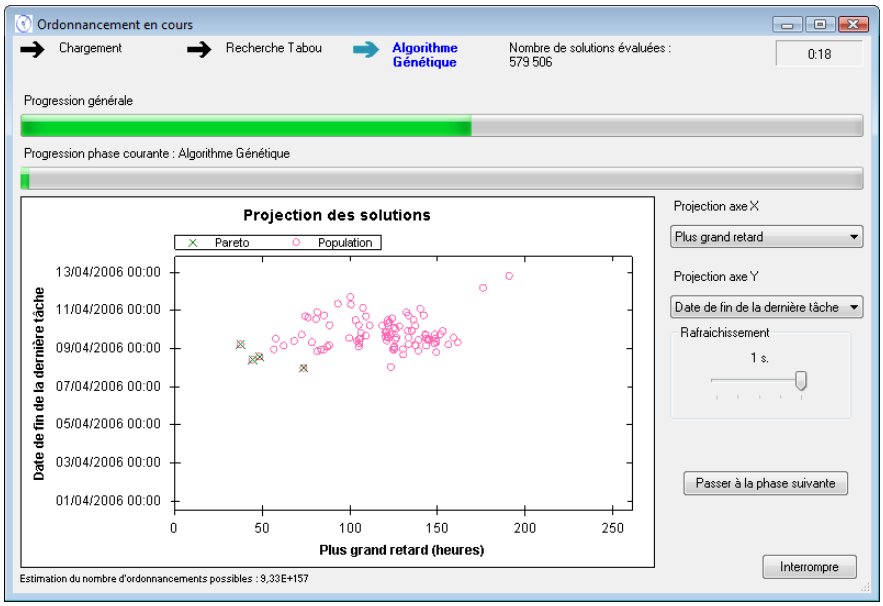

FIG. 5.5 – Exécution de l'ordonnancement

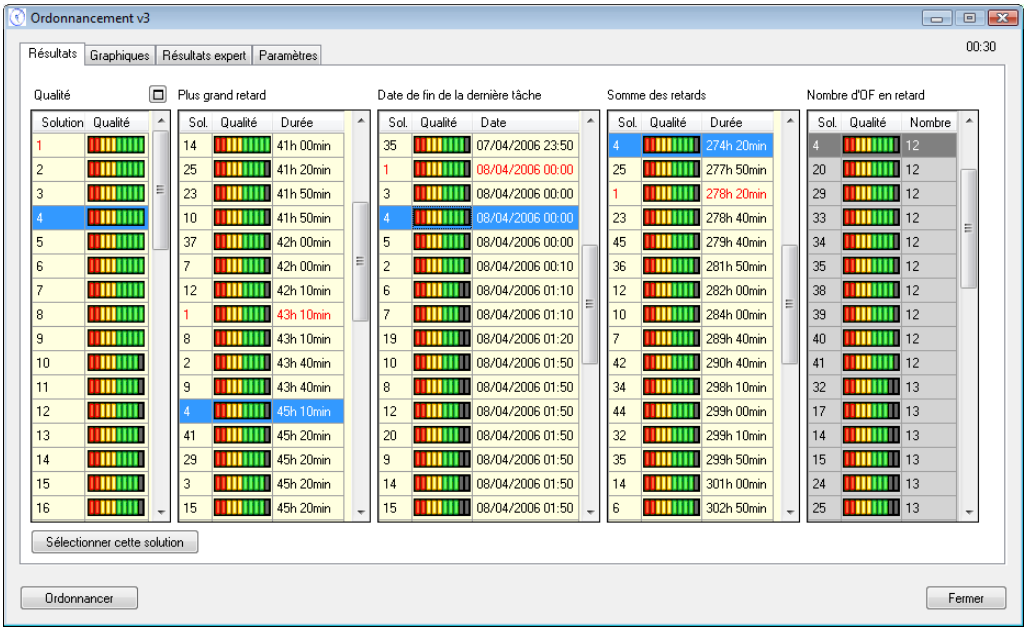

FIG. 5.6 – Résultats de l'ordonnancement sous forme tabulaire

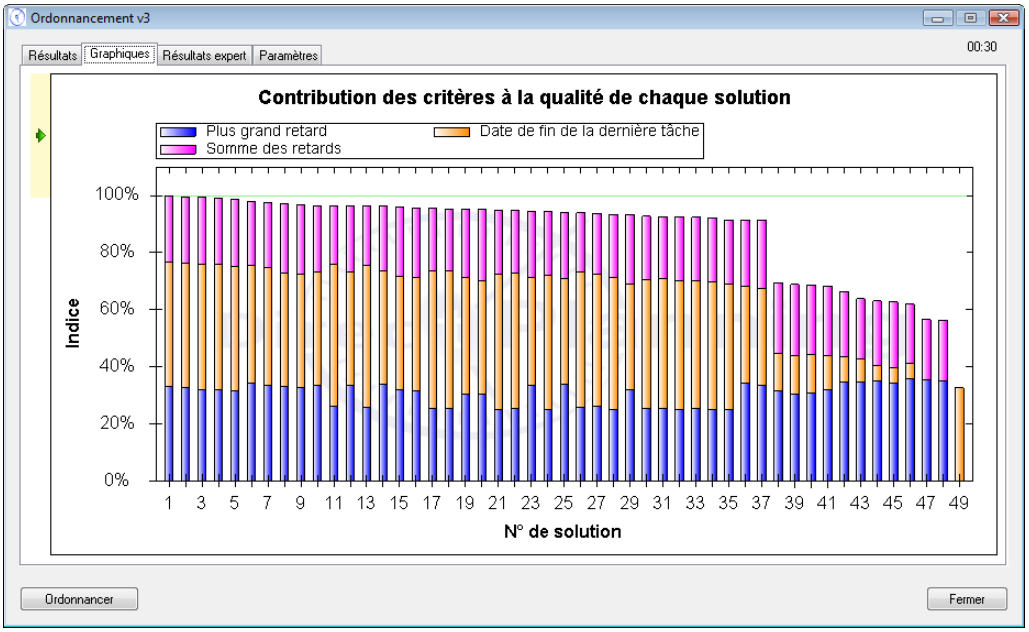

FIG. 5.7 – Résultats de l'ordonnancement sous forme graphique

5.3 Conclusion sur DirectPlanning

Dans ce chapitre nous avons présenté l'intégration du travail de recherche au sein d'un logiciel commercial. Un moteur d'ordonnancement basé sur nos travaux académiques a été développé. Cependant, il a fallu tenir compte des besoins « commerciaux » comme les opérations verrouillées ou les calendriers. Un part non négligeable du travail concernait la mise en place de l'interface manuelle permettant au décideur de réaliser lui même son ordonnancement en tenant compte de diverses contraintes.

Conclusion

Dans ce document, nous avons abordé des problèmes d'ordonnancement d'atelier issus du monde industriel. Nous avons proposé de résoudre ces problèmes pas des méthodes approchées.

Dans le premier chapitre, nous avons donné des généralités sur les problèmes d'ordonnancement et en particulier sur les problèmes de job shop flexible et de job shop multiressource. Nous avons aussi donné quelques bases de l'optimisation multicritère. Nous avons aussi présenté les schémas des principales méthodes que nous avons utilisées, à savoir la recherche Tabou et l'algorithme génétique.

Le second chapitre était dédié à l'état de l'art sur les problèmes présentés dans ce document. Nous avons commencé avec une présentation des principaux articles sur le problème de job shop flexible (et ses variantes), puis nous avons étudié les articles traitant des problèmes de job shop multiressource et enfin, nous nous sommes intéressés aux problèmes de job shop dans le cas multicritère.

Dans le chapitre trois, nous avons étudié les problèmes de job shop flexible multicritère. Nous avons proposé deux recherches Tabou, deux algorithmes génétiques (dont un qui peut être partiellement initialisé par une recherche Tabou) et un algorithme mémétique. Nous avons cherché à obtenir les meilleurs paramètres pour nos méthodes et nous les avons comparés entre elles. Nous avons étudié deux jeux de critères à optimiser, à savoir le couple date de fin de la dernière opération et plus grand retard et le couple date de fin de la dernière opération et somme des retards. Nous avons conclu que notre algorithme génétique à codage direct avec une initialisation de la population par une recherche Tabou donnait les meilleurs résultats. La recherche Tabou optimise une combinaison linéaire des critères, grâce à son voisinage connexe, toute solution peut être atteinte. L'algorithme génétique est basé sur le schéma NSGA-II, son codage direct lui permet d'avoir une bijection entre un individu et une solution au problème considéré. Ainsi nous pouvons injecter dans la population des solutions provenant d'autre algorithmes.

Dans le chapitre quatre, nous avons étudié les problèmes de job shop multiressource multicritère. Nous avons adapté notre recherche Tabou et notre algorithme génétique du chapitre trois. Nous avons aussi développé un programme linéaire en nombres entiers. Si ce dernier ne peut prétendre résoudre des instances industrielles, il nous a permis d'évaluer les bonnes performances de nos deux autres algorithmes. Comme pour le problème de job shop flexible, la combinaison de l'algorithme génétique et de la recherche Tabou présente les meilleurs performances.

Enfin, le chapitre cinq a présenté le logiciel *DirectPlanning*, principalement au niveau des apports de ce travail de recherche pour un logiciel commercial.

Les perspectives de recherche sont nombreuses. La première étant évidemment de pouvoir traiter le problème de job shop flexible multiressource multicritère. On peut déjà entre-apercevoir que le modèle de graphe pour le job shop flexible multiressource avec un sommet par sous-opération peut être étendu à ce problème.

Ensuite, le développement de procédures par séparation et évaluation multicritères permettrait, peut-être, d'évaluer de façon précise les performances de nos méta-heuristiques, au moins pour des instances de taille moyenne.

Enfin, une réflexion peut être menée autour de la pertinence de l'adaptation des critères classiques de littérature aux problèmes concrets du monde industriel. En effet, un critère comme le *Makespan* a peu de sens dans l'imprimerie puisque des travaux arrivent tous les jours. L'utilisation de critères spécifiques au métier, prenant en compte l'amortissement des machines par exemple, seraient plus intéressants pour le décideur. On peut penser aussi à des critères de type juste-à-temps pour minimiser le coût des stocks. Ces réflexions sur la pertinence des mesures de performance en ordonnancement dépassent bien entendu le cadre de cette étude.

Annexe A

Expérimentations préliminaires

A.1 Paramétrage des algorithmes pour le job shop flexible

Dans cette partie, nous cherchons à trouver les meilleurs paramètres pour nos algorithmes dédiés au problème de job shop flexible ($TS\epsilon$, $TS\ell c$, AG_{ind}, AG_{dir} et Mem). Ces expérimentations préliminaires utilisent les instances la01, la06, la11, la16, la21, la26, la31 et la36 pour chaque ensemble `sdata`, `edata`, `rdata` et `vdata`. Soit au total 32 instances différentes.

Les indicateurs sont ceux de la section 3.5.2 (p. 62). Dans les tableaux suivants, la valeur de SOw correspond au nombre de fois où l'ensemble des solutions non-dominées avec le paramètre considéré a dominé les autres. On utilise le même principe avec une moyenne au lieu d'une somme pour les indicateurs $\overline{NFC}$ et le temps d'exécution moyen.

A.1.1 Paramétrage des recherches Tabou

Les recherches Tabou possèdent chacune deux paramètres : le nombre maximum d'itérations sans amélioration $\#Iter$ et la taille de la liste tabou $|T\ell|$. Les plages de valeurs testées sont : $\#Iter \in \{100, 200, 500, 800, 1000\}$ et $|T\ell| \in \{10, 40, 70, 100, 150\}$.

Resultats pour $TS\epsilon$ Le tableau A.1 montre les résultats pour la recherche Tabou basée sur l'approche ϵ-contrainte en faisant varier le nombre maximum d'itérations sans amélioration. Pour ces expérimentations, $|T\ell|$ a été fixé à 100.

$\#Iter$	SOw	$\overline{NFC}$	Temps d'exécution moyen (s)
100	3	35%	53
200	13	51%	137
500	24	66%	339
800	41	78%	593
1000	48	81%	818

TAB. A.1 – Impact de $\#Iter$ sur les performances de $TS\epsilon$

Le temps d'exécution est clairement lié au paramètre $\#Iter$, nous avons cherché à avoir un bon compromis entre la qualité des solutions et le temps de calcul. La figure A.1 montre la variation de SOw et de $\overline{NFC}$ avec le temps de calcul.

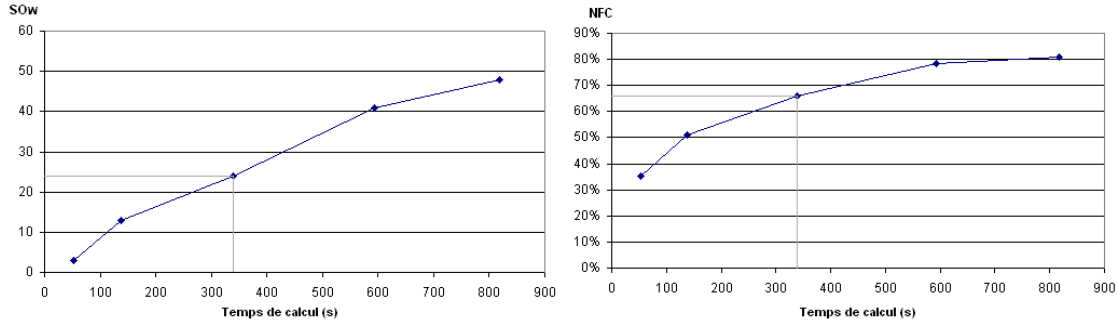

FIG. A.1 – Compromis entre la qualité et le temps de calcul selon la variation de $\#Iter$ pour $TS\epsilon$

On peut constater qu'au delà de 500 secondes de calculs, l'amélioration n'est pas significative pour chacune des deux mesures. Ainsi, pour l'algorithme $TS\epsilon$ nous avons décidé de fixer $\#Iter$ à 500.

Le tableau A.2 et la figure A.2 montrent les résultats pour $TS\epsilon$ selon la variation de $|T\ell|$. Pour ces expérimentations, $\#Iter$ a été fixé à 500.

$\|T\ell\|$	SOw	$\overline{NFC}$
10	0	15%
40	37	55%
70	46	67%
100	53	70%
150	53	71%

TAB. A.2 – Influence de $|T\ell|$ pour $TS\epsilon$

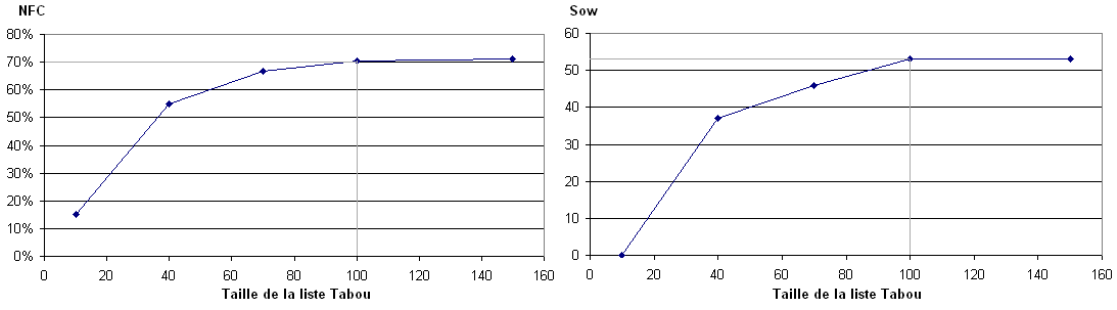

FIG. A.2 – Compromis entre la qualité et le temps de calcul selon la variation de $|T\ell|$ pour $TS\epsilon$

On peut constater que la qualité n'est pas beaucoup améliorée au dessus d'une taille

de liste tabou supérieure à 100. Ainsi, pour l'algorithme $TS\epsilon$ nous avons décidé de fixer $|T\ell|$ à 100.

Resultats pour $TS\ell c$ Le tableau A.3 et la figure A.3 montrent les résultats pour la recherche Tabou basée sur la combinaison linéaire des critères en faisant varier le nombre maximum d'itérations sans amélioration. Pour ces expérimentations, $|T\ell|$ a été fixé à 100.

$\#Iter$	SOw	$\overline{NFC}$	Temps d'exécution moyen (s)
100	1	19%	77
200	20	43%	155
500	47	68%	369
800	48	74%	547
1000	63	81%	650

TAB. A.3 – Impact de $\#Iter$ sur les performances de $TS\ell c$

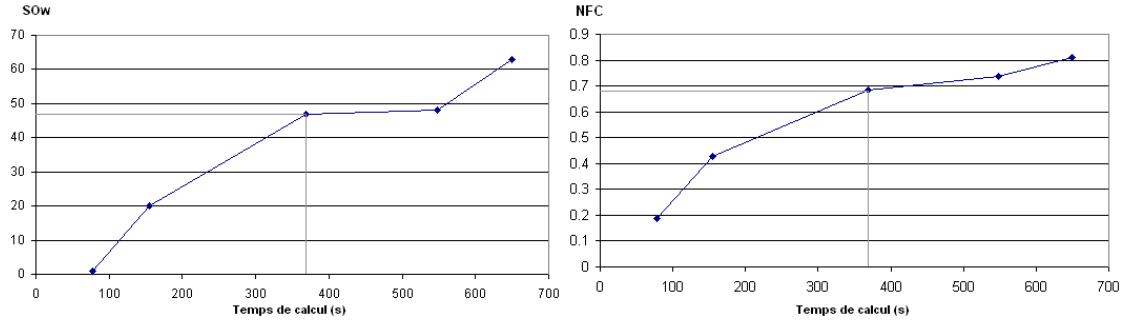

FIG. A.3 – Compromis entre la qualité et le temps de calcul selon la variation de $\#Iter$ pour $TS\ell c$

Comme pour $TS\epsilon$, on constate que le gain de qualité au-delà de 500 itérations est faible et demande une forte augmentation du temps de calcul. Nous avons décidé de fixer ce paramètre à 500.

Le tableau A.4 et la figure A.4 montrent les résultats pour $TS\ell c$ selon la variation de $|T\ell|$. Pour ces expérimentations, $\#Iter$ a été fixé à 500.

$\lvert T\ell\rvert$	SOw	$\overline{NFC}$
10	4	20%
40	23	50%
70	37	67%
100	47	69%
150	44	64%

TAB. A.4 – Influence de $|T\ell|$ pour $TS\ell c$

On constate ici que le meilleur choix pour la taille de la liste tabou est 100.

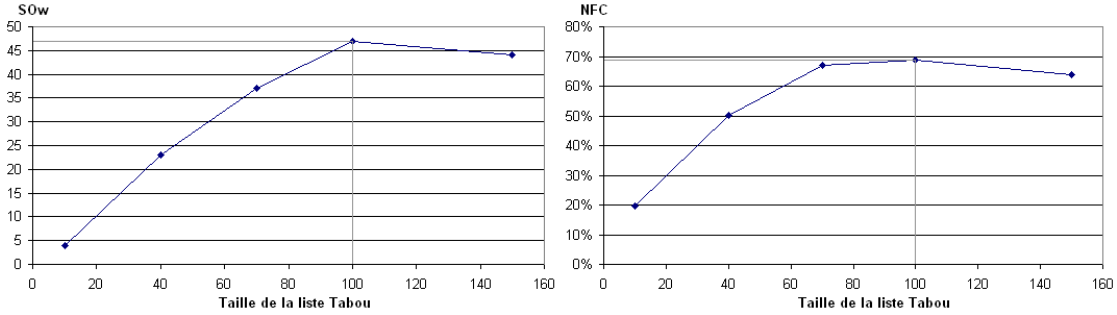

FIG. A.4 – Compromis entre la qualité et le temps de calcul selon la variation de $|T\ell|$ pour $TS\ell c$

A.1.2 Paramétrage de l'algorithme génétique à codage indirect

L'algorithme génétique à codage indirect AG_{ind} (cf. section 3.4.1) possède quatre paramètres à régler : la taille de la population N, le nombre maximum de générations sans amélioration $\#MaxGen$, le taux de renouvellement de la population tx et la probabilité de mutation Mut. Pour ces expérimentations, le temps d'exécution a été limité à 10 minutes.

Pour chaque ligne dans les tableaux A.5 à A.8, la valeur maximale pour SOw est 96 (32 instances $\times$ 3 paramètres différents à comparer).

La taille de la population On teste le paramètre N avec les valeurs $\{50, 100, 200, 500\}$. Le tableau A.5 montre les performances de AG_{ind} selon la taille de la population. Pour cette série, les autres paramètres ont été fixés à $\#MaxGen = 500$, $tx = 0.7$, $Mut = 0.1$.

N	SOw	$\overline{NFC}$
50	3	21%
100	12	63%
200	10	55%
500	40	85%

TAB. A.5 – Influence de N pour AG_{ind}

Clairement, une taille de population de 500 donne les meilleurs résultats. On peut cependant noter que l'augmentation de la taille de la population ne signifie pas pour autant une amélioration. En effet, d'après les résultats, il vaut mieux avoir $N = 100$ plutôt que $N = 200$.

Nombre maximum de générations sans amélioration Le paramètre $\#MaxGen$ peut prendre les valeurs $\{100, 500, 1000, 2000\}$. Le tableau A.6 montre les performances de AG_{ind} selon le nombre maximum de générations sans amélioration. Pour cette série, les autres paramètres ont été fixés à $N = 100$, $tx = 0.7$, $Mut = 0.1$.

$\#MaxGen$	SOw	$\overline{NFC}$
100	0	16%
500	64	91%
1000	50	84%
2000	89	100%

TAB. A.6 – Influence de $\#MaxGen$ pour AG_{ind}

Notons que sur les 96 exécutions de l'algorithme, seules 2 ont atteint les 10 minutes d'exécution, le 94 autres lancements ont atteint leur nombre maximal de générations. Dans ces conditions, on constate que la qualité augmente avec $\#MaxGen$. Avec $\#MaxGen = 2000$, AG_{ind} donne toujours le meilleur ensemble de solutions non-dominées (ie. $\overline{NFC} = 100\%$).

Taux de renouvellement de la population Le paramètre tx peut prendre les valeurs $\{0.3, 0.5, 0.7, 1\}$. Le tableau A.7 montre les performances de AG_{ind} selon le taux de renouvellement de la population. Pour cette série, les autres paramètres ont été fixés à $N = 100$, $\#MaxGen = 500$, et $Mut = 0.1$.

tx	SOw	$\overline{NFC}$
0.3	15	57%
0.5	5	54%
0.7	14	63%
1	0	19%

TAB. A.7 – Influence de tx pour AG_{ind}

On peut constater que la variation de ce paramètre a une influence plus faible sur la qualité. En effet, à part pour $tx = 1$, la variation des indicateurs est faible. On peut quand même remarquer que le meilleur paramètre est $tx = 0.7$.

Probabilité de mutation Le paramètre Mut peut prendre les valeurs $\{0.001, 0.01, 0.1, 0.5\}$. Le tableau A.8 montre les performances de AG_{ind} selon la probabilité de mutation. Pour cette série, les autres paramètres ont été fixés à $N = 100$, $\#MaxGen = 500$, et $tx = 0.7$.

Mut	SOw	$\overline{NFC}$
0.001	40	58%
0.01	48	67%
0.1	31	48%
0.5	0	7%

TAB. A.8 – Influence de Mut pour AG_{ind}

Ici, la meilleure probabilité de mutation est $Mut = 0.01$.

Au final d'après les expérimentations préliminaires, les meilleurs paramètres sont :
– $N = 500$

- $\#MaxGen = 2000$
- $tx = 0.7$
- $Mut = 0.01$

A.1.3 Paramétrage de l'algorithme génétique à codage direct

L'algorithme génétique à codage direct AG_{dir} (cf. section 3.4.2) possède deux paramètres : la taille de la population N et le probabilité maximale de mutation TM_{max}. Pour ces expérimentations, le temps d'exécution a été limité à 10 minutes.

On initialise partiellement la population avec des solutions issues de $TS\ell c$. Pour cela on a lancé $TS\ell c$ avec $\alpha \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}$. Le nombre maximum de générations sans amélioration a été fixé à 1000.

Pour chaque ligne dans les tableaux A.9 et A.10, la valeur maximale pour SOw est 96 (32 instances $\times$ 3 paramètres différents à comparer).

Taille de la population On teste le paramètre N avec les valeurs $\{50, 100, 200, 500\}$. Le tableau A.9 montre les performances de AG_{dir} selon la taille de la population. Pour cette série, la probabilité maximale de mutation a été fixée à $TM_{max} = 0.1$.

N	SOw	$\overline{NFC}$
50	8	52%
100	41	79%
200	37	79%
500	43	85%

TAB. A.9 – Influence de N pour AG_{dir}

Ici les meilleurs résultats sont obtenus avec $N = 500$.

Probabilité maximale de mutation On teste le paramètre TM_{max} avec les valeurs $\{0.001, 0.01, 0.1, 0.5\}$. Le tableau A.10 montre les performances de AG_{dir} selon la probabilité maximale de mutation. Pour cette série, la taille de la population a été fixée à $N = 100$.

TM_{max}	SOw	$\overline{NFC}$
0.001	10	50%
0.01	34	81%
0.1	42	75%
0.5	70	94%

TAB. A.10 – Influence de TM_{max} pour AG_{dir}

On constate que le meilleur paramètre est $TM_{max} = 0.5$. Cela peut sembler beaucoup comme probabilité de mutation. Cependant, ceci est une probabilité maximale et la mu-

tation porte sur un individu et non un gène comme c'est habituellement le cas pour un algorithme génétique.

Au final d'après les expérimentations préliminaires, les meilleurs paramètres sont :
- $N = 500$
- $TM_{max} = 0.5$

A.1.4 Paramétrage de l'approche mémétique

L'algorithme mémétique possède quatre paramètres : deux pour la partie algorithme génétique (la taille de la population N et la probabilité maximale de mutation TM_{max}) et deux pour la partie recherche Tabou (le nombre maximum d'itérations sans amélioration $\#Iter$ et la taille de la liste tabou $|T\ell|$).

Le temps d'exécution a été limité à 10 minutes.

Pour chaque ligne dans les tableaux A.11 à A.14, la valeur maximale pour SOw est 128 (32 instances $\times$ 4 paramètres différents à comparer).

Taille de la population La population a été testée avec quatre tailles différentes : $N \in \{50, 100, 150, 200, 300\}$. Le tableau A.11 montre l'influence N sur Mem. Pour cette série, $TM_{max} = 0.1$, $\#Iter = 100$ et $|T\ell| = 100$.

N	SOw	$\overline{NFC}$
50	28	62%
100	30	65%
150	26	62%
200	30	64%
300	20	49%

TAB. A.11 – Influence de N pour Mem

On peut constater que ce paramètre a assez peu d'influence sur la méthode. En effet, l'ordre de grandeur de SOw est de 25 alors que la valeur maximale est de 128, soit 23%. D'autre part, la contribution au front idéal est relativement élevée avec une valeur supérieure à 60%, si on excepte $N = 300$ qui est manifestement un mauvais choix pour N. On peut tout de même constater que la meilleure valeur est $N = 100$.

Probabilité maximale de mutation La probabilité maximale de mutation a été testée avec quatre valeurs différentes : $TM_{max} \in \{0.001, 0.01, 0.1, 0.5, 0.8\}$. Le tableau A.12 montre l'influence TM_{max} sur Mem. Pour cette série, $N = 100$, $\#Iter = 100$ et $|T\ell| = 100$.

A part pour la valeur $TM_{max} = 0.001$, ce paramètre à lui aussi une faible influence sur la méthode. On peut considérer que $TM_{max} = 0.8$ est un bon compromis pour ce paramètre.

TM_{max}	SOw	$\overline{NFC}$
0.001	13	48%
0.01	30	67%
0.1	37	64%
0.5	25	58%
0.8	29	67%

TAB. A.12 – Influence de TM_{max} pour Mem

Nombre maximum d'itérations sans amélioration dans les recherches Tabou Le nombre maximum d'itérations sans amélioration dans les recherches Tabou a été testé avec quatre valeurs différentes : $\#Iter \in \{50, 100, 200, 300, 500\}$. Le tableau A.13 montre l'influence $\#Iter$ sur Mem. Pour cette série, $N = 100$, $TM_{max} = 0.1$ et $|T\ell| = 100$.

$\#Iter$	SOw	$\overline{NFC}$
50	30	53%
100	25	65%
200	21	55%
300	38	66%
500	29	55%

TAB. A.13 – Influence de $\#Iter$ pour Mem

On remarque que ce paramètre a peu d'influence sur la méthode. On peut retenir que la meilleure valeur est $\#Iter = 300$.

Taille de la liste Tabou La liste Tabou a été testée avec quatre tailles différentes : $|T\ell| \in \{10, 50, 70, 100, 200\}$. Le tableau A.14 montre l'influence $|T\ell|$ sur Mem. Pour cette série, $N = 100$, $TM_{max} = 0.1$ et $\#Iter = 100$.

$\|T\ell\|$	SOw	$\overline{NFC}$
10	9	29%
50	32	59%
70	43	65%
100	39	65%
200	52	71%

TAB. A.14 – Influence de $|T\ell|$ pour Mem

On constate ici que la meilleure valeur est $|T\ell| = 200$.

Au final on peut remarquer que l'algorithme mémétique est très robuste aux variations de paramétrage. On peut tout de même retenir comme valeurs correctes :
- $N = 100$
- $TM_{max} = 0.8$
- $\#Iter = 300$
- $|T\ell| = 200$

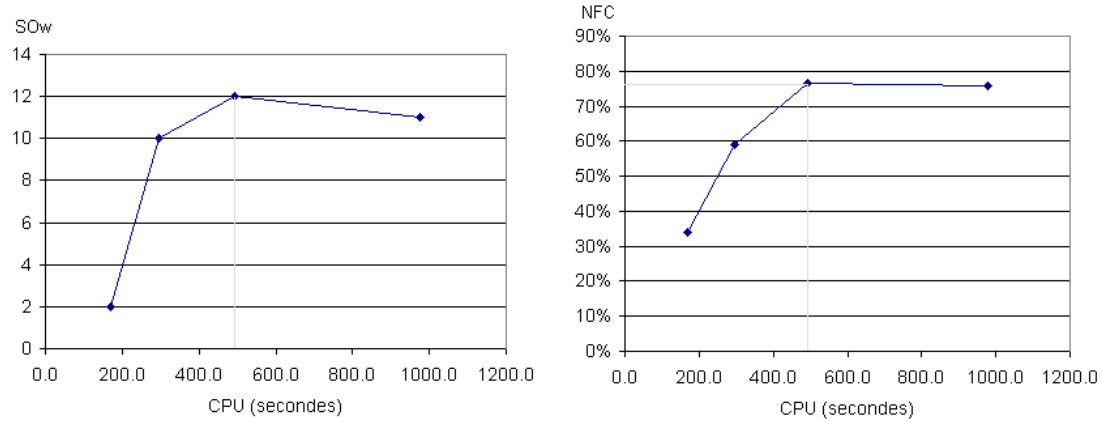

FIG. A.5 – Compromis entre la qualité et le temps de calcul selon la variation de $\#Iter$ pour TS_{MPT}

A.2 Paramétrage des algorithmes pour le job shop multiressource

Dans cette partie, nous cherchons à trouver les meilleurs paramètres pour nos algorithmes dédiés au problème de job shop multiressource (TS_{MPT} et AG_{MPT}). Ces expérimentations préliminaires utilisent les instances la01, la06, la11, la16, la21, la26, la31, la36, GJ01, GJ06, GJ11, GJ16, GJ21, GJ26 et GJ31. Soit au total 15 instances différentes.

A.2.1 Paramétrage de la recherche Tabou

La recherche Tabou TS_{MPT} possède un paramètre : le nombre d'itérations sans amélioration maximum, $\#Iter$. Les paramètres testés sont : $\#Iter \in \{500, 1000, 2000, 3000\}$. Le tableau A.15 montre les résultats en faisant varier le nombre d'itérations maximum sans amélioration. La figure A.5 montre l'évolution de la qualité en fonction du temps de calcul.

$\#Iter$	SOw	$\overline{NFC}$	Temps d'exécution moyen (s)
500	2	34%	169.3
1000	10	59%	294.8
2000	12	77%	494.1
3000	11	76%	976.9

TAB. A.15 – Impact de $\#Iter$ sur les performances de TS

On constate que le meilleur paramétrage est $\#iter = 2000$. On peut être surpris de la légère décroissance de qualité entre 2000 et 3000, cela peut s'expliquer par le fait que la méthode est stochastique.

A.2.2 Paramétrage de l'algorithme génétique

L'algorithme génétique pour le problème de job shop multiressource, AG_{MPT}, possède trois paramètres : la taille de la population N, le nombre de générations maximum sans amélioration $\#MaxGen$ et la probabilité de mutation Mut.

Taille de la population La population a été testée avec quatre tailles différentes : $N \in \{50, 100, 200, 500\}$. Le tableau A.16 montre l'influence N sur AG_{MPT}. Pour cette série, $Mut = 0.5$ et $\#MaxGen = 500$.

N	SOw	$\overline{NFC}$	Average computation time (s)
50	10	50%	294.0
100	13	60%	376.1
200	9	51%	364.5
500	12	56%	550.4

TAB. A.16 – Influence de N pour AG_{MPT}

On constate que la meilleure taille de population est $N = 100$.

Nombre maximum de générations sans amélioration Le nombre maximum de générations sans amélioration a été testé avec quatre valeurs différentes : $\#MaxGen \in \{100, 200, 500, 1000\}$. Le tableau A.17 montre l'influence $\#MaxGen$ sur AG_{MPT}. Pour cette série, $N = 100$ et $Mut = 0.5$.

$\#MaxGen$	SOw	$\overline{NFC}$	Average computation time (s)
100	11	51%	167.0
200	13	51%	251.1
500	8	49%	393.1
1000	19	67%	472.5

TAB. A.17 – Influence de $\#MaxGen$ pour AG_{MPT}

On constate que la meilleure valeur est $\#MaxGen = 1000$.

Probabilité de mutation La probabilité de mutation a été testée avec quatre valeurs différentes : $Mut \in \{0.001, 0.01, 0.1, 0.5\}$. Le tableau A.18 montre l'influence Mut sur AG_{MPT}. Pour cette série, $N = 100$ et $\#MaxGen = 500$.

La première chose qu'on peut remarquer est que Mut a une influence assez faible sur le temps d'exécution, en effet le temps de calcul est pus faible pour $Mut = 0.5$ que pour $Mut = 0.1$ et la qualité est meilleure.

Au final, le meilleur paramétrage pour AG_{MPT} est :
- $N = 100$

Mut	SOw	$\overline{NFC}$	Average computation time (s)
0.001	1	20%	157.9
0.01	8	43%	293.0
0.1	16	61%	351.0
0.5	31	81%	322.1

TAB. A.18 – Influence de Mut pour AG_{MPT}

- $\#MaxGen = 1000$
- $Mut = 0.5$

Bibliographie

[Adams *et al.*, 1988] Adams, J., Balas, E. et Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401.

[Alvarez-Valdes *et al.*, 2005] Alvarez-Valdes, R., Fuertes, A., Tamarit, J. M., Giménez, G. et Ramos, R. (2005). A heuristic to schedule flexible job-shop in glass factory. *European Journal of Operational Research*, 165:525–534.

[Artigues et Roubellat, 2000] Artigues, C. et Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127:297–316.

[Benayoun *et al.*, 1971] Benayoun, R., de MontGolfier, J., Tergny, J. et Laritchev, O. (1971). Linear programming with multiple objective functions : Step method (stem). *Mathematical Programming*, 1:366–375.

[Brandimarte, 1993] Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):158–183.

[Brucker *et al.*, 1997] Brucker, P., Jurisch, B. et Krämer, A. (1997). Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research*, 70:57–73.

[Brucker et Schlie, 1990] Brucker, P. et Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 44:369–375.

[Carlier et Pinson, 1989] Carlier, J. et Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

[Chambers et Barnes, 1996] Chambers, J. B. et Barnes, J. W. (1996). Flexible job shop scheduling by tabu search. Rapport technique, Departement of Computer Sciences, TAY 2.124, The University of Texas at Austin, Austin, Texas 78712.

[Chambers et Barnes, 1998] Chambers, J. B. et Barnes, J. W. (1998). Reactive search for flexible job shop scheduling. Rapport technique, Departement of Computer Sciences, TAY 2.124, The University of Texas at Austin, Austin, Texas 78712.

[Chen et Lee, 1999] Chen, J. et Lee, C.-Y. (1999). General multiprocessor task scheduling. *Naval Research Logistics*, 46:57–74.

[Cheng *et al.*, 1999] Cheng, T. E., Wang, G. et Sriskandarajah, C. (1999). One-operator-two-machine flowshop scheduling with setup and dismounting times. *Computer & Operations Research*, 26:715–730.

[Colorni *et al.*, 1991] Colorni, A., Dorigo, M. et Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. *In* Varela, F. et Bourgine, P., éditeurs : *Proceedings of the First European Conference on Artificial Life (ECAL)*, pages 134–142. MIT Press, Cambridge, Massachusetts.

[Czyak et Jaszkiewicz, 1996] Czyak, P. et Jaszkiewicz, A. (1996). A multiobjective metaheuristic approach to the localization of a chain of petrol stations by the capital budgeting model. *Control and Cybernetics*, 25(1):177–187.

[Dauzère-Pérès et Paulli, 1997] Dauzère-Pérès, S. et Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Resarch*, (70):281–306.

[Dauzère-Pérès *et al.*, 1998] Dauzère-Pérès, S., Roux, W. et Lasserre, J. B. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107:289–305.

[Dauzère-Pérès et Pavageau, 2001] Dauzère-Pérès, S. et Pavageau, C. (2001). Différencier les durées opératoires dans une approche intégrée pour l'ordonnancement multi-ressource. *In 3éme Conférence Francophone de MOdélisation et SIMulation "Conception, Analyse et Gestion des Systèmes Industriels".*

[Deb *et al.*, 2002] Deb, K., Pratap, A., Agarwal, S. et Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[Deckro *et al.*, 1982] Deckro, R. F., Hebert, J. E. et Winkofsky, E. P. (1982). Multiple criteria job-shop scheduling. *Computer and Operations Research*, 9:279–285.

[Demirkol *et al.*, 1998] Demirkol, E., Mehta, S. et Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141.

[Downsland, 1998] Downsland, K. (1998). Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407.

[Dupas, 2004] Dupas, R. (2004). *Amélioration de performance des systèmes de production : apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles.* HDR, Université d'Artois.

[Esquivel *et al.*, 2002] Esquivel, S., Ferrero, S., Gallard, R., Salto, C., Alfonso, H. et Schütz, M. (2002). Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem. *Knowledge-Based Systems*, 15:13–25.

[Garey et Johnson, 1979] Garey, M. R. et Johnson, D. S. (1979). *Computers and intractability : a guide to the theory of NP-completeness.* W.H. Freeman and Company, New York.

[Glover, 1989] Glover, F. (1989). Tabu search - part I. *ORSA Journal on Computing*, 1:190–206.

[Graham *et al.*, 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K. et Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5:287–326.

[Ho *et al.*, 2007] Ho, N. B., Tay, J. C. et Lai, E. M.-K. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316–333.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor.

[Huckert *et al.*, 1980] Huckert, K., Rhode, R., Roglin, O. et Weber, R. (1980). On the interactive solution to a multicriteria scheduling problem. *Zeitschrift für Operations Research*, 24:47–60.

[Hurink *et al.*, 1994] Hurink, J., Jurisch, B. et Thole, M. (1994). Tabu search for the job-shop scheduling problem with multipurpose machines. *OR Spektrum*, 15(4):205–215.

[Jaszkiewicz, 2004] Jaszkiewicz, A. (2004). *Metaheuristics for Multiobjective Optimisation*, chapitre Evaluation of multiple objective metaheuristic, pages 65–89. Springer.

[Jurisch, 1992] Jurisch, B. (1992). *Scheduling Jobs in Shops with Multi-Purpose Machines.* Thèse de doctorat, Universität Osnabrück.

[Kacem *et al.*, 2002] Kacem, I., Hammadi, S. et Borne, P. (2002). Pareto-optimality approach for flexible job-shop scheduling problems : hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60:245–276.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt, C. D. et Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

[Kis, 2003] Kis, T. (2003). Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151:307–332.

[Manne, 1960] Manne, A. (1960). On the job-shop scheduling problem. *Operations Research*, 8.

[Mastrolilli et Gambardella, 2000] Mastrolilli, M. et Gambardella, L.-M. (2000). Effective neighborhood function for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

[Mati et Xie, 2004] Mati, Y. et Xie, X. (2004). The complexity of two-job shop problems with multi-purpose unrelated machines. *European Journal of Operational Research*, 152:159–169.

[Mauguière, 2004] Mauguière, P. (2004). *Etude de problèmes d'ordonnancement disjonctifs avec contraintes de disponibilité des ressources et de préparation.* Thèse de doctorat, Université François-Rabelais Tours.

[Mauguière *et al.*, 2005] Mauguière, P., Billaut, J.-C. et Bouquard, J.-L. (2005). New single machine and job-shop scheduling problems with availability constraints. *Journal Of Scheduling*, 8:211–231.

[Mesghouni, 1999] Mesghouni, K. (1999). *Application des algorithmes évolutionnistes dans les problèmes d'optimisation en ordonnancement de la production.* Thèse de doctorat, Université des Sciences et Technologies de Lille.

[Nowicki et Smutnicki, 1993] Nowicki, E. et Smutnicki, C. (1993). A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813.

[Oğuz *et al.*, 2003] Oğuz, C., Ercan, M. F., Cheng, T. E. et Fung, Y. (2003). Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. *European Journal of Operational Research*, 149:390–403.

[Paulli, 1995] Paulli, J. (1995). A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research*, 86:32–42.

[Penz, 1994] Penz, B. (1994). *Constructions agrégatives d'ordonancements pour des job-shops statiques, dynamiques et réactifs.* Thèse de doctorat, Université Joseph Fourier - Grenoble 1.

[Perregaard, 1995] Perregaard, M. (1995). *Branch and bound method for the multi-processor Job shop and flow shop scheduling problem.* Thèse de doctorat, University of Copenhagen.

[Portmann et Vignier, 2001] Portmann, M.-C. et Vignier, A. (2001). *Ordonnancement de la production*, chapitre Algorithmes génétiques et ordonnancement, pages 95–130. Hermès.

[Roy et Sussmann, 1964] Roy, B. et Sussmann, B. (1964). Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS N.9 bis, SEMA, Paris.

[Serifoğlu et Ulusoy, 2004] Serifoğlu, F. S. et Ulusoy, G. (2004). Multiprocessor task scheduling in multistage hybrid flow-shops : a genetic algorithm approach. *The Journal of the Operational Research Society*, 55:504–512.

[Shachnai et Turek, 1999] Shachnai, H. et Turek, J. J. (1999). Multiresource malleable task scheduling to minimize response time. *Information Processing Letters*, 70:47–52.

[Suresh et Mohanasundaram, 2006] Suresh, R. et Mohanasundaram, K. (2006). Pareto archived simulated annealing for job shop scheduling with multiple objective. *International Journal of Advanced Manufacturing Technology*, 29:184–196.

[Tahar *et al.*, 2004] Tahar, D. N., Yalaoui, F., Amodeo, L. et Chu, C. (2004). Ordonnancement dans un atelier d'impression de type job shop hybride. *In 5éme Conférence Francophone de Modélisation et Simulation (MOSIM'04).*

[T'kindt et Billaut, 2006] T'kindt, V. et Billaut, J.-C. (2006). *Multicriteria scheduling : theory, models, algorithms. Second edition.* Springer, Berlin.

[Vilcot et Billaut, 2006] Vilcot, G. et Billaut, J.-C. (2006). Un algorithme génétique pour un problème de job-shop flexible multicritère. *In 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'06)*, Lille (France).

[Vilcot et Billaut, 2007a] Vilcot, G. et Billaut, J.-C. (2007a). Deux algorithmes tabou pour résoudre un problème de job shop flexible multicritère. *In 8ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'07)*, Grenoble (France).

[Vilcot et Billaut, 2007b] Vilcot, G. et Billaut, J.-C. (2007b). A memetic algorithm for a multicriteria flexible job shop scheduling problem. *In 22nd European Conference on Operational Research (EURO XXII)*, Prague (République Tchèque).

[Vilcot et Billaut, 2007c] Vilcot, G. et Billaut, J.-C. (2007c). A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. *International Journal of Production Research.* Seconde lecture.

[Vilcot et Billaut, 2007d] Vilcot, G. et Billaut, J.-C. (2007d). A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research*. doi :10.1016/j.ejor.2007.06.039, A paraître.

[Vilcot *et al.*, 2006a] Vilcot, G., Billaut, J.-C. et Esswein, C. (2006a). A genetic algorithm for a bicriteria flexible job shop scheduling problem. *In IEEE International Conference on Services Systems and Services Management (ICSSSM06)*, Troyes (France).

[Vilcot *et al.*, 2006b] Vilcot, G., Billaut, J.-C. et Esswein, C. (2006b). Une recherche tabou pour un problème de job-shop flexible bi-critère. *In* Gourgand, M. et Riane, F., éditeurs : *6ème Conférence Francophone de Modélisation et Simulation. "Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités" (MOSIM'06)*, volume 2, pages 1373–1381, Rabat (Maroc). Lavoisier. ISBN 2-7430-0892-X.

[Vilcot *et al.*, 2006c] Vilcot, G., Billaut, J.-C. et Esswein, C. (2006c). A tabu search algorithm for solving a bicriteria flexible job shop scheduling problem. *In* Dolgui, A., Morel, G. et Pere, C., éditeurs : *Information Control Problems In Manufacturing 2006, A Proceedings volume from the 12th IFAC International Symposium*, volume 3, pages 57–62, Saint-Etienne (France).

[Vilcot *et al.*, 2006d] Vilcot, G., Esswein, C. et Billaut, J.-C. (2006d). Multiobjective genetic algorithm for the flexible job shop scheduling problem. *In 7th International Conference on Multi-Objective Programming and Goal Programming (MOPGP'06)*, Tours (France).

[Wagner, 1959] Wagner, H. M. (1959). An integer linear programming model for machine scheduling. *Naval Research Logistic Quarterly*, 6.

[Zhang et Gen, 2005] Zhang, H. et Gen, M. (2005). Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal of Complexity International*, 11:223–232.

[Zitzler *et al.*, 2004] Zitzler, E., Laumanns, M. et Bleuler, S. (2004). *Metaheuristics for Multiobjective Optimisation*, chapitre A tutorial on Evolutionary Multiobjective Optimization, pages 3–37. Springer.

Résumé :

Ce travail de thèse s'inscrit dans le cadre d'une collaboration industrielle avec la société Volume Software pour le développement du module d'ordonnancement du logiciel « *DirectPlanning* ». Dans ce travail, nous étudions le problème de job shop flexible multicritère et le problème de job shop multiressource multicritère. Notre objectif est de déterminer une approximation du front de Pareto. Nous avons proposé des algorithmes de résolution approchés et plus particulièrement des algorithmes de recherche Tabou et des algorithmes génétiques. Nous avons proposé différentes versions de nos méthodes pour les deux problèmes considérés. Des expérimentations ont été réalisées et montrent les bonnes performances de nos algorithmes, à la fois d'un point de vue qualité des résultats et d'un point de vue de la rapidité des méthodes.

Mots clés :

Ordonnancement, job shop flexible, job shop multiressource, recherche Tabou, algorithme génétique, multicritère

Abstract :

This thesis was realized in the context of an industrial collaboration. In this work, we studied the multicriteria flexible job shop scheduling problem and the multicriteria multiprocessor tasks job shop scheduling problem. Our purpose was to find an approximation of the pareto frontier for these problems. Different meta heuristics were developed, such as Tabu search algorithms and genetic algorithms. Different versions of our methods were proposed for the two considered problems. Experiments show the good performances of our algorithms, both qualitative and computational aspects of our methods show satisfying results . This thesis was done in collaboration with Volume Software company within the framework of the development of the '*DirectPlanning*' scheduling engine.

Keywords :

Scheduling, flexible job shop, multiprocessor tasks job shop, Tabu search algorithm, genetic algorithm, multicriteria

---

Université François-Rabelais de Tours, Laboratoire d'Informatique, EA 2101, Équipe Ordonnancement et Conduite (http ://www.li.univ-tours.fr).

Polytech'Tours, Département Informatique, 64 Avenue Jean Portalis, 37200 Tours (http ://www.polytech.univ-tours.fr).