Une approche génétique pour la résolution d'ordonnancements cycliques

THÈSE

présentée et soutenue publiquement le 11 Décembre 2000

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Automatique)

par

Guillaume Cavory

Composition du jury

Président :	Christian Tahon	Professeur à l'Université de Valenciennes
Rapporteurs :	Claire Hanen	Professeur à l'Université Paris X
	Abdelhakim Artiba	Professeur à la Faculté Universitaire Catholique de Mons
Examinateurs :	Gilles Goncalves	Professeur à l'Université d'Artois (Directeur de thèse)
	Daniel Jolly	Professeur à l'Université d'Artois
	Talbi El-Ghazali	Maître de conférence à l'Université de Lille I
	Hammadi Slim	Maître de conférence à l'Ecole centrale de Lille
	Rémy Dupas	Maître de conférence à l'Université d'Artois (codirecteur de thèse)

Université d'Artois – LAboratoire en Organisation et Gestion de la Production
rue de l'Université, 62408 Béthune Cedex
Tél : +33 (0)3 21 63 71 64 – Fax : +33 (0)3 21 61 17 80

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier en premier lieu Rémy pour qui cette page ne suffirait pas à lui témoigner ma profonde gratitude. Denis Fourmaux pour ses conseils, ses remarques pertinentes et le réconfort que j'ai trouvé en sa présence.

Mon directeur de thèse, Gilles Goncalves qui m'enseigna la rigueur d'un travail de longue haleine et l'ABC d'une méthode de recherche.

Monsieur Abdelhakim Artiba, professeur à la Faculté Universitaire Catholique de Mons et Madame Claire Hanen, professeur à l'Université Paris X, pour avoir accepté d'être rapporteurs de cette thèse et pour les remarques et conseils éclairés qu'ils m'ont donnés et qui ont contribué à améliorer ce manuscrit.

Je remercie vivement Monsieur Hammadi Slim, habilité à diriger des recherches à l'école centrale de Lille, Monsieur Christian Tahon, professeur à l'université de Valenciennes, Monsieur El-Ghazali Talbi, habilité à diriger des recherches à l'université de Lille I et Monsieur Daniel Jolly, professeur à l'université d'Artois pour avoir étudié ce travail et avoir accepté de participer à ce jury.

Stéphanie, Ange-Marie et Anne Marie pour la relecture de ce document.

Christophe Lecoutre pour m'avoir fait découvrir le LABOGP et ses membres remplis de bonnes humeurs et toujours prêt à donner de bons conseils.

Les enseignants du département OGP pour leur accueil et leur joie de vivre.

Ma famille et les amis pour leur soutien moral indispensable,

plus tous ceux que j'ai oubliés qui je l'espère me pardonneront.

À Stéphanie,

Table des matières

Table des figures

Liste des tableaux

Table des algorithmes

Introduction générale

Le travail présenté dans cette thèse traite de la résolution de problèmes d'ordonnancement cyclique. Deux applications sont présentées dans cette thèse. La première application traite du problème académique du Job-Shop cyclique. La seconde application traite de l'amélioration de performances d'une ligne de fabrication industrielle de type grande série soumise à des tâches de maintenance systématique. Ces deux applications sont composées de tâches cycliques à ordonnancer. L'ordonnancement de ces tâches cycliques est le principal objectif de cette thèse.

Le premier chapitre présente la problématique de l'ordonnancement. Dans ce chapitre, nous rappelons les différents éléments qui composent un problème d'ordonnancement, ainsi que les caractéristiques permettant de classer les solutions. Nous présentons une méthode de modélisation de ces problèmes par des graphes de précédences. Après avoir rappelé une notation permettant de classer les problèmes d'ordonnancement, nous nous sommes focalisés sur les problèmes d'ordonnancement d'atelier. Une liste non-exhaustive de méthodes de résolution de ces problèmes est proposée. Cette liste se décompose en deux parties principales ; nous trouvons les méthodes exactes et les méthodes approchées.

Le deuxième chapitre présente les algorithmes génétiques comme méthode approchée de résolution des problèmes d'ordonnancement d'atelier. Pour cela, une description complète du fonctionnement des algorithmes génétiques est faite. Plusieurs méthodes de résolution des problèmes d'ordonnancement basées sur les algorithmes génétiques sont détaillées. Nous présentons particulièrement les méthodes permettant de résoudre le problème d'atelier qu'est le Job-Shop. Ces méthodes étant souvent basées sur un codage utilisant des séquences, des opérateurs génétiques spécialement conçus pour ce type de codage sont présentés.

Le troisième chapitre se focalise sur les problèmes d'ordonnancement dont les tâches sont ordonnancées une infinité de fois. Ce type d'ordonnancement est dit cyclique. Les graphes de précédences à contraintes linéaires sont utilisés afin de modéliser ce type de problème. Nous nous focalisons ensuite sur le problème de Job-Shop cyclique. Après avoir détaillé ce problème, nous proposons une méthode génétique pour le résoudre. Un ordonnanceur est utilisé afin de construire la solution. Cet ordonnanceur utilise un graphe de précédences à contraintes linéaires afin de construire l'ordonnancement. La gestion des occurrences des tâches est basée sur le principe des réseaux de Petri T-temporisé. Les conflits de ressources sont gérés par des heuristiques qui sont définies par la séquence fournie par l'algorithme génétique. Un ensemble d'essais a été mis au point afin de valider le comportement de l'ordonnanceur. Ces essais ont permis de vérifier la qualité des résultats fournis par le couplage de l'algorithme génétique et de l'ordonnanceur, appelé par la suite GA-ORDO. Un autre ensemble de jeux d'essais a été construit afin de pouvoir comparer GA-ORDO avec de futures extensions de cette méthode. Ces jeux d'essais proposent des exemples de Job-Shop cycliques. La construction de ces essais ainsi que des instances sont détaillées.

Une application industrielle est présentée dans le chapitre quatre. Cette application a pour but d'améliorer la productivité d'une ligne de fabrication. Un plan d'expériences a prouvé que les tâches de maintenance systématique sont la principale cause de non-productivité de cette ligne de production.

Pour répondre à ce type de problème industriel, une première approche a été développée. Elle est basée sur le couplage d'un algorithme génétique et d'un simulateur à évènements discrets que nous avons développé. Nous avons ensuite généralisé ce type de problème en le ramenant à un problème d'ordonnancement cyclique que nous avons résolu par une approche basée sur les graphes de précédences à contraintes linéaires.

Pour valider nos deux approches de résolution, nous avons développé une plate-forme modulaire d'analyses et d'amélioration de performances. Elle est composée d'un module d'édition de modèle, d'un module d'analyse basé sur les plans d'expériences, d'un module d'optimisation basé sur les algorithmes génétiques et d'un module de simulation.

La conclusion présente quelques pistes d'évolution de l'ordonnanceur, les études à réaliser afin de maîtriser totalement l'algorithme génétique ainsi que des extensions futures de la plate-forme.

Chapitre 1

L'ordonnancement

L'ordonnancement fait appel à une terminologie précise. Les méthodes de résolution sont très souvent adaptées à un seul type de problème. C'est pour cela qu'il est nécessaire de bien maîtriser le vocabulaire et d'être capable de classer les problèmes d'ordonnancement.

Ce chapitre se propose de présenter dans le détail les constituants d'un problème d'ordonnancement et les outils permettant de visualiser graphiquement un ordonnancement. Une notation permettant de classer les problèmes d'ordonnancement y est également présentée. Quelques notions de complexité sont exposées afin de mieux comprendre la difficulté de résolution de certains problèmes. La fin de ce chapitre se focalise sur la résolution des problèmes d'ordonnancement. Pour cela, une liste non-exhaustive de méthodes exactes ou approchée est proposée. Cette liste s'intéresse plus particulièrement aux problèmes d'atelier qui représentent une sous-catégorie des problèmes d'ordonnancement.

1.1 Concepts de base

Nous allons dans un premier temps aborder les concepts de bases qui permettent de définir, noter, caractériser et classer un problème d'ordonnancement. Nous terminerons cette partie en présentant les problèmes d'atelier. Mais avant tout, la définition 1.1 est utilisée tout le long de cette thèse comme définition de base d'un ordonnancement.

Définition 1.1 *Ordonnancer un ensemble de tâches, c'est programmer dans le temps leur exécution.*

1.1.1 Éléments d'un problème d'ordonnancement

Un ordonnancement est constitué de quatre éléments qui sont les tâches, les ressources, les contraintes et les objectifs. Les paragraphes suivants se proposent de définir ces quatre éléments.

1.1.1.1 Les tâches

Une tâche i est une **entité élémentaire** de travail. Elle est caractérisée par une **date de début** s_i ou de **fin** c_i, appelée aussi "Completion Time", et une **durée de réalisation** p_i. Une tâche peut également être définie par une latence q_i, qui est définie comme un minorant de la durée entre la fin de la tâche i et la fin de l'ordonnancement. Une tâche i peut consommer des **moyens** m avec une intensité I_{im}. Une tâche peut se réaliser par morceaux, on parle alors de tâche **préemptive**, ou sans interruption, on parle alors de tâche **non-préemptive** [Carlier et al. 1988].

1.1.1.2 Les ressources

Une ressource est un *moyen* technique ou humain dont la **disponibilité** est connue à priori. La capacité d'une ressource peut être **limitée** ou **illimitée**.

Il existe plusieurs types de ressource. Si après son utilisation la ressource est à nouveau disponible avec la même capacité, on parle de ressource **renouvelable** Les hommes et les machines sont des ressources renouvelables. Par contre, si après son utilisation la ressource est disponible avec une quantité inférieure ou nulle, on parle de ressource **consommable** [Slowinski 1982]. L'argent est un bon exemple de ressource consommable.

Une ressource **cumulative** est une ressource capable de réaliser plusieurs tâches à la fois. Par contre si cette ressource est limitée à la réalisation d'une et une seule tâche à la fois cette ressource est dite **disjonctive**. Un homme seul est une ressource disjonctive, par contre, une équipe est une ressource cumulative.

1.1.1.3 Les contraintes

Une contrainte est une restriction sur les valeurs que peuvent prendre les variables. C'est ainsi que les éléments qui composent un problème d'ordonnancement sont assimilables à des contraintes. Il existe deux types de contraintes.

- Temporelles : Ce type de contrainte permet de déterminer une date d_i butoir à laquelle la tâche doit être finie : $s_i + p_i \leq d_i$; cette date butoir est également appelée "deadline". Les contraintes temporelles permettent également de déterminer des enchaînements entre des tâches i et j : $s_i + p_i \leq s_j$. Ce type de contraintes est également appelé contraintes de précédences. Les contraintes temporelles permettent également de localiser une tâche dans le temps : $t_1 \leq s_i \leq t_2$. Ici, t_1 représente une date au plus tôt et t_2 représente une date au plus tard.
- Ressources : Les caractéristiques de ressources peuvent induire des contraintes. Dans le cas d'une ressource consommable, il faut que la consommation de la ressource ne soit pas supérieure à la capacité de la ressource. Par contre, pour une ressource renouvelable, ce type de contraintes permet de limiter sa consommation. Cette limitation peut être due à une ressource disjonctive ou à une ressource cumulative.
  - Disjonctive : la réalisation des tâches se fait sur des intervalles de temps disjoints. Par exemple, la tâche 1 doit se réaliser avant la tâche 2 ou après. "$s_1+p_1 \leq s_2$ OU $s_2+p_2 \leq s_1$"
  - Cumulative : elle limite le nombre de tâches à réaliser en parallèle pour la ressource compte tenu de sa disponibilité et des quantités requises par les tâches. La disponibilité de la ressource peut être une fonction $A_i(t)$ du temps. Par exemple, si $A_1(t)$ est la disponibilité de la ressource 1, et $\Omega_1(t)$ l'ensemble des tâches affectées à la ressource 1 au cours du temps, alors, l'équation suivante doit toujours être vérifiée : $\forall t$, $\Sigma_{i \in \Omega_1(t)} I_{i1} \leq A_1(t)$. C'est à dire que la somme des intensités des tâches affectées à la ressource doit être, à chaque instant, inférieure ou égale à la capacité de la ressource. La figure 1.1 représente un exemple de charge pour la ressource 1. La partie grisée représente l'occupation $\Omega_1(t)$ de la ressource 1. Dans cette exemple, la disponibilité de la ressource 1 est constante dans le temps.

1.1.1.4 Objectifs et fonction d'évaluation

L'objectif d'un problème d'ordonnancement est de maximiser ou minimiser une fonction d'évaluation. Cette fonction est déterminée à partir d'un ou plusieurs *critères d'évaluation*. Les fonctions d'évaluation les plus courantes sont [GOTHA 1993] :

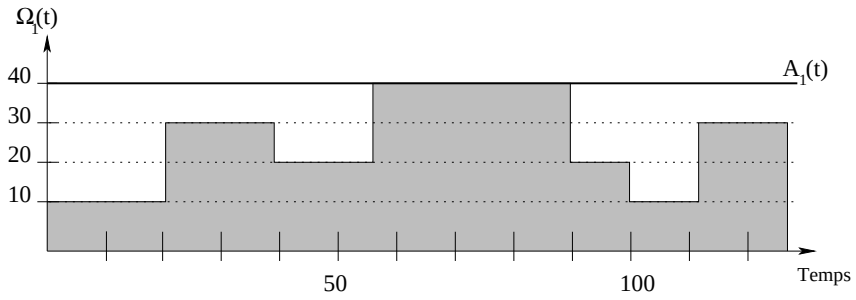

FIG. 1.1 – Courbe de consommation

- C_{max} : le temps de réalisation de l'ordonnancement, appelé également *makespan*. $C_{max} = max_{i \in T}\{c_i\}$, où T représente l'ensemble des tâches à ordonnancer et c_i est la date de fin de réalisation de la tâche i.
- L_{max} : l'écart par rapport au délai souhaité avec avance favorable et retard défavorable, appelé Lateness . Avec : $L_{max} = max_{i \in T}\{c_i - d_i\}$

Il est possible de rencontrer d'autres critères d'évaluation pour un ordonnancement. Ceux-ci sont basés sur des sommes pondérées ou non. Voici une liste non exhaustive de ces critères :

- $\sum_{i \in T} c_i$: somme des temps de réalisation
- $\sum_{i \in T} \mathcal{T}_i$: somme des retards dans l'ordonnancement, appelée aussi Tardiness . Avec : $\mathcal{T}_i = max\{L_i; 0\}$ et $L_i = c_i - d_i$.
- $\sum_{i \in T} w_i.c_i$: somme pondérée des temps de réalisation. Ici, w_i représente un poids associé à la tâche i qui traduit l'importance de la tâche i par rapport aux autres tâches..
- $\sum_{i \in T} w_i.\mathcal{T}_i$: somme pondérée des retards dans l'ordonnancement.

La fonction d'évaluation peut également porter sur le nombre de contraintes non satisfaites dans l'ordonnancement. Dans ce cas de figure, l'optimum est un ordonnancement admissible.

1.1.2 Caractéristiques d'un ordonnancement

Il est possible de caractériser un ordonnancement en fonction des améliorations de la performance sur un critère régulier qui sont encore réalisables sur ce dernier [Baker 1974]. Il existe ainsi des sous-ensembles d'ordonnancements dont certains présentent des propriétés de dominance vis à vis de tous critères réguliers.

Définition 1.2 *Un critère est dit* ***régulier*** *si on ne peut pas dégrader sa performance en avançant l'exécution d'une tâche.*

1.1.2.1 Ordonnancement admissible

Définition 1.3 *Un ordonnancement est dit* ***admissible*** *s'il respecte toutes les contraintes constituant le problème.*

Dans le but de mieux comprendre les différentes caractéristiques d'un ordonnancement, nous allons utiliser l'exemple 1.1.

Exemple 1.1 Soit cinq tâches 1,2,3,4,5. Ces tâches sont soumises aux contraintes de précédences suivantes : 1<2<5 et 3<4. La ressource capable de réaliser ces tâches est une ressource limitée, renouvelable et cumulative. La figure 1.2 représente un ordonnancement admissible.

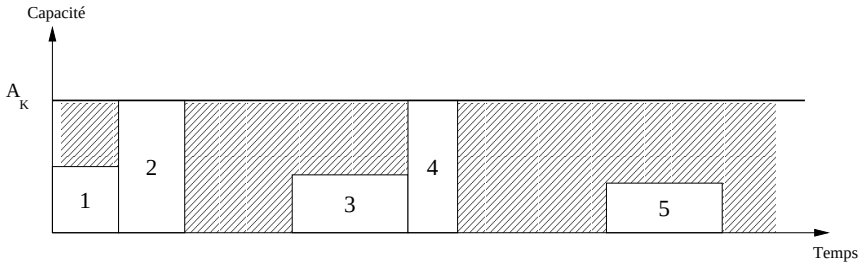

FIG. 1.2 – Ordonnancement admissible

Afin d'améliorer un ordonnancement pour un critère régulier , il est possible de réaliser un *glissement à gauche*. Ceci consiste à avancer dans le temps une ou plusieurs tâches. Il existe deux types de glissement à gauche :

- *Le glissement à gauche local* consiste à avancer le début d'une tâche sans remettre en cause l'ordre relatif entre les tâches. Sur l'exemple de la figure 1.2, l'ordonnancement pourrait être amélioré du point de vue du temps d'exécution en faisant glisser les tâches 3,4 et 5 sur la gauche.
- *Le glissement à gauche global* consiste à avancer le début d'une tâche en modifiant l'ordre relatif entre au moins deux tâches. Sur l'exemple précédent, le repositionnement de la tâche 5 juste au dessus de la tâche 3 est admissible. Ce repositionnement change la relation $4<5$ en $5<4$, mais ne viole aucune contrainte.

Définition 1.4 *L'*ordre relatif *est l'ordre de passage dans lequel les tâches devront être réalisées sur une ressource.*

1.1.2.2 Ordonnancement semi-actif

Dans un ordonnancement semi-actif, aucun *glissement à gauche local* n'est possible : on ne peut plus avancer une tâche sans modifier la séquence sur la ressource. Sur l'ordonnancement admissible de la figure 1.2, un glissement à gauche local permet par exemple d'obtenir l'ordonnancement semi-actif de la figure 1.3.

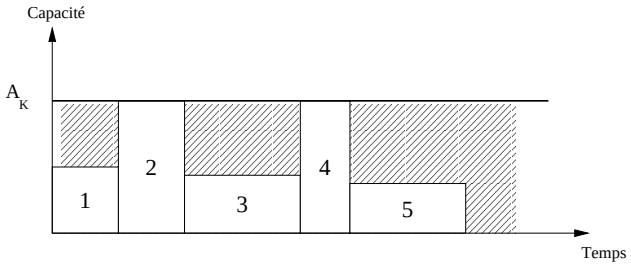

FIG. 1.3 – Ordonnancement semi-actif

1.1.2.3 Ordonnancement actif

Dans un ordonnancement actif aucun *glissement à gauche local* ou *global* n'est possible. Aucune tâche ne peut être commencée plus tôt sans reporter le début d'une autre. La figure 1.4 présente un ordonnancement actif de l'exemple 1.1.

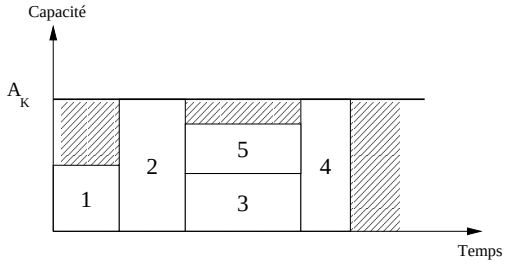

FIG. 1.4 – Ordonnancement actif

1.1.2.4 Ordonnancement sans retard

Dans un ordonnancement sans retard, si une tâche est en attente et si la ressource nécessaire à son exécution est disponible alors on ne doit pas retarder l'exécution de cette tâche. La figure 1.5 présente un ordonnancement sans retard pour l'exemple 1.1.

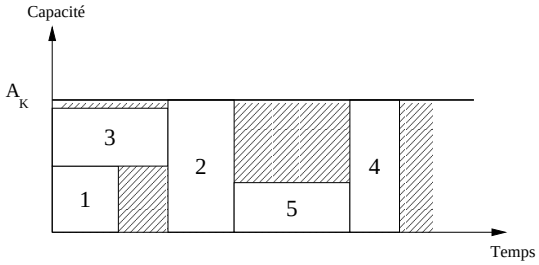

FIG. 1.5 – Ordonnancement sans retard

Propriété 1.1 *Un ordonnancement sans retard est actif [Esquirol et al. 1999].*

1.1.2.5 Classification des ordonnancements

Définition 1.5 *Un sous-ensemble de solutions est dit* ***dominant*** *pour l'optimisation d'un critère donné, s'il contient au moins un optimum pour ce critère. La recherche d'une solution optimale ou admissible peut ainsi se limiter au sous ensemble dominant.*

C'est ainsi que si quelqu'un cherche le point culminant d'un pays, il peut focaliser sa recherche sur les massifs montagneux car l'ensemble des massifs montagneux est un sous-ensemble dominant sur le critère de la hauteur.

Propriété 1.2 *Il est important de noter que l'ensemble des ordonnancements semi-actifs est dominant pour tous critères réguliers [Esquirol et al. 1999].*

Propriété 1.3 *Un ordonnancement actif est semi-actif. De plus, l'ensemble des ordonnancements actifs est également dominant pour tous critères réguliers [Esquirol et al. 1999].*

Pour l'optimisation d'un critère régulier, il est suffisant de considérer l'ensemble des ordonnancements semi-actifs qui domine l'ensemble de tous les ordonnancements. Cependant, leur nombre important amène à considérer souvent les hypothèses plus restrictives des ordonnancements actifs pour améliorer la recherche de solutions optimales. L'ensemble des ordonnancements actifs s'avère être le plus petit ensemble dominant.

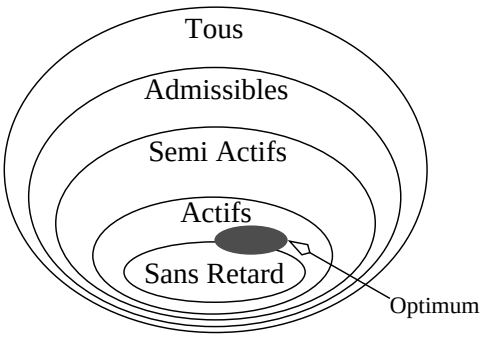

FIG. 1.6 – Classification des ordonnancements pour un critère régulier

L'ensemble des ordonnancements sans retard est un sous-ensemble des ordonnancements actifs, mais ne constitue pas un ensemble dominant vis à vis d'un critère régulier [Bellman et al. 1982]. Néanmoins, il est préférable de générer des ordonnancements sans retard qui constituent généralement de bonnes solutions et dont la construction est plus aisée. En effet, leur nombre est sensiblement moins élevé que celui des ordonnancements actifs.

La figure 1.6 présente une classification existante entre les différentes propriétés des ordonnancements sur un critère régulier ainsi que la position de l'optimum par rapport aux propriétés de ceux-ci [Bierwirth et al. 1999].

Ces diverses notions seront exploitées dans la suite pour limiter l'espace de recherche des solutions. On verra notamment les conditions de dominance qu'il est possible d'établir selon les problèmes traités et les caractéristiques des ordonnancements.

1.1.3 Les problèmes d'ordonnancement d'atelier

Dans les problèmes d'atelier, un ensemble $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ de n *jobs*, appelé aussi travaux, doit être réalisé dans un atelier composé de m machines.

Un job i est caractérisé par un ensemble $T_i = \{o_{i1}, o_{i2}, ..., o_{ix}\}$ de x tâches à réaliser et par une *gamme* $\mathcal{G}_i$. La gamme $\mathcal{G}$ représente la *séquence* à suivre pour réaliser les tâches. Cette séquence décrit les *contraintes de précédences* à respecter. Un job J_i se note de la façon suivante : $J_i = \{T_i \mid \mathcal{G}_i\}$. Ainsi, un job J_1 composé de trois tâches $\{o_{11}, o_{12}, o_{13}\}$ devant être réalisées dans l'ordre suivant : o_{11} puis o_{13} puis o_{12}, se note de la façon suivante :

$$J_1 = \{\{o_{11}, o_{12}, o_{13}\} \mid o_{11} < o_{13} < o_{12}\}$$

Ici, $o_{11} < o_{13} < o_{12}$ représente la gamme $\mathcal{G}_1$ du job J_1. Pour réaliser une tâche, il faut une machine (ressource) qui est dédiée à cette tâche pendant la durée p_{ij} (durée de réalisation de la tâche j sur la machine i). Les machines sont des ressources limitées, renouvelables et disjonctives.

Une classification sur les problèmes d'atelier peut s'opérer sur la gamme des jobs et sur l'organisation des ressources. On rencontre le *Flow-Shop*, le *Job-Shop* et l'*Open-Shop*.

1.1.3.1 Flow-Shop

Le Flow-Shop est rencontré sur des lignes de production dédiées à la production de grande série où les changements de type de produit sont rares. Ici, pour réaliser un 'job', il faut n tâches. Les tâches sont exécutées par les machines et les 'jobs' sont les 'programmes' de fabrication. Ces derniers renferment des données telles que le type de pièce à produire ou encore le nombre de

pièces à produire. En général, pour une tâche on trouve une seule machine dédiée, mais il est possible de trouver plusieurs machines pour réaliser une tâche. On parle de 'Flow-Shop hybride'.

Un cas particulier important est celui d'un Flow-Shop simplifié où la séquence des travaux affectée à la première machine est la même pour toutes les autres machines. On peut encore dire que les machines sont gérées selon une règle *FIFO* (First In First Out) qui impose que les travaux quittent la machine dans le même ordre que celui dans lequel ils sont arrivés. Ce cas particulier est appelé **Flow-Shop de permutation** .

On dit également que les ateliers de type Flow-Shop, sont des ateliers à *cheminement unique* où toutes les gammes sont identiques. Seuls les temps de réalisation des activités (p_{ij}) peuvent être différents. Voici une représentation d'un Flow-Shop à deux jobs :

$$J_1 = \{ \{o_{11}, o_{12}, \dots, o_{1n}\} \mid o_{11} < o_{12} < \dots < o_{1n} \}$$
$$J_2 = \{ \{o_{21}, o_{22}, \dots, o_{2n}\} \mid o_{21} < o_{22} < \dots < o_{2n} \}$$

1.1.3.2 Job-Shop

Le Job-Shop est associé à des lignes de production dédiées à la production de moyenne et petite série où les changements de type de produit sont fréquents. Comme pour le Flow-Shop, pour réaliser un 'job' il faut n tâches. Mais contrairement à ce dernier, dans un Job-Shop, les gammes sont différentes. Ceci veut dire que l'ordre d'exécution des tâches est différent d'un job à un autre et que le nombre de tâches qui compose les jobs peut également être différent d'un job à l'autre. On parlera d'atelier à cheminement multiple. Voici une représentation d'un Job-Shop à deux jobs :

$$J_1 = \{ \{o_{11}, o_{12}, \dots, o_{1n}\} \mid o_{11} < o_{12} < \dots < o_{1n} \}$$
$$J_2 = \{ \{o_{21}, o_{22}, \dots, o_{2m}\} \mid o_{22} < o_{24} < \dots < o_{21} \}$$

1.1.3.3 Open-Shop

Dans un atelier Open-Shop, le cheminement des jobs est multiple, mais à la différence du Job-Shop, ici, les jobs ne possèdent pas de gamme. Nous pouvons citer en exemple l'organisation des examens médicaux dans un hôpital. Chaque personne doit réaliser une suite d'examens dont l'ordre de réalisation n'a pas d'importance. On parle d'atelier à cheminement libre. Voici la représentation d'un Open-Shop à deux jobs :

$$J_1 = \{o_{11}, o_{12}, ..., o_{1n}\}$$
$$J_2 = \{o_{21}, o_{22}, ..., o_{2m}\}$$

Le tableau 1.1 présente un résumé des différents éléments qui composent un ordonnancement ainsi que les "valeurs" qu'ils peuvent prendre [GOTHA 1993].

1.1.3.4 Notation

Dans le but de décrire rapidement un problème d'ordonnancement, une notation a été adoptée. Cette notation permet également de classer les problèmes d'ordonnancement par famille. La notation retenue est celle décrite par Graham en 1979 [Graham et al. 1979] qui est étendue par Lawler [Lawler et al. 1985], et par Brucker [Brucker 1995]. Cette notation est basée sur la concaténation de trois champs α, β et γ. Un problème d'ordonnancement se note donc : $\alpha \mid \beta \mid \gamma$ avec :

<table>
<tr><td>Caractéristiques</td><td colspan="5">Choix possibles</td></tr>
<tr><td rowspan="3">Ressources</td><td colspan="2">illimitées</td><td colspan="3">limitées</td></tr>
<tr><td colspan="2">consommables</td><td colspan="3">renouvelables</td></tr>
<tr><td colspan="2">cumulatives</td><td colspan="3">disjonctives</td></tr>
<tr><td rowspan="2">Organisation des ressources</td><td rowspan="2">une machine</td><td rowspan="2">m machines en parallèle</td><td colspan="2">machines en série</td><td rowspan="2">gammes non linéaires</td></tr>
<tr><td>“Flow-Shop”</td><td>“Job-Shop”</td></tr>
<tr><td>Tâches</td><td colspan="2">préemptives</td><td colspan="3">non préemptives</td></tr>
<tr><td>Contraintes</td><td colspan="2">temporelles</td><td colspan="3">ressources</td></tr>
<tr><td>Objectifs</td><td colspan="2">minimiser</td><td colspan="3">maximiser</td></tr>
</table>

TAB. 1.1 – Caractéristiques d'un ordonnancement

α : Ce champ permet de fixer les paramètres de l'architecture du problème d'ordonnancement. Il permet de représenter le nombre de machines, le nombre de jobs ou encore le type d'organisation des jobs.

β : C'est dans ce champ que les contraintes sont fixées. L'ordre d'exécution des jobs, la préemption des tâches sont des contraintes définies par ce champ.

γ : Les critères d'optimisation sont définis ici. Nous pouvons citer à titre d'exemple la minimisation du retard maximal comme critère d'optimisation.

Cette notation est détaillée complètement dans l'annexe à la page 111. Néanmoins, voici deux exemples permettant de mieux comprendre cette notation.

Exemple 1.2 Soit un problème de Job-Shop composé de 3 jobs qui se réaliseront sur 5 machines. Les tâches sont préemptives et leur durée de réalisation est unitaire. Il existe des relations de précédences entre les tâches des jobs. L'objectif de ce problème est de minimiser le makespan.

Ce problème se note donc :

$$J, 5, 3 \mid Prec, pmtn, p_{ij} = 1 \mid Min\{C_{max}\}$$

Exemple 1.3 Soit un problème d'Open-Shop composé de 10 ressources et 5 jobs. Il n'existe aucune relation de précédences entre les tâches d'un job. Le “splitting” est autorisé, c'est à dire que Les tâches peuvent être éclatées en plusieurs sous tâches pouvants être réalisées sur des machines différentes au même moment. L'objectif de ce problème est de minimiser le makespan.

Ce problème se note donc :

$$O, 10, 5 \mid split \mid Min\{C_{max}\}$$

1.2 Modélisation d'un problème d'ordonnancement

Pour modéliser un problème d'ordonnancement, il est nécessaire de trouver un moyen de représenter les contraintes, les tâches ainsi que les ressources. Pour cela, il existe la méthode analytique et la méthode graphique. Une modélisation par des graphes puis par des équations analytiques permettront dans un premier temps de représenter le problème. Les diagrammes de Gantt seront ensuite présentés pour visualiser les résultats de ces problèmes.

1.2.1 Modélisation par les graphes

Un problème d'ordonnancement est modélisable par un graphe. Les noeuds représentent les tâches à réaliser, les contraintes de précédences sont représentées par des arcs valués de la durée de réalisation de la tâche et les contraintes de ressources sont représentées par des arêtes pointillées [GOTHA 1993, Jain et al. 1999].

La figure 1.7 représente un problème d'ordonnancement avec 3 ressources et 8 tâches à ordonnancer. Sur cette figure, les arcs représentent les contraintes de précédences entre les tâches. Il existe une contrainte de précédences entre les tâches 1 et 2. De plus, le temps de réalisation de la tâche 1 est représenté sur cet arc. Il est de 3 unités de temps pour cet exemple. Trois ressources sont disponibles pour réaliser la tâche. La première, appelée A, peut réaliser les tâches 1, 5 et 6. La seconde, appelée B, réalise les tâches 2, 4 et 7 et la troisième, appelé C, réalise les tâches 3 et 8.

Le graphe est dit disjonctif car les contraintes de ressources sont représentées sur le graphe par des arêtes. Le fait de donner un sens à ces arêtes détermine un ordonnancement [Roy 1970]. On parle également de graphe potentiel-tâche.

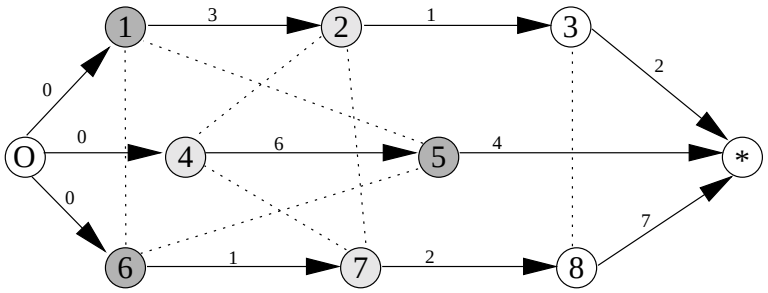

FIG. 1.7 – Exemple de graphe disjonctif

1.2.2 Modélisation analytique

Un problème d'ordonnancement peut également être modélisé sous une forme analytique. Pour cela, il faut modéliser les deux ensembles de contraintes qui représentent un problème d'ordonnancement ainsi que l'objectif. Nous allons donc montrer dans un premier temps comment modéliser les contraintes de ressources, puis dans un deuxième temps les contraintes temporelles enfin dans un troisième temps les objectifs.

Une contrainte de ressources peut se représenter par une fonction du temps, $C(t)$, capable de prendre un seul état dans un ensemble de valeurs $\mathcal{V}$ à un instant donné. Ainsi, une ressource, A, capable d'exécuter les tâches a et c peut s'écrire de la façon suivante : $C_A(t) \subset \mathcal{V}_A$ avec $\mathcal{V}_A = \{\emptyset, a, c\}$. La valeur $\emptyset$ représente le fait que la ressource peut être disponible. Il est également possible de représenter cette contrainte par une inéquation du type : $d_a + p_a \leq d_c \, OU \, d_c + p_c \leq d_a$

Les contraintes temporelles sont représentées par des inéquations. Ainsi, la contrainte entre les tâches *1* et *2* de la figure 1.7 peut s'écrire de la façon suivante : $d_1 + 3 \leq d_2$ avec d_1 la date de début de la tâche 1, et $+3$ représente le temps de réalisation de la tâche 1.

L'objectif de l'ordonnancement est de minimiser ou maximiser une fonction donnée. Une liste non-exhaustive de ces fonctions est donnée dans l'annexe B.3. Ainsi, le problème d'ordonnancement présenté par la figure 1.7 peut s'écrire des deux façons suivantes pour le cas de la minimisation du makespan :

$$obj. : Min(C_{max})$$
$$Sous : \begin{cases} d_1 + p_1 \leq d_2 \\ d_2 + p_2 \leq d_3 \\ d_4 + p_4 \leq d_5 \\ d_6 + p_6 \leq d_7 \\ d_7 + p_7 \leq d_8 \\ d_1 + p_1 \leq d_5 \quad OU \quad d_5 + p_5 \leq d_1 \\ d_1 + p_1 \leq d_6 \quad OU \quad d_6 + p_6 \leq d_1 \\ d_5 + p_5 \leq d_6 \quad OU \quad d_6 + p_6 \leq d_5 \\ d_2 + p_2 \leq d_4 \quad OU \quad d_4 + p_4 \leq d_2 \\ d_2 + p_2 \leq d_7 \quad OU \quad d_7 + p_7 \leq d_2 \\ d_4 + p_4 \leq d_7 \quad OU \quad d_7 + p_7 \leq d_4 \\ d_3 + p_3 \leq d_8 \quad OU \quad d_8 + p_8 \leq d_3 \end{cases}$$

ou

$$obj. : Min(C_{max})$$
$$Sous : \begin{cases} d_1 + p_1 \leq d_2 \\ d_2 + p_2 \leq d_3 \\ d_4 + p_4 \leq d_5 \\ d_6 + p_6 \leq d_7 \\ d_7 + p_7 \leq d_8 \\ C_A(t) \subset \{\emptyset\, 1, 5, 6\} \\ C_B(t) \subset \{\emptyset\, 2, 4, 7\} \\ C_C(t) \subset \{\emptyset\, 3, 8\} \end{cases}$$

Avec la première formulation, trouver un ordonnancement consiste à donner à une valeur à chaque s_i avec $i \in [1; 8]$ en respectant toutes les contraintes et en éliminant l'un des deux choix possibles dans les contraintes de ressources, en évitant de former une incohérence du type : $d_1 + p_1 \leq d_5$ et $d_6 + p_6 \leq d_1$ et $d_5 + p_5 \leq d_6$. En effet, ici, la tâche 5 doit être réalisée après la tâche 1, la tâche 1 doit être réalisée après la tâche 6, et la tâche 6 doit être réalisé après la tâche 5. Ceci forme un cycle qui ne permet pas de savoir laquelle de ces trois tâches doit être réalisée en premier.

1.2.3 Représentation graphique des solutions

Le diagramme de Gantt est certainement le type de représentation le plus ancien, le plus répandu et le plus simple pour visualiser graphiquement l'exécution des tâches et/ou l'occupation des ressources au cours du temps.

A chaque tâche est associé un segment, ou barre, horizontal de longueur proportionnelle à la durée de traitement. En ordonnancement de projet sans considération des contraintes de ressources, les tâches apparaissent en ordonnée, l'abscisse représentant le temps.

Exemple 1.4 Un projet X est composé de six tâches dont les caractéristiques sont représentées par le tableau 1.2. La figure 1.8 représente un diagramme de Gantt associé à une solution minimisant la durée. Le projet X est achevé en onze jours.

Tâche i	A	B	C	D	E	F
Durée p_i (jours)	5	2	2	4	8	1
Contrainte	-	-	après B	après A et C	après B	après D et E

TAB. 1.2 – Exemple de projet

La figure 1.8 ne tient pas compte des problèmes de conflit d'utilisation de ressources. Pour des problèmes d'atelier où les ressources disjonctives sont en nombre limité, le diagramme de Gantt est composé d'une ligne horizontale pour chaque ressource. Il permet ainsi de visualiser les périodes d'occupation et les périodes d'oisiveté des ressources ainsi que la séquence des opérations sur les ressources et la durée totale de l'ordonnancement.

Exemple 1.5 Soit un atelier dans lequel deux produits, A et B, sont à réaliser sur trois machines. Les gammes des produits A et B sont les suivantes :

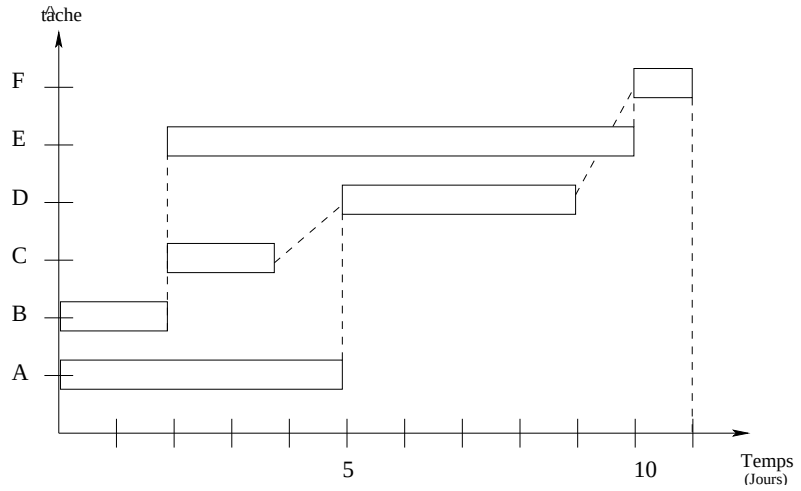

FIG. 1.8 – Diagramme de Gantt associé au projet

A : $o_{A1} < o_{A2} < o_{A3}$
B : $o_{B2} < o_{B1} < o_{B3}$
La tâche o_{ij} du job i est réalisée par la machine j. Le tableau 1.3 donne la valeur des durées de réalisation des tâches en minutes.

Produits	Temps de réalisation		
A	$o_{A1}(60)$	$o_{A2}(20)$	$o_{A3}(30)$
B	$o_{B1}(40)$	$o_{B2}(30)$	$o_{B3}(10)$

TAB. 1.3 – Temps de réalisation des produits

La figure 1.9 représente un ordonnancement faisable de durée totale de 120 minutes. Avec un diagramme de Gantt atelier/ressources, il est possible de visualiser les temps d'attente des ressources. Par exemple, la machine 2 de la figure 1.9 attend trente minutes alors que la machine 3 attend quatre vingts minutes. Par contre, il peut être intéressant de visualiser les temps durant lesquels les produits ne sont pas en cours de traitement. La figure 1.10 représente un diagramme de Gantt atelier/produits. Sur ce diagramme, nous pouvons voir que le produit B est en attente de ressources pendant un total de quarante minutes.

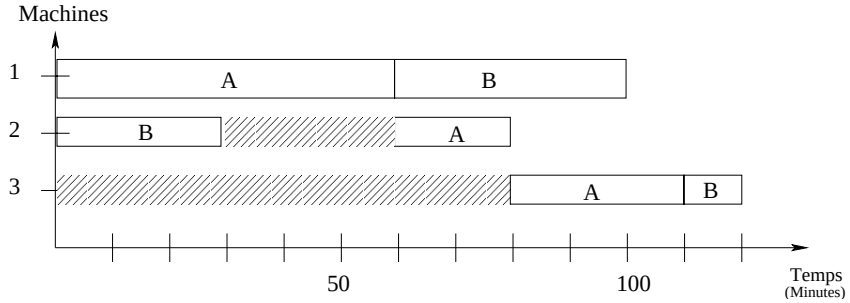

FIG. 1.9 – Diagramme de Gantt atelier/ressources

Il est possible de "surcharger" les diagrammes de Gantt atelier/ressource avec des diagrammes de charge, tel que celui présenté par la figure 1.1, pour visualiser des problèmes d'ordonnancement

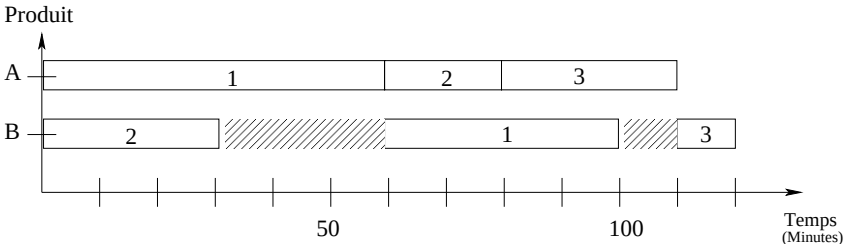

FIG. 1.10 – Diagramme de Gantt atelier/produits

avec des contraintes cumulatives.

1.3 Résolutions des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont des problèmes NP-difficiles pour lesquels il existe des méthodes exactes, mais uniquement pour certaines instances de problèmes. L'utilisation de méthodes approchées est souvent nécessaire afin de pouvoir résoudre ces problèmes [Garey et al. 1979]. Ce paragraphe présente dans un premier temps une liste des principales méthodes de résolution exacte en fonction du nombre de ressources disponibles, puis dans un deuxième temps une liste non-exhaustive des méthodes de résolution approchées. Les problèmes présentés comportent successivement aucune ressource, une seule ressource, n ressources, pour finir par les problèmes d'atelier. Cet enchaînement a été utilisé dans le but de montrer que les méthodes de résolution dépendent du problème abordé et que le nombre de ressources est l'un des facteurs de la complexité.

1.3.1 Notions de complexité

L'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre. Une théorie de la complexité a été développée et permet de classer les problèmes faciles et difficiles en deux classes : les classes $\mathcal{P}$ et $\mathcal{NP}$. Nous exposons dans la partie qui suit les grands principes de la théorie de la complexité. Le lecteur intéressé par de plus amples informations pourra consulter différents ouvrages complètement ou partiellement dédiés à la complexité dont les livres de Garey et Johnson [Garey et al. 1979] , Sakarovitch [Sakanovitch 1984], Xuong [Xuong 1992] ou encore Brucker [Brucker 1995].

Pour pouvoir exposer la notion de classe de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est "oui" ou "non". Il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil k correspondant à la fonction d'objectif f. Le problème d'ordonnancement qui vise à minimiser le makespan se transforme en problème de décision suivant : "existe-t-il un ordonnancement réalisable S tel que $C_{max} \leq k$?"

Il est alors possible de définir la classe $\mathcal{P}$ regroupant les problèmes de décision qui peuvent être résolus par des *algorithmes polynômiaux*. Un algorithme polynômial est défini comme un algorithme dont le temps d'exécution est borné par $O(p(x))$, où p est un polynôme et x est la longueur d'entrée d'une instance du problème. Les algorithmes dont la complexité ne peut être bornée polynômialement sont qualifiés d'*exponentiels*.

La classe $\mathcal{NP}$ regroupe les problèmes qui peuvent être résolus en un temps polynômial par un algorithme non déterministe. Ce type d'algorithme ne fournit pas toujours la même solution à un même problème. C'est le cas des algorithmes réalisant des choix grâce à des heuristiques.

Pour ces algorithmes, si à chaque instruction, le bon choix est effectué, le temps de calculs est polynômial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calculs devient exponentiel.

Les algorithmes "ordinaires" sont évidemment des cas particuliers des algorithmes non déterministes. Aussi tout problème de décision qui peut être résolu par un algorithme polynômial, et qui appartient à la classe $\mathcal{P}$, appartient également à la classe $\mathcal{NP}$. D'où $\mathcal{P} \subseteq \mathcal{NP}$.

Le tableau 1.3.1 présente quelques résultats de complexité pour des problème d'ordonnancement.

Problème	Complexité
BCS	$O(n^3 log(n))$
Machines parallèles	NP-difficile
Flow-Shop de permutation	$O(nlog(n))$
Flow-Shop avec délai d'attente	NP-difficile au sens fort
Job-Shop	NP-difficile
Job-Shop périodique	$O(nlog(n))$
Open-Shop	NP-difficile

TAB. 1.4 – Exemple de complexité pour des problèmes d'ordonnancement

1.3.2 Résolution exacte

Il n'existe pas encore de méthodes de résolution exactes pour chaque problème d'ordonnancement. Par contre, il est possible que pour une instance précise d'un problème d'ordonnancement, une telle méthode existe et que pour le cas général, cette dernière soit inefficace.

Nous allons présenter dans un premier temps les méthodes de résolution exactes pour le problème central défini ci-dessous, puis dans un deuxième temps pour les problèmes d'ordonnancement à ressource unique puis multiple.

1.3.2.1 Le problème central

Un problème central est un problème d'ordonnancement sans contrainte de ressources ou pour lequel les ressources sont en nombre suffisant pour ne pas avoir de conflit.

Une des questions fondamentales dans la résolution concerne le calcul du délai le plus court pour la réalisation du "projet", appelé aussi $Min\{C_{max}\}$. Ceci revient à rechercher le chemin le plus long sur le graphe conjonctif entre le début et la fin du "projet", problème qui compte parmi les plus anciens de la théorie des graphes et pour lequel on dispose d'algorithmes performants. L'algorithme de Bellman-Ford permet de résoudre ce problème de façon efficace.

La méthode la plus connue reste sans conteste PERT (Program Evaluation and Review Technique). Elle est issue du contrôle de la réalisation des engins Polaris à la fin des années cinquante aux États Unis. A la même époque, l'entreprise Du Pont de Nemours développe CPM (Critical Path Method) mise en oeuvre pour la construction d'une usine chimique. Par rapport à PERT, CPM permet de gérer des coûts qui sont fonction des durées ; d'un autre côté, PERT enrichit CPM par l'introduction d'incertitudes dans l'estimation des durées.

En France, toujours à la même époque, Bernard Roy met au point MPM (Méthode des Potentiels-Metra) à l'occasion de la construction du paquebot France. La différence essentielle avec les méthodes précédentes tient au fait que PERT et CPM sont basés sur une modélisation par les graphes potentiels-étapes, alors que MPM utilise les graphes potentiels-tâches.

1.3.2.2 Problèmes d'ordonnancement à ressource unique

Ici, l'objectif est d'ordonnancer n job sur *1* machine. Chaque job est composé d'une et une seule tâche. La résolution de ce problème est directement liée à l'objectif que l'on se fixe.

Minimisation des encours ($Min(\sum c_i.w_i)$)

Ici, c_i représente la date de fin d'exécution de la tâche i et w_i un poids associé à la tâche i. Ce problème se résout de façon exacte, si les *tâches sont indépendantes*. La méthode de résolution consiste à ranger les jobs dans l'ordre croissant des $\frac{p_i}{w_i}$ avec p_i la durée de la tâche i. Ceci est la règle de Smith [Smith 1956], appelée aussi WSPT (Weighted Shortest Processing Time first). Par contre, si les tâches sont dépendantes, le problème devient NP-difficile dans le cas général et polynômial pour des contraintes pouvant s'exprimer sous la forme arborescente [Jr 1976].

Minimisation du plus grand retard ($Min(Max(c_i - d_i))$)

Si les tâches sont dépendantes, la méthode consiste à ranger les tâches dans l'ordre croissant des $d_i^{'}$ où $d_i^{'} = min\{d_j / j\, descendant\, de\, i\}$. Dans le cas contraire, ranger les tâches dans l'ordre des dates échues (d_i) croissantes permet d'obtenir une solution optimale (règle de Jackson [Jackson 1955]).

Minimisation de la somme des retards ($Min(\Sigma \mathcal{T}_i)$)

En présence de date de disponibilité, la minimisation de la somme des retards est un problème NP-difficile [Kan 1976]. Dans le cas particulier où toutes les durées sont unitaires, la règle de Jackson fournit une solution optimale. Dans le cas où les dates de disponibilité sont distinctes, une condition suffisante d'optimalité locale [Chu et al. 1989, Chu et al. 1992] induit une nouvelle règle de séquencement noté PRTT (Priority Rule for Total Tardiness), qui généralise la règle de Smith [Smith 1956].

Minimisation de la durée totale

On s'intéresse ici à l'ordonnancement d'un ensemble de I tâches indépendantes sur une machine unique dans le but de minimiser la durée totale. Chaque tâche est caractérisée par sa date de disponibilité r_i, sa durée p_i et sa latence q_i. Pour tous sous-ensembles K de tâches,

$$h(K) = \min_{i \in K} r_i + \sum_{i \in K} p_i + \min_{i \in K} q_i$$

est une borne inférieure de la durée d'un ordonnancement. Il en résulte que $Max\{h(K), K \subseteq I\}$ est une évaluation par défaut de la durée minimale.

Pour résoudre ce problème, il existe aussi les méthodes suivantes :

- L'ordonnancement de Jackson : C'est un ordonnancement de listes basé sur la règle de priorité MWR (Most Work Remainning). Le défaut majeur de cette méthode est qu'elle n'intègre pas la possibilité de différer une tâche plus prioritaire.
- Relaxation de la contrainte de non préemption : Baker et Su ont proposé cette méthode dans le but d'obtenir une version préemptive de l'algorithme de Jackson. Cet algorithme permet d'interrompre une tâche si une tâche plus prioritaire devient disponible.
- Méthode arborescente : Carlier a proposé cette méthode pour des problèmes non préemptifs [Carlier 1982].

Les problèmes d'ordonnancement à ressource unique représentent en général des problèmes rencontrés en informatique. Par contre, les problèmes de production industrielle font appel à un nombre de ressources très supérieur à l'unité.

1.3.2.3 Problèmes à ressources multiples

Ce paragraphe présente les problèmes d'ordonnancement à ressources multiples. Dans un premier temps, les problèmes à ressources consommables seront présentés. Le cas général à m machines est présenté dans un deuxième temps pour aborder par la suite les problèmes d'atelier, et plus particulièrement le Flow-Shop, le Job-Shop et l'Open-Shop.

Problème à ressource consommable

Un ensemble de tâches liées par des contraintes de précédences doit être ordonnancé en une durée minimale. Chaque tâche i consomme durant son exécution I_{ik} unités d'une ressource consommable k. A la date $u_1 = 0$ nous disposons de b_1 unités de cette ressource. Des quantités additionnelles $b_2, ..., b_q$ deviennent disponibles aux dates $u_2, ..., u_q$. Un ordonnancement doit satisfaire la contrainte associée à la consommation de la ressource. La consommation de la ressource peut se représenter sous la forme d'une courbe de consommation qui doit toujours être en dessous de celle de l'offre. Pour cela, il faut décaler la courbe de consommation vers la droite afin qu'elle reste toujours en dessous de la courbe de l'offre [Carlier et al. 1982, Carlier 1989]. La figure 1.11 nous montre un exemple dont la contrainte sur la ressource est satisfaite.

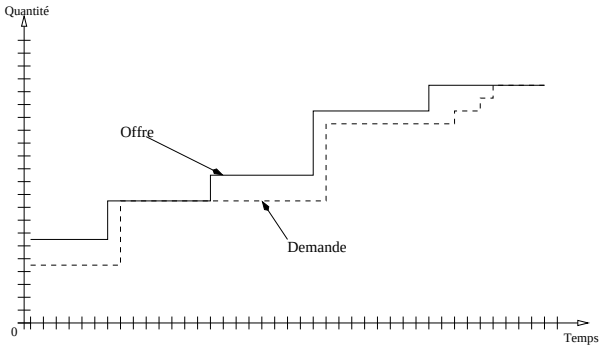

FIG. 1.11 – Courbes de la demande et de l'offre.

Problèmes à m machines

Chaque tâche requiert une machine et l'on dispose de m exemplaires identiques de cette machine. Ces problèmes sont presque toujours NP-difficiles [GOTHA 1993].

Les études théoriques ont surtout porté sur la recherche d'algorithmes approchés avec garantie de performances. Dans le cas de tâches indépendantes, Graham a montré [Graham et al. 1979] que la liste des tâches rangées par durées décroissantes impliquait une erreur majorée, par rapport à l'optimum, comprise entre 1/3 et 1/3m.

Dans le cas de tâches préemptives, les problèmes sont plus simples. L'algorithme classique de Mac Naughton concerne des tâches indépendantes et fournit un ordonnancement de durée minimale égale à $B = Max\{P_i, \sum P_i/m\}$ avec au plus *m-1* préemptions pour une complexité en *O(n)*.

Si la tâche i doit être exécutée dans un intervalle $[r_i, d_i]$, on résout polynômialement le problème en le modélisant par une recherche de flot maximal dans un réseau de transport biparti.

Lorsque les tâches sont indépendantes et que leurs durées dépendent de la machine qui leur est allouée, le problème est résolu en déterminant d'abord la solution optimale d'un programme linéaire qui fournit les durées globales de passage de chaque tâche sur les machines, puis en calculant le découpage en morceaux de l'ordonnancement par un algorithme itératif fondé sur un calcul de flot canalisé à chaque itération [Slowinski 1978].

Problèmes d'ordonnancement d'atelier

Il existe pour certaines instances du Flow-Shop, du Job-Shop et de l'Open-Shop des méthodes de résolution exactes. Ces méthodes sont utilisables sous certaines conditions.

Flow-Shop

Flow-Shop de base

Le problème du Flow-Shop de base est d'ordonnancer n jobs sur m machines. Les jobs sont composés de m tâches $(o_{i1}, ..., o_{im})$ qui se réalisent sur les machines 1 à m. Dans la version de base, il est possible de permuter l'ordre d'exécution des jobs sur deux machines successives. Par exemple, la machine i réalise le job J_1 puis le job J_2 alors que la machine $i+1$ réalisera le job J_2 puis le job J_1. Bien entendu, il est nécessaire de posséder les moyens technologiques pour réaliser une telle opération.

Cas à 2 machines

Jonhson propose en 1954 une solution au problème à 2 machines. Il propose un algorithme, appelé **J** par la suite, qui permet d'ordonnancer les jobs. Il suffit de placer en premier les jobs dont le temps opératoire sur la machine 1 est faible. Ensuite, on place les jobs dont le temps opératoire est important sur la machine 2 [Jonhson 1954].

Cas à m machines

Gonzales et Sahni proposent dans [Gonzalez et al. 1978] une extension de l'algorithme **J**. En 1992, Proust [Proust 1992] propose une extension de **J** pour résoudre des problèmes à m machines.

Flow-Shop sans attente

Pour ce problème, lorsqu'une pièce i sort d'une machine m_j, elle doit passer sans attendre dans la machine m_{j+1}. L'objectif est de minimiser les temps de réalisation des jobs. Ce problème se traite comme le problème du voyageur de commerce, connu sous le nom de TSP (Trade Salseman Problem).

Flow-Shop de permutation

Ce cas particulier du Flow-Shop ne permet pas de réaliser de permutation dans l'ordre de réalisation des jobs entre deux machines successives. Lorsqu'une séquence de réalisation des jobs est définie sur la première machine, cette séquence est utilisée par toutes les autres machines. Ce problème de Flow-Shop de permutation se modélise par le graphe présenté par la figure 1.12.

Le critère d'optimisation est la minimisation de la durée totale. Cet objectif est atteint en recherchant le chemin le plus long dans le graphe décrit par la figure 1.12.

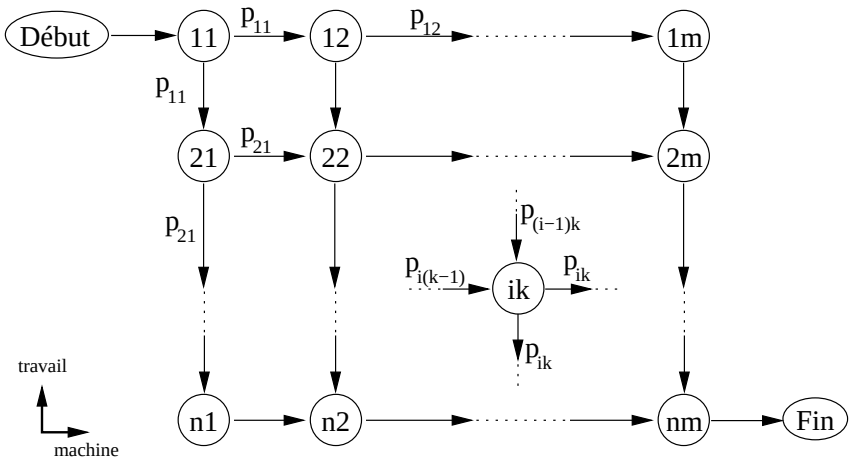

FIG. 1.12 – Graphe potentiels-tâches pour le calcul de la durée totale dans un Flow-Shop de permutation

Job-Shop

Les premiers résultats théoriques concernant le Job-Shop datent des années cinquante avec la résolution du problème à 2 machines et d'un cas particulier du problème à 3 machines. On trouve sa définition formelle dans l'ouvrage "Industrial Scheduling" de Muth et Thompson paru en 1963, où trois exemples de Job-Shop, considérés depuis comme données tests, sont proposés. Les deux premiers ont été résolus dans les années soixante-dix. En revanche, il a fallu attendre les années quatre vingts pour le troisième, l'optimalité de la meilleure solution connue, due à Lagewed, n'ayant été prouvée qu'en 1986 [Carlier et al. 1989b].

Une première approche consiste à déterminer la longueur du chemin critique dans le graphe conjonctif représentant le problème d'ordonnancement [Mattfeld 1995]. Une deuxième approche consiste à utiliser des heuristiques. Dans cette approche, de nombreuses heuristiques différentes peuvent être utilisées. Nous pouvons citer par exemple l'utilisation de l'algorithme de Giffler et Tompson [Giffler et al. 1959] qui permet de créer des ordonnancements actifs, l'utilisation d'un algorithme de type Branch and Bound [Moursli 1997] ou l'utilisation d'un algorithme évolutif tel que les algorithmes génétiques.

Il existe également des méthodes permettant de résoudre le Job-Shop de façon exacte. Ces méthodes sont basées sur l'utilisation des graphes disjonctifs. On peut attribuer à Roy et Balas [Roy 1970] les premières applications de ce type de modélisation.

Open-Shop

Ce type de problème est moins contraint que les problèmes de Job-Shop et de Flow-Shop. En effet, ici, l'ordre des activités n'a pas d'importance car les jobs ne possèdent pas de gamme. Néanmoins, ce type de problème reste NP-difficile [Gonzales et al. 1976]. Il est possible de le résoudre de façon exacte si les tâches sont préemptives. Pour cela, il suffit d'utiliser la méthode des "deux phases" décrite dans [Slowinski 1978]. Étant donné la complexité du problème d'Open-Shop, la plupart des méthodes de résolution sont des méthodes approchées. Elles appartiennent à deux principaux groupes : les méthodes par construction progressive, et les techniques de couplage.

1.3.3 Résolutions approchées

Les problèmes d'ordonnancement sont généralement NP-difficiles. Peu d'algorithmes polynômiaux existent pour les résoudre. C'est pourquoi il est parfois nécessaire d'utiliser des méthodes basées sur des heuristiques, afin de trouver une solution approchée. Deux types d'heuristiques existent. Il y a les heuristiques de construction qui travaillent sur des solutions partielles et les méta-heuristiques qui utilisent des solutions complètes.

1.3.3.1 Heuristiques de construction

Les méthodes de construction progressive travaillent sur des solutions partielles. En effet, l'objectif de ces méthodes est de construire pas à pas la solution finale. Il existe trois classes d'heuristiques de construction : la construction progressive qui complète une solution partielle de façon itérative, la décomposition qui découpe le problème en sous-problèmes qui seront résolus de façon indépendante et la modification de contraintes qui transforme le problème en un problème plus simple [Portmann 1997].

Construction progressive

Cette méthode consiste à compléter une solution partielle à chaque itération [Nawaz et al. 1983]. Si le choix de la tâche à ajouter à l'ordonnancement est défini par l'application d'un théorème de dominance, la méthode peut fournir une solution optimale. C'est le cas pour le problème de base à une machine et la minimisation de la somme des temps de présence ou du plus grand retard, noté $1|d_i|T_{max}$ (règle de Smith [Smith 1956] ou de Jackson [Jackson 1955]). En général, le choix de l'opération fixé par les règles de priorité (faisant intervenir ou non le hasard) ne conduit qu'à des solutions approchées. C'est en particulier le cas pour la résolution des problèmes de type cumulatif par des méthodes dites "sérielles", encore appelées algorithmes de listes, qui fabriquent des solutions à partir d'une liste ordonnée des tâches. Lorsqu'il existe des dates échues impératives, la méthode peut conduire à des solutions non réalisables ; dans ce cas, nous effectuons en général un "backtracking" par une remise en cause récurrente du dernier choix. Nous développons ainsi une arborescence de recherche jusqu'à la détermination d'une solution réalisable.

Décomposition

Cette méthode décompose le problème en un ensemble de sous-problèmes. La définition du critère de décomposition permet de mettre en évidence plusieurs méthodes de décomposition (décomposition hiérarchique, structurelle, spatiale ou temporelle) [Portmann 1988].

- Décomposition hiérarchique [Axsater et al. 1984, Bitran et al. 1982] : elle consiste à décomposer les problèmes en plusieurs niveaux, par exemple un niveau supérieur où l'on travaille sur des données agrégées et où l'on décide d'un certain nombre de paramètres globaux et un niveau inférieur détaillé où les décisions prises sur ces paramètres globaux deviennent des contraintes pour le niveau inférieur, ce qui limite le nombre des solutions à explorer.
- Décomposition structurelle [Roy 1970] : cette décomposition considère dans un premier temps que les moyens sont illimités pour résoudre le problème temporel. Puis, les dates étant fixées, on cherche la meilleure affectation possible des moyens. On revient ensuite au problème temporel en ajoutant des contraintes liées à l'utilisation des moyens.
- Décomposition temporelle [Portmann 1988] : dans le cas des ordonnancements dynamiques où les travaux arrivent à des dates différentes et dispersées dans le temps, la résolution se

fait par itérations successives. A la première itération on ignore les travaux qui arrivent au delà d'une date choisie d_1, puis l'on réalise un ordonnancement partiel jusqu'à une autre date fixée d_2. A l'itération suivante, on décale les deux dates (d_1 et d_2) pour construire la portion suivante de l'ordonnancement.
- Décomposition spatiale [Portmann 1988] : ici, on décompose l'atelier en sous ateliers en minimisant les déplacements de produits entre les sous-ateliers. Ensuite, on construit une méthode itérative dans laquelle chaque itération ordonnance un sous-atelier.

Il est bien sûr possible de combiner de différentes façons toutes ces méthodes par décomposition [Portmann 1988].

Modification de contraintes

Il s'agit ici de méthodes où l'on change le modèle du problème que l'on doit résoudre. Ce peut être, par exemple, de transformer un Flow-Shop normal en Flow-Shop de permutation de manière à avoir moins de solutions à explorer, mais on trouve surtout ici toutes les méthodes dites de "relaxation" : relaxation de contraintes d'intégrité, relaxation Lagrangienne, relaxation "surrogate" [Glover 1975], ...

Le fait de relâcher une contrainte conduit à des solutions qui ne sont plus réalisables, mais qui fournissent des évaluations par défaut qui peuvent être intégrées dans une méthode par séparation et évaluation, ou dans des méthodes itératives approchées utilisant le dual Lagrangien.

1.3.3.2 Méthodes Méta Heuristiques

Ces méthodes sont plus exactement des principes de résolution qui consiste à adapter le problème à résoudre afin de trouver une bonne solution. De nombreux travaux ont abordé les problèmes d'ordonnancement à l'aide de ces méthodes. Il est possible de citer la propagation de contraintes, les systèmes experts, les systèmes à base de connaissance, le recuit simulé [Kirkpatrick et al. 1983], les colonies de fourmis ou les algorithmes génétiques [Goldberg 1989]. Il est également possible de combiner ces approches avec les autres méthodes déjà présentées. On parle ainsi de méthode hybride.

Les méthodes méta-heuristiques sont des méthodes généralistes. Elles permettent de répondre rapidement à des problèmes dit NP-difficiles. L'avantage de ces méthodes est de ne pas explorer de façon itérative la totalité de l'espace de recherche. Le recuit simulé et la méthode tabou étant basés sur la méthode de descente, cette dernière est présentée dans un premier temps, suivi du recuit simulé, de la méthode tabou et des méthodes évolutives.

Améliorations successives ou voisinage

Cette méthode travaille sur des solutions complètes. Elle consiste à passer d'une solution complète à une autre solution complète qui est meilleure du point de vue du critère [Dannenbring 1977]. Pour utiliser cette méthode, il faut disposer d'une fonction $f(\overrightarrow{x})$ capable d'évaluer la solution complète $\overrightarrow{x}$ et d'une fonction de voisinage $v(\overrightarrow{x})$ capable de choisir la solution voisine de $\overrightarrow{x}$. Selon les propriétés que doit vérifier le voisin choisi. Plusieurs méthodes par voisinage peuvent être distinguées :
- Méthode de plus forte pente : choix du meilleur voisin de $\overrightarrow{x}$, ou arrêt s'il n'en n'existe pas.
- Méthode de descente : choix aléatoire d'un voisin meilleur que $\overrightarrow{x}$ et arrêt s'il n'en n'existe pas.

- Recuit simulé : choix d'un voisin quelconque ; si ce voisin est meilleur que $\overrightarrow{x}$, il est définitivement accepté ; sinon il est accepté avec la probabilité $\exp(-perte/T))$ où T représente une température que l'on fait décroître par paliers [Kirkpatrick et al. 1983].
- Méthode Tabou : Les méthodes de descente et de plus forte pente peuvent être utilisées, mais il est possible de "remonter" lorsqu'un optimum local est atteint tout en évitant de repasser par le chemin déjà emprunté [Hertz et al. 1990].

Le recuit simulé

Le recuit simulé copie le processus physique du recuit d'un métal décrit par Kirkpatrick, Gelatt et Vecchi en 1983 [Kirkpatrick et al. 1983]. Ce processus physique a été modélisé informatiquement par des méthodes basées sur *Monte Carlo* par Binder en 1978 [Binder 1978]. Cet algorithme était déjà connu sous le nom d'algorithme de Métropolis [Metropolis et al. 1953]. Cet algorithme était utilisé dans les années 50 pour simuler le refroidissement de bains de métaux en fusion.

Le recuit simulé est une méthode utilisant une solution complète. Partant d'un état initial, on lui fait subir une perturbation, c'est à dire une modification d'amplitude limitée. Si cette perturbation a pour effet d'améliorer le critère, le nouvel état est retenu. On appelle *état* l'ensemble des valeurs prises par les variables d'un problème. Si, au contraire, elle provoque une détérioration ΔE du critère, le nouvel état est retenu avec une probabilité $P = \exp(-\Delta E/T)$, où T est un paramètre qui diminue avec l'augmentation du nombre des itérations. Ce processus est réitéré soit avec le nouvel état, s'il est retenu, soit avec l'ancien état dans le cas contraire.

Les avantages du recuit simulé sont importants. D'abord, cette approche est en mesure de fournir des résultats satisfaisants dès que l'on sait trouver un état initial et une manière de perturber l'état du système. Ensuite, il permet de traiter les problèmes d'optimisation combinatoire pour lesquels il n'existe aucune méthode générale. De plus, il est important de noter que sa convergence est garantie, mais pas la qualité du résultat trouvé.

Notons également que le critère du problème sert uniquement à évaluer les solutions, et non à participer à leur recherche comme dans une méthode du gradient. Aucune propriété de continuité et/ou de dérivabilité n'est donc requise pour le critère.

Le recuit simulé est utilisé pour résoudre des problèmes de Job-Shop [Matsuo et al. 1998, Yamada et al. 1996]

Méthode tabou

Cette méthode à été créée dans les années 1970. Elle a été présentée pour la première fois par Glover en 1986 [Glover 1986]. La formulation finale de cette méthode est apparue à partir de 1989 dans [Glover 1989], [Glover 1990] et [Wera et al. 1989]. Cette méthode est basée sur l'amélioration des algorithmes de plus forte pente ou de gradient.

Cette méthode part d'une solution initiale qui est améliorée successivement par une série de déplacements dans le voisinage. Les déplacements réalisés sont mémorisés dans une liste afin d'éviter de revenir sur une solution déjà explorée [Hertz et al. 1990].

Les *déplacements* réalisés par la solution initiale sont soumis à un ensemble de *conditions tabous* ainsi qu'à un ensemble de *conditions d'aspiration*. Ces deux ensembles de conditions permettent d'augmenter l'efficacité du choix d'une solution dans l'ensemble des solutions fournies par la *fonction de voisinage*.

La liste des déplacements réalisés permet d'établir une liste T_r de *conditions tabous* contenant les futurs déplacements interdits. Cette liste est amenée à être changée à chaque itération. La liste

des conditions tabous du déplacement m se note de la façon suivante : $t_r(i,m) \in T_r$ $(r = 1,..,t)$ où t représente le nombre d'itérations.

Les "*conditions d'aspiration*" représentent des seuils d'amélioration de performance que doit franchir le nouveau déplacement pour être retenu. Ainsi, un déplacement m est retenu si son seuil $a(i,m)$ est supérieur à l'ensemble des seuils $A(i,m)$, même si celui-ci viole une des *conditions tabous*. Une condition d'aspiration commune consiste à accepter le déplacement si celui-ci produit une solution meilleure par rapport au(x) critère(s) choisi(s). Il est également possible de réaliser des conditions d'aspiration plus complexes.

Une *fonction de voisinage* $\mathcal{N}(i,k)$ permet de créer tous les voisins de la solution i qui respectent les *conditions tabous* pour une itération k de l'algorithme ou qui respectent une des *conditions d'aspiration*.

La création de la solution initiale peut se faire de plusieurs façons. La méthode la plus courante est de créer une solution aléatoire. Il est également possible d'utiliser une solution déjà connue afin de l'améliorer.

La méthode tabou est une méta-heuristique souvent rencontrée dans les problèmes industriels car elle représente une amélioration importante des algorithmes de plus forte pente. On parle également de technique de recherche guidée. Un exemple de résolution de Flow-Shop a été traité dans [Grabowski et al. 1997] ainsi que plusieurs problèmes de Job-Shop [Brucker et al. 1997, Taillard 1994]. Néanmoins, le point négatif d'une telle méthode est l'absence de preuve de convergence.

Les Algorithmes Génétiques

Il existe de nombreuses méthodes qui miment le comportement de la nature. Le principe de ces méthodes est de reproduire un comportement ou le fonctionnement d'une entité biologique ou d'un ensemble d'entités. C'est ainsi que les réseaux de neurones, les colonies de fourmis ou les algorithmes évolutifs sont apparus.

Les algorithmes génétiques font parti des algorithmes dit évolutifs. Ces algorithmes sont décrits de manière approfondie dans le deuxième chapitre.

1.4 Conclusion

Les problèmes d'atelier étant souvent des problèmes NP-difficiles, il est préférable de chercher la solution dans les ordonnancements actifs. Pour cela, il faut que le critère d'optimisation soit régulier.

La modélisation des problèmes d'ordonnancement par les graphes permet leur résolution par des techniques de la théorie des graphes. Ces techniques font souvent appel à la détermination du chemin critique qui permet d'obtenir une solution exacte. Néanmoins, il existe encore beaucoup de problèmes d'ordonnancement pour lesquels il n'existe pas de méthode de résolution exacte. La difficulté de ces problèmes vient de l'explosion combinatoire de l'espace de recherche.

Bien entendu, il existe de nombreuses méthodes heuristiques permettant de trouver une bonne solution. Le chapitre suivant présente les algorithmes génétiques comme une méthode permettant de résoudre les problèmes d'ordonnancement d'atelier [Jain et al. 1999].

Chapitre 2

Les algorithmes génétiques comme méthode de résolution des problèmes d'ordonnancement

C'est Holland qui a formulé le premier le principe des Algorithmes Génétiques en 1975 [Holland 1975]. Goldberg [Goldberg 1989] apportera également beaucoup à la création de ces algorithmes. Les algorithmes génétiques font partie des algorithmes dit évolutifs. Ces algorithmes sont des méthodes d'optimisation stochastiques pouvant opérer dans des espaces de taille extrêmement vastes. Le terme évolutif fait référence aux techniques basées sur les mécanismes de l'évolution naturelle. La figure 2.1 présente une hiérarchie des méthodes de résolution méta heuristique d'un problème. Cette figure permet de placer les algorithmes génétiques par rapport aux autres méthodes.

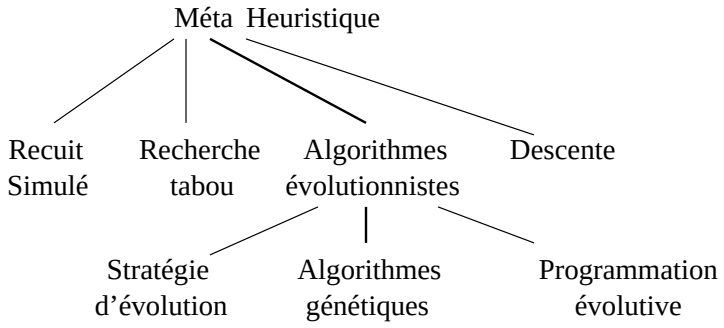

Fig. 2.1 – Méthodes de résolution

Les algorithmes génétiques sont souvent utilisés pour traiter les problèmes d'ordonnancement et plus particulièrement le problème du Job-Shop. Dans un premier temps, nous allons présenter le fonctionnement de ces algorithmes, puis dans un deuxième temps nous allons présenter des possibilités d'utilisation des algorithmes génétiques comme méthode de résolution des problèmes d'ordonnancement d'atelier. La fin de ce chapitre propose un bilan sur les opérateurs génétiques adaptés aux problèmes d'ordonnancement.

2.1 Fonctionnement d'un algorithme génétique

Le principe des algorithmes génétiques est de s'inspirer du fonctionnement de l'évolution de la nature. Bien entendu, ce principe n'est pas encore totalement maîtrisé d'un point de vue biologique, c'est pour cela que certains points de cette évolution sont simplifiés ou font appel à une expérimentation importante. Les algorithmes génétiques sont basés sur le principe *d'évolution* d'une *population d'individus*. L'*évolution* se décompose en deux phases qui consistent à sélectionner des individus dans la population puis de les faire se reproduire. Il existe de nombreux processus de reproduction dans la nature. Les plus courants sont la *mutation* et la *recombinaison*, appelée aussi croisement ou encore cross-over.

Dans cette partie, le principe de base des algorithmes génétiques est développé dans un premier temps, puis dans un deuxième temps, les opérateurs d'évolution seront abordés, suivi de quelques explications sur la fonction d'évaluation, du paramétrage de l'algorithme et de la notion de voisinage.

2.1.1 Principe de base

Le principe des algorithmes génétiques est de faire évoluer une population d'individus vers un optimum. Chaque individu représente une solution au problème à optimiser. La capacité de survie d'un individu est directement liée à son évaluation dans le problème traité. L'ensemble de la population est soumis à un cycle de vie décrit dans le paragraphe 2.1.2. Il existe plusieurs versions d'algorithmes génétiques :

- Canonique : Cette version s'intéresse à des espaces binaires. La fonction F à optimiser est de type $F : \Omega = \{0,1\}^N \rightarrow \mathbb{R}^+$
- Générale : Dans cette version l'espace de recherche peut être des entiers ou des réels. La fonction F à optimiser prend la forme générale : $F : \Omega = \mathbb{R}^N \rightarrow \mathbb{R}$

Les opérateurs utilisés dans le cas général s'adaptent également au cas canonique. C'est pour cette raison que seuls les algorithmes génétiques de type général seront alors présentés.

2.1.2 Cycle de vie

Le cycle de vie de la figure 2.2 décrit les différents éléments et mécanismes qui participent à l'évolution de la population. Dans un premier temps, les individus de la population sont initialisés, puis une partie d'entre-eux est sélectionnée pour être recombinée (croisement). Ces nouveaux individus ainsi créés remplaceront une partie de la population initiale après mutation de certains d'entre-eux. L'ensemble des opérateurs sera décrit ci après.

2.1.3 Population initiale

La construction d'une population initiale représente l'étape de départ de l'algorithme génétique. En général, cette population est créée de façon aléatoire. Il est possible de placer les individus initiaux de façon uniforme dans l'espace de recherche. Une partie de ces individus peut être initialisée avec des résultats connus du problème afin de le faire converger plus vite. Cette dernière méthode permet d'accroître la vitesse de convergence, mais empêche l'algorithme de se focaliser sur une autre partie de l'espace de recherche qui peut contenir l'optimum.

Les opérateurs génétiques sont liés au codage du problème. Nous allons donc expliquer comment coder le problème avant de présenter les opérateurs génétiques.

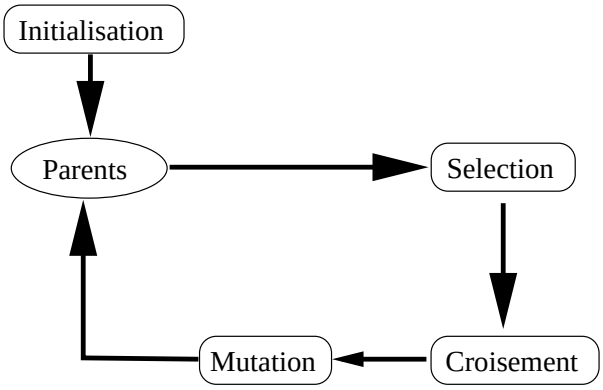

FIG. 2.2 – Cycle de vie

2.1.4 Le codage

Chaque individu représente une solution au problème à optimiser. Un individu est représenté par un *chromosome*. Ce chromosome est constitué de *gènes* qui peuvent prendre des valeurs appelées *allèles*. La position du gène dans le chromosome se nomme *locus*.

Il existe plusieurs manières de coder ces individus. Chacune de ces manières dépend du problème à traiter. Le contenu du gène peut être soit des nombres binaires, soit des nombres entiers ou réels, soit des caractères.

A chaque individu est associé une évaluation (appelée aussi *fitness*). Le chromosome est en fait une représentation du codage du problème instanciée de valeurs. Chaque gène représente une partie élémentaire du problème. Ainsi, un gène peut être assimilé à une variable du problème. L'évaluation représente la performance de l'individu vis à vis du problème. L'évaluation d'un individu est détaillée dans le paragraphe 2.1.7. On parle également de *génotype* et de *phénotype*. Le génotype représente l'ensemble des valeurs des gènes du chromosome de l'individu alors que le phénotype représente la solution réelle.

Prenons l'exemple de la figure 2.3 où le but du problème est de créer une fleur. Le phénotype de chaque individu est une plante dont la forme et la taille diffère d'un individu à l'autre. Le génotype de l'exemple de la figure est : {4,6,8,2,1}.

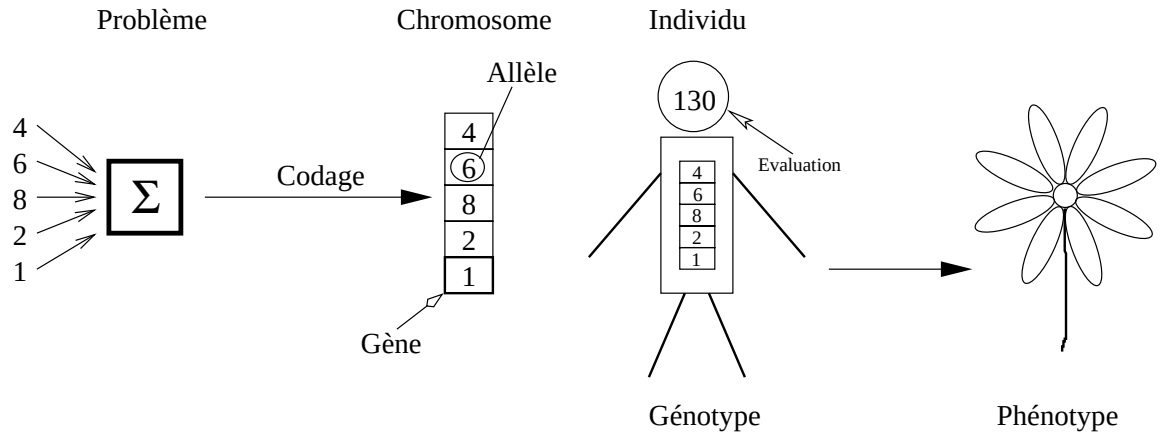

FIG. 2.3 – Du problème à l'individu

Les deux exemples suivants présentent deux problèmes de codage :

Exemple 2.1 L'objectif est de maximiser une fonction $f(r, s, t, u, v)$ tel que $\{r, s, t, u, v\} \in \mathbb{R}^5$ et $f : \mathbb{R}^5 \rightarrow \mathbb{R}$.

Ici, un individu est codé par le chromosome suivant :

α	β	γ	δ	ϵ

Les cinq gènes du chromosome sont respectivements représentés par α, β, γ, δ et ϵ.

Exemple 2.2 L'objectif est de maximiser la production d'une ligne industrielle composée de deux machines. Il existe trois paramètres influants sur ces machines qui sont : la capacité du stock amont CS_{at}, la quantité de pièces pouvant être usinées en une seule fois Q et le numéro R de la ressource affectée sur la machine. Une première possibilité de codage serait de mettre tous les paramètres du problème les uns derrière les autres. Cette possibilité est représentée par :

CS_{at_1}	Q_1	R_1	CS_{at_2}	Q_2	R_2

L'utilisation d'un codage spécifique au problème peut permettre d'améliorer les performances de l'algorithme génétique. Néanmoins, l'utilisation d'un codage spécifique nécessite l'utilisation d'opérateurs génétiques spécifiques. Le paragraphe 2.1.6 présente les différents opérateurs génétiques. La notion de voisinage est présentée dans le paragraphe suivant afin de mieux comprendre les différents opérateurs génétiques.

2.1.5 Le voisinage

La notion de voisinage est définie par une distance. On dit que deux individus I_1 et I_2 sont voisins si la distance qui les sépare est faible ou ne dépasse pas un seuil fixé. Cette distance peut être une distance euclidienne ou tout autre type de distance. Il peut néanmoins arriver que cette notion de distance n'ait aucun sens pour certains problèmes. Imaginons le problème de la construction d'un robot mobile. L'objectif de l'algorithme génétique serait de trouver un ensemble de composants mécaniques, tel que le type de moteur ou le type de transmission d'énergie, permettant de créer un robot mobile autonome ayant une certaine performance. Ici, le chromosome représente une liste de composants. Dire que le voisinage est défini par la distance séparant deux individus n'a de sens que si la notion de distance est correctement définie.

Il est donc possible d'étendre la notion de voisinage en disant que deux individus sont voisins s'ils appartiennent au même hyper-plan, ou plus simplement en disant que seul un paramètre doit les différencier. Par exemple, $I_1 = \{Piston, Chaîne, Hydraulique\}$ est voisin de $I_2 = \{Piston, Chaîne, Pneumatique\}$ car seul le troisième paramètre est différent. Par contre, $I_3 = \{Piston, Arbre, Pneumatique\}$ n'est pas voisin de I_1 car deux paramètres différents les séparent. Dans ce type de problème où la notion de voisinage classique ne peut pas être utilisée, il faut parfois créer des opérateurs adaptés.

2.1.6 Les opérateurs génétiques

Les opérateurs génétiques ont une double mission : celle de permettre à l'algorithme de converger vers un optimum et celle d'explorer au mieux l'espace de recherche. Ce dilemme est identifié par Goldberg sous le nom d'EVE (Exploitation Versus Exploration) [Goldberg 1989]. Dans la mesure où la taille de la population est constante, ces deux objectifs sont antagonistes :

il faut une partie de la population pour exploiter le voisinage des solutions optimales et une autre partie de la population pour explorer la "terra incognita".

Parmi les trois étapes du cycle de vie, la sélection est orientée vers l'exploitation seule. Le croisement et la mutation permettent également l'exploitation et l'exploration, avec la différence suivante : les enfants obtenus par croisement sont en général loin des parents, mais appartiennent à une région réduite de l'espace de recherche, qui dépend des parents. Par opposition, les enfants obtenus par mutation sont en général proches du parent, mais peuvent être situés dans *tout* l'espace de recherche.

2.1.6.1 Sélection

L'opérateur de sélection génère une population P'_t, intermédiaire à la population P_t, d'individus qui seront ensuite croisés et mutés pour former la population P_{t+1}. Les individus sélectionnés représentent les individus les mieux adaptés.

Il existe de nombreuses techniques de sélection. La plus simple, connue sous le nom de *ranking* consiste à ranger les n individus de la population par ordre croissant de leur évaluation respective, ou décroissant selon l'objectif. Les m premiers individus sont ensuite sélectionnés. Ainsi, seuls les meilleurs individus sont conservés. L'inconvénient est de fixer une limite à la sélection. Prenons l'exemple décrit par le tableau 2.1. Le seuil de sélection de la génération 1 peut se faire à l'individu 6. Alors qu'à la génération 10, tous les individus semblent être 'bons'. De plus, l'individu 7 de la première génération peut contenir une partie génétique de bonne qualité.

Individu	Génération 1	Génération 10
1	120	121
2	119	121
3	119.5	120.5
4	118	120
5	118	119.8
6	117.9	119.3
7	100	119
8	90	118.8
9	80	118.5
10	70	118

TAB. 2.1 – Exemple de population après un tri croissant

Il est donc très facile de voir que cette technique permet une mise en oeuvre simple de l'algorithme, mais risque de supprimer de la population des individus de bonne qualité. C'est ainsi qu'est née l'idée de choisir les individus avec une probabilité équivalente à leur évaluation. La technique de la *roulette* consiste donc à affecter à chaque individu une portion d'une roue qui est proportionnelle à son évaluation. Le tableau 2.2 présente un exemple d'une population de 4 individus I_1, I_2, I_3 et I_4 ayant pour évaluation respective 150, 75, 37.5 et 37.5. La roue associée à cet exemple est représentée par la figure 2.4. La sélection des individus est réalisée en faisant tourner la roue en face d'un pointeur fixe. Le pointeur est arrêté après un temps aléatoire et l'individu pointé est sélectionné.

C'est ainsi que même les individus les plus faibles ont une chance d'être sélectionnés. L'inconvénient de cette technique est le cas du "super héros". Si un individu est très supérieur au reste de la population alors sa probabilité d'être sélectionnée est presque totale. Ce "super héros"

Individu	Évaluation	$P_{selection}$
I_1	150	50%
I_2	75	25%
I_3	37.5	12.5%
I_4	37.5	12.5%

TAB. 2.2 – Exemple de sélection à la roulette

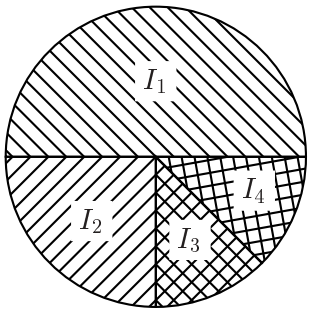

FIG. 2.4 – Sélection à la roulette

est alors sélectionné à chaque tirage.

Une autre technique appelée *tournoi* peut permettre de remédier à ce problème de "super héros". Cette technique tire au hasard deux ou plusieurs individus de la population et les fait "combattre". Le plus fort est sélectionné. L'inconvénient est qu'un bon individu peut ne jamais être sélectionné pour un tournoi. Cet inconvénient peut être atténué en réalisant un tirage avec un nombre d'individus supérieur à deux. Par contre, si le nombre d'individus sélectionnés pour le tournoi est important, le problème du "super héros" risque d'apparaître à nouveau.

L'opérateur de sélection est difficile à mettre en oeuvre car il doit :

- permettre d'éliminer les individus les moins adaptés.
- permettre de dupliquer en nombre suffisant les meilleurs individus afin d'explorer leur voisinage.
- conserver la diversité génétique de la population.

Dans le but d'améliorer l'opérateur de sélection, il est préférable de l'adapter au problème traité. Dans le paragraphe suivant, nous allons voir comment sont croisés les individus sélectionnés par l'opérateur de sélection.

2.1.6.2 Croisement (cross-over)

Le croisement permet d'accoupler m parents pour former m enfants. L'idée est que les enfants garderont les meilleures parties de leurs parents. Prenons l'exemple de la fonction $f(x)$ décrite par la figure 2.5. Le principe du croisement consiste à recombiner les bonnes parties des chromosomes des deux parents $P1$ et $P2$ pour engendrer deux enfants $E1$ et $E2$ de meilleure qualité. La difficulté réside dans le fait que la bonne partie du chromosome n'est pas connue et qu'il est possible de créer des enfants de plus mauvaise qualité que leurs parents. L'enfant $E3$ est un exemple de cette difficulté dans le cas d'une minimisation de la fonction $f(x)$.

Il existe de nombreuses techniques de croisement. La figure 2.6 présente quatre techniques de base de croisement.

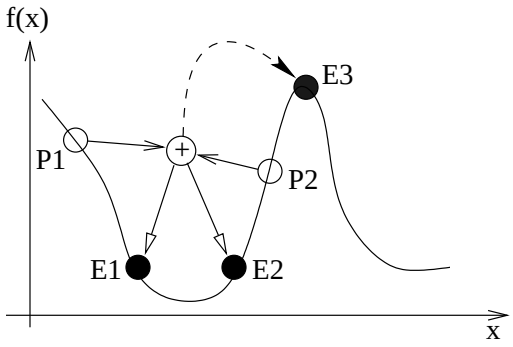

FIG. 2.5 – Principe du croisement

- 1 point : un point de coupure est déterminé de façon aléatoire sur les gènes des parents. Les enfants sont générés en échangeant les morceaux de parents ainsi constitués.
- 2 points : deux points de coupure sont déterminés de façon aléatoire sur les gènes des parents. Les enfants sont des copies des parents dans lesquels la partie centrale, délimitée par les deux points de coupure, est inter-changée.
- masque : plusieurs points de coupures sont déterminés de façon aléatoire, ou prédéfinis, afin de découper les parents en parties paires et impaires. Les enfants sont créés en recombinant les parties paires d'un parent et les parties impaires de l'autre. Cette technique est une extension de la technique à 2 points qui est elle même une extension de la technique à 1 point.
- Orgie : cette technique utilise n parents pour générer n enfants. $(n-1)$ points de coupure sont créés de façon aléatoire. Les enfants sont créés en utilisant un morceau de chaque parent de façon cyclique. La figure 2.6 présente un exemple à 3 parents. [Eiben et al. 1995]

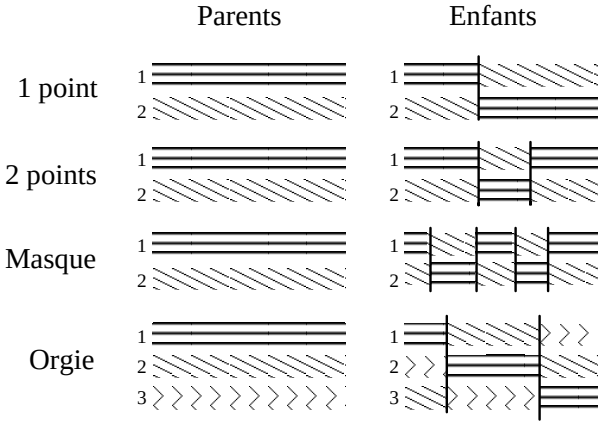

FIG. 2.6 – Exemples de recombinaison

Tous les parents choisis par l'opérateur de sélection n'auront pas la chance d'être croisés. Une probabilité P_{cross} permet de décider si les parents seront croisés entre-eux ou s'ils seront tout simplement recopiés. Le croisement est davantage basé sur l'exploitation du voisinage des parents que sur l'exploration de l'espace de recherche. La figure 2.7 présente un exemple d'une fonction à deux paramètres binaires x et y. Le croisement de $P1$ et $P2$ définis par : $P1 = \{0, 0\}$ et $P2 = \{1, 1\}$; peut générer les enfants $E1 = \{0, 1\}$, $E2 = \{1, 0\}$,$E3 = P1$ et $E4 = P2$. Le croisement choisi est celui à 1 point. Les enfants générés par $P1$ et $P2$ sont tous voisins de

leurs parents. De la même façon, la figure 2.8 représente un cas de croisement à 1 point et un cas de croisement à 2 points pour une fonction dont les paramètres x et y sont codés sur trois bits. Ces deux cas de voisinage sont uniquement présentés pour donner une idée de la notion d'exploration/exploitation dans le croisement

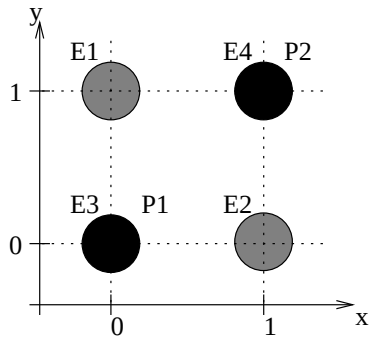

FIG. 2.7 – Exemple de croisement sur un problème binaire à deux dimensions

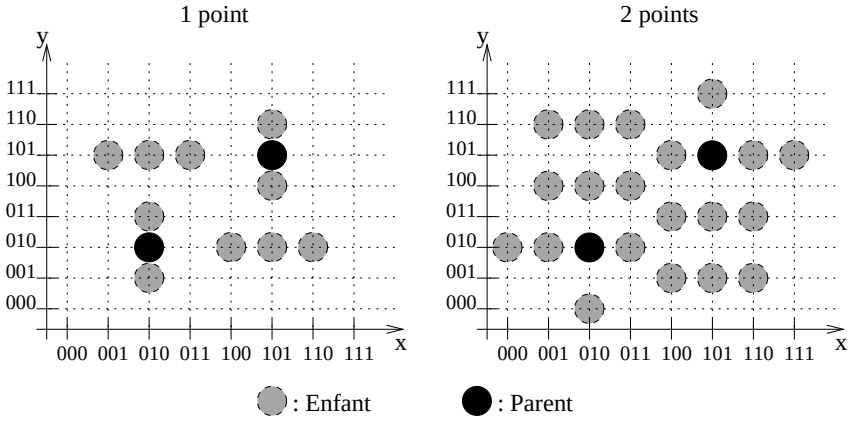

FIG. 2.8 – Exemples de croisement

Contrairement à l'opérateur de croisement qui se base plus sur l'exploitation, la mutation se base sur l'exploration.

2.1.6.3 Mutation

La mutation permet d'explorer l'espace de recherche évitant à l'algorithme de converger vers un optimum local. La figure 2.9 présente un exemple de fonction $f(x)$ à optimiser. Les jetons noirs représentent les individus de la population qui ont tous convergés vers un optimum local. Le principe de la mutation est donc de prendre un de ces individus et de le modifier afin d'explorer une autre partie de l'espace de recherche, en espérant que cette partie soit plus intéressante.

Il existe de nombreuses façons de réaliser une mutation sur un individu. Cet opérateur fait appel à la notion de voisinage qui est définie dans le paragraphe 2.1.5. La figure 2.10 présente trois techniques basiques de mutation ou de voisinage. Le *swap* est la technique la plus répandue. Elle consiste à prendre au hasard 2 gènes du chromosome, 2 et 8 dans l'exemple, et à les inverser. L'*insertion* supprime au hasard un gène, 2 dans l'exemple, et le réinsère ailleurs de façon aléatoire. La technique appelée *Random* modifie aléatoirement un gène choisi de façon aléatoire.

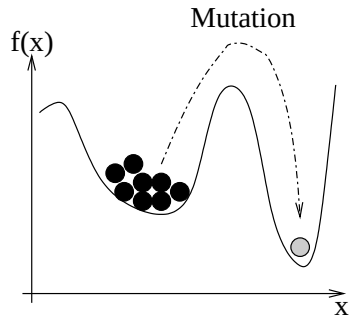

FIG. 2.9 – Principe de la mutation

Origine	0 1 2 3 4 5 6 7 8 9
Swap	0 1 8 3 4 5 6 7 2 9
Insertion	0 1 3 4 5 2 6 7 8 9
Random	0 1 2 3 4 5 2 7 8 9

FIG. 2.10 – Exemples de mutation

L'opérateur de mutation ne s'applique pas à tous les individus. Chaque individu a une probabilité, P_{mut}, d'être muté. Cette probabilité est en général très faible. Une fois les individus croisés et mutés, il faut les évaluer afin de pouvoir sélectionner la prochaine génération. Cette évaluation se réalise à l'aide d'une fonction d'évaluation

2.1.7 Fonction d'évaluation

La fonction d'évaluation permet de quantifier la capacité d'un individu à survivre. Cette quantité, appelée aussi *Fitness*, est le résultat d'une fonction d'évaluation. La fonction d'évaluation dépend directement du problème à résoudre. Elle est constitué d'un objectif (minimiser ou maximiser) et d'un ou plusieurs critères d'évaluation. L'évaluation du critère de performance peut prendre plusieurs formes qui peuvent se classer en deux catégories :

- Mathématique : cette catégorie regroupe tous les problèmes pouvant s'écrire sous la forme d'équations algébriques, ou pouvant être évalués par une fonction mathématique.
- Simulation : ici, les problèmes abordés ne peuvent pas être modélisés par un outil mathématique. Le seul moyen d'évaluer un individu se fait par le biais d'un simulateur. C'est généralement le cas des fonctions non linéaires, non continues et/ou non dérivables.

La simulation est souvent employée [Cavory et al. 2000b, Pierreval 1997] comme fonction d'évaluation mais son utilisation est généralement gourmande en temps de calculs et représente la plus grande partie du temps d'optimisation. Pour remédier à ceci, il est possible d'utiliser des opérateurs adaptés au problème ou de bien paramétrer son algorithme.

2.1.8 Le paramétrage de l'algorithme génétique

Le paramétrage d'un algorithme de recherche est souvent une tâche difficile. Dans le cas des algorithmes génétiques, de nombreux travaux sont en cours pour déterminer la valeur de ces paramètres. A ce jour, aucun n'a abouti sur des résultats concluants. Il existe néanmoins des valeurs, proches de celle de la nature, qui permettent d'obtenir de bon résultats. Par exemple, la probabilité de croisement appartient à l'intervalle $0.7 < P_{cross} < 0.99$. De même, la probabilité de mutation est souvent définie comme : $P_{mut} < 0.001$. Bien entendu, ces chiffres ne sont là qu'à

titre d'exemple. Ils ne représentent que des valeurs couramment utilisées et ne sont en aucun cas les valeurs idéales.

Il existe d'autres méthodes permettant d'améliorer l'algorithme génétique. Il est possible d'y inclure des mécanismes supplémentaires tel que *l'élitisme*. Ce mécanisme permet d'éviter la perte d'individus de bonne qualité. Nous avons vu que les enfants remplacent leurs parents par application de l'opérateur de croisement. Mais il est possible que les enfants soient moins performants que leurs parents, créant ainsi la perte d'individus de bonne qualité. L'élitisme permet de conserver le ou les meilleurs individus d'une population P_t pour le ou les réinjecter dans la population P_{t+1}. Néanmoins, si le nombre d'individus ainsi conservé est trop important, l'algorithme peut être ralenti dans sa convergence ou tout simplement resté bloqué dans un optimum local. La convergence d'un algorithme génétique peut également être améliorée en définissant une bonne fonction de voisinage ou en utilisant des paramètres dynamiques de mutation et de cross-over [Eiben 1998].

2.2 Résolution des problèmes d'ordonnancement d'atelier par les algorithmes génétiques

De nombreux travaux sur l'optimisation d'ordonnancement utilisant des algorithmes génétiques ont été réalisés. Ces travaux peuvent être classés en deux parties. La première partie regroupe les travaux utilisant un codage direct alors que la deuxième partie regroupe les travaux utilisant un codage indirect. Malgré cette différence de codage, les problèmes d'ordonnancement traités par les algorithmes génétiques portent en majeur partie sur les problèmes de Job-Shop.

2.2.1 Différence entre codage direct et codage indirect

Il existe deux cas de figures pour coder un problème d'ordonnancement d'atelier. Le premier utilise un codage direct dans lequel toute l'information est présente dans le chromosome. Le deuxième déporte la difficulté du problème d'ordonnancement à l'extérieur du codage, en utilisant un codage indirect [Lee et al. 2000].

2.2.1.1 Direct

Dans un codage direct, le gène doit représenter une solution complète. Ce dernier doit contenir toutes les informations utiles à la création de l'ordonnancement. Toutes les dates de début de chaque tâche ainsi que leur affectation doivent être représentées dans le gène pour chaque occurrence. De plus, les informations dans le gène doivent tenir compte des contraintes de précédences. Les opérateurs de croisement et de mutation ne doivent en aucun cas générer des individus 'infaisables'. Un individu est dit infaisable si au moins une contrainte du problème n'est pas satisfaite. La difficulté d'un tel codage réside dans la représentation des données du problème ainsi que sur le choix des opérateurs génétiques.

2.2.1.2 Indirect

Dans ce type de codage, le génotype ne représente pas directement la solution. Il est nécessaire de passer par une étape intermédiaire pour obtenir la solution. La figure 2.11 présente deux codages différents. Dans le codage direct, toute l'information est disponible. Ainsi, on sait directement que la tâche *1* doit se réaliser sur la ressource *A* à l'instant *0*. Le passage du génotype au phénotype se fait directement. Par contre dans le codage indirect, c'est au cours d'un traitement

intermédiaire, ici appelé *ordonnanceur*, que l'ordonnancement est construit. On parle d'individu *légal* s'il est possible de transformer ce dernier en une solution faisable ou non. Un individu est dit *faisable* si le phénotype respecte toutes les contraintes du problème. Il est parfois possible de "réparer" un individu qui est illégal ou infaisable [Beasley 1999]. Les méthodes de réparation dépendent directement du problème et de son codage.

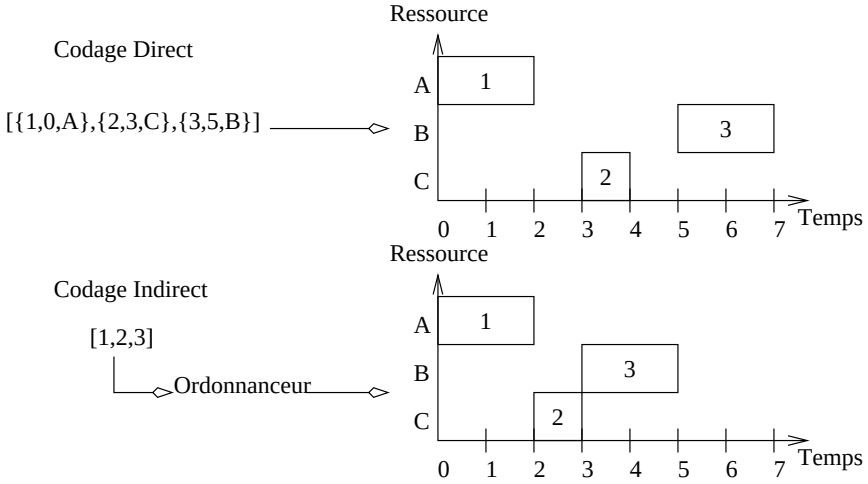

FIG. 2.11 – Différence entre codage direct et codage indirect

2.2.2 Résolution par un codage direct

La difficulté d'utiliser un codage direct pour un problème d'ordonnancement d'atelier tel que le Job-Shop réside dans le fait que toute l'information doit être contenue dans le gène en tenant compte de toutes les contraintes qui composent le problème d'ordonnancement.

L'idée : La représentation qui semble la plus naïve est celle présentée par Yamada et Nakano dans [Yamada et al. 1992]. Elle consiste à utiliser dans le chromosome les dates de fin de réalisation des tâches ainsi qu'un algorithme réalisant des ordonnancements actifs.

Le codage : Le chromosome utilisé peut être vu comme un chromosome à deux dimensions. La première représente les tâches alors que la deuxième représente les jobs. Les valeurs contenues dans le gène correspondent aux dates de fin de réalisation des tâches à effectuer. Le tableau 2.3 présente un exemple de chromosome pour un problème à 6 jobs composés chacun de 6 tâches, avec la notation suivante : $o_{13}(22)$ représente le fait que la 3^{ieme} tâche du job 1 se termine à la date 22.

Job	Date de fin de réalisation de l'opération : Opération(p_{ij})					
J_1	$o_{11}(1)$	$o_{12}(4)$	$o_{13}(22)$	$o_{14}(37)$	$o_{15}(45)$	$o_{16}(55)$
J_2	$o_{21}(8)$	$o_{22}(13)$	$o_{23}(23)$	$o_{24}(38)$	$o_{25}(48)$	$o_{26}(52)$
J_3	$o_{31}(6)$	$o_{32}(10)$	$o_{33}(18)$	$o_{34}(27)$	$o_{35}(28)$	$o_{36}(49)$
J_4	$o_{41}(13)$	$o_{42}(18)$	$o_{43}(27)$	$o_{44}(30)$	$o_{45}(38)$	$o_{46}(54)$
J_5	$o_{51}(22)$	$o_{52}(25)$	$o_{53}(30)$	$o_{54}(42)$	$o_{55}(51)$	$o_{56}(53)$
J_6	$o_{61}(16)$	$o_{62}(19)$	$o_{63}(28)$	$o_{64}(32)$	$o_{65}(42)$	$o_{66}(43)$

TAB. 2.3 – Un chromosome selon Yamada et Nakano contenant les temps de réalisation

Les opérateurs génétiques : Yamada et Nakano utilisent un cross-over, basé sur l'algo-

rithme de Giffler et Tompson, qui permet de générer des ordonnancements actifs à partir d'ordonnancements semi-actifs [Nakano et al. 1991]. L'opérateur de cross-over utilisé est l'opérateur GA/GT décrit par l'algorithme 2.1. Cet opérateur génétique construit un ordonnancement actif à partir d'une matrice d'héritage et de deux parents, nommés PÈRE et MÈRE par la suite, qui sont eux-mêmes des ordonnancements actifs. La matrice d'héritage permet de déterminer de quel parent l'enfant hérite lors du choix de l'opération à ordonnancer durant le croisement. Un opérateur de mutation proposé consiste à modifier l'héritage de l'enfant au moment de la sélection du PÈRE ou de la MÈRE. Cet algorithme considère que la sélection des parents est déjà réalisée par un autre opérateur.

Algorithme 2.1 Cross-over GA/GT

Initialiser aléatoirement une matrice binaire Héritage de taille $m \times n$ (m est le nombre de machines et n est le nombre de tâches)
Faire
Trouver une opération $\mathcal{O}^*$ qui représente l'opération non ordonnancée qui possède la date de fin de réalisation la plus petite. Appelons M_p la machine sur laquelle se réalise $\mathcal{O}^*$.
Trouver l'ensemble C qui est constitué des opérations non ordonnancées qui se réalisent sur la ressource M_p et dont l'intervalle de réalisation recouvre celui de $\mathcal{O}^*$.
Supposons que c'est le moment d'ordonnancer la k^{ieme} opération de la machine M_p
Si Héritage(p,k)=1 **Alors**
Sélectionner l'opération $\mathcal{O}$, dont la date de début de réalisation est la plus faible dans l'ordonnancement MÈRE, dans l'ensemble $\mathcal{C}$.
Sinon
Sélectionner l'opération $\mathcal{O}$, dont la date de début de réalisation est la plus faible dans l'ordonnancement PÈRE, dans l'ensemble $\mathcal{C}$.
FinSi
Ordonnancer l'opération $\mathcal{O}$ au plus tôt dans l'ordonnancement FILS.
TantQu'il n'y a plus d'opération non ordonnancée.

L'initialisation de la population est réalisée à partir d'un algorithme quelconque capable de réaliser des ordonnancements actifs. Le codage est direct dans ce cas car il est possible de trouver directement la valeur du fitness. Ce codage permet de représenter la totalité des ordonnancements faisables. De plus, une solution infaisable ne peut pas être créée car l'opérateur GA/GT est capable de modifier les enfants pour les rendre faisable. Les auteurs utilisent une technique appelée 'forçage génétique' qui consiste à conserver les enfants modifiés dans la population courante.

2.2.3 Résolution par un codage indirect

La résolution d'un problème d'ordonnancement par un codage génétique indirect est la méthode la plus répandue. En effet, l'utilisation d'un codage direct implique l'utilisation d'opérateurs génétiques spécifiques à ce codage. Il est possible que la détermination de ces opérateurs soit très difficile. L'idée est donc de chercher des éléments qui permettent de construire une solution au problème. Cette nouvelle recherche permet d'utiliser des opérateurs génétiques connus.

Nous allons présenter trois situations possibles de codage indirect, tout d'abord une approche de codage utilisant des heuristiques dans le gène, puis, un codage utilisant des listes de priorité par ressource et enfin, un codage utilisant des permutations. Les méthodes décrites ci-dessous

ne représentent en rien une liste complète des méthodes de résolution, mais présentent plutôt certains travaux originaux qui permettent d'obtenir une solution.

2.2.3.1 Utilisation d'heuristiques dans le gène

Dans ce cas particulier, nous traitons les problèmes d'atelier qui consistent à déterminer l'ordre de passage des tâches sur les ressources.

L'idée proposée par Ghedjati [Ghedjati 97], pour un Job-Shop, et par Norenkov et Goodman [Norenkov et al. 1999], pour un Flow-Shop, est de coder des heuristiques de choix qui permettent à une ressource de choisir la tâche qui va être réalisée. Norenkov classe les heuristiques de choix en deux catégories :

- Catégorie A : regroupe des heuristiques de choix basées sur des indicateurs calculés à partir des dates de fin de la tâche (c_i) ou des dates de libération des ressources.
- Catégorie B : regroupe des heuristiques de choix basées sur les temps de réalisation de la tâche en fonction de l'affectation de cette dernière. Le choix portera sur la ressource la plus rapide, la plus lente ou tout simplement sur la première disponible.

Le codage est réalisé en affectant à chaque ressource une heuristique de choix. Ainsi, le chromosome est constitué d'autant de gènes qu'il y a de ressources. De plus, les individus créés sont tous faisables.

Les opérateurs génétiques : Cette méthode, appelée aussi HCM (Heuristics Combination Method) par Norenkov, permet d'utiliser une initialisation aléatoire ainsi que des opérateurs génétiques classiques tels que le croisement à $n-$points.

Les individus étant tous faisables, une éventuelle réparation de ces derniers est donc inutile. Par contre, l'espace de recherche exploré par cette méthode est limité par les heuristiques utilisées.

2.2.3.2 Liste de priorité par ressource

Les méthodes présentées ici concernent toutes la résolution du problème du job-shop. Ce problème est à l'origine de nombreux travaux [Lee et al. 2000, Cheng et al. 1999] utilisant les algorithmes. Les trois méthodes présentées utilisent toutes un codage par liste de priorité sur les ressources. La première vise à rendre le graphe de précédences conjonctif, alors que les deux suivantes utilisent des séquences de réalisation des tâches sur les ressources.

Graphe disjonctif-conjonctif

L'idée : nous avons vu dans le paragraphe 1.2.1 qu'il est possible de représenter un problème de Job-Shop par un graphe de précédences, appelé aussi graphe potentiel-tâche. Ce graphe est disjonctif car le sens des arêtes représentant les contraintes de ressources n'est pas défini. Roy [Roy 1970] propose de donner un sens aux arêtes du graphe de précédences afin de le rendre conjonctif. Tamaki et Nishikawa [Tamaki et al. 1992] ont repris l'idée de Roy et utilisant un algorithme génétique afin de déterminer le sens des arêtes.

Prenons l'exemple décrit par la figure 2.12. Rendre ce graphe conjonctif c'est choisir un sens aux arêtes en pointillées. La figure 2.13 représente un choix pour le sens des arêtes en pointillées. Le problème d'ordonnancement consiste donc à déterminer un sens aux arêtes représentant les contraintes de ressources.

Le codage : Cette méthode peut facilement être codée par une chaîne de bits où chacun des bits représentent le sens d'une arête. Le tableau 2.4 représente le codage associé à la figure

2.13. Ici, $d_{15} = 0$ veut dire que l'arc est orienté du noeud 1 vers le noeud 5 ; de même $d_{38} = 1$ veut dire que l'arc est orienté du noeud 8 vers le noeud 3.

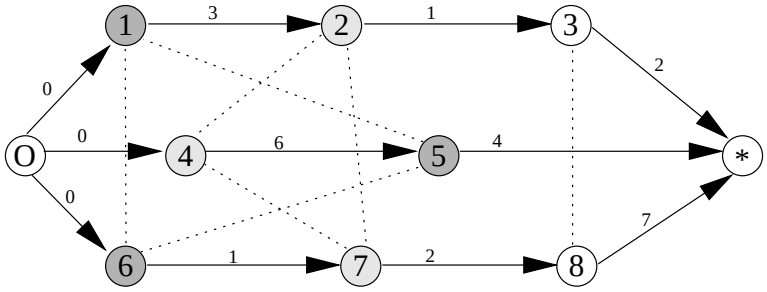

FIG. 2.12 – Exemple de graphe disjonctif

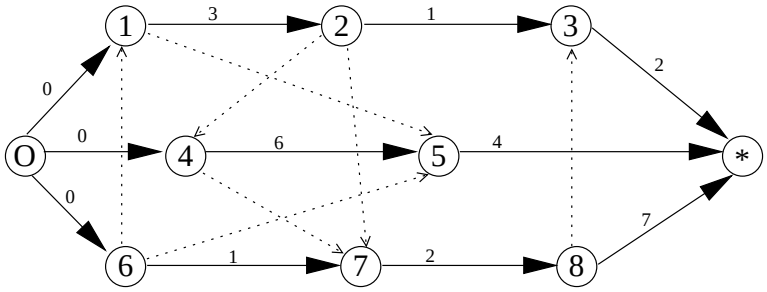

FIG. 2.13 – Exemple de graphe conjonctif

0	1	1	0	0	0	1
d_{15}	d_{16}	d_{56}	d_{24}	d_{27}	d_{47}	d_{38}

TAB. 2.4 – Codage binaire de Tamaki

Les opérateurs génétiques : Le codage employé permet d'utiliser des opérateurs génétiques classiques tels que les cross-over à 1 ou n-points.

L'évaluation d'un ordonnancement se réalise en déterminant la valeur du chemin critique. Néanmoins, ce codage n'interdit pas la formation de cycles dans le graphe qui correspond à une incohérence du type : a<b<c<a. Ces cycles empêchent la détermination du chemin critique. Pour remédier à ceci, Tamaki et Nishikawa utilisent l'algorithme 2.2, appelé "Construct feasible schedule from chromosome". Cet algorithme calcule le chemin critique et les dates au plus tôt de réalisation de chaque tâche en ignorant les conflits de ressource. Les conflits potentiels sont supprimés en utilisant l'information contenue dans le chromosome ; les opérations définies comme étant moins prioritaires par les allèles du chromosome, sont décalées dans le temps. Ceci est répété tant qu'il y a au moins un conflit.

Cet algorithme est capable de réparer le chromosome, si ce dernier possède un cycle, lors du calcul du chemin critique. Néanmoins, la réparation n'est réalisée que lors de la construction de l'ordonnancement. Le chromosome nécessitant une réparation est conservé en l'état.

Cette méthode permet toutefois d'optimiser un problème d'ordonnancement comportant des contraintes de ressource et pouvant se modéliser par un graphe de précédences. L'espace de recherche exploré de cette méthode est déterminée par les choix réalisés dans l'algorithme de

Algorithme 2.2 Construct_feasible_schedule_from_chromosome

Faire

Calculer le chemin critique et la date au plus tôt de début de chaque noeud (tâche) en ignorant les conflits de ressource

Si au moins 2 opérations de la même ressource ont leur intervalle de temps de réalisation en conflit

Alors Résoudre le conflit en utilisant l'information du chromosome. L'opération non prioritaire est décalée dans le temps.

FinSi

TantQu'il y a conflit entre au moins deux opérations

construction de l'ordonnancement. Cet espace de recherche ne représente pas la totalité des ordonnancements faisables.

Séquence d'affectation des tâches

Les codages indirects sont très souvent basés sur le codage d'une séquence. La séquence représente une liste des jobs à réaliser selon un ordre prédéfini.

L'idée : Kobayashi et al [Kobayashi et al. 1995] proposent une méthode de codage avec plusieurs séquences. Les séquences déterminent l'ordre de réalisation des tâches sur chaque ressource.

Le codage : Le chromosome est constitué d'autant de séquences qu'il y a de ressources. Les séquences peuvent être mises bout à bout dans le chromosome ou peuvent être vue comme un chromosome à plusieurs dimensions.

Les opérateurs génétiques : L'opérateur de cross-over utilisé dans cette méthode est basé sur l'échange de sous-séquences échangeables. Deux sous-séquences sont dites échangeables si elles possèdent les même valeurs. Par exemple les deux sous-séquences suivantes sont échangeables : $S_1 = \{J_1, J_3, J_5\}$ et $S_2 = \{J_5, J_1, J_3\}$.

La figure 2.14 présente un exemple de Job Shop à deux machines et cinq jobs. Le croisement de deux parents P1 et P2 donne naissance aux deux enfants E1 et E2.

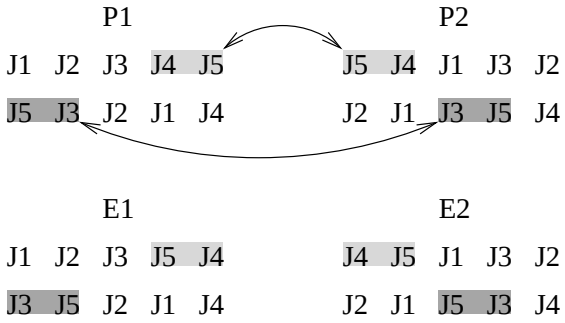

FIG. 2.14 – Exemple de cross-over pour la méthode de Kobayashi

Il est possible que cet opérateur génère des solutions infaisables. Pour remédier à ceci, l'algorithme de Giffler et Tompson est utilisé pendant la phase d'évaluation de l'individu afin de "réparer" l'ordonnancement. L'initialisation de la population est réalisée à l'aide d'un algorithme qui génère des ordonnancements actifs.

L'opérateur de mutation utilisé est la mutation swap.

Cette méthode se résume donc à trouver pour chaque ressource une bonne séquence permettant de satisfaire l'objectif. L'espace de recherche de cette méthode se limite aux ordonnancements actifs. Cette limitation est due à l'utilisation de l'algorithme de Giffler et Tompson lors de l'évaluation des individus. Cette méthode souffre également de la possibilité de créer des individus illégaux ; une méthode de réparation est nécessaire.

2.2.3.3 Séquence de réalisation des tâches

L'idée : Pour le problème de Job-Shop, Lee et Yamakawa [Lee et al. 1996] proposent un codage qui tient compte des contraintes de précédences entre les tâches, mais ne tient pas compte des contraintes d'affectation.

Le codage : Ce problème est codé par une séquence de tâches à réaliser au plus tôt sur les ressources dès que celles-ci sont libres. L'évaluation de l'individu est réalisée par un *ordonnanceur* qui affecte les tâches aux ressources. Le chromosome est donc constitué d'une séquence de tâches.

Les opérateurs génétiques : Les opérateurs génétiques employés doivent tenir compte de l'ordre des tâches dans le gène ainsi que des contraintes de précédences. Lee et Yamada ont utilisé les opérateurs POX et PPS décrits ci-dessous.

Croisement POX : Precedence preserving ORder based cross-over

Cet opérateur génétique prend en compte l'existence de contraintes de précédences entre les allèles du chromosome. Les opérateurs présentés jusqu'ici ne prenaient pas en compte ces contraintes supplémentaires. Néanmoins, les allèles doivent toujours être représentés par un et un seul exemplaire.

Prenons les contraintes de précédences suivantes : $1<7<9$, $2<6$, $4<8$ et $3<5$. Nous allons construire les deux enfants résultants du croisement des deux parents $Parent_1$ et $Parent_2$. Chacun des enfants se réalise en quatre étapes successives qui sont :

1. Sélectionner un allèle dans le parent i.

$$\begin{array}{lccccccccc} Parent_1 = & 1 & 2 & 3 & 4 & 5 & \underline{6} & \overline{7} & 8 & 9 \\ Parent_2 = & 4 & 1 & 2 & 8 & 7 & \overline{6} & 9 & 3 & 5 \end{array}$$

2. Chercher ses contraintes de précédences C_{Parent_i}.

$$\begin{array}{l} C_{parent_1} = \{1, 7, 9\} \\ C_{Parent_2} = \{2, 6\} \end{array}$$

3. Placer les allèles de C_{Parent_i} au même locus dans l'$Enfant_i$.

$$\begin{array}{lccccccccc} Enfant_1 = & 1 & \bullet & \bullet & \bullet & \bullet & \bullet & 7 & \bullet & 9 \\ Enfant_2 = & \bullet & \bullet & 2 & \bullet & \bullet & 6 & \bullet & \bullet & \bullet \end{array}$$

4. Retirer du $Parent_j$ les allèles de C_{Parent_i} et recopier le résultat dans l'ordre d'apparition dans l'$Enfant_i$.

$$\begin{array}{lllll} Parent_2 - C_{Parent_1} = \{4\ 2\ 8\ 6\ 3\ 5\} & Enfant_1 = & 1 & 4 & 2\ 8\ 6\ 3\ 7\ 5\ 9 \\ Parent_1 - C_{Parent_2} = \{1\ 3\ 4\ 5\ 7\ 8\ 9\} & Enfant_2 = & 1 & 3 & 2\ 4\ 5\ 6\ 7\ 8\ 9 \end{array}$$

Mutation PPS : Precedence Preserving Shift mutation

La mutation PPS permet de conserver les contraintes de précédence sur les allèles dans le chromosome. L'explication de cet opérateur se fera en prenant les contraintes de précédences suivantes : 2<4<6, 1<3<7 et 5<8<9. Pour créer la mutation d'un chromosome, il faut réaliser les quatre étapes suivantes :

1. Tirer aléatoirement un locus i sur le chromosome à muter.

				i					
Chromosome :	1	2	3	4	5	6	7	8	9

2. Créer l'ensemble Pc_i des allèles appartenant à une même contrainte de précédence que l'allèle au locus i.

$$Pc_i = \{2, 4, 6\}$$

3. Chercher le locus des allèles de Pc_i le plus à gauche (lg) dans le chromosome. Faire de même avec le locus le plus à droite (ld).

		lg		i		ld			
Chromosome :	1	2	3	4	5	6	7	8	9

4. Choisir aléatoirement un locus p entre les positions lg et ld. Déplacer l'allèle du locus i au locus p.

		lg		i	p	ld			
Chromosome :	1	2	3	5	4	6	7	8	9

L'ordonnanceur utilisé par Lee affecte les tâches sur les ressources dès que ceci est possible. Ainsi, l'espace de recherche de cette méthode se limite aux ordonnancements "au plus tôt". L'avantage de cette méthode est de toujours fournir des solutions faisables et qu'aucune réparation n'est nécessaire.

2.2.3.4 Permutation avec répétition

L'idée : Une autre stratégie basée sur les séquences est de considérer le problème du Job-Shop comme étant un problème de permutation [Davis 1985b]. Bierwirth [Bierwirth 1995] propose une représentation *complète*, dans le sens où toutes les solutions faisables sont codables et que les solutions infaisables ne le sont pas. Le problème du Job-Shop est représenté comme un problème de permutation avec répétitions. Cette représentation permet de décrire des problèmes de Job-Shop à plusieurs jobs $(J, n, m//)$. Cette représentation est basée sur les permutation avec répétitions car dans le chromosome, il est possible que le même allèle apparaisse plusieurs fois.

Le codage : Prenons par exemple un problème de Job-Shop à 3 jobs et 3 machines. Le chromosome est composé d'une succession de neuf allèles ayant pour valeur soit J_1, soit J_2 ou soit J_3. Les allèles J_1, J_2 et J_3 seront présents trois fois dans le chromosome car chaque job doit passer sur les trois machines. La figure 2.15 représente ce chromosome. Le chromosome est ensuite évalué par un ordonnanceur qui affecte les tâches des jobs aux machines. En utilisant le chromosome de la figure 2.15, l'ordonnanceur affectera en priorité la première tâche du job J_1, puis la deuxième tâche du job J_1, puis la première tâche du job J_2 et ainsi de suite jusqu'à la troisième tâche du job J_2.

J_1	J_1	J_2	J_3	J_2	J_1	J_3	J_3	J_2

FIG. 2.15 – Exemple de chromosome pour un problème de permutation avec répétition

Les opérateurs génétiques : De cette façon, le problème de Job-Shop peut être vue comme le problème du voyageur de commerce. Il est donc possible d'utiliser, comme pour le voyageur de commerce, des opérateurs génétiques tels que la mutation swap ou le croisement PMX décrit ci dessous.

PMX : Partial Mapped cross-over

Cet opérateur a été conçu par Goldberg. Il est basé sur des permutations multiples. Pour cela, il faut dans un premier temps créer deux points de coupure sur les deux parents. Nous allons prendre les parents $Parent_1$ et $Parent_2$ ainsi que les points de coupure suivants :

$$\begin{array}{lc|c|c} Parent_1 & = 123 & 456 & 789 \\ Parent_2 & = 412 & 876 & 935 \end{array}$$

Dans un deuxième temps, nous allons recopier le $Parent_1$ dans l'$Enfant_1$ et le $Parent_2$ dans l'$Enfant_2$. Ensuite, nous allons, dans un troisième et dernier temps, réaliser les permutations. Le nombre de permutations à réaliser sur un enfant correspond au nombre de gènes compris entre les points de coupure. Pour l'exemple traité, ce nombre est égal à 3. Les allèles à permuter sont donnés en comparant les allèles, d'un même locus, compris entre les points de coupure des $Parent_1$ et $Parent_2$. Ainsi, les allèles 4 et 8, 5 et 7, 6 et 6 seront permutés sur les enfants. Ces permutations se réalisent les unes après les autres. Ceci donne pour l'enfant 1 :

$Permutation$	$Enfant_1$	$Enfant_2$	$Permutation$
1	$1\ 2\ 3\ \overline{8}\ 5\ 6\ 7\ \overline{4}\ 9$	$\overline{8}\ 1\ 2\ \overline{4}\ 7\ 6\ 9\ 3\ 5$	$4 \leftrightarrow 8$
2	$1\ 2\ 3\ 8\ \overline{7}\ 6\ \overline{5}\ 4\ 9$	$8\ 1\ 2\ 4\ \overline{5}\ 6\ 9\ 3\ \overline{7}$	$5 \leftrightarrow 7$
3	$1\ 2\ 3\ 8\ 7\ \overline{6}\ 5\ 4\ 9$	$8\ 1\ 2\ 4\ 5\ \overline{6}\ 9\ 3\ 7$	$6 \leftrightarrow 6$

L'ordonnanceur de Bierwirth permet d'améliorer la solution correspondant au chromosome en décalant à gauche certaines tâches. De plus, la technique de *forçage génétique* de Yamada et Nakano est utilisée afin de conserver dans la population courante cette solution modifiée.

En général, les méthodes utilisant un codage direct sont capables d'explorer la totalité de l'espace de recherche du problème traité. Par contre, l'espace de recherche, utilisé par les méthodes basées sur un codage indirect, est limité par l'étape intermédiaire de transformation du chromosome. Par exemple, l'espace de recherche exploré par la méthode employée par Kobayashi [Kobayashi et al. 1995] est limité aux ordonnancements actifs. Néanmoins, cette limitation peut s'avérer avantageuse si l'espace de recherche de la méthode est dominant.

2.3 Bilan sur les opérateurs

La forme la plus souvent employée pour coder un problème de Job-Shop est la séquence. Cette forme permet de représenter facilement un ordre de réalisation des tâches ou des jobs. Néanmoins, l'utilisation d'opérateurs génétiques tels que le cross-over à n-points est proscrit. Ce type d'opérateur détruit la séquence en supprimant ou en multipliant des allèles. L'exemple

suivant présente un exemple de destruction d'une séquence par l'application d'un cross-over à 1-point.

$$\begin{array}{llll} P_1 : & 1234|56789 & E_1 : & 1234|54321 \\ P_2 : & 9876|54321 & E_2 : & 9876|56789 \end{array}$$

Selon le problème traité, l'ordre relatif ou absolu des allèles a son importance. Par exemple, l'ordre relatif des villes pour le Traveling Salesman Problem (TSP) est important. En effet, la séquence 12345 est équivalente à la séquence 45123 car la séquence représente un parcours fermé. Par contre, dans un problème d'ordonnancement, c'est l'ordre dans lequel les tâches sont réalisées qui est important. Cet ordre est dit absolu car la position d'une tâche dans la séquence est plus importante que sa position relative par rapport aux autres tâches.

De nombreux travaux ont permis de déterminer quels sont les meilleurs opérateurs de cross-over à utiliser en fonction du problème [Michalewicz 1996]. Le tableau 2.5 résume ces travaux.

Problème	Cross-Over	
	Meilleur	Pire
Ordre relatif	OX, Order Based, Position Based	PMX, CX
Ordre absolu	Position Based, Order Based, CX, PMX	OX

TAB. 2.5 – Comparaison des cross-over en fonction du type de problème

La méthode utilisée dans le chapitre 3 est basée sur un ordre absolu. Les opérateurs génétiques énoncés dans le tableau 2.5 sont décrits dans l'annexe D.

2.4 Conclusion

Les algorithmes génétiques sont des algorithmes stochastiques permettant l'exploration et l'exploitation d'espaces de recherche de grande taille. Ils représentent une bonne heuristique pour résoudre des problèmes NP-difficile. Néanmoins, aucune garantie sur leur convergence existe.

Le choix des paramètres de l'algorithme génétique est parfois difficile et fait souvent appel à une expérimentation importante. De plus, le codage d'une solution n'est pas toujours facile à réaliser. Les algorithmes génétiques permettent d'obtenir des résultats intéressants à condition de connaître le paramétrage adapté au problème traité.

Les problèmes d'ordonnancement, et plus particulièrement les problèmes de Job-Shop ont captivé l'attention de la communauté génétique grâce à leur complexité et par le fait que beaucoup de problèmes ne disposent pas encore de solution exacte. De nombreux travaux sur le codage et sur les opérateurs génétiques ont déjà permis d'obtenir des résultats encourageants.

Le chapitre suivant se propose de décrire les problèmes d'ordonnancement cyclique. Il propose également une résolution basée sur les algorithmes génétiques pour le problème de Job-Shop cyclique.

Chapitre 3

Résolution du problème de Job-Shop cyclique

Dans ce chapitre, nous nous intéresserons aux problèmes cycliques, et plus particulièrement au problème du Job-Shop cyclique. Ce problème est fréquemment rencontré en milieu industriel dit de production de masse. Ce problème se note '$J, n, m/Prec_L, Cyclique/*$' en utilisant une notation étendue de Lawler [Lawler et al. 1985]. Cette notation indique qu'il s'agit d'un problème de Job-Shop à n jobs et m machines. Les "tâches" qui composent les jobs sont liées par des contraintes de précédences linéaires ($Prec_L$).

Ce problème a très peu été abordé dans la littérature par rapport aux problèmes de Job-Shop "classiques". Nous allons présenter dans un premier temps les problèmes d'ordonnancement cyclique, puis dans un deuxième temps nous allons proposer une méthode basée sur les algorithmes génétiques qui permet de résoudre le problème du Job-Shop cyclique. Cette méthode utilise d'un ordonnanceur qui est présenté dans un troisième temps. La validation de cette méthode est réalisée dans une quatrième temps.

3.1 Présentation des problèmes d'ordonnancement cyclique

Les problèmes d'ordonnancement de type cyclique ont de nombreuses applications dans le milieu industriel mais également dans l'informatique. Nous pouvons citer par exemple, l'exécution de calculs vectoriels dans une architecture à processeurs parallèles.

Les paragraphes suivants abordent les différents éléments qui composent un ordonnancement cyclique afin de pouvoir comprendre les spécificités d'un ordonnancement cyclique. Quelques classes de problèmes d'ordonnancement cycliques sont présentées dans un deuxième temps, suivi de leur résolution.

3.1.1 Éléments d'un ordonnancement cyclique

Un problème d'ordonnancement est dit cyclique si les tâches qui le composent sont amenées à être répétées. Ceci apporte des changements importants dans les définitions des éléments qui le composent. Les paragraphes suivants présentent ces différences. Ils proposent également une extension de la notation décrite au chapitre 1.

3.1.1.1 Tâches génériques

Dans un problème d'ordonnancement classique, chaque tâche se réalise une seule fois puis l'ordonnancement est terminé. Dans un ordonnancement cyclique, les tâches sont dites *génériques* et elles se répètent une infinité de fois. Ainsi, l'ordonnancement cherché est lui même infini. Nous pouvons citer par exemple une addition dans un processeur. On note $\ll i, n \gg$ la $n^{ième}$ occurrence de la tâche i. Une occurrence de tâche représente une réalisation de celle-ci.

La solution d'un problème cyclique peut être périodique. C'est à dire qu'il existe un enchaînement de tâches, appelé un *motif* , qui se répète indéfiniment sur un intervalle de temps régulier. Un problème d'ordonnancement cyclique est dit périodique si la solution recherchée à ce problème est elle même périodique. Nous verrons que les solutions périodiques peuvent être dominantes sur certains critères.

En notant $t(i, n)$ la date de début de réalisation de la $n^{ième}$ occurrence de la tâche i, et T l'ensemble des tâches qui composent le problème à résoudre, nous pouvons définir un ordonnancement périodique par la définition 3.1.

Définition 3.1 *Un ordonnancement est dit* périodique *avec une période δ si pour chaque tâche générique i on a, $\forall i \in T$, $\forall k > 1$, $t(i, k) = t(i, 1) + \delta * k$ [Hanen 1994].*

La définition 3.2 permet de définir la *fréquence* de réalisation d'une tâche i où T représente l'ensemble des tâches qui composent le problème à résoudre. La fréquence représente le *débit* de réalisation de la tâche i.

Définition 3.2 *Si $S = \{t(i, n), i \in T, n > 0\}$ est un ordonnancement qui affecte à chaque occurrence n de la tâche i une date de début $t(i, n)$, alors la* fréquence *de la tâche générique i est définie par :* $\lim_{n\to\infty} \frac{n}{t(i,n)}$*, si la limite existe. [Munier 1996]*

3.1.1.2 Contraintes de précédences

Dans un ordonnancement "classique", les contraintes de précédences portent sur la tâche. Elles permettent de représenter des relations de succession entre les tâches. Dans un ordonnancement cyclique, les contraintes de précédences portent sur les occurrences de la tâche.

Il existe deux types de contraintes de précédences dans les problèmes d'ordonnancement cyclique : les contraintes uniformes et les contraintes linéaires. Elles sont définies de la façon suivante :

- Une contrainte uniforme $e = (i, j)$, entre les tâches i et j, valuée de (p_i, β, ω) se note : $\forall n > 0$, $t(i, n + \beta) + p_i \leq t(j, n + \omega)$ [Munier 1996].
- Une contrainte linéaire $e = (i, j)$, entre les tâches i et j, valuée de $(p_i, \alpha, \beta, \gamma, \omega)$ se note : $\forall n > 0$, $t(i, \alpha n + \beta) + p_i \leq t(j, \gamma n + \omega)$ [Munier 1996].

Les paramètres β et ω sont également appelés : *hauteur*. De même, les paramètres α et γ sont appelés : *longueur*.

Remarque 3.1 Dans le cas de contraintes linéaires les paramètres α et γ sont strictement positifs alors que $\beta \in]-\alpha; +\infty]$ et $\omega \in]-\gamma + 1; +\infty]$.

Ces deux équations peuvent se lire de la façon suivante :

- Contrainte uniforme : Pour toutes valeurs de n strictement positives, la $n + \omega$ ième occurrence de la tâche j, notée $\ll j, n + \omega \gg$, ne peut commencer au plus tôt que lorsque la $n + \beta$ ième occurrence de la tâche i, notée $\ll i, n + \beta \gg$, est terminée.

– Contrainte linéaire : Pour toutes valeurs de n strictement positives , la $\gamma n + \omega$ ième occurrence de la tâche j, notée $\ll j, \gamma n + \omega \gg$, ne peut commencer au plus tôt que lorsque la $\alpha n + \omega$ ième occurrence de la tâche i, notée $\$ \ll i, \alpha n + \beta \gg$, est terminée.

Les contraintes uniformes permettent ainsi d'obtenir un écart constant entre les numéros d'occurrence de deux tâches alors que les contraintes linéaires permettent d'obtenir un écart constamment croissant entre les numéros d'occurrence de deux tâches. Une contrainte uniforme est en fait une contrainte linéaire particulière où $\alpha = \gamma = 1$.

3.1.1.3 Visualisation des contraintes

Afin d'appréhender plus facilement les contraintes de précédence, nous proposons un 'diagramme d'occurrences' qui permet de représenter une contrainte entre deux tâches i et j. Ce diagramme d'occurrences permet de relier toutes les occurrences de i avec les occurrences de j. La figure 3.1 représente le cas d'une contrainte uniforme $e = (i, j)$ valuée de (p_i, β, ω). La durée de réalisation de la tâche i, noté p_i, n'est pas représenté sur ce diagramme car cette figure décrit uniquement la contrainte $e = (i, j)$ entre les occurrences de la tâche i et j. Ce diagramme permet de déterminer les occurrences de j réalisables en fonction des occurrences de i déjà réalisées. Les triangles noirs de ce diagramme représentent les occurrences de j réalisables par rapport à une occurrence i déjà réalisée. La figure 3.1 présente une contrainte de précédence uniforme $e = (i, j)$, valuée de $(p_i, 2, 3)$. Sur cette figure, nous pouvons voir que les trois premières occurrences de la tâche j ne sont pas contraintes par la tâche i. De plus, lorsque la tâche i est terminée pour la troisième fois, l'occurrence quatre de la tâche j peut commencer. Ensuite, lorsque la tâche i termine une occurrence $n + \beta$ alors la tâche j peut commencer l'occurrence $n + \omega$. Le paramètre β représente donc le nombre d'occurrences supplémentaires de la tâche i à réaliser pour satisfaire la première fois la contrainte uniforme. Le paramètre ω représente le nombre d'occurrences de j non contraintes par la tâche i.

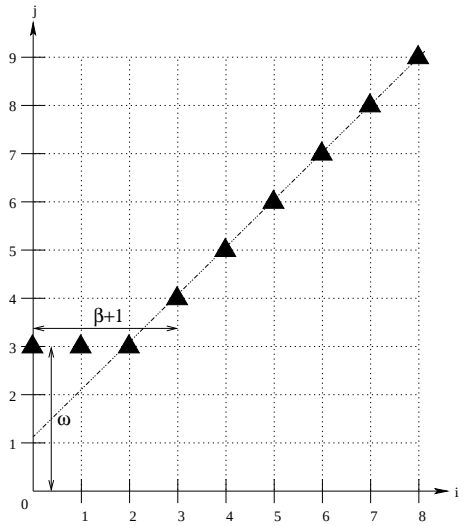

FIG. 3.1 – Diagramme d'occurrences pour une contrainte uniforme

La figure 3.2 représente le cas d'une contrainte linéaire $e = (i, j)$ valuée de $(p_i, 2, 2, 3, 1)$. En traçant une droite passant par les sommets des 'marches', on obtient un coefficient de linéarité représentant l'écart uniformément croissant entre les occurrences de i et j. Ce coefficient de linéarité est égal à $\frac{\gamma}{\alpha}$. Dans le cas d'une contrainte linéaire, ce coefficient de linéarité est strictement

positif. Dans le cas d'une contrainte uniforme, ce coefficient est égal à $\frac{\gamma}{\alpha} = 1$.

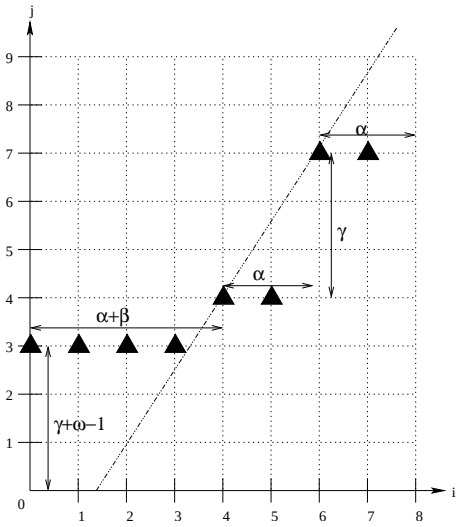

FIG. 3.2 – Diagramme d'occurrences pour une contrainte linéaire

Exemple 3.1 La figure 3.3 représente le diagramme d'occurrences de la contrainte linéaire entre les occurrences des tâches i et j de l'exemple de la figure 3.4. Les triangles noirs du diagramme représentent les occurrences de j réalisables par rapport à une occurrence i déjà réalisée. Ce diagramme se lit de la façon suivante : lorsque la $5^{ième}$ occurrence de i est terminée, le diagramme nous montre que seules l'occurrence *1* de la tâches j peut être réalisées ou en cours de réalisation. L'occurrence 2 de la tâche j peut démarrer au plus tôt lorsque la 5^{ieme} occurrence de i vient de se terminer.

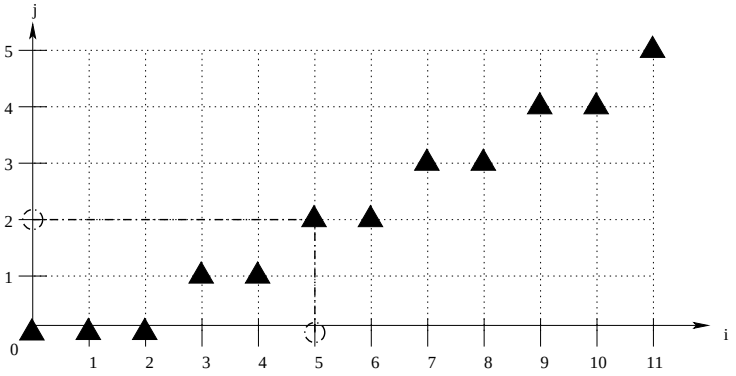

FIG. 3.3 – Diagramme d'occurrences du graphe de la figure 3.4

3.1.1.4 Modélisation par les graphes de précédences à contraintes linéaires

Pour modéliser l'ensemble T des tâches à ordonnancer et l'ensemble P des contraintes linéaires, un graphe de précédences $G = \{T, P\}$ est utilisé [Hanen et al. 1993]. Sur ce graphe, les noeuds représentent les tâches, les arcs représentent les contraintes de précédences entre les tâches et les arêtes correspondent aux contraintes de ressources.

Les noeuds

Les noeuds représentent les tâches à réaliser. Les noeuds sont graphiquement représentés par un label encerclé. Le label représente le numéro ou le nom d'une tâche.

Les arcs

Les arcs représentent les contraintes de précédences existantes entre les occurrences des tâches. Le nombre de valeurs sur les arcs dépendra du type de contrainte.

Une contrainte uniforme $e = (i, j)$ est représentée par un arc valué d'un triplet (p_i, β, ω), alors qu'une contrainte linéaire $e = (i, j)$ est représentée par un arc comportant les cinq paramètres suivants $(p_i, \alpha, \beta, \gamma, \omega)$. La figure 3.4 présente un exemple comportant deux tâches i et j et une contrainte linéaire $e = (i, j)$ avec pour valeurs *(3,2,1,1,0)*. Cette contrainte peut également s'écrire sous la forme suivante : $\forall n > 0$, $t(i, 2n + 1) + 3 \leq t(j, n)$. Dans cet exemple, la tâche i a un temps de réalisation de 3 unités de temps.

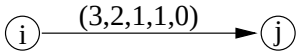

FIG. 3.4 – Exemple de graphe de précédences

La figure 3.5 présente le diagramme de Gantt associé à l'exemple 3.4. Sur cette figure, deux parties apparaissent : une partie dite transitoire et une partie stationnaire. La partie transitoire représente la phase de "démarrage" où les décalages causés par les paramètres β, γ et ω sont pris en compte, alors que la partie stationnaire représente un fonctionnement permanent où seuls les paramètres α et γ sont utilisés.

En effet, avec l'exemple de la figure 3.4, il faut réaliser 3 occurrences de i, correspondant à $\alpha + \beta$, avant de pouvoir réaliser la première occurrence de j, correspondant à $\gamma + \omega$. Ensuite, toutes les deux (α) occurrences de i nous pouvons réaliser une (γ) occurrence de j . Dans cette figure, les flèches en pointillées représentent "l'autorisation" de commencer une tâche au plus tôt.

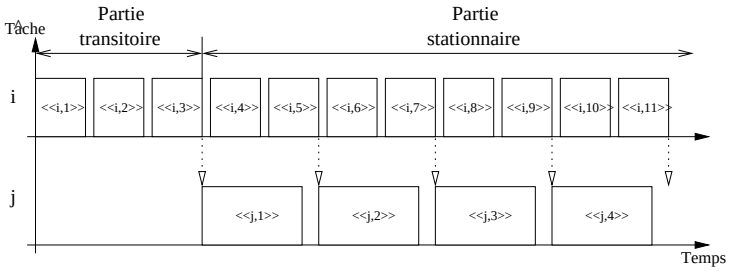

FIG. 3.5 – Gantt associé à la figure 3.4

Le paramètre β représente le nombre d'occurrences supplémentaires de la tâche i à réaliser pour satisfaire la première fois la contrainte. Par contre, le terme $(\gamma + \omega - 1)$ représente le nombre d'occurrences non contraintes que la tâche j peut réaliser dans la phase transitoire.

Les arêtes

Les contraintes de ressources sont représentées par des arêtes. Ces arêtes représentent la partie disjonctive du graphe car ils ne possèdent pas d'orientation. Donner une orientation à ces arêtes permet de déterminer un ordonnancement. La figure 3.6 représente un problème cyclique

avec des contraintes de précédences linéaires. Il est composé de 4 tâches génériques et de deux ressources capables de réaliser les tâches 1,2 et 3 pour la première ressource et la tâche 4 pour la deuxième ressource. Les arêtes associées aux ressources sont en pointillées.

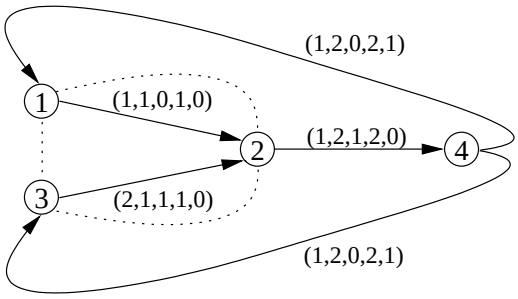

FIG. 3.6 – Exemple de graphe

Les arêtes représentant une ressource forment un graphe complet avec les noeuds que la ressource est capable de réaliser. Ainsi, un noeud relié à aucune arête indique que ce noeud est affecté à une ressource capable de réaliser uniquement la tâche représentée par ce noeud.

3.1.1.5 Critères d'évaluation

Les tâches étant répétées une infinité de fois, il semble inutile d'utiliser les critères des ordonnancements "classiques". L'évaluation d'un ordonnancement cyclique porte en général sur la notion de fréquence, de temps de cycle ou de latence.

- la fréquence d'une tâche a été définie précédemment par la définition 3.2.
- Le temps de cycle d'une tâche représente le temps moyen qui sépare deux occurrences successives d'une même tâche.
- La latence représente le nombre de temps de cycle nécessaire pour la réalisation complète de toutes les tâches génériques qui composent un job.

La figure 3.7 présente un exemple de job composé de 5 tâches qui se réalisent sur 3 ressources. Les contraintes de précédence de ce job sont les suivantes : 1<2, 1<3, 2<4, 3<4, 4<5. Sur cette figure, un motif est répété périodiquement. La détermination de la latence d'un job se réalise en respectant les contraintes de précédences. Les occurrences encadrées représentent une occurrence du job. Sur cet exemple, il a fallu trois débuts de motif pour finir complètement une occurrence de job. La latence de ce job est donc de 3 [Draper et al. 1999].

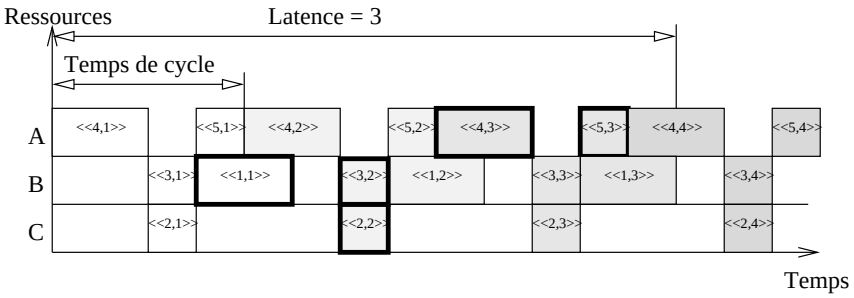

FIG. 3.7 – Diagramme de Gantt

3.1.1.6 Extension de la notation

La notation décrite dans le paragraphe 1.1.3.4 est insuffisante pour décrire les problèmes d'ordonnancement cyclique. Pour cela, nous proposons d'étendre cette notation au niveau du champ des critères d'évaluation γ et du champ des contraintes du système β.

Le champ β est composé de cinq sous-champs β_1 à β_5. Le champ β_2 permet de représenter les contraintes existantes entre les tâches. Ce champ est étendu par :

- $\beta_2 = Prec_L$: relations de précédences avec des contraintes linéaires.
- $\beta_2 = Prec_U$: relations de précédences avec des contraintes uniformes.

Un nouveau sous-champ β_6 doit être ajouté afin de spécifier si le problème est cyclique ou périodique. Ceci se note de la façon suivante :

- $\beta_6 = Cyclic$: Le problème d'ordonnancement est un problème cyclique.
- $\beta_6 = Périodique$: Le problème d'ordonnancement est un problème cyclique dont la solution recherchée est périodique.

Le champ γ est étendu avec les critères suivants :

- $Deb_i = \lim_{n\to\infty} \frac{n}{t(i,n)}$: Le débit Deb_i d'une tâche i donnée.
- $Deb_{min} = \min_{i\in T}\{Deb_i\}$: Le plus petit débit de l'ensemble T des tâches.
- $Tc_i = \lim_{n\to\infty} \frac{t(i,n)}{n}$: Le temps de cycle d'une tâche i.
- $Tc_{max} = \max_{i\in T}\{Tc_i\}$: Le temps de cycle le plus long où T représente l'ensemble des tâches à réaliser.
- $MoyTc = \sum_{i\in T} \frac{Tc_i}{Card(T)}$: La moyenne des temps de cycle des jobs.
- $Tc_{moy} = \lim_{n\to\infty} \frac{\sum_{i\in J} q_{i,n}}{Max_{j\in T}(t(j,n)+p_j)}$: Le temps de cycle moyen d'un job, avec $q_{i,n}$ le nombre d'occurrence réalisées du job i pour une valeur n donnée.
- $Lt_i = \frac{\lceil t(der,n)+p_{der}-t(prem,m)\rceil}{K}$: La latence d'un job J_i où :
  - K est la période de l'ordonnancement
  - $t(der,n)$ la date de début de réalisation de la dernière tâche du job J_i
  - $t(prem,m)$ la date de début de réalisation de la première tâche du job J_i.
  - $\frac{\lceil a\rceil}{b}$ représente la division entière par excès de a par b.
- $Lt_{max} = \max_{i\in J}\{Lt_i\}$: La latence la plus longue parmi l'ensemble J des jobs.

Des critères hybrides peuvent être créés en couplant plusieurs critères. Par exemple, il est possible d'utiliser le temps de cycle moyen des jobs pour une latence donnée. Ce critère se note : $Tc_{moy} \setminus Lt_{max} = k$.

3.1.2 Les classes de problèmes cycliques

Ce paragraphe présente quatre classes de problèmes d'ordonnancement cyclique qui sont : le Basic Cyclic Scheduling (BCS), le BCS avec contraintes de précédences linéaires, le Job-Shop périodique et cyclique.

3.1.2.1 Le Basic Cyclic Scheduling problem

Le BCS est considéré comme le problème d'ordonnancement cyclique le plus simple. Il est défini par :

Définition 3.3 *Le "Basic Cyclic Scheduling" [Hanen et al. 1995] est composé d'un ensemble $T = \{1,...,n\}$ de tâches qui seront exécutées une infinité de fois. Le temps de réalisation de chaque tâche est noté p_i, $i \in T$. Les tâches sont contraintes par un ensemble P de contraintes uniformes.*

Si $S = \{t(i,n), i \in T, n > 0\}$ *est un ordonnancement affectant à chaque occurrence* n *de la tâche générique* i *une date de début de réalisation* $t(i,n)$*, alors le temps de cycle* $w(S)$ *est donné par :*

$$w(S) = \limsup_{k \to \infty} \frac{\max\{t(i,k) + p_i \mid i \in T\}}{k}$$

Dans ce type de problème avec contraintes de précédences uniformes et sans contrainte de ressources, l'objectif est de minimiser le temps de cycle.

3.1.2.2 BCS avec contraintes de précédences linéaires

Cette classe de problème est dérivée de la classe BCS. Elle est notée par la suite BCSL. La seule différence est que les contraintes de précédences entre les tâches sont linéaires [Munier 1996].

3.1.2.3 Le Job-Shop cyclique et périodique

Cette classe est dérivée de la classe des Job-Shop [Draper et al. 1999]. Les jobs sont composés de tâches génériques. Il existe entre ces tâches des contraintes de précédences uniformes ou linéaires. Les ordonnancements recherchés peuvent être des ordonnancements périodiques. Des contraintes de ressources sont présentes.

3.2 Résolution des problèmes d'ordonnancement cyclique

Très peu de méthodes exactes permettent de résoudre les problèmes d'ordonnancement cyclique. Face à leur complexité, les méthodes méta-heuristiques semblent être adaptées à leur résolution. Pour réduire la difficulté, beaucoup d'auteurs limitent leurs recherches aux ordonnancements périodiques car ces ordonnancements sont dominants pour tous critères réguliers s'il n'y a pas de contrainte de ressources [Hanen 1994].

Nous allons présenter la méthode de résolution exacte du BCS, puis celle du BCS avec contraintes linéaires ainsi qu'une approche de résolution pour le Job-Shop périodique.

3.2.1 Le BCS

Ramchandani a prouvé [Ramchandani 1973] que pour une valeur $w(G)$ du chemin critique du graphe G, il est possible de trouver en un temps polynômial un ordonnancement périodique. Cet ordonnancement périodique se trouve en $O(n^3 \log n)$, où n représente le nombre de tâches. Ici, le chemin critique est donné par la relation suivante : $w(G) = max_{c \in C(G)} \frac{L(c)}{H(c)}$ avec 'c' un circuit possible dans l'ensemble $C(G)$ des circuits possibles du graphe G, $L(c)$ la longueur du circuit 'c' et $H(c)$ sa hauteur. En appelant $e = (i,j)$ la contrainte uniforme reliant la tâche i à la tâche j, $E(c)$ représente l'ensemble des contraintes qui composent le circuit c. La longueur et la hauteur du circuit c sont alors données par : $L(c) = \sum_{i \in c} p_i$ et $H(c) = \sum_{e \in E(c)} \omega_e - \beta_e$ avec ω_e et β_e les valeurs de la contrainte uniforme $e = (i,j)$ [Hanen et al. 1993].

3.2.2 Le BCSL

Munier [Munier 1996] a montré qu'il est possible de transformer un BCS linéaire en BCS. Pour cela, les tâches génériques seront dupliquées afin de transformer les contraintes linéaires en contraintes uniformes.

Si le nombre de duplications n_i de la tâche i et le nombre de duplications n_j de la tâche j du problème d'ordonnancement cyclique vérifient le théorème 3.1 alors il est possible de transformer le BCS linéaire en BCS. Le BCS se résolvant de façon exacte, il est alors possible de résoudre le BCS linéaire de façon exacte.

Théorème 3.1 *Si le nombre de duplications n_i et n_j vérifient :*

$$\frac{n_i}{\alpha_e} = \frac{n_j}{\gamma_e} = x \in \mathbb{N}$$

alors la contrainte linéaire $e = (i, j)$, valuée de $(p_i, \alpha_e, \beta_e, \gamma_e, \omega_e)$, est équivalente à x contraintes uniformes.

Prenons l'exemple d'une contrainte $e = (2, 1)$ avec les valeurs $(2, 2, 0, 3, 1)$. La tâche générique 1 est remplacée par les trois tâches génériques $(1, 1)$, $(1, 2)$ et $(1, 3)$. La $n^{ième}$ occurrence de $(1, 1)$, $(1, 2)$ et $(1, 3)$ remplace respectivement les occurrences $3(n-1)+1$, $3(n-1)+2$ et $3n$ de la tâche générique 1. De même, la tâche générique 2 est remplacée par les tâches génériques $(2, 1)$ et $(2, 2)$. Ainsi, $e = (2, 1)$ est équivalent à $e' = ((2, 2), (1, 1))$ avec pour valeurs $\beta_{e'} = 0$ et $\omega_{e'} = 1$. La figure 3.8 représente le résultat de la transformation.

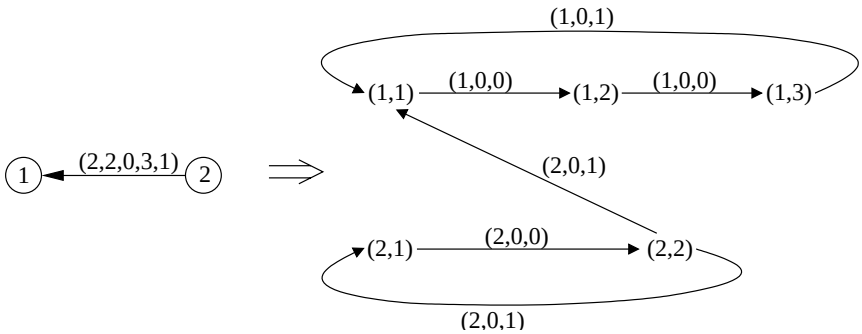

FIG. 3.8 – Contrainte uniforme équivalente

3.2.3 Le Job-Shop périodique

Nous allons dans un premier point présenter un cas particulier de ce problème où les contraintes de précédences sont uniformes avec des hauteurs nulles et dont le graphe représentant le problème ne possède pas de circuit. Un second point présente le cas général où les hauteurs des contraintes sont quelconques et le graphe représentant le problème possède des circuits.

3.2.3.1 Cas particulier de Draper

Pour un problème de Job-Shop périodique à contraintes de précédences uniformes dont les hauteurs sont nulles, noté $(J, m, 1/Prec_U, cyclic/*)$, Draper et al. [Draper et al. 1999] proposent une méthode originale. Cette méthode consiste à chercher un motif d'ordonnancement et à le répéter. La figure 3.10 présente un motif possible pour un problème de Job-Shop à *5* tâches et *3* ressources dont les contraintes de précédences et de ressources sont représentées par la figure 3.9. Dans ce problème, trois ressources sont disponibles. Les tâches 1 et 2 sont affectées à la ressource A, la tâche 4 à la ressource B et les tâches 3 et 5 sont affectées à la ressource C. Ce motif est ensuite recopié une 'infinité' de fois en essayant au maximum de réaliser des recouvrements entre

deux motifs successifs. On parle de recouvrement entre deux motifs successifs s'il est possible de commencer la réalisation d'un motif $i+1$ avant la fin de réalisation du motif i. En appelant T_m le temps de réalisation du motif alors le temps de cycle Tc d'une tâche quelconque du motif est donné par : $Tc = T_m - T_R$, avec T_R le temps de recouvrement.

La figure 3.11 représente l'ordonnancement associé au motif. Il est possible de voir sur cette figure que deux motifs successifs sont recouvrables. Il est parfois possible que le recouvrement de deux motifs soit impossible. Ensuite, il faut affecter à chaque occurrence d'une tâche son numéro d'occurrence. Ceci se détermine en respectant les contraintes de précédences. Le temps de cycle d'un job peut donc être calculé. Sur la figure 3.11, les occurrences des tâches sont représentées par un contour plus épais représente une occurrence du job. Il est donc possible que les tâches appartenant à un motif ne portent pas forcément le même numéro d'occurrence.

Le problème de Job-Shop cyclique devient un problème où il faut trouver un motif qui maximise ou minimise un critère de performance donné. Cette recherche peut être réalisée par une méthode quelconque d'exploration.

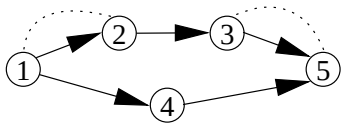

FIG. 3.9 – Exemple de problème de Job-Shop cyclique

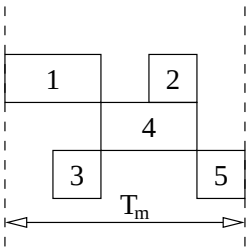

FIG. 3.10 – Exemple de motifs d'ordonnancement

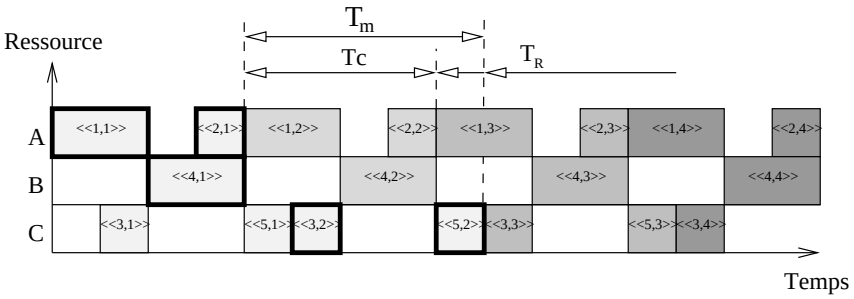

FIG. 3.11 – Exemple d'ordonnancement cyclique avec recouvrement de motifs

3.2.3.2 Cas général du Job-Shop périodique

Le problème de Draper reste simple car le graphe qui le représente ne possède pas de circuit. Dans le cas général du Job-Shop périodique de période w, avec contraintes de ressource et de

précédences uniformes, il est possible de déterminer la valeur du chemin critique. Néanmoins, cette détermination reste difficile à cause des éventuels circuits dans le graphe et à cause des disjonctions des contraintes de ressource. Un circuit est chemin passant commençant par un noeud i, et se terminant par le même noeud i. La figure 3.12 possède plusieurs circuits. Par exemple, $1, 2, 3$ est un circuit possible.

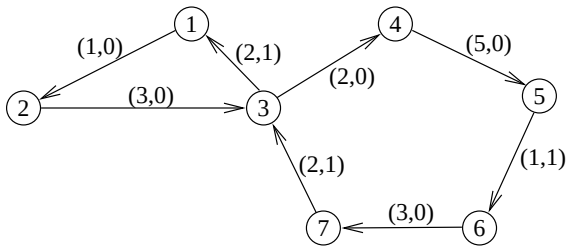

FIG. 3.12 – Exemple de graphe comportant des circuits

Prenons l'exemple d'une contrainte disjonctive entre deux tâches i et j se réalisant sur la même ressource. Pour chaque occurrence $\ll i, k \gg$ et $\ll j, l \gg$ de ces deux tâches, nous avons :

$$soit\, t(i,1) + p_i + (k-1)w \geq t(j,1) + (l-1)w$$
$$ou\, t(j,l) + p_j + (l-1)w \geq t(i,k) + (k-1)w$$

Cette contrainte disjonctive peut également s'écrire [Hanen 1994] :

$$\exists k_{ij} \in \mathbb{Z},\, t(j,1) \geq t(i,1) + p_i - k_{ij}\, et\, k_{ij} + k_{ji} = 1$$

Nous appelons sélection d'une disjonction entre les tâches i et j le fait de choisir une valeur $k_{ij} \in \mathbb{Z}$. Une sélection est complète si toutes les disjonctions font l'objet d'un choix. Réaliser une sélection complète permet de calculer la valeur du chemin critique du graphe combinant le graphe G représentant le problème et le graphe uniforme induit par la sélection complète.

Ainsi, le problème de job-shop périodique revient à déterminer la meilleure sélection complète.

Le Job-Shop cyclique est une version plus générale du Job-Shop périodique dans le sens où la solution recherchée n'est pas forcément périodique. Ce problème ainsi qu'une approche de résolution sont détaillés dans le paragraphe suivant.

3.3 Proposition de résolution du Job-Shop cyclique à contraintes linéaires

Le Job-Shop cyclique avec contraintes de précédences linéaires et contraintes de ressources est un problème encore peu étudié à ce jour. Nous allons dans un premier temps décrire complètement le problème traité. Une approche de résolution basée sur un algorithme génétique est proposée dans un deuxième temps.

3.3.1 Présentation du problème

Le problème traité est un problème d'ordonnancement de Job-Shop cyclique décrit comme suit :

- Un ensemble $T = \{1, ..., n\}$ de n tâches génériques est à ordonnancer sur un ensemble $R = \{1, ..., m\}$ de m ressources. Ces tâches se regroupent en job. Ainsi, il existe un ensemble $J = J_1, ..., J_m$ de jobs. Chaque tâche appartient à un et un seul job.

- Les ressources sont disjonctives et à exemplaire unique.
- Les tâches sont non-préemptives et elles sont pré-affectées aux ressources.
- Les tâches sont non-réentrantes. C'est à dire qu'il ne peut y avoir qu'une seule occurrence de la tâche en cours de réalisation à chaque instant t.
- Le temps de réalisation des tâches sur les ressources est connu à l'avance et il est noté p_i pour la tâche i.
- Une tâche est réalisée par une seule ressource, par contre plusieurs tâches peuvent être affectées à la ressource.
- Un ensemble P de contraintes linéaires de précédences reliant les tâches génériques.
- L'objectif est de minimiser le temps de cycle moyen des jobs.

Les ordonnancements périodiques n'étant pas dominants pour le critère choisi, nous ne nous sommes pas focalisés sur ce type d'ordonnancement. Nos recherches porteront sur l'ensemble des ordonnancements cycliques possibles.

3.3.1.1 Modélisation par les graphes de précédences linéaires

Il est possible de modéliser ce problème par un graphe de précédences à contraintes linéaires. La figure 3.13 représente un problème de Job-Shop cyclique à trois jobs et deux jobs de synchronisation. Ici, les tâches 0 et 9 représentent les jobs de synchronisation J_{sync1} et J_{sync2} entre les différents jobs. Les jobs sont dit *dépendants* car il faut attendre la fin de réalisation de l'occurrence i de tous les jobs pour pouvoir démarrer l'occurrence $i+1$ de chaque job. Il est possible que ces points de synchronisation n'existent pas. Dans un tel cas, un job n'est pas obligé d'attendre la fin de réalisation des autres jobs pour démarrer une nouvelle occurrence. Les jobs sont alors appelés : jobs *indépendants*. La figure 3.14 présente un autre exemple de Job-Shop cyclique dont les jobs sont indépendants.

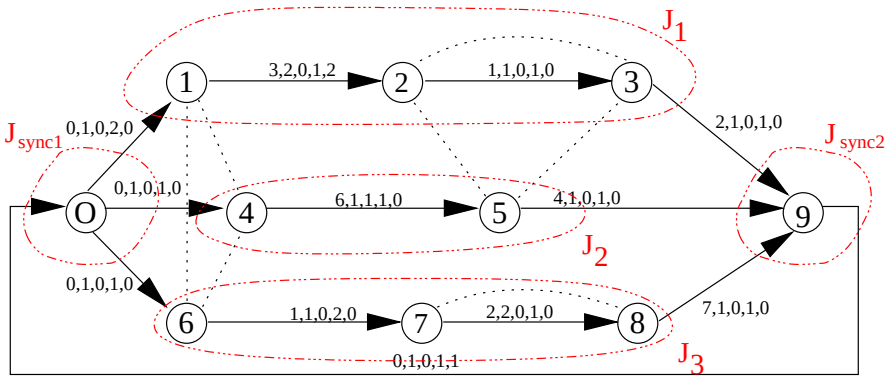

FIG. 3.13 – Exemple de Job-Shop cyclique à contraintes linéaires avec jobs dépendants

De manière générale, deux jobs I et J sont dépendants si il existe au moins un arc $e = (i, j)$ tel que la tâche i soit une tâche du job I et que la tâche j soit un tâche du job j.

3.3.2 Une approche de résolution par les Algorithmes Génétiques

Étant donné la complexité du problème due à la taille de l'espace de recherche, les algorithmes génétiques semblent une méthode de résolution adaptée. Pour ce type de problème, deux approches différentes sont présentées. Ces deux approches sont toutes les deux basées sur une méthode appelée par la suite GA-ORDO. Cette méthode couple un ordonnanceur avec un algo-

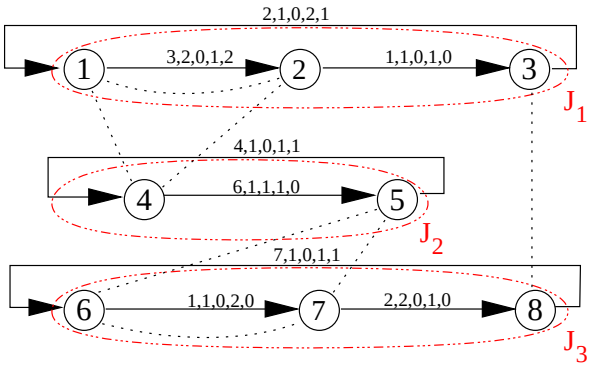

FIG. 3.14 – Exemple de Job-Shop cyclique avec jobs indépendants

rithme génétique. Nous allons présenter dans un premier temps le principe de cette approche, puis dans un deuxième temps les deux approches employées.

3.3.2.1 Principe de GA-ORDO

La méthode GA-ORDO utilise un codage indirect. Le principe consiste à utiliser un ordonnanceur qui crée un ordonnancement qui est lui-même évalué ensuite [Dupas et al. 2000b].

La figure 3.15 représente le couplage entre l'algorithme génétique et l'ordonnanceur. L'ordonnanceur utilise le GPCL représentant le problème à optimiser et le transforme en Réseau de Petri. Cette transformation est décrite dans le paragraphe 3.4. Ensuite, l'ordonnanceur crée l'ordonnancement en utilisant le RdP pour déterminer les dates de réalisation des occurrences des tâches. En cas de conflit d'utilisation d'une ressource, l'ordonnanceur utilise une gestion des conflits basée sur l'emploi d'heuristiques. Chaque ressource dispose d'une heuristique qui est fixée par le chromosome transmis par l'algorithme génétique. Lorsque l'ordonnancement est terminé, l'ordonnanceur évalue celui-ci en fonction du critère choisi. Ce résultat est transmis à l'algorithme génétique. Il représente le fitness de l'individu à évaluer.

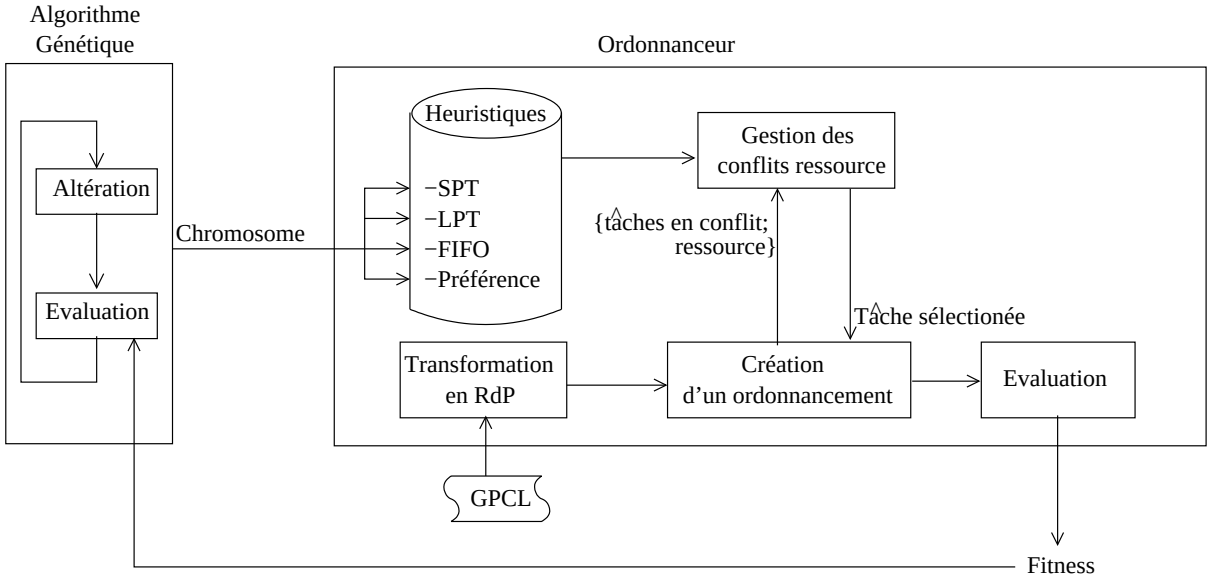

FIG. 3.15 – Résolution des conflits ressource

3.3.2.2 Les différentes approches de résolution

Il est possible de caractériser un ordonnancement infini par les heuristiques qui permettent de le construire. Deux approches basées sur cette idée sont proposées. Les heuristiques employées dans ces approches permettent de résoudre les conflits de ressource lors de la construction de l'ordonnancement. La première heuristique est basée sur une liste de préférences de réalisation des tâches alors que la deuxième est basée sur le pilotage des ressources par le choix d'une heuristique. Une troisième approche qui consiste à créer un hybride des deux premières approches est également proposée.

a) Approche basée sur les tâches

Dans ce problème de Job-Shop cyclique, une ressource étant disjonctive, une seule tâche est réalisée à la fois. Ce type de ressource rend le problème disjonctif. L'idée est donc de rendre le problème conjonctif en donnant une *liste de préférences* de réalisation des tâches sur une ressource.

Cette idée est proche de celle de Tamaki et Nishikawa [Tamaki et al. 1992, Falkenauer et al. 1991] qui voulaient orienter chaque arête représentant les contraintes de ressources dans un problème de Job-Shop. Le fait d'orienter les arêtes représentant les contraintes de ressources revient à créer une *liste de préférences* de réalisation des tâches sur la ressource. Avec cette méthode, la formation de cycles empêche la construction de l'ordonnancement par une recherche de chemin critique.

De plus, dire qu'une ressource doit réaliser les tâches 1, puis 2 puis 3 n'a pas de signification dans un problème cyclique à contraintes de précédences linéaires. Prenons l'exemple d'une contrainte de précédences linéaire e=(1,2) avec pour valeur (1,2,0,1,0). Dans cet exemple, il faut deux occurrences de 1 pour réaliser une occurrence de 2. Ainsi, dire que la ressource doit réaliser la tâche 1 puis la tâche 2 bloquera l'ordonnancement car il faut réaliser deux fois la tâche 1 avant de réaliser une fois la tâche 2.

La méthode de Tamaki et de Nishikawa ne peut donc pas être utilisée directement. L'idée est donc d'interpréter l'orientation des arêtes représentant les contraintes de ressources comme une *priorité de réalisation* d'une tâche par rapport à une autre tâche. Les *priorités de réalisation* sont utilisées lorsque l'ordonnanceur crée l'ordonnancement et qu'un conflit de ressource apparaît. Ceci permet de supprimer les cycles et les problèmes liés aux occurrences multiples des tâches.

Ainsi, l'algorithme génétique fourni à l'ordonnanceur une *liste de préférences* pour chaque ressource présente dans le problème à ordonnancer. L'heuristique permettant de résoudre les conflits de ressource est toujours la même. L'heuristique employée est appelée par la suite "Préférence". Cette heuristique est utilisée pour toutes les ressources du problème. La *liste de préférences* permet de déterminer la *priorité de réalisation* de chaque tâche.

Lors d'un conflit de ressource, l'heuristique "Préférence" possède deux niveaux de traitement :

- ⋆ Le premier niveau porte sur le nombre d'occurrences déjà réalisées des tâches en conflit. Les tâches ayant le nombre d'occurrences déjà réalisées le plus faible sont retenues. En effet, ceci empêche une tâche de monopoliser une ressource pour réaliser toutes ses occurrences et permet de finir les jobs commencés.
- ⋆ Le second niveau porte sur les priorités de réalisation des tâches. Chaque ressource dispose d'une *liste de préférences* qui permet de déterminer quelle tâche doit être réalisée en premier, en cas de conflit.

Cette approche peut se résumer par l'algorithme 3.1.

Algorithme 3.1 Choisir une tâche (approche basée sur les tâches)

Entrée : T_r : Ensemble de tâches en conflit.
Entrée : R_l : Ressource sur laquelle il y a conflit.
Sortie : T_i : La tâche choisie dans l'ensemble T_r.

Créer l'ensemble T'_r des tâches en conflit ayant le nombre d'occurrences réalisées le plus faible.
Choisir T_i comme étant la tâche de T'_r ayant la priorité de réalisation la plus élevée sur R_l.

Codage de la solution

Dans cette approche, nous codons dans le chromosome la *liste de préférences*. La *priorité de réalisation* d'une tâche se détermine selon la position de la tâche dans la *liste de préférences*. Ainsi, plus la tâche se situe près du début de la liste, plus la tâche est prioritaire par rapport aux tâches se situant après. De cette façon, la liste $\{5, 1, 3\}$ indique que la tâche 5 est prioritaire par rapport aux tâches 1 et 3 et que la tâche 1 est prioritaire par rapport à la tâche 3. La tâche 3 se voit donc attribuer la *priorité de réalisation* la plus faible. Le chromosome représente directement la *liste de préférences*. Ainsi, chaque individu est légal et faisable.

Nous allons illustrer le codage de cette approche par l'exemple de la figure 3.16 où les tâches *2* et *4* sont réalisées par une ressource A alors qu'une ressource B réalise les tâches *1*, *3* et *5*.

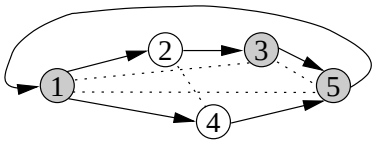

FIG. 3.16 – Graphe de précédences d'un Job-Shop cyclique

Chaque ressource dispose d'une *liste de préférences*. Cette *liste de préférences* peut également être vue comme un ordre préférentiel dans lequel les tâches doivent se réaliser sur cette ressource. En prenant l'exemple donné, nous pouvons obtenir le chromosome suivant :

4	2	5	1	3

Dans ce chromosome, les deux premiers allèles correspondent à la première ressource et les trois suivants à la deuxième ressource.

Imaginons maintenant quelques situations possibles afin de mieux comprendre le fonctionnement de cette approche.

Exemple 3.2 Les occurrences suivantes, $\ll 1, 2 \gg$, $\ll 5, 2 \gg$ et $\ll 3, 2 \gg$, sont réalisables au même instant t. Le premier niveau de la méthode de choix conserve ces trois occurrences car elles sont toutes les trois au même numéro d'occurrence. Par contre, le deuxième niveau choisi l'occurrence $\ll 5, 2 \gg$ car d'après le chromosome ci-dessus, c'est la tâche 5 la plus prioritaire. C'est donc l'occurrence $\ll 5, 2 \gg$ qui est retenue.

Exemple 3.3 Dans cet exemple, les occurrences $\ll 1, 2 \gg$, $\ll 5, 3 \gg$ et $\ll 3, 2 \gg$ sont réalisables au même instant t. Le premier niveau de choix retient les occurrences $\ll 1, 2 \gg$ et $\ll 3, 2 \gg$ car la tâche 5 a une occurrence de plus déjà réalisée par rapport aux tâches 1 et 3. Le deuxième niveau

de choix retient l'occurrence $\ll 1,2 \gg$ car c'est elle qui est prioritaire d'après le chromosome. C'est donc l'occurrence $\ll 1,2 \gg$ qui est retenue pour cet exemple.

Opérateurs génétiques

Cette approche étant basée sur une *liste de préférences*, les allèles contenus dans le chromosome ne peuvent en aucun cas être modifiés par mutation et ils doivent tous être présents à un et un seul exemplaire dans le chromosome. C'est pour ces raisons que les opérateurs classiques, tels que le cross-over à n-points, sont rejetés. De même, les mutations de type "aléatoire" ne peuvent être utilisées ici.

Nous avons vu dans le paragraphe 2.3 qu'il existe des opérateurs génétiques adaptés au traitement de séquences. Des opérateurs tels que PMX et Order Based peuvent être utilisés ici pour le cross-over.

Nous avons réalisé un test comparatif entre les opérateurs PMX et Order Based. Pour cela, nous avons réalisé une moyenne sur trente expériences successives de l'algorithme génétique pour chaque opérateur. A chaque expérience, la même population initiale a été utilisée pour les deux opérateurs. La figure 3.17 nous montre que l'opérateur PMX est sensiblement plus performant que l'opérateur Order Based.

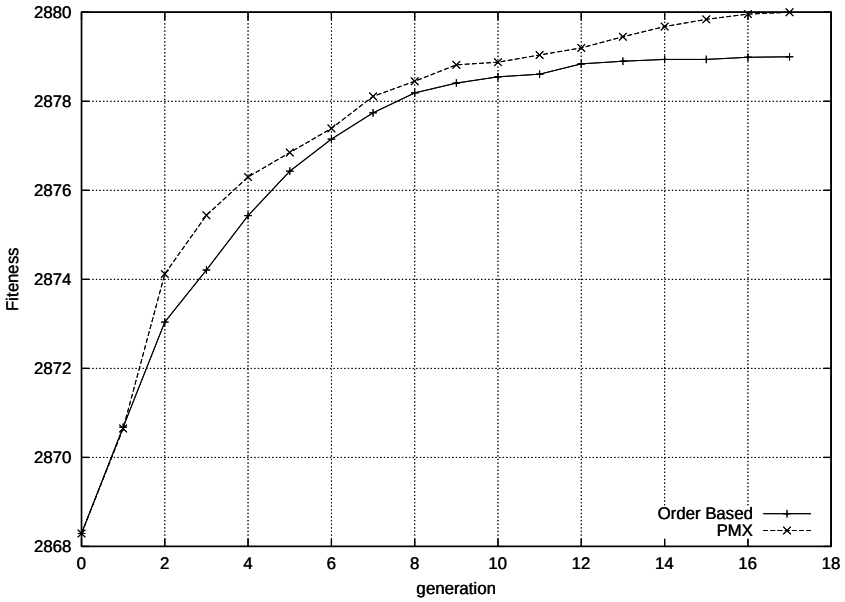

FIG. 3.17 – PMX face à Order Based

Les opérateurs génétiques retenus pour cette approche sont :
- Cross-over : PMX.
- Mutation : "swap".

La mutation "swap" consiste à échanger deux allèles choisis aléatoirement.

Cette approche consiste donc à utiliser toujours la même heuristique, appelée "Préférence", et *une liste de préférences*. Ainsi, la tâche sélectionnée en cas de conflit est déterminée en fonction de sa place dans la *liste de préférences*. Le but est donc de trouver une bonne séquence pour la *liste de préférences*.

b) Approche basée sur le pilotage des ressources

La seconde approche est très proche de la précédente car, elle aussi, utilise un codage indirect et un ordonnanceur. Contrairement à la première approche basée sur les tâches, l'heuristique employée dans cette approche n'utilise pas la *liste de préférences*. Par contre, l'heuristique n'est plus fixe et peut prendre une valeur appartenant à un ensemble $\mathcal{H}_c$ d'heuristiques pré-définies. Cette heuristique, sélectionnée par l'algorithme génétique, *pilote* la ressource lors du choix d'une tâche en cas de conflit. L'algorithme génétique sélectionnant uniquement des heuristiques, l'heuristique "Préférence" n'est pas présente dans l'ensemble $\mathcal{H}_c$ car cette heuristique nécessite une *liste de préférences* que l'algorithme génétique ne fournit pas. Cette approche a été inspirée des travaux de Ghedjati [Ghedjati 97] et de Hart et Ross [Hart et al. 1999].

L'ensemble $\mathcal{H}_c$ est défini par des heuristiques dont voici quelques exemples :
- SPT (Shortest Processing Time) : temps de réalisation le plus court.
- LPT (Longest Processing Time) : temps de réalisation le plus long.
- FIFO (First In First Out) : première arrivée, première servie.

Cette approche permet de choisir une tâche T_c dans un ensemble T_r de tâches en conflit. Comme dans la première approche, deux niveaux de traitement sont nécessaires à ce choix. Le premier sélectionne les tâches ayant le nombre d'occurrences déjà réalisées le plus faible. Le second niveau utilise l'heuristique sélectionnée par l'algorithme génétique. Cette approche peut se résumer par l'algorithme 3.2.

Algorithme 3.2 Choisir une tâche (approche par pilotage de ressource)

Entrée : T_r : Ensemble de tâches réalisables.
Entrée : R_l : Ressource sur laquelle le choix doit se faire.
Sortie : T_c : La tâche choisie dans l'ensemble T_r.

Créer T'_r ensemble des tâches réalisables ayant le nombre d'occurrences réalisées le plus faible.
Selon l'heuristique H_{R_l} de la ressource R_l
SPT : Choisir T_c comme étant la tâche de T'_r ayant le temps de réalisation le plus court.
LPT : Choisir T_c comme étant la tâche de T'_r ayant le temps de réalisation le plus long.
⋮
FinSelon

Codage d'une solution

Avec cette approche, chaque ressource dispose d'un gène dans le chromosome. Ainsi, le chromosome est constitué de x gènes où x représente le nombre de ressources. Avec l'exemple de la figure 3.16 qui comporte deux ressources, nous obtenons le chromosome suivant :

H_1	H_3

où H_1 et H_3 représentent deux heuristiques différentes de l'ensemble $\mathcal{H}_c$.

Opérateurs génétiques

Avec cette approche, les opérateurs génétiques simples peuvent être utilisés. Contrairement à une séquence où la position d'un allèle ainsi que sa présence dans le chromosome a une importance, la présence ou non d'une heuristique dans le chromosome n'est pas capitale. En effet, l'objectif est de trouver une combinaison de règles qui permet d'obtenir de bons résultats. C'est pour ces raisons que l'emploi d'un cross-over à n-points et d'une mutation de type "aléatoire" sont possibles.

c) Approche hybride

Il est possible de combiner ces deux approches afin d'obtenir une approche hybride. En effet, la première approche représente une heuristique particulière de l'ensemble $\mathcal{H}_c$ de la seconde approche. Il est donc possible d'inclure la première approche dans la seconde.

Le chromosome est constitué de deux listes. La première représente les heuristiques et la deuxième les *listes de préférences*. Selon l'heuristique sélectionnée, la *liste de préférences* peut ne pas être utile. Il est également possible d'ajouter de nouvelles heuristiques tel que l'heuristique PSPT qui sélectionne la tâche ayant le temps de réalisation le plus court et ayant la *priorité de réalisation* la plus importante.

Deux opérateurs de cross-over travaillent chacun sur une partie du chromosome. Un cross-over à $n - points$ gère la partie avec les heuristiques et un PMX les *listes de préférences*.

3.4 L'ordonnanceur

L'ordonnanceur permet d'évaluer un individu en simulant l'exécution du graphe de précédences à contraintes linéaires. Cette simulation construit un ordonnancement qui est évalué en fonction des critères d'évaluation choisis. La simulation est basée sur le principe de fonctionnement des Réseaux de Petri (RdP) [Carlier et al. 1989a]. Les conflits de ressource sont gérés par une base de règles.

Ce paragraphe s'articule autour de six points. Le premier explique la transformation du graphe de précédences à contraintes linéaires (GPCL) en Réseau de Petri T-temporisé. Cette classe de Réseau de Petri affecte les temporisations aux transitions.

Le second détaille l'initialisation de ce Réseau de Petri afin de créer le marquage initial. Le troisième généralise cette modélisation à des graphes à structure quelconque. Le quatrième détaille la gestion des ressources. Le cinquième explique la "simulation" du Réseau de Petri et le sixièmz indique comment est évaluer l'ordonnancement.

3.4.1 Modélisation d'un GPCL avec les RdP

L'idée de l'ordonnanceur est de construire un ordonnancement en "simulant" le GPCL. Pour cela,un RdP a été construit dans lequel les places représentent les tâches à réaliser et les jetons les occurrences des tâches. Les transitions doivent ainsi permettre de gérer le temps de réalisation des tâches et la satisfaisabilité des contraintes de précédences linéaires.

La figure 3.18 présente le RdP associé à un GPCL contenant une seule contrainte $e = (1, 2)$ de valeur $(p_1, \alpha, \beta, \gamma, \omega)$. Cette représentation nécessite un marquage initial qui est réalisé par un algorithme d'initialisation.

Nous allons maintenant détailler la transformation qui permet d'obtenir un RdP à partir d'un GPCL.

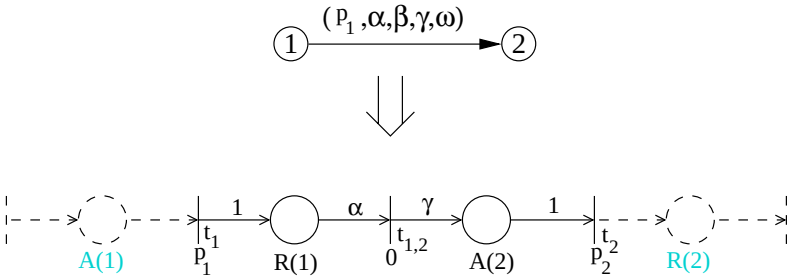

FIG. 3.18 – Equivalence entre GPCL et RdP

Les noeuds du GPCL

Un noeud i du GPCL est représenté par deux places $A(i)$ et $R(i)$ séparées par une transition t_i. La place $A(i)$ représente la liste des occurrences réalisables de la tâche i. $R(i)$ représente la "liste" des dernières occurrences de la tâche i réalisées et attendant de "franchir" la contrainte linéaire. La transition t_i représente le temps de réalisation de la tâche i. Si le noeud i ne possède aucun arc entrant, la place A(i) devient inutile car dans ce cas, toutes les occurrences de la tâche i sont réalisables sans contrainte. C'est pour cette raison que sur la figure 3.18 la place A(1) est en pointillée.

Les arcs du GPCL

Les arcs du GPCL représentent la contrainte linéaire qui bloque la réalisation de certaines occurrences de tâches. Un arc du GPCL est représenté dans le RdP par une transition $t_{i,j}$, pour une contrainte linéaire $e = (i, j)$ valuée de $(p_i, \alpha, \beta, \gamma, \omega)$. Cette transition sépare les places $R(i)$ et $A(j)$. Le temps de tirage est nul car la transition représente la satisfaisabilité de la contrainte linéaire. Pour tirer cette transition, il faut que la tâche i soit réalisée α fois pour autoriser les $\gamma^{ième}$ occurrences suivantes de la tâche j. Lors du tirage de la transition $t_{i,j}$, α jetons sont retirés de $R(i)$, et γ jetons sont placés dans la place $A(j)$.

Si aucun arc ne sort d'un noeud i, alors la place $R(i)$ devient inutile. C'est pour cette raison que la place R(2) est en pointillée dans la figure 3.18.

Les réseaux de Petri sont conçus pour des fonctionnements en régime permanent. Néanmoins, il est possible de prendre en compte le régime transitoire dû au paramètre β des contraintes de précédences linéaires. Pour cela, une fonction B(i) est créée. Cette fonction est définie de la façon suivante :

$$B(i) = \begin{cases} \beta & si\ premier\ tirage\ de\ t_{i,j} \\ 0 & sinon \end{cases}$$

Ainsi, il est possible de représenter les β jetons supplémentaires pour tirer la première fois la transition t_i. La figure 3.19 représente un exemple de RdP avec une fonction B(i). Cette fonction s'ajoute à la valeur de l'arc entre la place $R(i)$ et la transition $t_{i,x}$. Ici, les places A(1) et R(2) ne sont pas représentées car aucun arc n'arrive sur la tâche 1 et aucun arc ne sort de la tâche 2.

3.4.2 Initialisation du RdP

L'algorithme 3.3 permet de réaliser le marquage initial. Cet algorithme tient compte des paramètres β, γ et ω qui induisent des décalages initiaux entre les numéros d'occurrences des tâches.

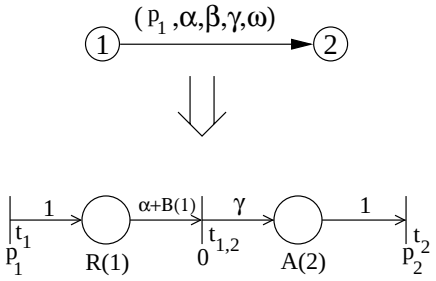

FIG. 3.19 – Exemple de RdP prenant en compte le régime transitoire

Algorithme 3.3 Algorithme d'initialisation du RdP

Pour tous les noeuds $n_i \in N$
  Si il existe une contrainte $e_{j,i}$ de valeur $(p_j, \alpha, \beta, \gamma, \omega)$
    Ajouter dans la place $A(i)$, $(\gamma + \omega - 1)$ jetons
  FinSi
FinPour

Il est important de noter que d'après la remarque 3.1, il faut que $\gamma + \omega > 0$.

3.4.3 Extension aux graphes à structure quelconque

La représentation proposée permet d'utiliser des graphes simples. Nous allons proposer une extension de cette représentation aux graphes à structure quelconque. L'extension doit tenir compte des graphes à structure divergente et des graphes à structure convergente. L'extension de l'algorithme d'initialisation des jetons est détaillée dans un dernier temps. Les figures présentées ci-après ne tiennent pas compte du régime transitoire. Seul le régime permanent est modélisé.

Graphes à structure convergente

La figure 3.20 représente la convergence de deux contraintes issues de deux tâches différentes sur une troisième tâche. Nous pouvons remarquer que la représentation des tâches 1 et 2 n'a pas changé par rapport à la modélisation du paragraphe 3.4.1. Seule la tâche 3 est différente. En effet, lorsque n arcs convergent sur une tâche i, cette dernière est représentée par n places $A_x(i)$ et une place $R(i)$ si nécessaire. La place $A_x(i)$ représente les occurrences de la tâche i qui sont autorisées par la tâche x. C'est ainsi que la tâche i est réalisable pour une occurrence s'il y a au moins un jeton dans chaque place $A_x(i)$.

Graphes à structure divergente

La figure 3.21 représente un cas où une tâche contraint la réalisation des occurrences de deux autres tâches. Sur cette figure, lorsque des occurrences de la tâche 1 sont réalisées, ces dernières peuvent autoriser la réalisation des occurrences des tâches 2 et 3. Il est donc nécessaire de maintenir deux "listes" $R_2(1)$ et $R_3(1)$ des dernières occurrences réalisées de la tâche 1.

Ainsi, lorsque n arcs partent d'une tâche i, cette dernière est représentée par une place $A(i)$, si nécessaire, et n places $R_x(i)$. Ici, la place $R_j(i)$ représente la liste des dernières occurrences réalisées de la tâche i qui permettront de déterminer la liste des occurrences autorisées de la tâche j.

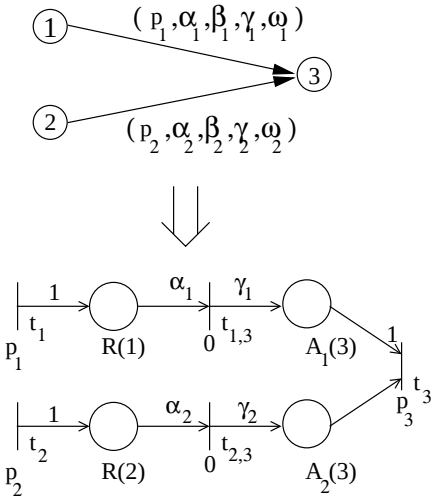

FIG. 3.20 – Équivalence entre un GPCL à structure convergente et un RdP

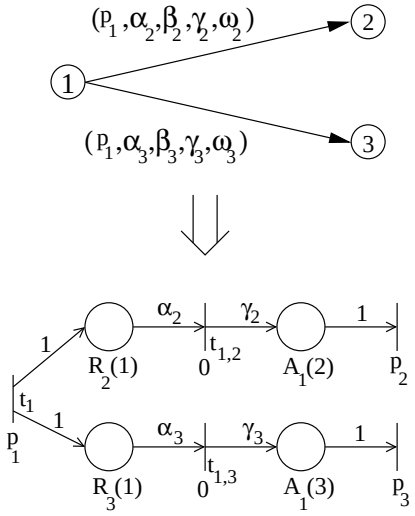

FIG. 3.21 – Équivalence entre un GPCL à structure divergente et un RdP

Représentation générique d'une tâche

Une tâche générique i du GPCL est susceptible d'avoir n arcs entrants et m arcs sortants. C'est ainsi que cette tâche i est représentée par une transition t_i et n places $A_x(i)$ qui convergent sur la transition t_i et m places $R_y(i)$ qui divergent de la transition t_i. Cette représentation est inspirée de celle présentée par Carlier [Carlier et al. 1989a] qui permet de représenter des graphes de précédences à contraintes uniformes.

La transition $t_{x,y}$ qui sépare une place $R_y(x)$ et une place $A_x(y)$ représente la contrainte linéaire $e = (x, y)$ valuée de $(p_x, \alpha_x, \beta_x, \gamma_x, \omega_x)$. Cette transition a une durée de tirage nulle. Lors de son tirage, $\alpha_x + B_y(x)$ jetons sont retirés de la place $R_y(x)$, et γ_x jetons sont déposés dans la place $A_x(y)$.

La figure 3.22 représente les trois éléments de base d'un GPCL transformé. Une tâche générique se transforme en deux parties. La première représente les n arcs entrants et la deuxième les m arcs sortants. Elles sont respectivement représentées par la partie convergente et la partie divergente. Une troisième partie représentant la contrainte de précédence permet de relier une place de la partie divergente d'une tâche à une place de la partie convergente d'une autre tâche.

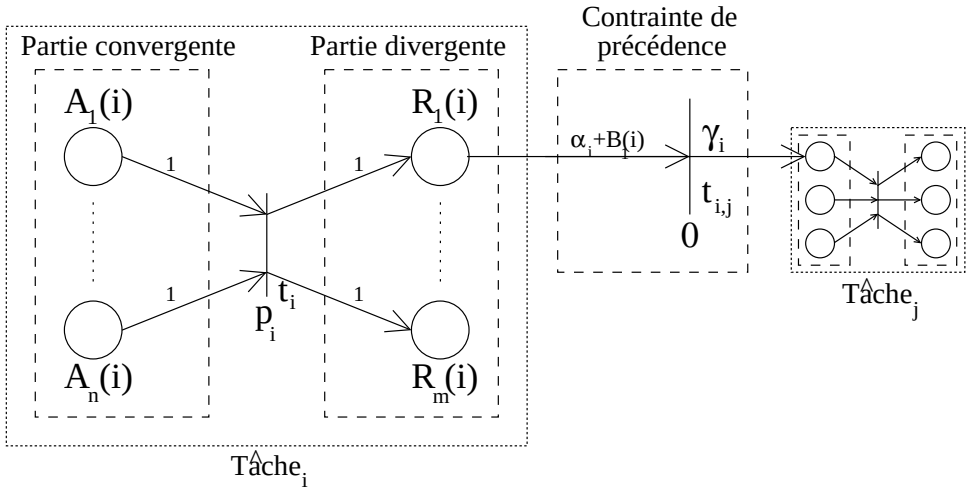

FIG. 3.22 – Les trois éléments de la représentation

Initialisation générale

La présence de structure divergente ou convergente dans un graphe, modifie l'initialisation proposée pour des graphes simples. En effet, l'initialisation doit tenir compte de tous les arcs entrants et sortants d'un noeud. L'initialisation d'un graphe à structure quelconque est définie par l'algorithme 3.4.

Algorithme 3.4 Initialisation d'un graphe quelconque

Pour tous les noeuds $n_i \in N$
  Pour toutes les contraintes linéaires $e_{j,i}$ de valeur $(p_j, \alpha, \beta, \gamma, \omega)$
    Ajouter dans la place $A_j(i)$, $(\gamma + \omega - 1)$ jetons
  FinPour
FinPour

Pour tenir compte du régime transitoire, il faut ajouter à chaque arc du RdP se situant entre une place $R_y(i)$ et une transition $t_{i,y}$, une fonction $B_y(i)$ qui permet de tenir compte du paramètre $\beta_{i,y}$ qui se situe sur la contrainte de précédences linéaire $e = (i, y)$ valuée de $(P_i, \alpha_{i,y}, \beta_{i,y}, \gamma_{i,y}, \omega_{i,y})$.

3.4.4 Gestion des ressources

Dans l'exemple de la figure 3.23, nous disposons d'une ressource à exemplaire unique, capable de réaliser les tâches 1 et 2. Cette ressource est représentée par un label "$Ress_1$" dans le RdP. Ce label fait référence à une place dans laquelle se situe au moins un jeton. Le nombre de jetons représente la capacité de la ressource.

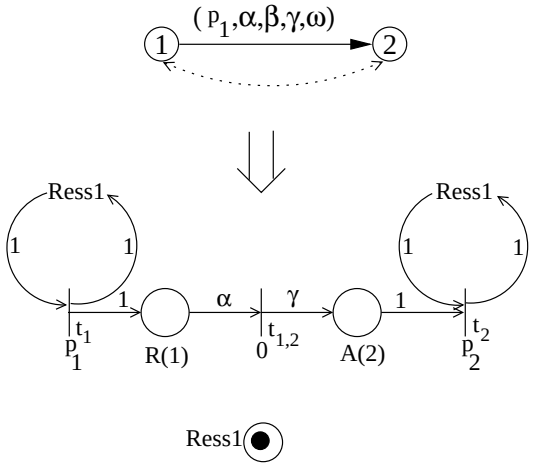

FIG. 3.23 – Equivalence entre GPCL et RdP avec des ressources

Si plusieurs tâches doivent être réalisées par la même ressource au même moment, nous devons éliminer le conflit en utilisant une heuristique. Cette heuristique est définie en fonction de l'approche utilisée. Ces approches ont été décrites dans le paragraphe 3.3.2.2.

3.4.5 Simulateur du GPCL transformé

Le GPCL transformé en Rdp est "simulé" pour construire l'ordonnancement. Cette simulation porte sur trois points :

- la création d'une liste de tâches réalisables est produite par l'algorithme 3.5.
- le choix d'une tâche se fait par l'utilisation d'une des deux approches décrites par les algorithmes 3.1 et 3.2 .
- l'ordonnancement de la tâche choisie se réalise en ordonnançant cette tâche au plus tôt.

La création de la liste des tâches réalisables $L_R(Ress)$ sur une ressource $Ress$ est réalisée avec l'algorithme 3.5. Cet algorithme vérifie que chaque tâche ayant besoin de la ressource est réalisable. Pour cela, il vérifie que la transition t_i est tirable. Une transition est tirable si le nombre de jetons requis pour la tirer est présent dans toutes les places en amont. Si aucune place se situe en amont d'une transition, alors cette dernière est toujours transitoire. Une telle transition est appelée "transition source".

Algorithme 3.5 Liste des tâches réalisables

Entrée : $Ress$: Ressource
Sortie : $L_R(Ress)$: Liste des tâches réalisables sur la ressource $Ress$

Pour toutes les tâches i ayant besoin de $Ress$
  Si la transition t_i est tirable
    Placer la tâche i dans la liste $L_R(Ress)$
  FinSi
FinPour

Exemple de simulation

Nous allons "simuler" le graphe de la figure 3.24 afin de visualiser le fonctionnement de l'ordonnanceur.

Pour cela, nous avons transformé ce graphe de précédences à contraintes linéaires en réseau de Petri. Cette transformation est représentée par la figure 3.25. Il est important de noter que chaque tâche dispose de sa ressource et que les fonctions $B_x(y)$ sont inutiles car toutes les valeurs β des contraintes linéaires sont nulles.

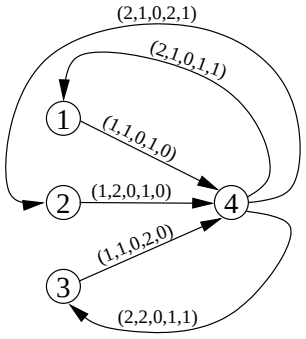

FIG. 3.24 – Exemple de graphe

Le tableau 3.1 décrit l'évolution dans le temps des marquages des places de la figure 3.25. C'est ainsi que nous pouvons voir qu'après l'initialisation du graphe, des jetons sont placés dans les places $A(1)$, $A(2)$, $A(3)$ et $A_3(4)$.

Les premières occurrences de tâche à être réalisées sont $\ll 1, 1 \gg$, $\ll 2, 1 \gg$ et $\ll 3, 1 \gg$. Nous trouvons donc à la date 1 des jetons, dans les places $R(1)$, $R(2)$ et $R(3)$. Ces jetons proviennent des places $A(i)$ avec $i \in \{1, 2, 3\}$. Dans le tableau 3.1, la date $1'$ correspond à l'instant où les contraintes linéaires ont été satisfaites. C'est ainsi que le jeton dans la place $R(1)$ est retiré et qu'un jeton est déposé dans la place $A_1(4)$ lors du tirage de la transition $t_{1,4}$. De même, lors du tirage de la transition $t_{3,4}$, le jeton dans la place $R(3)$ est retiré, et deux jetons sont déposés dans la place $A_3(4)$.

La figure 3.26 représente l'ordonnancement qui est créé au fur et à mesure du temps. Dans le tableau 3.1, une date marquée d correspond au moment où les occurrences de tâches en cours de réalisation sont terminées. Par contre, une date marquée d' représente l'instant suivant où les contraintes de précédences linéaires sont satisfaites.

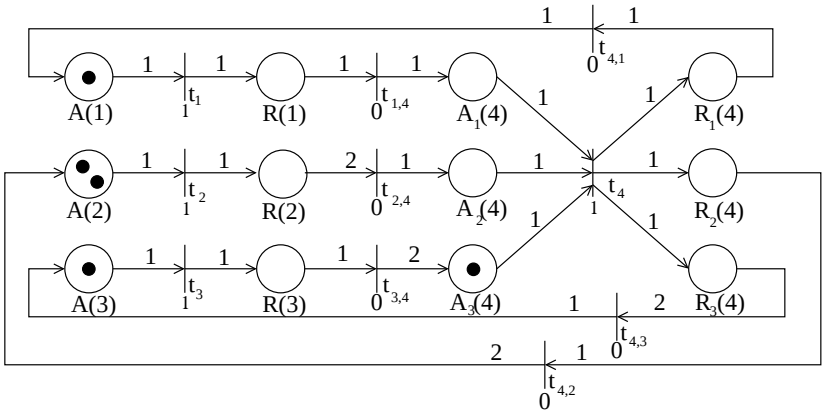

FIG. 3.25 – Transformation en RdP du graphe de la figure 3.24

	Tâche 1		Tâche 2		Tâche 3		Tâche 4					
date	$A(1)$	$R(1)$	$A(2)$	$R(2)$	$A(3)$	$R(3)$	$A_1(4)$	$A_2(4)$	$A_3(4)$	$R_1(4)$	$R_2(4)$	$R_3(4)$
Init	1	0	2	0	1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	0	0	1	0	0	0
1'	0	0	1	1	0	0	1	0	3	0	0	0
2	0	0	0	2	0	0	1	0	3	0	0	0
2'	0	0	0	0	0	0	1	1	3	0	0	0
4	0	0	0	0	0	0	0	0	2	1	1	1
4'	1	0	2	0	0	0	0	0	2	0	0	1
5	0	1	1	1	0	0	0	0	2	0	0	1
5'	0	0	1	1	0	0	1	0	2	0	0	1
6	0	0	0	2	0	0	1	0	2	0	0	1
6'	0	0	0	0	0	0	1	1	2	0	0	1
8	0	0	0	0	0	0	0	0	1	1	1	2
8'	1	0	2	0	1	0	0	0	1	0	0	0

TAB. 3.1 – Évolution dans le temps du marquage des places

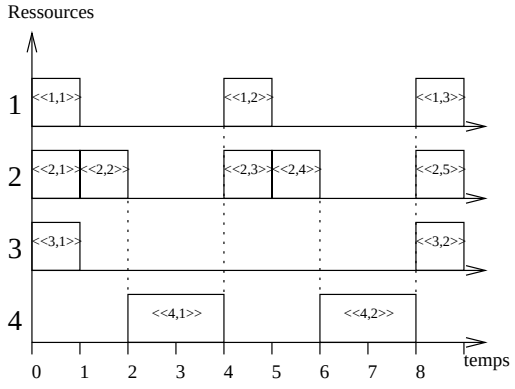

FIG. 3.26 – Ordonnancement associé à la figure 3.24

3.4.6 Évaluation de l'ordonnancement

Une fois l'ordonnancement réalisé, l'ordonnanceur propose de l'évaluer en fonction d'un critère choisi. L'évaluation choisie porte sur le temps de cycle moyen des tâches. Ce critère se note Tc_{moy} d'après la notation étendue dans le paragraphe 3.1.1.6. Pour déterminer ce critère, nous simulons le graphe de précédences à contraintes linéaires sur un horizon de temps important, puis nous le calculons. De cette façon, nous réduisons les effets de la partie transitoire.

3.5 Validation

Afin de valider la méthode GA-ORDO une série d'essais a été préparée. Nous allons présenter la méthode employée pour valider le fonctionnement de GA-ORDO, ainsi que les essais mis en place.

Lors de la validation de la capacité de GA-ORDO à résoudre un problème de Job-Shop cyclique, seule l'approche basée sur les tâches a été utilisée.

3.5.1 Méthode de validation

Le fonctionnement de GA-ORDO passe par deux phases. La première consiste à vérifier le code de l'ordonnanceur. Le but de cette première phase est de vérifier que l'ordonnanceur ne viole aucune contrainte et que les résultats sont corrects. La seconde phase porte sur la méthode. Pour cela, nous allons tester le couplage de l'ordonnanceur avec des algorithmes génétiques. L'objectif est de vérifier la convergence ainsi que la qualité de la solution trouvée.

3.5.1.1 Première phase : validation de l'ordonnanceur

L'idée de cette première phase est de vérifier que l'ordonnanceur fourni des résultats corrects. Pour cela, nous avons vérifié que toutes les solutions données par l'ordonnanceur étaient faisables. Nous avons ensuite demandé à l'ordonnanceur de créer l'ordonnancement d'un BCS.

Un problème de BCS est un problème d'ordonnancement sans ressource. Ainsi, lors de la création de l'ordonnancement, aucun conflit de ressource n'apparaît. Donc, si l'ordonnanceur crée l'ordonnancement d'un BCS, il doit trouver la valeur du chemin critique. La valeur de ce chemin critique est donnée par le temps de cycle le plus long (Tc_{max}). Nous avons vérifié sur plusieurs essais que l'ordonnanceur permet de trouver ce chemin critique.

La phase suivante concerne l'approche globale de résolution du problème de Job-Shop cyclique.

3.5.1.2 Deuxième phase : validation par l'heuristique "Préférence"

L'objectif principal de cette phase est de valider l'approche de résolution. La convergence ainsi que la qualité des résultats sont les deux critères de validation. Pour vérifier la qualité des résultats trouvés, nous avons besoin de les comparer avec des résultats connus. Malheureusement, aucun jeu d'essais pour des problèmes cycliques n'existe. Afin de remédier à ceci, nous avons transformé des jeux d'essais non cycliques en jeux d'essais cycliques.

Dans cette série d'essais, nous avons utilisé les jeux d'essais de Lawrence [Lawrence 1984]. Ces jeux d'essais sont des problèmes de Job-Shop non cyclique. La transformation consiste à créer une tâche supplémentaire de durée nulle. Cette tâche permet de synchroniser le début des jobs. Elle empêche ainsi un job de commencer une occurrence $i+1$ tant que les occurrences i de tous les autres jobs ne sont pas terminées. Ainsi, nous pouvons dire que le temps de cycle de

cette tâche de synchronisation est identique au temps de réalisation complet du Job-Shop non cyclique, appelé aussi makespan.

Prenons l'exemple de la figure 3.27. Sur cet exemple, deux jobs, Job_1 et Job_2, sont à réaliser. Ils sont composés des tâches suivantes : $Job_1 = \{1, 2, 3\}$ et $Job_2 = \{4, 5, 6\}$. Trois ressources A, B et C sont disponibles. L'affectation des tâches sur les ressources est la suivante : $A = \{1, 5\}$, $B = \{2, 6\}$ et $C = \{3, 4\}$. Les valeurs sur les arcs représentent les temps de réalisation des tâches. Dans cet exemple, les contraintes de précédences sont des contraintes uniformes dont les hauteurs sont nulles.

Un ordonnancement possible de ce Job-Shop est représenté par le diagramme de Gantt de la figure 3.29. Ce problème étant un problème d'ordonnancement non cyclique, lorsque toutes les tâches ont été réalisées, l'ordonnancement s'arrête. Le makespan est donné par la date de fin de réalisation la plus grande. La figure 3.28 présente le résultat de la transformation du problème non cyclique de la figure 3.27 en problème cyclique. L'objectif dans les problèmes d'ordonnancement cyclique est de maximiser le débit ou de minimiser le temps de cycle d'une tâche. Ici, c'est la minimisation du temps de cycle de la tâche de synchronisation ($Min\{Tc_i\}$) qui nous intéresse. La figure 3.30 présente le diagramme de Gantt associé au graphe de précédences de la figure 3.28.

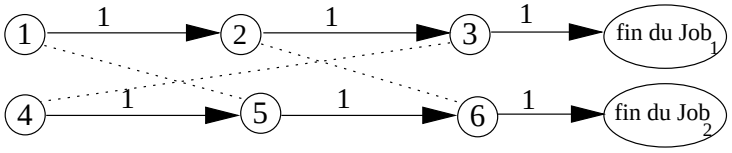

FIG. 3.27 – Exemple de problème de Job-Shop non cyclique

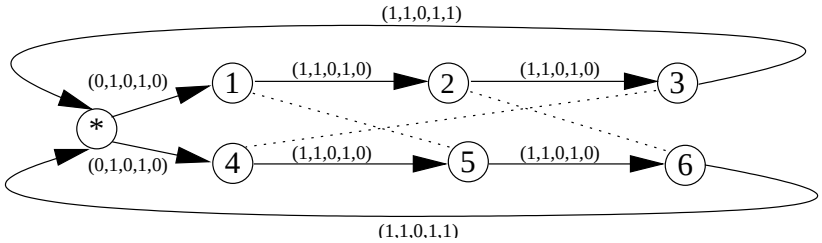

FIG. 3.28 – Job-Shop non-cyclique transformé

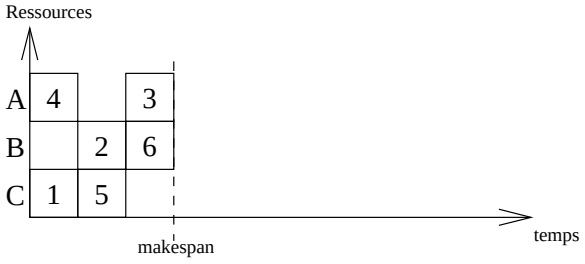

FIG. 3.29 – Ordonnancement associé à la figure 3.27

Après transformation des jeux d'essais de Lawrence, nous avons obtenu les résultats représentés dans le tableau 3.2. La méthode GA-ORDO a permis d'obtenir dans 48% des cas l'optimum et en cas d'échec, l'écart mesuré est d'environ 6% au dessus de l'optimum.

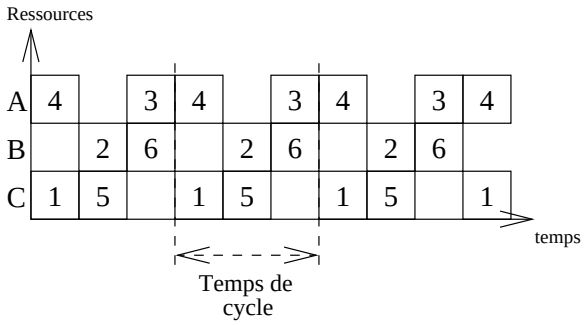

FIG. 3.30 – Ordonnancement associé à la figure 3.28

Benchmark	Optimum	GA-ORDO
10 jobs 5 machines		
la1	666	666
la2	655	672
la3	597	623
la4	590	611
la5	593	593
15 jobs 5 machines		
la6	926	926
la7	890	980
la8	863	863
la9	951	951
la10	958	958
20 jobs 5 machines		
la11	1222	1222
la12	1039	1039
la13	1150	1150
la14	1292	1292
la15	1207	1207
10 jobs 10 machines		
la16	945	1004
la17	784	793
la18	848	869
la19	842	875
la20	902	939
15 jobs 10 machines		
la21	1046	1121
la22	927	989
la23	1032	1040
la24	935	982
la25	977	1067

TAB. 3.2 – Résultats obtenus par la méthode GA-ORDO

3.5.1.3 Jeux d'essais cycliques

Dans le but de pouvoir comparer GA-ORDO avec d'autres méthodes de résolutions, nous proposons un ensemble de jeux d'essais représentant des problèmes de Job-Shop cyclique.

Les jeux d'essais sont composés de n jobs et de m machines. Les jobs sont tous constitués de m tâches à réaliser dans un ordre prédéfini. Une machine réalise une seule tâche de chaque job. Il y a donc n tâches qui lui sont affectées. Les jobs sont non ré-entrants, c'est à dire, qu'il ne peut y avoir qu'une seule occurrence active à la fois du même job. Les jeux d'essais sont créés par l'algorithme 3.6. Le type de graphe de précédences à contraintes linéaires créé par cet algorithme est représenté par la figure 3.31. Dans ces essais, chaque job forme un sous graphe. Les tâches composants le job se réalisent les unes après les autres formant ainsi un cycle. Nous obtenons donc autant de cycles indépendants qu'il y a de jobs.

Algorithme 3.6 Création de jeux d'essais cycliques

Pour les n jobs
  Créer une liste de m tâches
  Créer $(m-1)$ contraintes linéaires entre les m tâches
  Créer la contrainte linéaire qui referme le graphe représentant le job
FinPour
Créer l'affectation des tâches sur les ressources

Cet algorithme sert à créer une liste de m tâches successives. Ensuite, entre chacune de ces tâches, nous créons une contrainte linéaire dont le temps de réalisation, les paramètres α et γ sont déterminés de façon aléatoire. Les paramètres β et ω sont égaux à 0.

L'étape suivante crée une contrainte linéaire entre la dernière tâche du job et la première tâche. Le temps de réalisation de la dernière tâche est déterminée selon le même procédé que précédemment. Le paramètre ω est égal à 1 et le paramètre β égal à 0. Par contre, pour pouvoir dérouler l'exécution du graphe de façon infinie, les paramètres α et γ doivent respecter le théorème suivant définit par Hanen et Munier dans [Hanen et al. 1995]

$$\prod_{i \in \mathcal{N}} \frac{\alpha_i}{\gamma_i} = 1$$

avec $\mathcal{N}$ l'ensemble des tâches composant un cycle, α_i et γ_i les valeurs de la contrainte linéaire $e = (i, x)$.

Ces étapes sont répétées autant de fois qu'il y a de jobs à créer.

L'affectation des tâches sur les ressources se fait de façon aléatoire. Chaque machine est capable de réaliser une seule tâche de chaque job. Nous créons ainsi une liste qui définit pour chaque machine l'ensemble des tâches qu'elle réalise. La figure 3.31 présente une instance avec 3 jobs et 3 machines.

Résultats des essais

Neuf essais ont été réalisés. Ces essais concernent trois instances de trois classes différentes. Chaque classe représente un nombre de jobs et de machines différent.

Ce essais ont tous été optimisés par la méthode GA-ORDO. L'approche employée est celle basée sur les tâches. Le chromosome est constitué des *listes de préférences* de chaque ressource. Les paramètres de l'algorithme génétique sont les suivants :

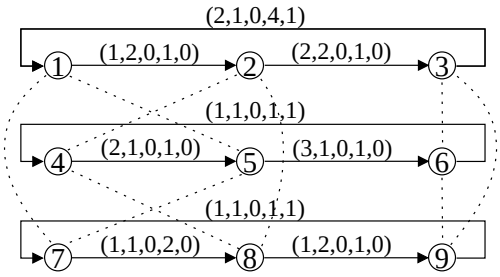

FIG. 3.31 – Jeux d'essais cycliques : instance comportant 3 jobs et 3 machines

- Initialisation : aléatoire.
- Sélection : roulette.
- Taille de la population : 80.
- Cross-over : PMX (80%).
- Mutation : swap (20%).
- Élitisme : oui.
- Objectif : minimiser.

Le critère d'optimisation est le temps de cycle moyen de réalisation d'un job (Tc_{moy}). L'objectif est de minimiser ce temps de cycle afin de permettre de maximiser le nombre de jobs réalisés durant une période de temps déterminé. Dans le but de déterminer la performance de notre méthode, nous avons indiqué le nombre de solutions explorées pendant l'optimisation.

Jeux d'essais	Temps de cycle moyen d'un job	Nombre de solutions explorées	$B_{\overline{Ress}}$
10 jobs 15 machines			
10_15a	4.832	1624	1.051
10_15b	6.169	1402	1.110
10_15c	7.468	2329	1.110
5 jobs 10 machines			
5_10a	6.944	1058	1.090
5_10b	7.590	1347	1.318
5_10c	5.901	1452	0.884
5 jobs 5 machines			
5_5a	10.599	2030	2.877
5_5b	18.975	2420	9.913
5_5c	5.961	1250	1.936

TAB. 3.3 – Résultats obtenus par GA-ORDO sur les jeux d'essais cycliques

Le tableau 3.3 donne les résultats, exprimés en unités de temps, obtenus après optimisation de ces neuf essais. Les essais sont téléchargeables à l'adresse suivante : "http ://www.skyscaper.com/edit/899/benchmark ". La notation employée pour nommer un essai est la suivante : X_Yz, où 'X' représente le nombre de jobs, 'Y' le nombre de machines et 'z' le nom de l'instance.

Recherche d'une borne

Nous allons utiliser une borne dans le but de déterminer la performance de la solution proposée par l'algorithme généntique. Cette borne est déterminée en supprimant les contraintes de ressources et représente donc une valeur minimale inférieure à l'optimum. Une fois les contraintes de ressources supprimées, nous calculons le temps de cycle moyen des jobs (Tc_{moy}). Cette borne représente un moyen naïf d'évaluer la performance des solutions proposées. La colonne $B_{\overline{Ress}}$ du tableau 3.3 correspond aux valeurs des bornes des différents jeux d'essais.

L'écart entre la valeur trouvée par la méthode GA-ORDO et la borne proposée se situe dans un intervalle compris entre 3.781 et 9.062, avec pour valeur moyenne 5.875.

3.6 Conclusion

La transformation d'un graphe de précédences à contraintes linéaires en Réseau de Petri permet de créer un ordonnancement à partir du GPCL. Couplé à un algorithme génétique, cette méthode de création d'ordonnancement aide à optimiser le critère d'évaluation de cet ordonnancement.

La transformation d'un jeu d'essais de Job-Shop en jeu d'essais cyclique démontre la performance de cette méthode. Néanmoins, ce type de jeu d'essais utilisent des contraintes uniformes. Ce type de contrainte génère des essais avec un espace de recherche réduit par rapport aux jeux d'essais cycliques proposés qui eux utilisent des contraintes linéaires.

Due à l'utilisation des algorithmes génétiques, cette méthode n'a pas de garantie de performances. En effet, la convergence de l'algorithme n'est pas prouvée. De plus, cet algorithme étant stochastique, il est nécessaire de réaliser plusieurs expériences.

La méthode GA-ORDO ne permet pas seulement de traiter des problèmes académiques, elle a également permis de traiter un problème industriel. La suite de ce document présente la résolution d'un problème de type industriel.

Chapitre 4

Application : amélioration d'un processus industriel

Dans ce chapitre, nous présentons un problème industriel de production. Ce problème concerne une ligne de fabrication de moteurs automobile. Un premier point décrit la ligne de fabrication. Ce premier point aborde l'architecture de la ligne de fabrication, les caractéristiques des éléments qui composent cette ligne ainsi qu'une analyse par les plans d'expériences de l'impact de ces éléments sur la productivité. Deux approches d'amélioration de cette ligne de fabrication sont ensuite présentées. La première consiste à utiliser un simulateur à événements discrets. Pour cela, nous utilisons une modélisation du problème selon une approche objet. L'algorithme génétique employé pour l'optimisation est expliqué ensuite, suivi de l'expérimentation qui a permis de déterminer les paramètres de cet algorithme, adaptés à ce problème. Une seconde approche de modélisation par les graphes de précédences à contraintes linéaires est proposée afin d'optimiser le problème industriel.

4.1 Présentation du problème industriel

Nous allons nous intéresser dans ce chapitre à une ligne de fabrication de moteurs automobile. Elle est de type "grande série" dédiée. Après une description de la problématique générale, une analyse de l'existant est présentée, suivie d'une proposition de méthode d'amélioration [Dupas et al. 1998b].

4.1.1 Structure et exploitation de la ligne de fabrication

La ligne de fabrication est composée d'une succession de machines et de stocks tampons. La figure 4.1 présente un schéma constitué de trois machines et de deux stocks tampons. Les machines sont des machines de type multi-postes capables de traiter en même temps plusieurs moteurs à des stades de production différents. Les stocks tampons sont des convoyeurs à accumulation pouvant atteindre des dimensions importantes. Leur rôle est de convoyer et de stocker les moteurs d'une machine à une autre.

Sur chaque machine, un ensemble de tâches est à réaliser de façon périodique. On y trouve des tâches de maintenance systématique et des tâches de contrôle. Les changements d'outils (CO) représentent les tâches de maintenance systématique et les contrôles sont appelés des contrôles inter-opérations (CIO). Pour réaliser un CIO, un moteur est extrait, de façon périodique, du flux

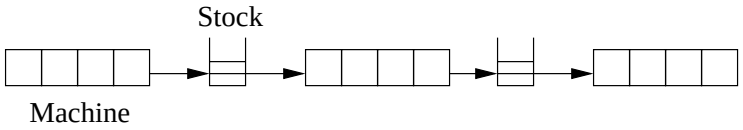

FIG. 4.1 – Architecture de la ligne de fabrication

de production. Une fois les contrôles de qualité réalisés par l'opérateur, le moteur est remis dans le flux. Le CIO est une tâche préemptive et non prioritaire par rapport aux autres tâches.

Un ensemble d'opérateurs est présent sur la ligne pour réaliser les changements d'outils et les contrôles inter-opérations. Les opérateurs sont affectés à tous les changements d'outils d'une ou de plusieurs machines ainsi qu'à tous les contrôles inter-opérations de ces machines. La figure 4.2 présente un exemple d'affectation de deux opérateurs sur trois machines.

Lors de la réalisation d'un changement d'outils, la machine sur laquelle est réalisé ce changement ne peut plus produire. La production reprend au moment où le changement d'outils est terminé. Par contre, un contrôle inter-opérations n'arrête pas la production. Le contrôle inter-opérations est réalisé en temps masqué.

La ligne est dite équilibrée car toutes les machines ont le même temps de cycle. Pour des raisons internes à l'entreprise, la ligne de fabrication est complètement arrêtée le dimanche.

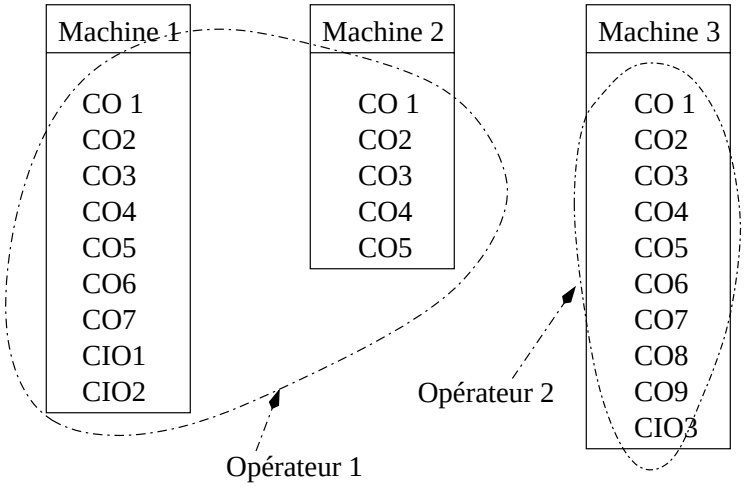

FIG. 4.2 – Exemple d'affectation opérateur

4.1.2 Analyse de l'existant

La ligne de fabrication est composée de cinq éléments qui sont : les opérateurs, les machines, les stocks tampons, les changements d'outils et les contrôles inter-opérations. Les caractéristiques de ces éléments sont décrites dans un premier temps. La détermination par les plans d'expériences des éléments responsables de la non productivité de la ligne de fabrication est réalisée dans un second temps.

4.1.2.1 Les caractéristiques des éléments du problème industriel

Chaque élément est constitué d'un ensemble de caractéristiques. Une caractéristique est considérée comme variable si cette caractéristique peut être modifiée sans remettre en compte les choix

stratégiques et économiques de l'entreprise. Par contre, une caractéristique est considérée comme fixe si le fait de changer sa valeur remet en cause les choix de l'entreprise. Les caractéristiques de chaque élément sont décrites de la façon suivante :

- Les machines :
  - Temps de cycle : temps nécessaire à la réalisation d'une opération de production sur une machine.
  - Nombre de postes : représente le nombre de moteurs pouvant être simultanément en cours d'usinage dans la même machine.
  - Taux de panne : quantifie la disponibilité de la machine.
  - Liste de changements d'outils : liste des changements d'outils à réaliser sur la machine.
- Les changements d'outils :
  - Déclencheur : nombre d'occurrences de la tâche de production séparant deux occurrences successives de la tâche de maintenance.
  - Durée : temps nécessaire à la réalisation de cette tâche de maintenance.
  - Décalage : nombre d'occurrences supplémentaires nécessaires de tâches de production pour déclencher la première occurrence de la tâche de maintenance. Si cette caractéristique est négative, alors la tâche de maintenance est anticipée.
  - Ressource : ressource nécessaire à la réalisation du changement d'outils. Cette valeur est fixée au début du programme de fabrication.

  La figure 4.3 présente un diagramme de Gantt d'une machine sur laquelle une tâche de maintenance doit être réalisée avec les caractéristiques suivantes : $D\acute{e}clencheur = 3$, $Durée = 3$, $D\acute{e}calage = -2$. La valeur indiquée dans la tâche représente le numéro d'occurrence.

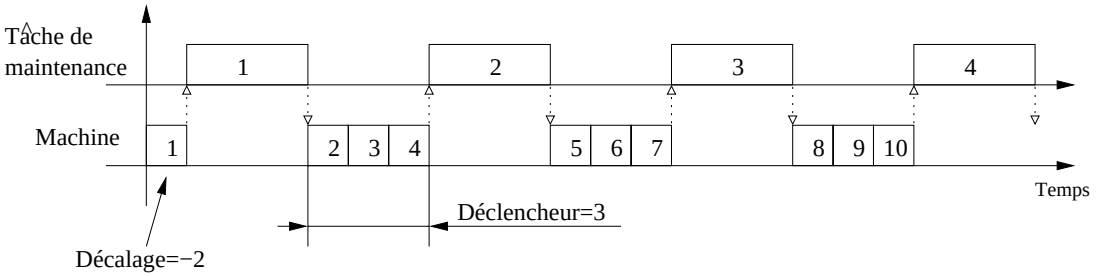

FIG. 4.3 – Occurrences d'une tâche de maintenance affectée sur une machine (diagramme de Gantt)

- Les contrôles inter-opérations :
  - Déclencheur : nombre d'occurrences de la tâche de production séparant deux occurrences successives de la tâche de contrôle.
  - Durée : temps nécessaire à la réalisation de cette tâche de contrôle.
  - Ressource : ressource nécessaire à la réalisation du contrôle.
- Les stocks tampons :
  - Taille du stock : indique le nombre maximal de moteurs pouvant être stockés.
  - Temps de convoyage : représente le temps qu'il faut à un moteur pour traverser le stock afin d'être disponible pour la machine suivante.
- Les ressources :
  - Nombre : nombre de ressources disponibles.

Une liste des caractéristiques variables et fixes a été réalisée afin de déterminer les possibilités d'amélioration. Cette liste est représentée par le tableau 4.1. Les caractéristiques variables sont

représentées par la lettre "V" et les caractéristiques fixes par la lettre "F".

Machine		CO		CIO		Stock		Ressource	
Temps de cycle	F	Durée	F	Durée	F	Taille	F	Nombre	F
Nombre de postes	F	Déclencheur	F	Déclencheur	F	Temps de cconvoyage	F		
Taux de panne	V	Ressource	F	Ressource	F				
Liste de CO	F	Décalage	V						

TAB. 4.1 – Résumé des caractéristiques

4.1.2.2 Analyse par les plans d'expériences

L'idée de cette analyse est de déterminer quels sont les éléments qui influencent le plus la production, et plus particulièrement quels éléments génèrent le plus de non productivité [Dupas et al. 1998a]. Pour cela, nous avons réalisé un plan d'expériences dont les paramètres sont les éléments [Fourmaux et al. 1998, Dupas et al. 1998c]. La détermination de l'influence d'un élément se réalise en lui affectant plusieurs niveaux possibles. Par exemple, pour un changement d'outils, les deux niveaux sont : présent ou absent. Le niveau absent signifie que lors de l'évaluation, les changements d'outils sont retirés du modèle de simulation. De cette manière, il est possible de voir leur impact sur la production. Par contre, pour une machine, les niveaux possibles sont : temps de cycle normal et temps de cycle multiplié par deux.

L'évaluation d'un essai ne pouvant en aucun cas se réaliser sur la ligne de fabrication réelle, un simulateur de lignes de fabrication a été utilisé. Le simulateur utilisé est le logiciel commercial WITNESS.

Un premier plan d'expériences comprenant tous les éléments comme paramètre a permis de nous focaliser sur trois éléments importants. Les changements d'outils, les contrôles inter-opérations et les ressources sont les causes les plus importantes de non productivité. Ces trois éléments identifiés, il est nécessaire de déterminer le plus "important".

Un deuxième plan d'expériences avec uniquement ces trois éléments comme paramètres a été réalisé. Chaque paramètre prend deux niveaux possibles : présent ou absent. En réalisant toutes les combinaisons possibles des niveaux, nous obtenons huit essais à réaliser. Une ligne du tableau 4.2 [Dupas et al. 1998b] représente un essai réalisé. Pour cet essai, chaque paramètre prend un niveau déterminé. La colonne évaluation de ce tableau représente un nombre de moteurs fabriqués en une semaine de production.

La seconde partie du tableau 4.2 représente les effets des paramètres. Ainsi, les opérateurs sont "responsables" de 11.81% de non productivité. Ce tableau indique également les effets engendrés par l'interaction des paramètres. Parmi ces paramètres, les changements d'outils et les opérateurs sont les deux principales causes de non-productivité. Ne pouvant ajouter d'opérateur supplémentaire pour des raisons économiques, nous nous sommes focalisés sur les changements d'outils.

4.1.3 Proposition d'une méthode d'amélioration

Le plan d'expériences a permis de prouver que le paramètre changement d'outils est la principale cause de non productivité sur la ligne de fabrication. Pour des raisons de qualité et de coût de production, seule la caractéristique "décalage" d'un changement d'outils est variable. Cette

Plan d'expériences			
Opérateur	CIO	CO	Évaluation
Présent	Présent	Présent	10754
Présent	Présent	Absent	12771
Présent	Absent	Présent	12098
Présent	Absent	Absent	13830
Absent	Présent	Présent	12837
Absent	Présent	Absent	10422
Absent	Absent	Présent	12837
Absent	Absent	Absent	10422

Effets des paramètres						
Opérateur	CIO	Opérateur+CIO	CO	CO+Opérateur	CIO+CO	CO+CIO+Opérateur
11,81%	2,8%	2,31%	**13,67%**	4,52%	0,62%	0,32%

TAB. 4.2 – Résultats du deuxième plan d'expériences

caractéristique permet de décaler les changements d'outils les uns par rapport aux autres. En effet, les changements d'outils génèrent beaucoup de non productivité car il existe des familles d'outils sur une machine dont les déclencheurs sont identiques ou multiples. Ainsi, l'opérateur est amené à réaliser les changements d'outils de toute une famille au même moment, générant de cette façon des arrêts machine de longue durée. L'idée est donc de décaler ces changements d'outils afin d'avoir un certain temps de production entre deux changements d'outils d'une même famille et d'éviter des temps d'arrêt trop longs [Dupas et al. 1997b]. De cette façon, le stock en amont est moins souvent plein, évitant ainsi à la machine en amont d'être bloquée. De même, le stock aval évite à la machine suivante d'être affamée par manque de moteurs dans le stock.

Dans le chapitre 3, lors d'un conflit de ressource, la ressource choisit une tâche en fonction d'une heuristique qui lui est affectée. Ici, le comportement d'une ressource opérateur est prédéfini. Lors d'un conflit de ressource, l'opérateur réalise en priorité les changements d'outils de la machine la plus en amont de la ligne de fabrication. Cette heuristique nous est imposée par l'industriel. Le paramètre utilisé pour l'optimisation est le *décalage* initial des changements d'outils.

Le gain maximal théorique pouvant être obtenu grâce au paramètre changement d'outils est de 13.67% (voir tableau 4.2). Ce pourcentage (13.67%) représente le gain possible si tous les changements d'outils sont absents de la ligne de fabrication ou si leur temps de réalisation est nul. Ce gain représente donc une borne maximale possible pour une optimisation utilisant le paramètre changement d'outils.

4.2 Amélioration de performance basée sur une simulation à événements discrets

Dans le but d'améliorer la production de cette ligne, il est nécessaire de la modéliser. Une modélisation basée sur un simulateur à événements discrets est proposée dans un premier temps. Le couplage d'un algorithme génétique avec ce simulateur est détaillé dans un deuxième temps. Une approche permettant de déterminer un bon paramétrage de l'algorithme génétique est présentée dans un troisième temps.

4.2.1 Modélisation de la ligne de fabrication

La modélisation de la ligne de fabrication consiste à créer un modèle de simulation qui est exécuté par le simulateur. Le simulateur employé est un simulateur à événements discrets. Son fonctionnement repose sur des automates d'états finis qui changent d'état en fonction d'événements qui leurs sont internes ou externes.

Il existe quatre familles d'éléments ayant chacune leur mode de fonctionnement. Les machines, les stocks, les CO et les opérateurs représentent ces quatre familles. Chacune de ces familles est représentée par une classe d'objet. Ainsi, chaque élément qui compose la ligne de production industrielle est représenté dans le modèle de simulation par une instance de l'objet représentant sa famille.

La simulation employée est une simulation par processus. Ainsi, au cours de la simulation, chaque élément du problème est représenté par un processus. Ce processus représente l'automate d'état fini qui décrit l'élément.

Nous avons employé un simulateur dit "allégé" car certains éléments de la réalité ont été supprimés. C'est le cas des pannes et des CIO. Nous avons observé que l'erreur occasionnée par cet allégement est négligeable. C'est pour cela que nous avons retiré ces éléments de la simulation afin d'accélérer les temps de calculs. Une explication plus détaillée sur le fonctionnement de ce simulateur à événements discrets est faite dans le chapitre 5.

4.2.2 Optimisation par les algorithmes génétiques

L'objectif du problème est donc de trouver un ensemble de valeurs pour les décalages des changements d'outils afin de maximiser le critère d'évaluation. Pour cela, nous avons utilisé une optimisation basée sur l'emploi d'un algorithme génétique [Cavory 2000, Dupas et al. 1997b]. Le principe général d'utilisation de l'algorithme génétique est décrit ci-dessous.

Codage du problème industriel

Les paramètres de l'optimisation sont les décalages des changements d'outils. Un codage élémentaire est utilisé comme chromosome. Dans ce chromosome, chaque décalage représente un gène. Le problème industriel étant composé de soixante changements d'outils, le chromosome est donc composé de soixante gènes de type entier. La figure 4.4 représente ce chromosome.

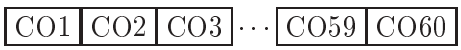

FIG. 4.4 – Exemple de codage du problème industriel

Chaque décalage est défini sur un domaine de définition qui est $]-\mathit{déclencheur}; 0]$. Ainsi, les gènes du chromosome sont eux-mêmes définis sur un domaine de définition qui correspond au domaine de définition du décalage.

L'opérateur génétique de cross-over

Le codage élémentaire du problème autorise l'emploi de cross-over à 1-point. En effet, le chromosome n'utilisant pas de structure particulière, il n'est pas nécessaire d'utiliser un cross-over particulier. Ici, l'objectif est de brasser les gènes d'un même locus. La probabilité de cross-over est fixée à 80%.

L'évaluation d'un individu

L'évaluation de l'individu est assurée par le simulateur à événements discrets. Pour cela, l'algorithme génétique fournit au simulateur les valeurs de chaque décalage des changements d'outils par le biais du chromosome de l'individu à évaluer. Une fois la simulation terminée, le simulateur renvoie à l'algorithme génétique la valeur de l'évaluation de l'individu. Cette valeur représente le nombre de moteurs fabriqués en une semaine de production. La figure 4.5 présente le couplage réalisé entre l'algorithme génétique et le simulateur à événements discrets.

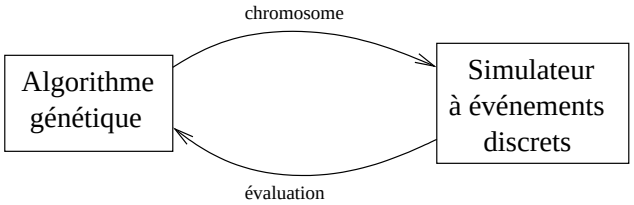

FIG. 4.5 – Couplage algorithme génétique - simulateur à événements discrets

Stratégie d'élitisme

Lors du croisement, il est possible de perdre un individu de bonne qualité. Pour éviter ceci, nous conservons une copie du meilleur individu de la population d'une génération à une autre. Cette stratégie s'appelle l'élitisme.

Les paragraphes suivants détaillent les résultats obtenus par cette approche ainsi que l'expérimentation qui a permis de déterminer les paramètres de l'algorithme génétique les mieux adaptés au problème.

4.2.3 Résultats

Le choix des paramètres dans un algorithme génétique représente une difficulté majeure [Sebag et al. 1996]. Des essais ont été réalisés afin de déterminer quels opérateurs génétiques permettent de faire converger l'algorithme génétique rapidement tout en conservant une certaine garantie sur les résultats pour le problème traité [Dupas et al. 1999, Dupas et al. 2000a]. Pour cela, nous avons utilisé un plan d'expériences sur les paramètres de l'algorithme génétique. Cette méthode a permis de déterminer un paramétrage adapté à ce type de problème.

Nous allons maintenant présenter cette expérimentation et une synthèse des résultats que nous avons obtenus.

4.2.3.1 Expérimentation

Les paragraphes suivants représentent la méthodologie employée pour déterminer les paramètres de l'algorithme génétique. Cette détermination comporte quatre phases : une expérimentation, une analyse basée sur des tests statistiques, une analyse du compromis entre la performance de l'algorithme génétique et le temps de calculs et enfin une comparaison des algorithmes génétiques avec une méthode d'optimisation naïve.

a) Détermination des paramètres de l'algorithme génétique

Nous avons choisi, de façon subjective, de ne pas faire varier certains des paramètres de l'algorithme génétique. Les paramètres inchangés sont les suivants :

- Nombre d'individus dans la population : 100
- Probabilité du cross-over à 1-point : 80%
- Nombre d'expériences : 32
- Condition d'arrêt : convergence de l'algorithme

Dans les plans d'expériences présentés ci-dessous, chaque ligne du plan représente un essai. Chaque essai est composé de trente deux expériences avec des populations initiales différentes. Une expérience correspond à une optimisation. Il est important de noter que les résultats indiqués ci-dessous représentent la moyenne des trente deux expériences réalisées pour un paramétrage donné. Ceci confère une certaine robustesse aux résultats obtenus par l'algorithme génétique qui est un algorithme de nature stochastique.

L'effet de trois paramètres considérés comme importants pour les performances de l'algorithme génétique est étudié ici. Ces trois paramètres sont : la sélection, l'initialisation et la mutation. La méthode des plans d'expériences de Tagushi est utilisée. Les niveaux associés à ces paramètres sont les suivants :

Sélection : deux méthodes de sélection sont testées. La première est la méthode du tournoi qui consiste à prendre aléatoirement deux individus de la population et à conserver le meilleur des deux. La seconde méthode est la méthode classique de la roulette où chaque individu à une probabilité d'être sélectionné qui est proportionnelle à son évaluation.

Initialisation : deux méthodes différentes sont testées. La première initialise la population de façon aléatoire en respectant les domaines de définition des gènes. La seconde, utilise une population aléatoire dont dix pour cent des individus sont des solutions déjà optimisées. Les valeurs de décalage sont déterminées machine par machine. Les valeurs sont ensuite regroupées pour former un chromosome. La méthode d'optimisation est une méthode locale. Cette méthode est rapide car l'espace de recherche est fortement réduit, mais les résultats obtenus sont de mauvaise qualité.

Mutation : Quatre mutations sont utilisées. Les mutations une et deux correspondent à une mutation aléatoire avec un taux de mutation respectivement faible et important. Les mutations trois et quatre utilisent une recherche locale comme opérateur de mutation. Cette recherche locale correspond à une recherche aléatoire dans une des dimensions du chromosome représentant l'ordonnancement.

Les différents niveaux des paramètres sont représentés par le tableau 4.3.

Mutation		**Initialisation**	**Sélection**
1 aléatoire faible	Probabilité = 0,1%	**1** aléatoire	**1** tournoi
2 aléatoire important	Probabilité = 10%	**2** optimisation locale	**2** roulette
3 recherche locale faible	Probabilité = 0,1%		
4 recherche locale importante	Probabilité = 10%		

TAB. 4.3 – Niveaux des paramètres

Nous avons utilisé le plan d'expériences fractionné présenté par le tableau 4.4 pour étudier les effets des paramètres du tableau 4.3 sur la production. Ce plan est un plan fractionné qui réduit de 50% les expériences à réaliser [Pillet 1992]. Ceci permet un gain de temps important car le temps de calcul d'un essai est important. La dernière colonne, appelée m, représente la moyenne des trente deux expériences réalisées avec les mêmes paramètres.

La moyenne de chaque niveau est représentée dans le tableau 4.5. Ce tableau permet d'obtenir la détermination du meilleur niveau de chaque paramètre. Par exemple, la moyenne des essais GA2, GA4, GA5 et GA7 pour lesquels le paramètre "Sélection" est fixé à deux est 17001. Cette

Essai	Mutation	Initialisation	Sélection	m
GA1	1	1	1	16706
GA2	2	1	2	17003
GA3	3	1	1	16719
GA4	4	1	2	16993
GA5	1	2	2	17007
GA6	2	2	1	17076
GA7	3	2	2	17003
GA8	4	2	1	17079

TAB. 4.4 – Plan d'expériences

valeur est meilleure que la moyenne des essais pour lesquels ce paramètre est fixé à un.

Niveaux	Mutation	Initialisation	Sélection
1	16856	16855	16895
2	*17040*	*17041*	*17001*
3	16861	-	-
4	17036	-	-

TAB. 4.5 – Moyennes des essais pour chaque niveau

C'est ainsi que la meilleure combinaison de paramètres est :
Mutation : aléatoire importante (2).
Initialisation : optimisation locale (2).
Sélection : roulette (2).

L'essai représenté par ces paramètres n'ayant pas été réalisé dans le plan fractionné du tableau 4.4, il est important de le réaliser pour confirmer les résultats. Les résultats de cet essai, appelé par la suite GAm, sont présentés dans le tableau 4.6 dans lequel est reportée la moyenne des trente deux expériences réalisées.

Essai	Mutation	Initialisation	Sélection	m
GAm	2	2	2	17088

TAB. 4.6 – Expérience de confirmation

Le résultat obtenu par GAm est supérieur à tous les autres résultats obtenus dans le tableau 4.4. Néanmoins, ce résultat est obtenu à partir d'un plan fractionné et ne tient pas compte des interactions entre les paramètres. Il est donc dangereux de conclure en disant que GAm est LE meilleur résultat pour les paramètres étudiés.

b)Tests statistiques

D'après les tableaux 4.4 et 4.6 les essais GA6, GA8 et GAm permettent d'obtenir les meilleurs résultats. L'idée est donc de vérifier si les trois moyennes de GA6, GA8 et GAm sont semblables. Pour cela, nous avons réalisé un test de comparaison de moyennes avec un niveau de confiance de 99.5%. Le résultat de ce test est représenté par le tableau 4.7. Avec le niveau de confiance choisi, la valeur critique à ne pas dépasser pour ce test est de 2.58.

Avec les valeurs du test de comparaison décrites dans le tableau 4.7, nous pouvons dire que les moyennes de GA6, GA8 et GAm sont semblables avec un niveau de confiance de 99.5%.

	GA1	**GA2**	**GA3**	**GA4**	**GA5**	**GA7**	**GA8**	**GAm**
GA6	70.54	7.51	55.79	7.65	7.76	10.16	**0.32**	**1.53**
GA8	63.31	7.48	51.85	7.65	7.69	9.84	-	**1.15**
GAm	63.21	8.32	52.19	8.41	8.59	10.88	**1.15**	-

TAB. 4.7 – Statistiques de comparaison des maximums obtenus

Nous pouvons donc conclure que :
- Les résultats obtenus par l'algorithme génétique sont meilleurs avec une initialisation non aléatoire de la population initiale.
- Une probabilité de mutation élevée permet d'obtenir de bons résultats aussi bien avec la mutation aléatoire qu'avec une mutation basée sur une recherche locale.
- Il n'est pas encore possible de choisir une méthode de sélection à ce niveau d'expérimentation car les essais GA6 et GA8 utilisent une sélection par tournoi et GAm utilise une sélection par roulette.

Ne pouvant pas se prononcer sur une méthode de sélection, nous avons poursuivi l'analyse afin de déterminer la méthode de sélection adaptée.

c) Analyse du compromis performance/temps de calculs

Notre objectif est de trouver une bonne solution dans un temps de calcul raisonnable. Le temps nécessaire à l'évaluation d'un individu n'est pas négligeable. Le temps de calcul nécessaire à la convergence de l'algorithme génétique est en moyenne d'une heure et demie sur un Pentium II 350 sous Linux pour une expérience. C'est pour cela que le nombre de solutions explorées est mesuré et que ce nombre sert d'indicateur.

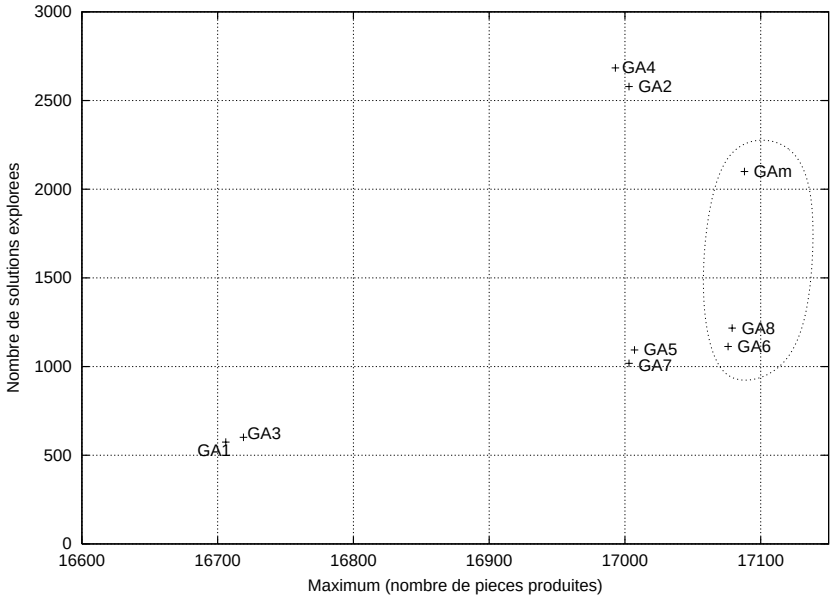

FIG. 4.6 – Performance par rapport au nombre de solutions explorées

La figure 4.6 permet de faire ressortir les essais GA6 et GA8 car le nombre de solutions explorées est plus faible que GAm pour un résultat équivalent.

Nous avons réalisé un test de comparaison des moyennes du nombre de solutions explorées afin de vérifier si GA6, GA8 et GAm sont semblables. Avec un niveau de confiance de 99.5%, le tableau 4.8 indique que GA6 et GA8 sont semblables.

	GA8	**GAm**
GA6	**1.71**	12.24
GA8	-	9.93

TAB. 4.8 – Statistiques de comparaison des temps de calcul

Le meilleur compromis entre le temps de calcul et le résultat trouvé est obtenu avec les essais GA6 et GA8. Le choix d'une sélection par tournoi est donc moins coûteux en nombre de solutions explorées. Néanmoins, il n'est toujours pas possible de déterminer, à ce stade, l'intérêt de la recherche locale comme opérateur de mutation. En effet, l'essai GA6 utilise une mutation aléatoire et l'essai GA8 utilise une recherche locale comme opérateur de mutation.

d) Comparaison de l'algorithme génétique avec une méthode naïve

Dans le but de valider l'approche génétique, nous l'avons comparée avec une recherche aléatoire. La figure 4.7 présente les courbes de convergence des essais GA6, GA8 et GAm ainsi que celle de la recherche aléatoire, appelée 'rand' sur la figure.

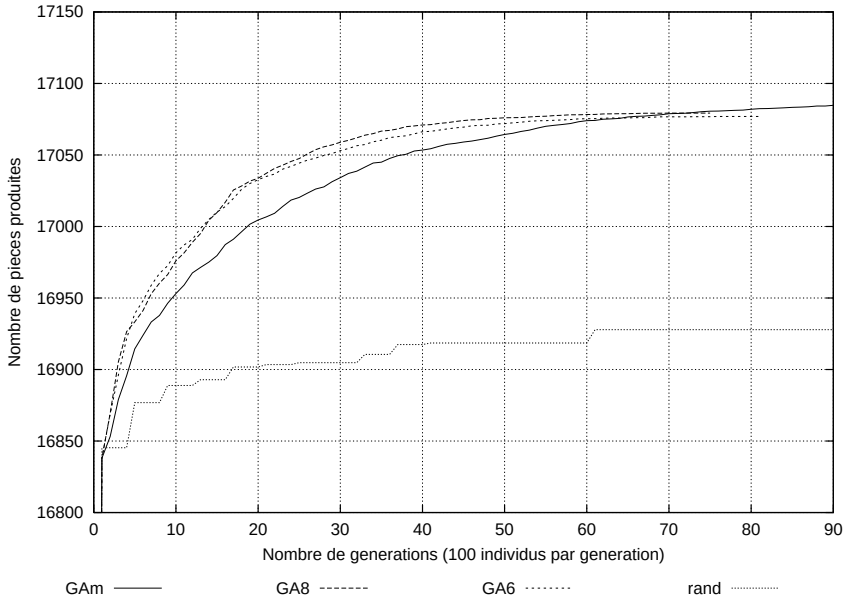

FIG. 4.7 – Comparaison de la recherche aléatoire avec l'algorithme génétique

La recherche aléatoire a été réalisée autant de fois que le nombre de solutions explorées par l'algorithme génétique. Ce nombre n'est pas égal à un nombre de générations multiplié par la taille de la population. En effet, il est possible qu'un individu survive d'une génération à une autre ou qu'il existe plusieurs exemplaires d'un même individu. Les trois essais présentés ici sont tous les trois supérieurs à la recherche aléatoire. De plus, un test statistique a permis de montrer

que l'écart est significatif avec un niveau de confiance de 99.5%. Par ailleurs, il est important de noter que les essais GA1 et GA3 sont inférieurs à la recherche aléatoire.

4.2.3.2 Synthèse

Les conclusions sur l'approche de résolution par le couplage d'un algorithme génétique et d'un simulateur à événements discrets sont les suivantes :

- Les algorithmes génétiques ont permis d'obtenir un gain d'environ 7.5%, sur les 13.67% espérés, par rapport à une approche sans anticipation des changements d'outils. Ce gain représente environ 50% des gains espérés en utilisant les décalages de changements d'outils comme "paramètre" d'optimisation.
- Cette approche est intéressante car elle permet d'obtenir des gains qui sont supérieurs à une approche d'optimisation naïve.
- Le choix des paramètres de l'algorithme génétique a une importance. Une étude statistique a prouvé l'intérêt d'utiliser une méthode d'initialisation non aléatoire. De plus, cette analyse statistique a permis de démontrer la supériorité d'une probabilité de mutation importante. La sélection par un tournoi est meilleure que la sélection par une roulette pour le compromis temps de calcul/résultat.

4.3 Amélioration par les Graphes de Précédences à Contraintes Linéaires

Lors du fonctionnement de la ligne de fabrication, toutes les tâches se réalisent de façon répétitive. De plus, certaines occurrences de tâches sont contraintes par les occurrences d'autres tâches. C'est le cas par exemple d'un changement d'outils qui doit attendre "déclencheur" occurrences de fabrication pour se réaliser. Les graphes de précédences à contraintes linéaires (GPCL) sont donc utilisables pour modéliser la ligne de fabrication.

Nous allons proposer dans un premier temps la modélisation de la ligne de fabrication par les GPCL [Dupas et al. 2000b]. L'ordonnanceur du chapitre 3 utilise cette modélisation pour permettre dans un deuxième temps de refaire les essais d'optimisation réalisés avec le simulateur à événements discrets dans le but de les comparer.

4.3.1 Modélisation par un Graphe de Précédences à Contraintes Linéaires

La ligne de fabrication peut être modélisée par les GPCL. Pour cela, les éléments qui composent cette dernière sont décrits un à un dans un premier temps. L'association de ces éléments afin de modéliser la ligne complète est réalisée dans un second temps.

4.3.1.1 Modélisation des éléments de la ligne de fabrication

Les éléments composant la ligne de fabrication sont détaillés dans les paragraphes suivants. Nous détaillerons les machines, les stocks tampons, les changements d'outils et les opérateurs. Les contrôles inter-opérations ne sont pas modélisés car leur absence dans le modèle génère une erreur négligeable.

Les machines

Les machines sont modélisées par une tâche de production. Cette tâche de production est graphiquement représentée par un noeud du GPCL. Chaque noeud dispose de sa ressource. Cette

ressource peut être vue comme la machine qui réalise la tâche de production. Cette représentation permet de modéliser des machines mono-poste. Néanmoins la modélisation de machines multi postes est possible. La figure 4.8 représente une machine multi-postes *MA1* avec trois postes. Les trois postes sont appelés *PO1*, *PO2* et *PO3* sur la figure. Cette modélisation permet d'affecter les changements d'outils au poste sur lequel il doit se réaliser. Ceci apporte au modèle une représentation proche de la réalité. Néanmoins, ce raffinement de modèle le rend plus complexe, et son évaluation devient donc plus longue. Pour les essais réalisés, la modélisation des machines par une approche mono-poste n'a pas engendré d'erreurs importantes. Nous avons donc choisi de modéliser les machines comme des machines mono-poste. Le temps de cycle de la machine est représenté par "Tfabrication" sur la figure.

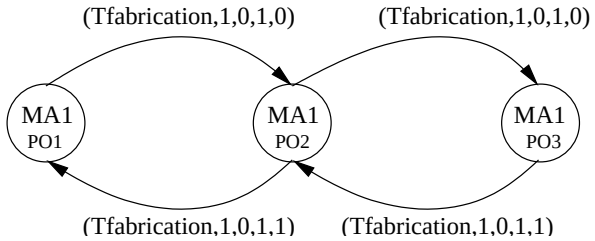

FIG. 4.8 – GPCL d'une machine multi-postes

Les stocks tampons

Les stocks tampons sont modélisés par une tâche de stockage. Cette tâche de stockage est graphiquement représentée par un noeud du GPCL. Chaque noeud dispose de sa ressource. Cette ressource représente physiquement le convoyeur qui stocke et transporte les moteurs.

La figure 4.9 représente le lien entre un stock tampon et deux machines. Le temps de réalisation de la tâche de stockage ou de convoyage et la capacité de stockage sont représentés sur les arcs du graphe reliant un stock et une machine.

Les changements d'outils

Les changements d'outils sont représentés par une tâche de maintenance. Les ressources nécessaires à la réalisation de cette tâche sont des ressources renouvelables limitées appelées ressources opérateurs. Cette tâche de maintenance est graphiquement représentée par un noeud du GPCL. Chaque tâche ne dispose pas de sa ressource. Il existe un nombre très limité de ressources capables de réaliser les tâches de maintenance. Une ressource opérateur est affectée à tous les changements d'outils d'une ou plusieurs machines.

La figure 4.11 représente le lien entre un changement d'outils et une machine. Le temps de réalisation, le déclencheur et le décalage sont des paramètres des arcs reliant une machine et un changement d'outils.

Les opérateurs

Les opérateurs sont représentés par des arêtes pointillées. De cette façon, toutes les tâches qui doivent être réalisées par une ressource sont reliées entre elles par ces arêtes pointillées. Le sous-graphe ainsi créé est dit complet car chaque tâche est reliée à toutes les autres tâches.

4.3.1.2 Modélisation de la ligne

Les éléments qui composent la ligne sont liés entre-eux par des contraintes de précédences. Ce paragraphe présente, dans un premier temps, les relations existantes entre les machines, les stocks et les changements d'outils. Dans un deuxième temps, un exemple de ligne est présenté.

Machine - stock

La difficulté de cette relation est que le stock doit empêcher une machine en amont de lui donner une pièce si celui-ci est plein, et empêcher une machine en aval de prendre une pièce si celui-ci est vide. La figure 4.9 présente un exemple de deux machines *MA1* et *MA2* reliées par un stock tampon *ST1*.

Les paramètres des arcs sont les suivants :
- Tfabrication : temps de cycle d'une machine.
- Tstockage : temps de convoyage d'un stock.
- capacitéSTx : taille du stock STx.

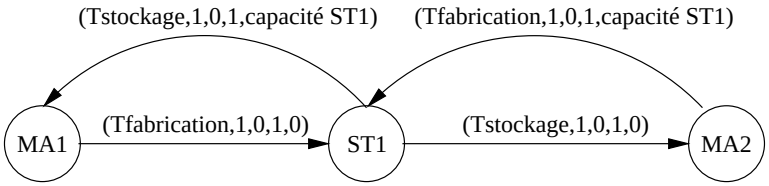

FIG. 4.9 – GPCL machine-stock

Exemple

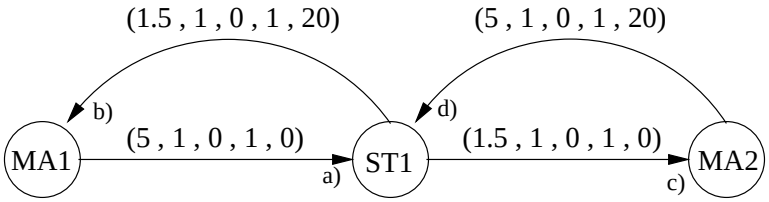

FIG. 4.10 – Modélisation d'une unité de stockage

La figure 4.10 représente le graphe de précédences à contraintes linéaires associé à une ligne de fabrication simple composée de 2 unités de production (*MA1* et *MA2*) et d'un convoyeur (*ST1*) reliant les deux machines. La capacité maximale de ST1 est de capacité "capacitéST1=20" et le temps de convoyage "Tconvoyage=1.5". Les temps de production pour *MA1* et *MA2* sont identiques et égaux à 5 unités de temps.

Les tâches *MAx* et *STx* doivent être interprétées comme suit :
- *MAx* : réaliser la tâche de production sur un moteur.
- *STx* : stocker et transporter un moteur de ma machine MA_x à une machine MA_{x+1}.

Le graphe de précédences de la figure 4.10 se lit de la façon suivante :

a) Dès qu'une occurrence n de la tâche $MA1$, notée $\ll MA1, n \gg$, est terminée alors l'occurrence n de la tâche $ST1$, notée $\ll< ST1, n \gg$, peut démarrer.

b) Dès qu'une occurrence n de la tâche $ST1$, notée $\ll ST1, n \gg$, est terminée alors l'occurrence $n + 20$ de la tâche $MA1$, notée $\ll MA1, n + 20 \gg$, peut démarrer.

c) Dès qu'une occurrence n de la tâche $ST1$, notée $\ll ST1, n \gg$, est terminée alors l'occurrence n de la tâche $MA2$, notée $\ll MA2, n \gg$, peut démarrer.

d) Dès qu'une occurrence n de la tâche $MA2$, notée $\ll MA2, n \gg$, est terminée alors l'occurrence $n + 20$ de la tâche $ST1$, notée $\ll ST1, n + 20 \gg$, peut démarrer.

La contrainte *b)* empêche qu'il y ait un écart entre le numéro de l'occurrence de la tâche *MA1* et le numéro de l'occurrence de la tâche *ST1* supérieur à 20. *MA1* est donc bloquée si *ST1* est plein. De même, la contrainte *d)* empêche qu'il y ait un écart entre le numéro de l'occurrence de la tâche *MA2* et le numéro de l'occurrence de la tâche *ST1* supérieur à 20. *MA2* est affamée si le stock *ST1* est vide.

a) représente le fait qu'une pièce produite par *MA1* est directement transmise dans *ST1* et *c)* représente le "convoyage" entre *ST1* et MA2.

Machine - changement d'outils

Pour cette relation, un changement d'outils doit bloquer la machine lors de sa réalisation. De plus, ce dernier doit se déclencher lorsque le nombre d'occurrences requis est réalisé. Lorsque sa réalisation est terminée, le changement d'outil doit autoriser la machine à fabriquer les pièces suivantes. La figure 4.11 présente un exemple de deux tâches de maintenance *TM1* et *TM2* affectées à une machine *MA1*. Une seule ressource est disponible pour réaliser les tâches *TM1* et *TM2*.

Les paramètres des arcs sont les suivants :

- DéclencheurX.Y, DuréeX.Y et DécalageX.Y : les paramètres constituant le changement d'outils numéro 'Y' réalisé sur la machine 'X'.
- Tfabrication : temps de cycle d'une machine.

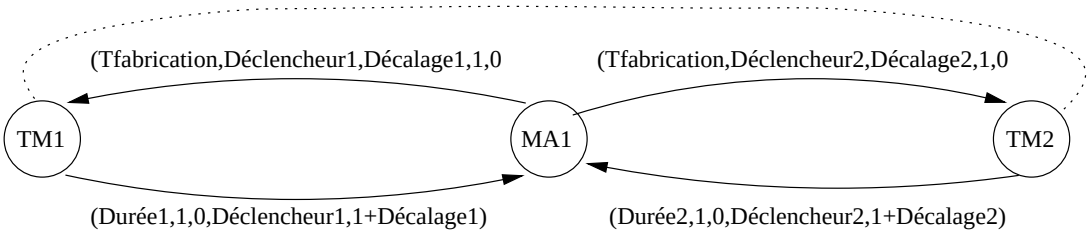

FIG. 4.11 – GPCL machine-changement d'outils

Machine - stock - changement d'outils

Pour représenter graphiquement la ligne complète, il suffit de reproduire les schémas décrits en 4.9 et en 4.11. La figure 4.12 représente un exemple de trois machines et deux stocks où les machines ont respectivement deux, trois et un changement d'outils. Chaque machine dispose de sa ressource pour réaliser les changements d'outils.

4.3.2 Optimisation basée sur l'évaluation d'un GPCL

Les essais réalisés avec une modélisation du problème par les GPCL se limitent aux essais GA6 et GA8 précédents car ils avaient fournis les meilleurs performances. Chaque essai est composé de trente deux expériences. Les paramètres de l'algorithme génétique sont les suivants :

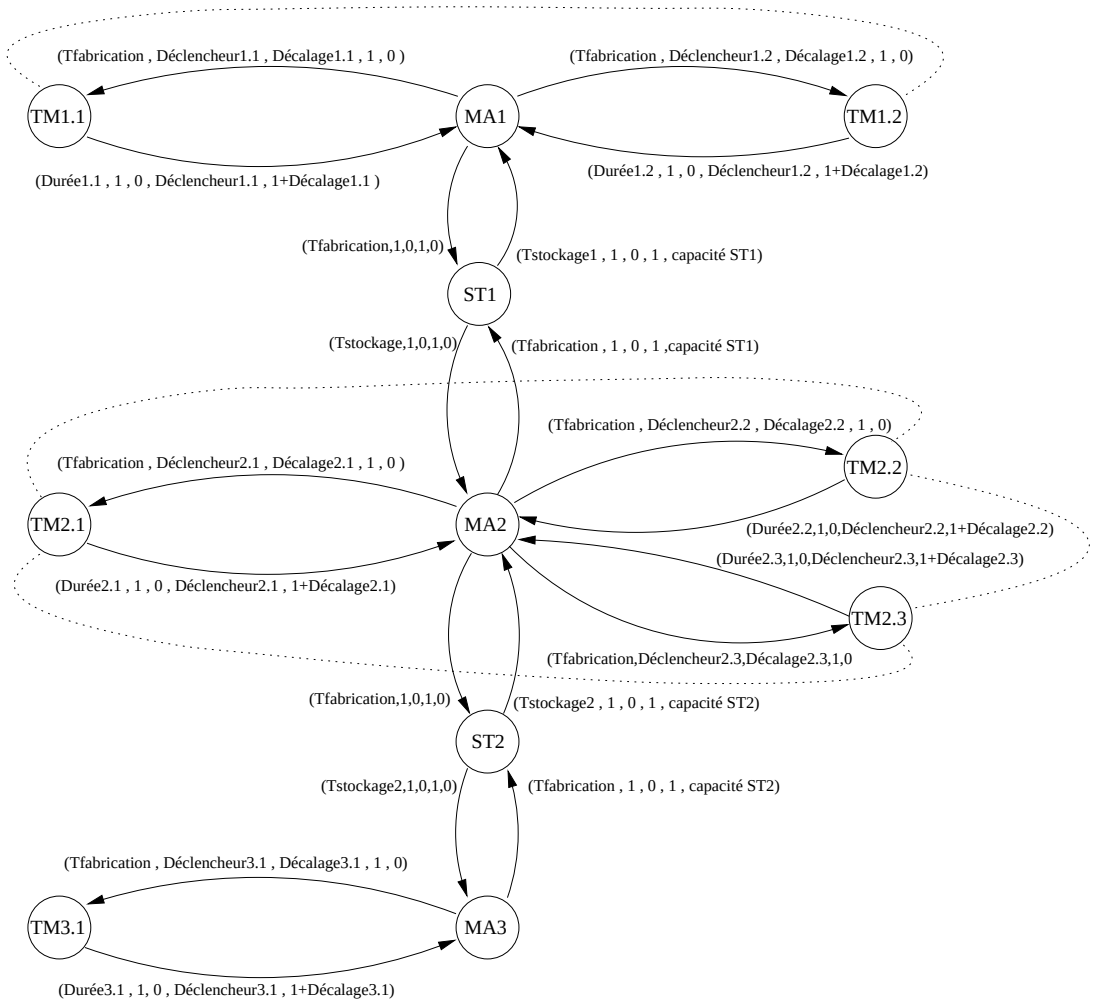

FIG. 4.12 – GPCL d'une ligne

Essai	SED	Ordonnanceur	Écart
GA6	17076	17315	+1.39%
GA8	10079	10210	+1.29%

TAB. 4.9 – Comparaison des résultats obtenus par les deux approches d'évaluation

Méthode d'évaluation	Gain
Sim. à Evt. Dis.	7,50%
Ordonnanceur	7,48%

TAB. 4.10 – Comparaison des gains obtenus par les deux approches d'évaluation

- Le codage : le codage est le même que précédemment. C'est à dire que chaque décalage de changement d'outils est représenté par un gène.
- L'initialisation : les individus de la population sont initialisés de façon aléatoire en respectant les domaines de définitions des gènes. De plus, 10% de cette population représentent des solutions déjà optimisées avec une optimisation locale [Dupas et al. 1997a].
- Opérateurs génétiques :
  - Cross-over : le cross-over utilisé est un cross-over à 1-point avec une probabilité de croisement de 80%.
  - Mutation : la probabilité de mutation des deux essais est de 10%. Par contre, le premier essai utilise une mutation aléatoire et le second utilise une recherche locale comme opérateur de mutation.
- La sélection : l'opérateur de sélection utilisé est la sélection par tournoi deux à deux.
- Élitisme : nous utilisons une stratégie d'élitisme afin de conserver d'une génération à une autre le meilleur individu de la population.
- L'évaluation : l'évaluation d'un individu se réalise par l'ordonnanceur. Pour cela, l'algorithme génétique fourni à l'ordonnanceur les valeurs des décalages des changements d'outils par le biais du chromosome. L'ordonnanceur crée l'ordonnancement associé à ces valeurs puis l'évalue en fonction des critères choisis.

4.3.3 Résultats

Le tableau 4.9 présente les résultats obtenus par l'optimisation basée sur un simulateur à événements discrets (SED) et l'optimisation basée sur l'utilisation d'un ordonnanceur. Les valeurs indiquées représentent la valeur moyenne obtenue sur les trente deux expériences. Elle représente le nombre de moteurs fabriqués en une semaine.

Nous pouvons voir sur le tableau 4.9 que l'écart entre les deux approches est d'environ 1,35%. Cet écart est dû en majeure partie à la modélisation mono-poste des machines qui sont réellement multi-postes. L'écart entre les deux optimisations est faible, par contre, le gain théorique obtenu par ces deux approches est équivalent. Le tableau 4.10 présente les gains obtenus par les deux approches. Ce gain est calculé en comparant une solution optimisée avec une solution où les décalages initiaux ne sont pas pris en compte.

4.4 Conclusion

Grâce aux plans d'expériences, nous avons réussi à déterminer les paramètres de l'algorithme génétique qui fournissent un bon résultat en un temps raisonnable. De plus, le couplage de l'algorithme génétique avec un simulateur à événements discrets permet d'obtenir des gains de production importants.

Nous avons réussi à diminuer d'environ 50% la non-productivité due aux changements d'outils. Ceci représente un gain de production de 7.5%. Il est important de dire qu'un gain de 1% apporte à l'industriel un gain financier de plusieurs centaines de milliers de francs par an. Étant donné le faible coût nécessaire à l'implantation de cette approche, le retour sur l'investissement est très rapide, bien que sa mise en oeuvre soit difficile.

Le couplage de l'algorithme génétique avec un ordonnanceur basé sur les GPCL représente une autre approche également intéressante. En effet, l'écart existant entre les deux approches est faible et le temps d'évaluation d'un individu par l'ordonnanceur est plus faible.

La meilleur approche du point de vue du compromis temps de calculs/résultats est fournie par le couplage de l'algorithme génétique et de l'ordonnanceur basé sur les GPCL. Cette approche permet de trouver une bonne solution dans des temps de calculs raisonnables pour un industriel. Néanmoins, cette approche sera efficace à condition de maîtriser les paramètres de l'algorithme génétique.

Chapitre 5

Plate-forme d'analyses et d'amélioration de performances

Dans le but de réaliser les différentes campagnes d'essais et d'optimisations, nous avons développé une plate-forme d'expérimentations. L'objectif de cette plate-forme consiste à proposer à l'utilisateur un outil simple d'utilisation qui s'appuie sur une méthodologie d'analyse et d'amélioration de performances [Dupas et al. 1998b]. Cette plate-forme doit être facile d'emploi et suffisamment générique pour répondre à de nombreux problèmes. Elle doit être modulable et permettre l'ajout de nouvelles fonctionnalités.

Le principe de la plate-forme est tout d'abord décrit. Sont ensuite présentés les modules qui la composent ainsi que leur fonctionnement.

5.1 Principe de la plate-forme

Pour répondre aux différents besoins, l'architecture logicielle choisie est décomposée en modules. Le module central de cette plate-forme est le module d'évaluation. Autour de celui-ci gravitent les autres modules : le module d'édition de modèles, le module d'analyse et le module d'optimisation. La figure 5.1 représente l'architecture logicielle adoptée.

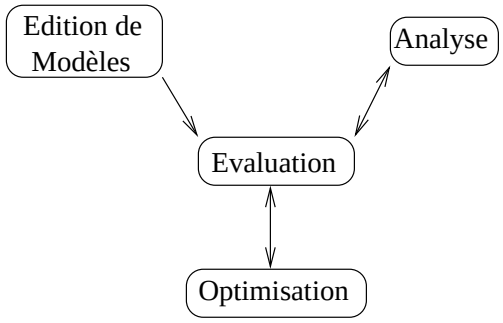

Fig. 5.1 – Architecture de la plate-forme

Cette architecture permet d'utiliser des technologies ou des méthodes adaptées au sous problème traité par chaque module et a permis d'éviter des compromis lors de la réalisation. Cette modularité permet également de remplacer facilement un module sans altérer le fonctionnement global de la plate-forme. En effet, les modules sont spécialisés dans la réalisation de tâches bien précises, ils peuvent donc être remplacés ou modifiés à condition qu'ils respectent un certain

protocole. Les paragraphes suivants sont consacrés à la description et au fonctionnement de ces modules.

5.2 Le module d'évaluation

Ce module doit permettre d'évaluer les performances d'un système quelconque en fonction d'indicateurs prédéfinis. Ces indicateurs peuvent porter sur des aspects différents du processus industriel. Nous pouvons citer à titre d'exemple le nombre d'occurrences d'une tâche, le taux d'occupation d'une ressource ou encore le coût d'utilisation d'une ressource. L'évaluation de ces indicateurs se fait actuellement selon deux méthodes utilisant la simulation. La première est basée sur la simulation à événements discrets alors que la deuxième est basée sur la simulation d'un graphe de précédences à contraintes linéaires (GPCL).

Ces deux méthodes sont basées sur le principe suivant : Un *modèle de simulation* est instancié par des valeurs pour former un *scénario de simulation*. Ce dernier est transmis au *modèle d'exécution* qui l'exécute. Le *modèle d'exécution* fournit après l'exécution un ensemble de résultats qui dépendent des *indicateurs* choisis.

Le *modèle de simulation* décrit l'architecture du processus industriel ainsi que les caractéristiques fixes du processus. Ce dernier décrit également les *indicateurs* qui sont exploités par le modèle d'exécution. Il est communément appelé "modèle". La figure 5.2 présente les différents modèles intervenant dans une simulation.

Le *modèle d'exécution*, appelé aussi simulateur, est composé de règles qui permettent de dérouler dans le temps le *scénario de simulation*. Ces règles dépendent du modèle de simulation et peuvent être paramétrables. Les paragraphes suivants décrivent les deux approches différentes utilisées.

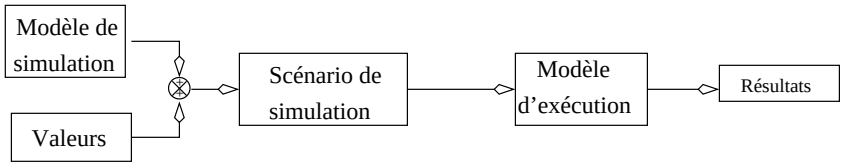

FIG. 5.2 – Les différents modèles d'une simulation

5.2.1 Le simulateur à événements discrets

Le simulateur utilisé est basé sur le principe des automates d'états finis. Chaque entité à simuler est un automate d'états finis qui change d'état en fonction d'événements qui lui sont internes ou externes [Fishwick 1995]. Les entités sont représentées par des classes d'objets. Nous avons classé les entités en quatre familles qui sont : les machines, les changements d'outils, les ressources et les stocks. Chaque famille possède son propre mode de fonctionnement.

Ces familles sont organisées selon la hiérarchie présentée par la figure 5.3. "Process" et "Entity" sont deux familles fournies par la librairie C++SIM [arjuna 1994]. Cette librairie basée sur une approche par processus de la simulation. "Process" représente la classe des processus non interruptibles. Par contre, "Entity" représente la classe des processus interruptibles. C'est pour cette raison que les entités Machine, CO, Ressource et Stock se retrouvent dans la classe Entity. En effet, la production d'une Machine doit pouvoir être interrompue, par exemple, lors d'un changement d'outils. La classe "Atelier" décrit l'architecture des lignes de productions à simuler.

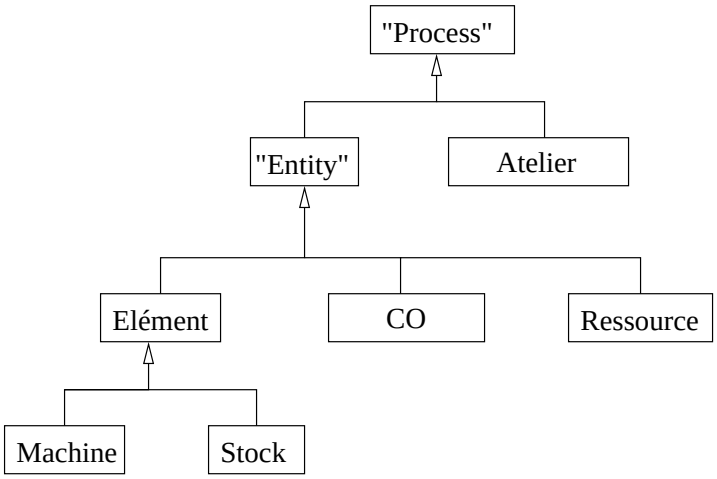

FIG. 5.3 – Hiérarchie des entités du simulateur à événements discrets

Le modèle d'exécution a été conçu [Dupas et al. 1999] en UML (Langage Unifié de Modélisation) [Lai 1997]. Ce langage a permis de décrire la dynamique du modèle d'exécution.

Le langage UML utilise différentes vues qui permettent de modéliser un problème sous différents aspects. Par exemple, les diagrammes d'états-transitions permettent de visualiser les changements d'états d'un objet en fonction des événements qui arrivent. Cette vue permet donc de modéliser le comportement dynamique d'un objet.

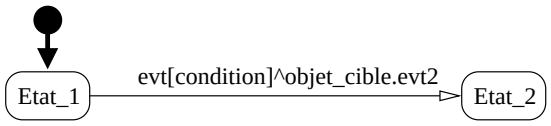

FIG. 5.4 – Exemple de changement d'état

La figure 5.4 présente un diagramme d'états-transitions d'un objet qui peut prendre deux états : $\underline{Etat_1}$ et $\underline{Etat_2}$. L'état initial est $\underline{Etat_1}$. Pour passer de $\underline{Etat_1}$ à $\underline{Etat_2}$, il faut que l'événement *evt* arrive sur l'objet et que la condition *condition* soit respectée. Au moment du changement d'état, un événement *evt2* est envoyé à l'objet *objet_ cible*.

Seuls les diagrammes d'états-transitions des changements d'outils et des ressources sont détaillés dans ce chapitre. Les diagrammes d'états-transitions des autres entités se trouvent dans l'annexe E.

5.2.1.1 Les changements d'outils

La figure 5.5 représente le diagramme d'états-transitions d'un changement d'outils. Cette entité peut être dans les trois états suivants :

- actif : le changement d'outils est en cours de réalisation.
- inactif : le changement d'outils attend d'être déclenché.
- attente ressource : le changement d'outils attend que la ressource dont il a besoin soit disponible pour sa réalisation.

L'état initial d'un changement d'outils est l'état $\underline{inactif}$. Lorsqu'un changement d'outils reçoit l'événement *réaliser* de la machine sur laquelle il est affecté, deux cas de figure sont envisageables :

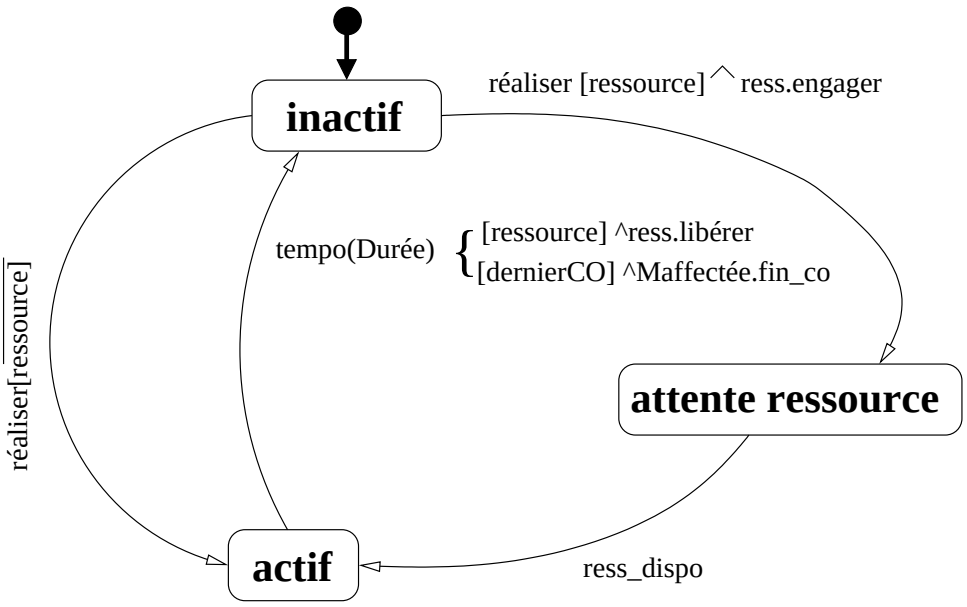

FIG. 5.5 – Diagramme d'états-transitions d'un changement d'outils

1. Aucune ressource n'est nécessaire à la réalisation du changement d'outils : dans ce cas, le changement d'outils passe dans l'état *actif*.
2. Une ressource est nécessaire à la réalisation du changement d'outils : dans ce cas, le changement d'outils passe dans l'état *attente ressource*. Au moment du changement d'état, un événement *engager* est envoyé à la ressource dont le changement d'outils a besoin pour être réalisé. Le changement d'outils quitte cet état pour l'état *actif* lorsque la ressource lui envoie l'événement *ress_ dispo*.

Le changement d'outils passe de l'état *actif* à l'état *inactif* lorsque "Durée" unités de temps se sont écoulées. Ce temps représente le temps de réalisation du changement d'outils. Lors du changement d'état, deux événements peuvent être générés :

1. Si le changement d'outils a utilisé une ressource, cette dernière est libérée par l'événement *libérer*. Ceci est représenté sur le diagramme par : "tempo(Durée)[ressource]^ress.libérer".
2. Si le changement d'outils est le dernier à être réalisé sur la machine, la machine est autorisée à reprendre la production en envoyant au changement d'outils l'événement *fin_ co*. Ceci est représenté par : "tempo(Durée)[dernierCO]^Maffectée.fin_co".

5.2.1.2 Les ressources

La figure 5.6 représente le diagramme états-transitions d'une ressource. Cette dernière peut se trouver dans deux états :

– occupée : la ressource est employée à la réalisation d'une tâche.
– libre : la ressource attend une tâche à réaliser. Cet état est l'état initial.

La ressource passe de l'état libre à l'état *occupée* lors de la réception de l'événement *engager*. A ce moment, la ressource renvoie à l'objet qui a envoyé cet événement, l'événement *ress_ dispo*. Ceci est représenté sur le diagramme par : "engager^Eaffectée.ress_dispo". Cet événement permet de débuter la réalisation de la tâche qui a besoin de la ressource.

Lorsque la ressource est dans l'état *occupée*, trois situations sont possibles :

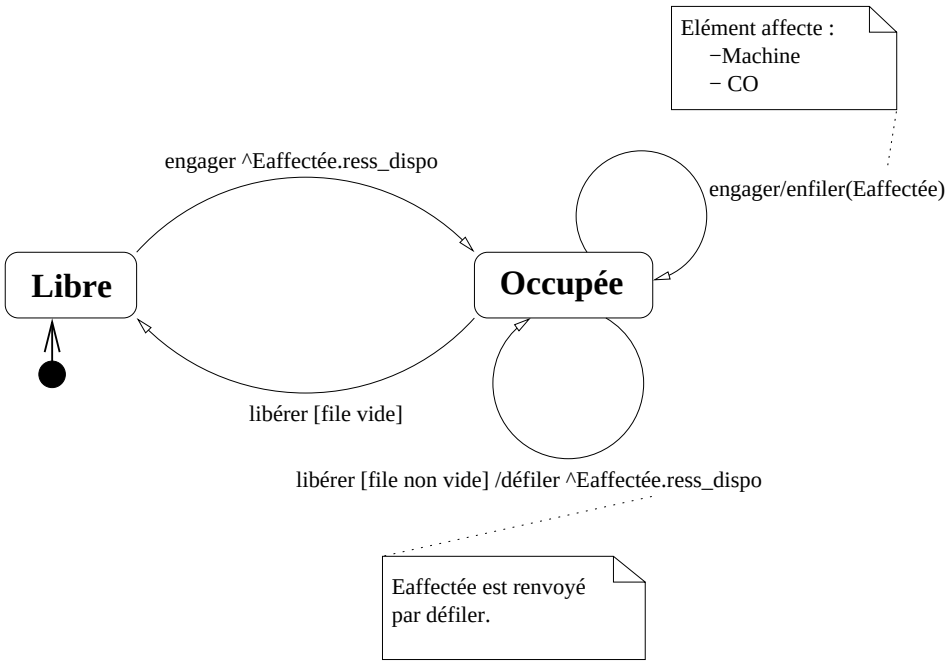

FIG. 5.6 – Diagramme d'états-transitions d'une ressource

1. Un autre événement *engager* arrive. La ressource étant déjà occupée, elle ne peut pas réaliser la tâche représentée par l'objet qui a envoyé l'événement. Pour cela, la ressource va garder en mémoire l'objet source de l'événement grâce à l'événement "engager/enfiler(Eaffectée)". La ressource reste dans l'état *occupée*.
2. Un événement *libérer* est reçu par la ressource et des objets attendent toujours que la ressource soit disponible. Dans ce cas, la ressource reste dans l'état *occupée* et elle renvoie l'événement *ress_ dispo* à l'objet suivant en attente dans la file. La commande 'défiler' permet de déterminer l'objet suivant en attente. Ceci se note "libérer[file non vide]/défiler^Eaffectée.ress_dispo" sur le diagramme.
3. Un événement *libérer* est reçu par la ressource et aucun objet n'est en attente. Dans ce cas, la ressource passe dans l'état libre.

La ressource gère une file de changements d'outils en *attente ressource*. Cette file permet de déterminer quels sont les changements d'outils en attente. La règle de gestion de cette file est la règle du premier arrivé, premier servi, connue également sous le nom de FIFO.

5.2.1.3 Ressources-CO

Les diagrammes de séquences permettent de visualiser les interactions entre plusieurs objets. La figure 5.7 décrit l'interaction qu'il y a entre une ressource "Ressource :1" et un changement d'outils "CO :1". Ce diagramme présente un scénario où un événement *réaliser* arrive sur le changement d'outils "CO :1". Ce dernier ayant besoin de la ressource "Ressource :1", il lui envoie l'événement *engager*. Le changement d'outils reste en attente ressource tant que la ressource ne lui a pas envoyé l'événement *ress_ dispo*. Une fois cet événement arrivé, le changement d'outils passe dans l'état actif et attend "Durée" unités de temps. Ce temps écoulé, le changement d'outils devient à nouveau inactif et libère la ressource par l'événement *libérer*. Cet événement permet à la ressource de devenir à nouveau libre car aucun autre changement d'outils n'est en attente.

Le changement d'outils étant le seul changement d'outils à réaliser sur la machine, l'événement *fin_ co* lui est envoyé.

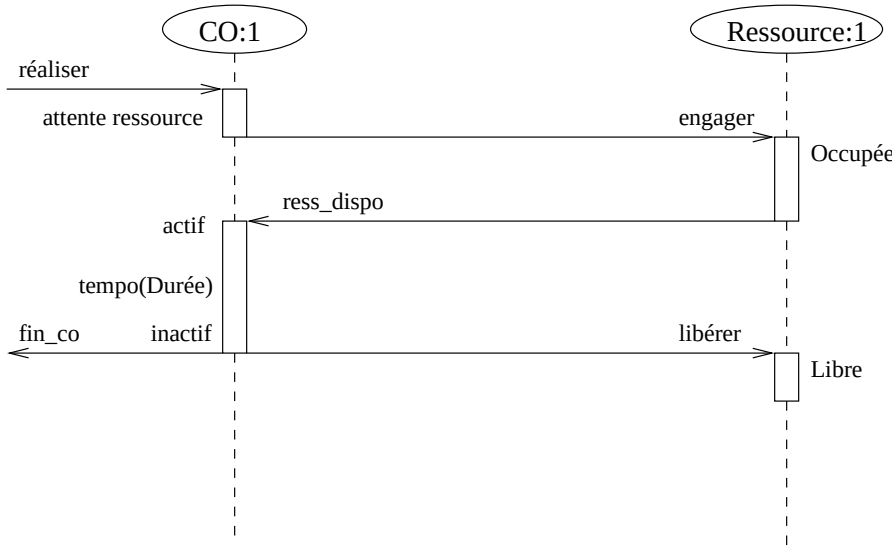

FIG. 5.7 – Diagramme de séquences entre une ressource et un CO

La mise en oeuvre informatique a été réalisée en C++ grâce à une librairie de fonctions appelée C++SIM [arjuna 1994]. Cette librairie a été choisie car elle a déjà permis de réaliser de nombreux projets de simulation dans plusieurs laboratoires. Elle permet de gérer facilement les processus légers et dispose d'outils de synchronisation tels que les sémaphores ainsi que d'une horloge virtuelle permettant de cadencer les tâches à réaliser. Ceci permet lors de l'exécution d'un scénario de simulation, de considérer chaque entité qui compose la ligne de production comme un processus léger (Thread).

Ce simulateur à événements discrets a été utilisé lors de nombreux travaux internes [Cavory et al. 2000a, Cavory 2000, Dupas et al. 1999, Cavory et al. 1999]. Sa stabilité a été validée au cours de ces travaux. Les résultats ont été comparés avec le simulateur industriel Witness afin de valider leur exactitude.

5.2.2 L'ordonnanceur

L'ordonnanceur, est un modèle d'exécution basé sur les priorités. Il permet "d'exécuter" un graphe de précédences à contraintes linéaires (GPCL). Son objectif est de créer un ordonnancement sur un horizon déterminé et de compter le nombre d'occurrences de chaque noeud du graphe. Il est composé de trois étapes : la lecture du GPCL, l'initialisation du GPCL et la construction de l'ordonnancement.

La lecture du GPCL permet d'instancier le modèle d'exécution. C'est dans cette étape que les caractéristiques du graphe sont lues.

L'initialisation du GPCL permet de transformer ce graphe en RdP et de déterminer les occurrences réalisables sur chaque noeud, au démarrage de la simulation. L'initialisation utilise l'algorithme 3.3.

La construction de l'ordonnancement se réalise en gérant des conflits de ressources grâce à des heuristiques. Les ordonnancements créés sont des ordonnancements du type *sans retard*,

introduit dans le chapitre 1. La méthode permettant de créer les ordonnancements a été décrite dans le paragraphe 3.4.

5.3 Le module d'édition de modèles et de scénarios de simulation

Ce module doit permettre à l'utilisateur de créer un modèle de simulation rapidement. Ce module est entièrement dédié à la méthode basée sur les événements discrets. Un langage de modélisation basé sur des activités simples tels que, par exemple, le stockage ou l'usinage est proposé à l'utilisateur.

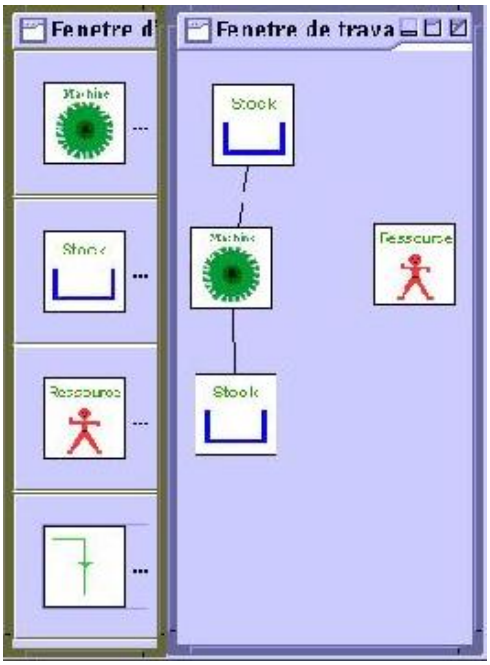

Fig. 5.8 – Éditeur de modèle

La création du modèle de simulation et du scénario de simulation se réalise par l'interface présentée en figure 5.8. Le volet gauche de cette interface propose les activités de base et le volet droit la zone de travail où l'on place les différentes activités qui composent la ligne de fabrication ainsi que leurs liens. Dans un deuxième temps, il faut compléter ce modèle de simulation en ajoutant les valeurs des caractéristiques grâce à une interface de saisie, créant ainsi un scénario de simulation. Ces caractéristiques dépendent du type d'activité. Par exemple, l'activité de stockage a comme caractéristique la capacité maximale de stockage. L'évaluation du scénario de simulation se fait dans un troisième temps à l'aide du module d'évaluation.

Le module d'édition a été réalisé avec le langage Java. Ce langage propose des outils permettant de créer rapidement des interfaces graphiques. En dépit de la lenteur actuelle de ce langage, ce module ne pénalise pas la rapidité de la plate-forme.

5.4 Le module d'analyse

Ce module vise à analyser l'effet des paramètres qui agissent sur le système de production tels que les opérateurs ou les opérations de maintenance. Cette analyse conduit à focaliser l'étude d'amélioration de performance sur certains paramètres qui engendrent le plus de non-productivité.

Ce module utilise une méthode globale sur l'ensemble du modèle de simulation qui est basée sur les plans d'expériences [Pillet 1992]. Cette méthode consiste à effectuer une suite d'essais organisés à l'avance de manière à déterminer en un minimum d'essais et avec un maximum de précision l'influence de multiples facteurs sur une ou plusieurs grandeurs de sortie ou réponses. Chacun de ces essais est représenté par un scénario de simulation qui est évalué par le module de simulation. A la fin de l'analyse, ce module indique quels sont les paramètres importants ainsi qu'une variation de la grandeur de sortie en fonction de l'état des paramètres.

Pour réaliser l'analyse d'une ligne, il faut dans un premier temps fixer les paramètres sur lesquels l'analyse porte ainsi que des niveaux qu'ils peuvent prendre. Dans l'exemple de la figure 5.8, il est possible, par exemple, de choisir la caractéristique capacités maximales des deux stocks, notée c_i pour le stock i, en autorisant les niveaux c_i et $2.c_i$. Ce module se charge, dans un deuxième temps, de créer la table des expériences, appelée aussi table de Tagushi, et d'exécuter les scénarios de simulation. Dans l'exemple cité précédemment, il y a quatre essais à réaliser. Chaque essai est évalué par le biais du module d'évaluation. Au cours du troisième et dernier temps, ce module synthétise les résultats des essais et calcule les gains envisageables ainsi que la corrélation entre les paramètres.

Ce module, encore en cours de développement, utilise également le langage Java pour sa simplicité à créer des interfaces graphiques. La vitesse d'exécution de ce module ne pénalisant pas la vitesse de réaction de la plate-forme, l'utilisation d'un langage semi-compilé peut être considérée comme un choix judicieux.

5.5 Le module d'optimisation

Ce module permet d'optimiser les performances du processus de production. Pour cela, il est nécessaire de se fixer un ou plusieurs paramètres à optimiser ainsi que leur domaine de définition et un ou plusieurs critères d'optimisation. Le module d'analyse présenté ci-dessus est utilisé pour aider l'utilisateur à trouver les paramètres sur lesquels l'optimisation est faite. Le module d'optimisation utilise le module d'évaluation à chaque fois qu'il doit évaluer un scénario de simulation.

Les problèmes traités par cette plate-forme sont très souvents des problèmes NP-difficiles. De plus, la fonction d'évaluation est non-linéaire et discontinue. La taille et la complexité de l'espace de recherche en font des problèmes non résolvables, dans des temps de calculs raisonnables, par des méthodes itératives. La méthode méta-heuristique choisie est celle des algorithmes génétiques. Ils permettent d'obtenir un résultat satisfaisant en réalisant un nombre limité d'évaluations [Venturini 1995]. Néanmoins, grâce à l'approche modulaire de cette plate-forme, il est possible d'utiliser un autre algorithme tel que, par exemple, le recuit simulé pour réaliser cette optimisation.

Pour réaliser une optimisation, il faut donner au module d'optimisation quatre éléments : le modèle de simulation, la liste des variables qui font l'objet de la recherche d'amélioration, les domaines de définition des paramètres et les paramètres de l'algorithme génétique. Le module d'évaluation est paramétré pour utiliser l'indicateur de performance choisi. A la fin des calculs,

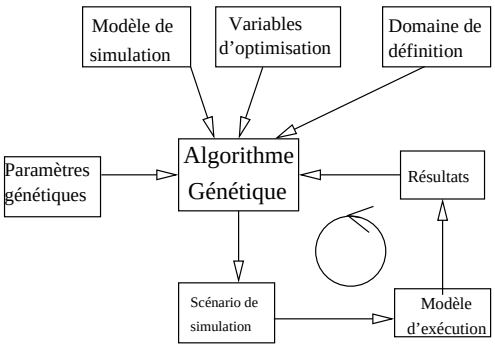

FIG. 5.9 – Utilisation de l'algorithme génétique

le module d'optimisation fournit un fichier résultat contenant un ensemble de solutions ainsi que leur évaluation. La figure 5.9 représente les différents éléments utilisés lors d'une optimisation.

Pour réaliser ce module, la librairie de fonctions libGA [Corcoran 1993] a été utilisée. De nombreuses modifications ont été apportées afin de diminuer le nombre d'évaluations à réaliser sans pour autant altérer la qualité des résultats. Ces modifications portent sur les opérateurs génétiques et sur l'utilisation d'un cache de calcul. En effet, il n'est pas rare de calculer l'évaluation d'un même individu plusieurs fois si celui-ci survit d'une génération à une autre ou tout simplement si celui-ci est recopié dans la population. Les temps de simulation étant "longs", l'utilisation d'un cache permettant d'éviter de recalculer un individu déjà calculé diminue le temps global d'optimisation. Le choix du langage s'est porté sur un langage permettant d'obtenir de bonnes performances lors de l'exécution. Le langage 'C' a été utilisé pour ce module car la librairie LibGA est elle-même programmée en C.

5.6 Communication entre les modules

L'échange d'informations entre les modules se fait grâce à un protocole prédéfini. Ce protocole est basé sur l'utilisation de fichiers textes. Cette méthode élémentaire permet de conserver la modularité de notre plate-forme et permet aussi l'ajout de nouveaux modules. L'utilisation de fichier pour communiquer ralentit légèrement la vitesse d'exécution à cause du temps d'ouverture et de fermeture de ces derniers. La syntaxe de description du fichier est volontairement explicite pour pouvoir réaliser avec un éditeur de texte un modèle de simulation. Ceci permet de maintenir facilement la plate-forme. La communication entre le module d'optimisation et le module d'évaluation par l'ordonnanceur est représentée pour l'exemple de la figure 3.24 par le fichier suivant, en supposant que chaque tâche dispose d'une ressource :

Nressource 4
Ntache 4
TempsSimul 1000
Affectation 1 1
Affectation 2 2
Affectation 3 3
Affectation 4 4
Noeud 1 Arc 2 1 0 1 1 4
Noeud 2 Arc 2 1 0 2 1 4
Noeud 3 Arc 2 2 0 1 1 4

Noeud 4 Arc 1 1 0 1 0 1 Arc 1 2 0 1 0 2 Arc 1 1 0 2 0 3
#ccommentaire de fin

Les informations contenues dans ce fichier s'interprète de la façon suivante : dans un premier temps, le nombre de ressources disponibles (*Nressource*) est déclaré, suivi du nombre de tâches (*Ntache*). Le temps durant lequel la simulation doit avoir lieu (*TempsSimul*) est déclaré ensuite. L'affectation des tâches sur les ressources vient ensuite. Pour cela, le numéro de la ressource est indiqué dans un premier temps puis la liste des tâches affectées à cette dernière. Par exemple, "*Affectation 1 1*" indique que la ressource 1 est capable de réaliser uniquement la tâche 1. Ces informations représentent l'en-tête du fichier. Les données des arcs et des noeuds représentent le corps du fichier. Pour cela, il faut dans un premier temps nommer le noeud sur lequel un arc arrive, par exemple *'Noeud 1'*, puis indiquer qu'un arc dont les paramètres sont : (2,1,0,1,1) a pour cible le noeud 1 et pour source le noeud 4, ce qui se fait par : *'Arc 2 1 0 1 1 4'*. Il est possible d'ajouter au fichier des commentaires.

5.7 Conclusion

L'utilisation de la plate-forme a permis de concrétiser les travaux décrits dans les chapitres 3 et 4. Lors de l'ajout de l'ordonnanceur comme méthode d'évaluation alternative au simulateur à événements discrets, l'approche modulaire de cette plate-forme s'est révélée efficace. En effet, ce nouveau module a été ajouté à la plate-forme en ne réalisant aucune modification sur les autres modules.

La hiérarchie de classes d'objets utilisée dans le simulateur à événements discrets facilite la création de nouveaux éléments. Les nouveaux éléments héritent des fonctionnalités d'éléments parents et peuvent ainsi spécifier leurs fonctionnalités.

Le langage UML a permis de concevoir le fonctionnement du modèle d'exécution du simulateur à événements discrets. Les différentes vues autorisent la possibilité de créer des 'mises en situation' du modèle d'exécution et permettent de visualiser sa dynamique.

Nous pouvons dire que tous les objectifs de la plate-forme sont atteints.

Conclusion générale et perspectives

Le travail de cette thèse a pour origine une étude industrielle dans le domaine de la production automobile. Le but était d'évaluer et d'améliorer les performances d'une ligne de fabrication à l'aide de la simulation. Pour cela, nous avons proposé une analyse de performances de la ligne de fabrication basée sur les plans d'expériences. Les résultats de cette analyse nous ont conduits à nous focaliser sur l'un des paramètres générateur de rendement, c'est à dire les tâches de maintenance systématique.

Dans le but d'améliorer la gestion de ces tâches de maintenance systématique, nous avons proposé une architecture basée sur un couplage entre un algorithme génétique et un simulateur à événements discrets "allégé" que nous avons développé au sein d'une plate-forme d'analyse et d'amélioration de performance. Nous avons ensuite étendu nos travaux à la gestion des problèmes d'ordonnancement cyclique. Dans ce cadre, nous avons proposé une nouvelle méthode pour résoudre de façon approchée le problème du Job-Shop cyclique, réputé NP-difficile.

Deux approches ont été utilisées dans cette méthode. La première est basée sur l'utilisation d'une liste de préférences de réalisation des tâches sur les ressources. L'objectif de cette approche est de trouver une liste de préférences par ressource qui maximise le critère d'évaluation choisi. La seconde, est basée sur le pilotage des ressources. Pour cela, il faut déterminer sur chaque ressource quelle heuristique utiliser en cas de conflit. Le but est donc d'affecter à chaque ressource une heuristique afin de maximiser le critère d'évaluation choisi. L'algorithme génétique permet de trouver la liste de préférences pour la première méthode et l'affectation des heuristiques sur les ressources pour la deuxième.

Les points plus significatifs de nos travaux sont :

- La mise en oeuvre des plans d'expériences pour analyser la ligne de fabrication avec un faible nombre d'essais.
- L'utilisation des plans d'expériences pour déterminer les paramètres de l'algorithme génétique qui permettent d'obtenir un bon compromis résultats/rapidité de convergence.
- La conception d'un simulateur "allégé" à événements discrets basée sur une approche par processus grâce au langage UML.
- Le couplage d'un algorithme génétique et du simulateur "allégé" pour améliorer la performance de la ligne de fabrication.
- La proposition d'un ordonnanceur qui "évalue" la performance d'un graphe de précédences à contraintes linéaires (GPCL). Pour cela, cet ordonnanceur traduit le GPCL en Réseau de Petri T-temporisé. Ensuite, il utilise ce Réseau de Petri pour déterminer les dates de réalisation des occurrences des tâches. Les conflits d'utilisation de ressource sont gérés par une heuristique qui est affectée à la ressource.
- La proposition d'une résolution du Job-Shop cyclique par une méthode méta-heuristique basée sur une évaluation du graphe de précédences à contraintes linéaires par un ordonnanceur que nous avons développé.

Les résultats obtenus sont déjà très encourageants. Néanmoins, les perspectives d'amélioration sont très nombreuses et portent essentiellement sur l'amélioration de la méthode de résolution ainsi que sur la plate-forme.

- L'amélioration de la méthode de résolution concerne les trois points suivants :
  - L'ordonnanceur actuel réalise des ordonnancements sans retard. Ce type d'ordonnancement ne garantissent pas de trouver *l'optimum*. L'utilisation de l'algorithme d'ordonnancement de Giffler et Thompson est envisagée pour placer les tâches sélectionnées. Cet algorithme génère des ordonnancements actifs qui sont dominants pour tous critères réguliers. De cette façon, nous garantissons que notre méthode est capable de trouver *l'optimum*.
  - L'influence des paramètres de l'algorithme génétique tels que la taille de la population, l'opérateur de cross-over ainsi que sa probabilité n'a pas été évaluée et constitue une source d'amélioration potentielle de la performance de l'algorithme génétique. De plus, une étude du paysage de ce problème aiderait aux choix des opérateurs génétiques. Cette étude nous fournira l'allure générale de la fonction à optimiser en fonction de l'opérateur de voisinage utilisé. Par ailleurs, l'utilisation de méthodes hybrides améliore dans le cas général les performances de l'algorithme génétique. Le choix d'une telle méthode représente donc une extension avantageuse et constitue donc un axe de développement intéressant. Les hybridations envisagées portent sur l'opérateur de mutation. L'exploration de l'espace de recherche par le biais d'un recuit simulée ou d'une méthode tabou est envisagée.
  - Les jeux d'essais présentés représentent des exemples académiques simples. La structure des graphes de précédences utilisée est rudimentaire. La recherche de jeux d'essais basés sur des structures plus complexes est primordiale. Ces jeux d'essais contribuent à la validation de la robustesse de la méthode GA-ORDO.
- Les fonctionnalités de la plate-forme peuvent être améliorées afin de fournir des résultats plus rapidement et plus précisément. Pour cela, deux voies d'amélioration sont possibles.
  - Actuellement, l'optimisation porte sur un seul critère d'évaluation. Ce système d'évaluation et très limitatif. En effet, il serait intéressant de pouvoir tenir compte d'autres critères tels que, par exemple, les coûts ou la satisfaction du client. Pour cela, l'utilisation de fonctions d'évaluation multicritères devient indispensable. La difficulté de ce type de fonction réside essentiellement dans leur fabrication. Pour cela, nous projetons d'utiliser une fonction "floue" afin de réaliser une fonction d'évaluation entre deux critères antagonistes.
  - La plate-forme peut être enrichie de nouvelles fonctionnalités par l'ajout de nouveaux modules. La fonctionnalité la plus intéressante est de permettre à la plate-forme de fonctionner sur un réseau informatique hétérogène. De plus, la répartition de l'évaluation sur plusieurs ordinateurs est une perspective intéressante. Cette répartition apportera un gain de temps pendant les optimisations.

Annexes

Annexe A

Ordonnancement : définitions et notation

Définition : *Ordonnancer un ensemble $T = \{1, ..., n\}$ de tâches, c'est déterminer, pour chaque tâche $i \in T$, une date de début d_i et leur allouer les ressources nécessaires.*

A.1 Liste des notations

- s_i : date de début de réalisation de la tâche i.
- c_i : date de de fin de réalisation de la tâche i, appelé aussi Completion time.
- d_i : deadline de la tâche i.
- p_i : durée de réalisation de la tâche i.
- m_i : moyens consomés par la tâche i.
- I_{ik} : intensité de consommation du moyen k par la tâche i.
- $t(i, n)$: date de début de la $n^{ième}$ exécution de la tâche i.
- p_{ij} : temps d'exécution de la tâche i sur la ressource j.
- $\ll i, n \gg$: $n^{ième}$ occurrence de la tâche i.
- q_i : latence de la tâche i.
- $L_i = c_i - d_i$: retard sur la réalisation de la tâche i, appelé aussi Lateness.
- $\mathcal{T}_i = max\{L_i; 0\}$: Tardiness.
- C_{max} : Makespan.

A.2 Contraintes de précédences

- T : ensemble des tâches à ordonnancer.
- P : ensemble des contraintes de précédences entre les tâches.
- $G = \{T, P\}$: graphe de précédence.

La **contrainte de précédence linéaire** $e = (i, j)$, valuée de $(p_i, \alpha, \beta, \gamma, \omega)$, entre les tâches i et j se note :

$$\forall n > 0\,,\ t(i, \alpha n + \beta) + p_i \leq t(j, \gamma n + \omega)$$

La **contrainte de précédence uniforme** $e = (i, j)$, valuée de (p_i, β, ω), entre les tâche i et j se note :

$$\forall n > 0\,,\ t(i, n + \beta) + p_i \leq t(j, n + \omega)$$

A.3 Dénomition

- Activité : travail élémentaire dont la réalisation nécessite un certain nombre d'unités de temps et d'unités de chaque ressource.
- Job : constitué d'une ou plusieurs activités
- Ressource : moyen technique ou humain dont la disponibilité limitée ou non est connue à priori.
- Gamme : relation de précédence entre les tâches qui composent le job.

Annexe B

Notation des problèmes d'ordonnancement

B.1 Paramètres d'architecture α

Le champ α permet d'identifier le nombre de machines disponibles, le type des machines ainsi que le nombre de jobs. Ce champ se décompose en trois sous champs qui sont α_1, α_2 et α_3.

- $\alpha_1 \in \{o, P, Q, R, O, F, J\}$
  - $\alpha_1 \in \{o, P, Q, R\}$: Dans ce cas de figure, chaque job est composé d'une activité qui peut être réalisée sur une des m machines disponibles.
    - $\alpha_1 = o$: Seule une machine est disponible pour réaliser les jobs. Les jobs sont donc composés d'une tâche et le temps de réalisation d'une tâche j se note : p_j. Chaque job est donc réalisé sur cette unique machine à des dates différentes.
    - $\alpha_1 = P$: Un ensemble de machines parallèles identiques est disponible pour réaliser les jobs. Le temps de réalisation du job i est le même sur toutes les machines. Nous avons donc la relation suivante : $p_{ij} = p_j \, \forall i$. Ceci est mieux connu sous le terme de *Identical Parrallel Machines*.
    - $\alpha_1 = Q$: Les machines disponibles pour la réalisation des jobs sont dites parallèles proportionnelles. Cela veux dire que toutes les machines sont identiques, mais elles possèdent chacune une vitesse d'exécution s_i. Nous avons donc un temps de réalisation $p_{ij} = p_j/s_i$ pour une machine M_i. Ceci est mieux connu sous le terme de : *Uniform Parallel Machines*.
    - $\alpha_1 = R$: Ici, les machines disponibles sont parallèles quelconques. Le temps de réalisation d'une activité dépend de la machine sur laquelle elle sera réalisée. On parlera de : *Unrelated Parallel Machine*.
  - $\alpha_1 \in \{O, F, J\}$ Chaque job est composé de plusieurs activités qui seront réalisées sur des machines différentes. L'ordre dans lequel sont exécutées les activités s'appelle : *gamme*.
    - $\alpha_1 = O$: Les jobs ne possèdent pas de gamme. Les activités peuvent se réaliser dans n'importe quel ordre. C'est le cas d'un *Open-Shop*.
    - $\alpha_1 = F$: Tous les jobs possèdent la même gamme. Ceci représente l'ensemble des *Flow-Shop*.
    - $\alpha_1 = J$: Les jobs possèdent des gammes qui sont différentes d'un job à l'autre. Ceci représente l'ensemble de *Job-Shop*.
- $\alpha_2 \in \mathbb{Z}_+$: Le nombre de machines $m \in \mathbb{Z}_+$ est constant.
- $\alpha_2 = o$: Le nombre de machines m est variable.

- $\alpha_3 \in \mathbb{Z}_+$: Le nombre de jobs n est constant.

B.2 Paramètres des Jobs β

Le champ β caractérise les contraintes sur les jobs et les ressources. Il permet de connaître les relations de précédence entre les jobs ainsi que des informations sur les dates de début et le temps de réalisation. Ce champ se décompose en cinq sous champs qui sont β_1, β_2, β_3, β_4, β_5.

- $\beta_1 \in \{pmtn, o, split\}$
  - $\beta_1 = pmtn$: La préemption de tâche est autorisée. Une tâche peut donc être interrompue dans sa réalisation pour être reprise ensuite.
  - $\beta_1 = o$: La préemption des tâches est interdite.
  - $\beta_1 = split$: Le *splitting* est autorisé. C'est à dire que les parties d'une tâche donnée peuvent être réalisées par plusieurs machines en même temps. La vérification d'un grand nombre de dossiers peut, par exemple, être "splitter" sur plusieurs opérateurs.
- $\beta_2 \in \{Prec, Tree, o\}$
  - $\beta_2 = Prec$: Il existe des relations de précédence entre les tâches qui composent un job.
  - $\beta_2 = Tree$: Les relations de précédence entre les tâches d'un job ont une forme d'arbre.
  - $\beta_2 = o$: Il existe aucune relation de précédence entre les tâches d'un job.
- $\beta_3 \in \{r_j, o\}$
  - $\beta_3 = r_j$: Chaque tâche j possède une date de début empêchant la planification de cette dernière avant la date r_j.
  - $\beta_3 = o$: Toutes les tâches peuvent être planifiées à partir de t=0.
- $\beta_4 \in \{p_j = 1, p_{ij} = 1, o\}$
  - $\beta_4 = p_j = 1$: Chaque tâche a une durée de réalisation de 1 unité de temps. Ceci est valable uniquement si $\alpha_1 \in \{o, P, Q\}$. .
  - $\beta_4 = p_{ij} = 1$: Chaque tâche a une durée de réalisation de 1 unité de temps. Ceci est valable uniquement si $\alpha_1 \in \{O, F, J\}$.
  - $\beta_4 = o$: Toutes les tâches ont des durées de réalisation >0.
- $\beta_5 \in \{S_{nsd}, R_{nsd}, R_{sd}, b_{j,j+1}\}$
  - $\beta_5 = S_{nsd}$: Il existe un temps de montage non dépendant de la séquence.
  - $\beta_5 = R_{nsd}$: Le temps de démontage est non dépendant de la séquence.
  - $\beta_5 = R_{sd}$: Le temps de démontage est dépendant de la séquence.
  - $\beta_5 = b_{j,j+1}$: Il existe un stock de taille limité entre les machines j et $j+1$.

B.3 Critère d'optimisation γ

Le champ γ représente le critère d'optimisation. Ces critères ont été détaillés dans le paragraphe 1.1.1.4.

- $\gamma = f_{max}$: Minimiser $f_{max} \in \{C_{max}, L_{max}\}$
- $\gamma = \sum f_i$: Minimiser $\sum f_i \in \{\sum C_i, \sum T_i, \sum w_i.C_i, \sum w_i.T_i$.

Annexe C

Algorithmes

C.1 Choisir une tâche

C.1.1 Approche basée sur les tâches

Entrée : T_r : Ensemble de tâches en conflit.
Entrée : R_l : Ressource sur laquelle il y a conflit.
Sortie : T_i : La tâche choisie dans l'ensemble T_r.

Créer l'ensemble T'_r des tâches en conflit ayant le plus faible nombre d'occurrences.
Choisir T_i comme étant la tâche de T'_r ayant la priorité de réalisation la plus élevée sur R_l. ___

C.1.2 Approche par pilotage de ressource

Entrée : T_r : Ensemble de tâches réalisables.
Entrée : R_l : Ressource sur laquelle le choix doit se faire.
Sortie : T_c : La tâche choisie dans l'ensemble T_r.

Créer T'_r ensemble des tâches réalisables ayant le plus faible nombre d'occurrences.
Selon l'heuristique H_{R_l} de la ressource R_l
    SPT : Choisir T_c comme étant la tâche de T'_r ayant le temps de réalisation le plus court.
    LPT : Choisir T_c comme étant la tâche de T'_r ayant le temps de réalisation le plus long.
    ⋮
FinSelon

C.2 Initialisation des RdP

Pour tous les noeuds $n_i \in N$
  Pour toutes les contraintes linéaires $e_{j,i}$ de valeur $(p_j, \alpha, \beta, \gamma, \omega)$
    Ajouter dans la place $A_j(i)$, $(\gamma + \omega - 1)$ jetons
  FinPour
FinPour

C.3 Liste des tâches réalisables

Entrée : $Ress$: Ressource
Sortie : $L_R(Ress)$: Liste des tâches réalisables sur la ressource $Ress$

Pour toutes les tâches i ayant besoin de $Ress$
  Si la transition t_i est tirable
    Placer la tâche i dans la liste $L_R(Ress)$
  FinSi
FinPour

C.4 Création d'un jeu d'essai cyclique

Pour les n jobs
  Créer une liste de m tâches
  Créer $(m-1)$ contraintes linéaires entre les m tâches
  Créer la contrainte linéaire qui referme le graphe représentant le job
FinPour
Créer l'affectation des tâches sur les ressources

C.5 Construc_feasible_schedule_from_chromosome

Faire

Calculer le chemin critique et la date au plus tôt de début de chaque noeud (tâche) en ignorant les conflits de ressource

Si au moins 2 opérations de la même ressource ont leur intervalle de temps de réalisation en conflit

Alors Résoudre le conflit en utilisant l'information du chromosome. L'opération non prioritaire est décalée dans le temps.

FinSi

TantQu'il y a conflit entre au moins deux opérations

Annexe D

Opérateurs génétiques adaptés aux problèmes d'ordonnancement

D.1 Cross-Over

D.1.1 Order based

Cet opérateur, proposé par Syswerda [Syswerda 1989] génère les enfants en trois étapes qui sont :

1. Choisir aléatoirement des locus dans les deux parents. Le nombre de locus est choisi aléatoirement.

$$\begin{array}{lccccccccc} Parent_1 : & 1 & \overline{2} & 3 & \overline{4} & 5 & \overline{6} & 7 & 8 & \overline{9} \\ Parent_2 : & 4 & \overline{1} & 2 & \overline{8} & 7 & \overline{6} & 9 & 3 & \overline{5} \end{array}$$

2. Supprimer du $Parent_1$ les valeurs sélectionnées dans le $Parent_2$ et recopier le résultat dans l'$Enfant_1$.

$$\begin{array}{lccccccccc} Enfant_1 : & \bullet & 2 & 3 & 4 & \bullet & \bullet & 7 & \bullet & 9 \\ Enfant_2 : & \bullet & 1 & \bullet & 8 & 7 & \bullet & \bullet & 3 & 5 \end{array}$$

3. Compléter les gènes vides de l'$Enfant_1$ avec les allèles sélectionnés du $Parent_2$ dans leur ordre d'apparition.

$$\begin{array}{lccccccccc} Enfant_1 : & 1 & 2 & 3 & 4 & 8 & 6 & 7 & 5 & 9 \\ Enfant_2 : & 2 & 1 & 4 & 8 & 7 & 6 & 9 & 3 & 5 \end{array}$$

D.1.2 Position Based

Proposé par Syswerda [Syswerda 1989], cet opérateur représente une variante de l'opérateur "Order Based". Il se réalise en quatre étapes qui sont :

1. Choisir aléatoirement des locus dans les deux parents. Le nombre de locus est choisi aléatoirement.

$$\begin{array}{lccccccccc} Parent_1 : & 1 & \overline{2} & 3 & \overline{4} & 5 & \overline{6} & 7 & 8 & \overline{9} \\ Parent_2 : & 4 & \overline{1} & 2 & \overline{8} & 7 & \overline{6} & 9 & 3 & \overline{5} \end{array}$$

2. Recopier les allèles des locus sélectionnés du $Parent_2$ dans l'$Enfant_1$ (et du $Parent_1$ dans l'$Enfant_2$).

$$\begin{array}{lcccccccccc} Enfant_1 : & \bullet & 1 & \bullet & 8 & \bullet & 6 & \bullet & \bullet & 5 \\ Enfant_2 : & \bullet & 2 & \bullet & 4 & \bullet & 6 & \bullet & \bullet & 9 \end{array}$$

3. Retirer du $Parent_1$ les allèles du $Parent_2$ sélectionnés,puis retier du $Parent_2$ les allèles du $Parent_1$ sélectionnés.

$$\begin{array}{rl} Parent_1 - Parent_2 = & 2\ 3\ 4\ 7\ 9 \\ Parent_2 - Parent_1 = & 1\ 8\ 7\ 3\ 5 \end{array}$$

4. Remplir les gènes vides de l'$Enfant_1$ avec les allèles obtenus par $Parent_1 - Parent_2$ dans l'ordre de leur apparition et les gènes vides l'$Enfant_2$ avec les allèles obtenus par $Parent_2 - Parent_1$ dans l'ordre de leur apparition.

$$\begin{array}{lccccccccc} Enfant_1 : & 2 & 1 & 3 & 8 & 4 & 6 & 7 & 9 & 5 \\ Enfant_2 : & 1 & 2 & 8 & 4 & 7 & 6 & 3 & 5 & 9 \end{array}$$

D.1.3 OX : Order cross-over

Cet opérateur, proposé par Davis [Davis 1985a], permet de conserver l'ordre relatif des allèles dans le gène. Prenons pour exemple les parents $Parent_1$ et $Parent_2$ suivants, ainsi que deux points de coupures choisis de façon aléatoire :

$$\begin{array}{rl|l|l} Parent_1 & = 123 & 456 & 789 \\ Parent_2 & = 412 & 876 & 935 \end{array}$$

Pour créer les enfants $Enfant_1$ et $Enfant_2$, il faut dans un premier temps recopier le $Parent_1$ dans l'$Enfant_1$ en retirant les allèles du $Parent_2$ se situant entre les points de coupures. Ce retrait forme un trou dans le gène, noté 'T' sur l'exemple. De la même façon nous créons l'$Enfant_2$ en retirant les allèles du $Parent_1$ se situant entre les points de coupure du $Parent_2$. Ceci nous donne :

$$\begin{array}{rl|l|l} Enfant_1 & = 123 & 45T & TT9 \\ Enfant_2 & = T12 & 87T & 93T \end{array}$$

Maintenant, l'objectif est de décaler les allèles vers la gauche en partant du point de coupure droit tout en considérant le chromosome comme étant circulaire jusqu'à ce que les "trous" se trouvent entre les points de coupure. Nous obtenons ainsi :

$$\begin{array}{rl|l|l} Enfant_1 & = 345 & TTT & 912 \\ Enfant_2 & = 287 & TTT & 931 \end{array}$$

Il ne reste plus qu'à placer dans les trous de l'$Enfant_1$ les allèles se situant entre les points de coupure du $Parent_2$ pour obtenir le résultat suivant :

$$\begin{array}{rl|l|l} Enfant_1 & = 345 & 876 & 912 \\ Enfant_2 & = 287 & 456 & 931 \end{array}$$

D.1.4 LOX : Linear Order cross-over

Créé par Falkenauer et Bouffouix [Falkenauer et al. 1991], cet opérateur génétique permet de conserver l'ordre absolu des allèles dans le chromosome. Cet opérateur sera expliqué en utilisant le même exemple que pour l'opérateur OX. Comme pour ce dernier, deux points de coupure sont à déterminer de façon aléatoire. Les parents $Parent_1$ et $Parent_2$ sont recopiés dans les enfants $Enfant_1$ et $Enfant_2$. Les allèles du $Parent_2$ sélectionnés entre les points de coupure sont supprimés de l'$Enfant_1$ formant ainsi des "trous" dans les gènes de l'enfant. L'opération inverse est réalisée sur l'$Enfant_2$. Cette opération nous donne le résultat suivant :

$$\begin{array}{lc|c|c} Enfant_1 & = 123 & 45T & TT9 \\ Enfant_2 & = T12 & 87T & 93T \end{array}$$

Ensuite, dans les deux enfants, les allèles sont poussés à droite et à gauche afin d'obtenir les trous entre les points de coupure. Nous obtenons ainsi :

$$\begin{array}{lc|c|c} Enfant_1 & = 123 & TTT & 459 \\ Enfant_2 & = 128 & TTT & 793 \end{array}$$

L'$Enfant_1$ est ensuite complété avec les allèles sélectionnés entre les points de coupure du $Parent_2$, et l'$Enfant_2$ avec les allèles du $Parent_1$. Le résultat final est donc :

$$\begin{array}{lc|c|c} Enfant_1 & = 123 & 876 & 459 \\ Enfant_2 & = 128 & 456 & 793 \end{array}$$

D.1.5 CX : Cycle cross-over

Cet opérateur génétique permet de conserver l'ordre absolu des allèles dans le chromosome [Oliver et al. 1987]. L'explication de la construction se fera uniquement sur l'$Enfant_1$. Dans un premier temps, il faut recopier la première valeur du $Parent_1$ dans l'$Enfant_1$,

$$\begin{array}{lccccccccc} Parent_1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ Parent_2: & 4 & 1 & 2 & 8 & 7 & 6 & 9 & 3 & 5 \\ Enfant_1: & 1 & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

puis chercher dans le $Parent_2$ l'allèle qui se trouve au locus 1 (4 pour cet exemple). Cet allèle sera recopié au locus de l'allèle 4 du $Parent_1$.

$$Enfant_1: \quad 1 \quad \bullet \quad \bullet \quad 4 \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet$$

Ensuite, l'allèle se situant au même locus dans le $Parent_2$ que l'allèle précédement placé dans l'$Enfant_1$ détermine le prochain allèle et locus de l'$Enfant_1$. Sur l'exemple utilisé l'allèle 4 vient d'être placé dans l'$Enfant_1$ au locus 4. L'allèle 8 se situe au même locus dans le $Parent_2$. Cet allèle est donc ajouté à l'$Enfant_1$ au locus 8 car l'allèle 8 se situe au locus 8 dans le $Parent_1$.

$$Enfant_1: \quad 1 \quad \bullet \quad \bullet \quad 4 \quad \bullet \quad \bullet \quad \bullet \quad 8 \quad \bullet$$

Ceci est répété jusqu'à ce que l'allèle recopié de cette manière ne soit pas égal à l'allèle se situant au locus 1 de l'$Enfant_1$. Ceci donne les étapes suivantes :

$$\begin{array}{lccccccccc} Enfant_1: & 1 & \bullet & 3 & 4 & \bullet & \bullet & \bullet & 8 & \bullet \\ Enfant_1: & 1 & 2 & 3 & 4 & \bullet & \bullet & \bullet & 8 & \bullet \end{array}$$

Une fois le cycle terminé, il faut remplir les gènes vides de l'$Enfant_1$ avec les allèles non encore utilisés du $Parent_2$ dans l'ordre de leur apparition dans le $Parent_2$. Ceci donne le résultat final suivant :

$$Enfant_1 : \quad 1 \quad 2 \quad 3 \quad 4 \quad 7 \quad 6 \quad 9 \quad 8 \quad 5$$

D.1.6 GOX : Generalization of OX

Ici, aucune contrainte de précédences n'existe entre les différents allèles. Il est possible que les allèles contenus dans le chromosome soient représentés plusieurs fois. L'indice d'un allèle représente le nombre d'occurrences de cet allèle déjà présentes dans le chromosome à des locus inférieurs. Bierwirth propose une généralisation de l'opérateur génétique OX [Bierwirth 1995] dans le but de résoudre des problèmes de Job-Shop codés sur le principe de la "permutation avec répétition". Le principe de fonctionnement de cet opérateur génétique est le suivant :

1. Recopier le $Parent_i$ dans l'$Enfant_i$.

$$\begin{array}{rcccccccccc} Parent_1 : & B & A & B & B & C & A & C & C & B & A \\ Parent_2 : & A & B & B & A & C & A & B & C & B & C \end{array}$$

2. Déterminer aléatoirement dans le $Parent_2$ un sous-ensemble $\mathcal{S}$.

$$\begin{array}{rcccccccccc} Parent_2 : & A & B & B & A & C & A & B & C & B & C \\ \mathcal{S} : & & & & A & C & A & B & & & \end{array}$$

3. Calculer les indices d'apparition des valeurs contenues dans $\mathcal{S}$.

$$\begin{array}{rcccc} \mathcal{S} : & A & C & A & B \\ Indice : & 2 & 1 & 3 & 3 \end{array}$$

4. Insérer les allèles contenus dans $\mathcal{S}$ dans l'$Enfant_1$ après le locus où se trouve le même allèle (et son indice) que le premier allèle (et son indice) de $\mathcal{S}$.

$$\begin{array}{rcccccccccc} Enfant_1 : & B & A & B & B & C & A & C & C & B & A \\ Indice : & 1 & 1 & 2 & 3 & 1 & 2 & 2 & 3 & 4 & 3 \\ Lieu\,d'insertion : & & & & & & \bullet & & & & \end{array}$$

$$\begin{array}{rcccccccccccccc} Enfant_1 : & B & A & B & B & C & A & A & C & A & B & C & C & B & A \\ Indice : & 1 & 1 & 2 & 3 & 1 & 2 & 2 & 1 & 3 & 3 & 2 & 3 & 4 & 3 \\ & & & & & & & \wedge & \wedge & \wedge & \wedge & & & & \end{array}$$

5. Détruire dans l'$Enfant_1$ les tuples {allèle-indice} qui correspondent à ceux de $\mathcal{S}$.

$$\begin{array}{rcccccccccccccc} Enfant_1 : & B & A & B & B & C & A & A & C & A & B & C & C & B & A \\ Indice : & 1 & 1 & 2 & 3 & 1 & 2 & 2 & 1 & 3 & 3 & 2 & 3 & 4 & 3 \\ Detruire : & & & & \uparrow & \uparrow & \uparrow & & & & & & & & \uparrow \end{array}$$

$$\begin{array}{rcccccccccc} Enfant_1 : & & & B & A & B & A & C & A & B & C & C & B \end{array}$$

Annexe E

Description du simulateur à évènements discrets

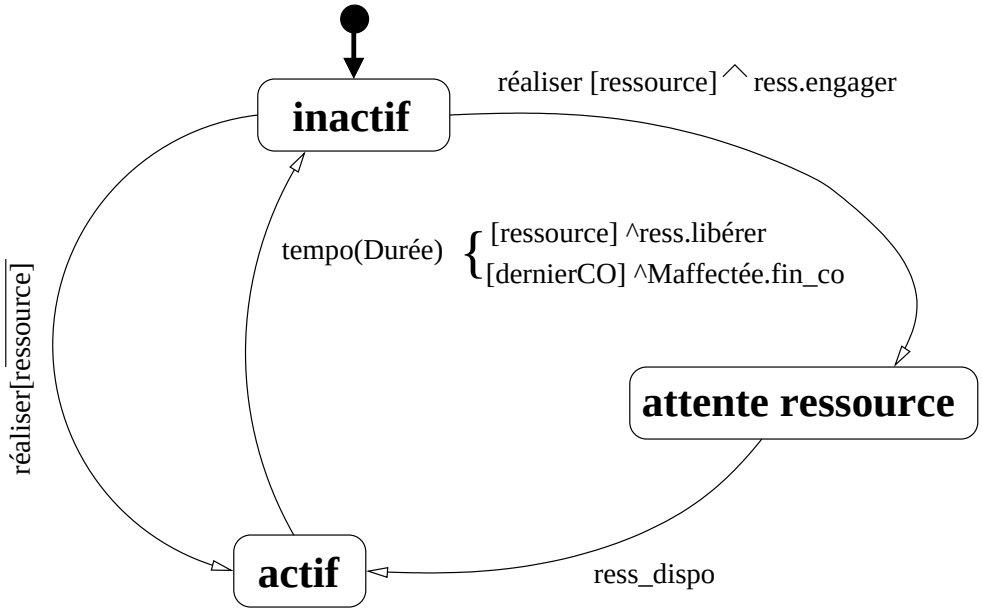

FIG. E.1 – Diagramme état transition d'un changement d'outils

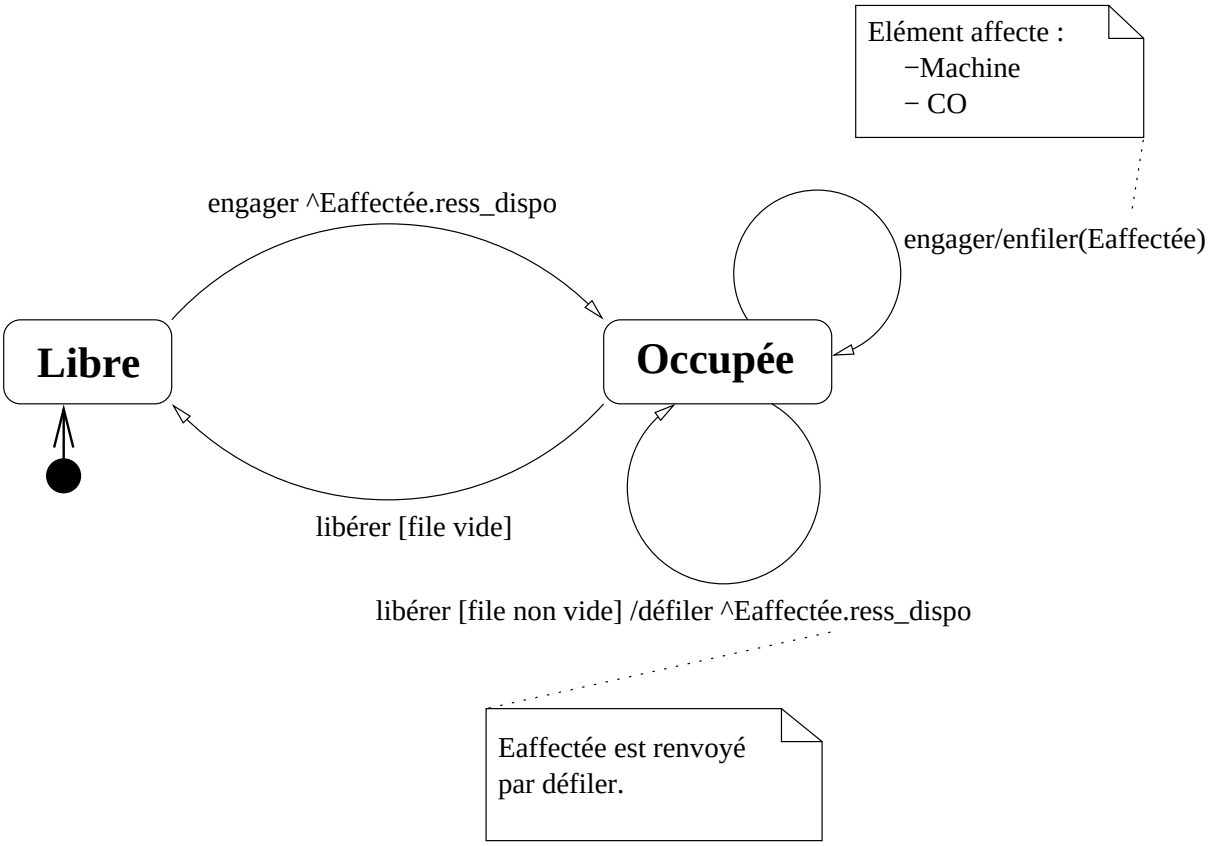

FIG. E.2 – Diagramme état transition d'une ressource

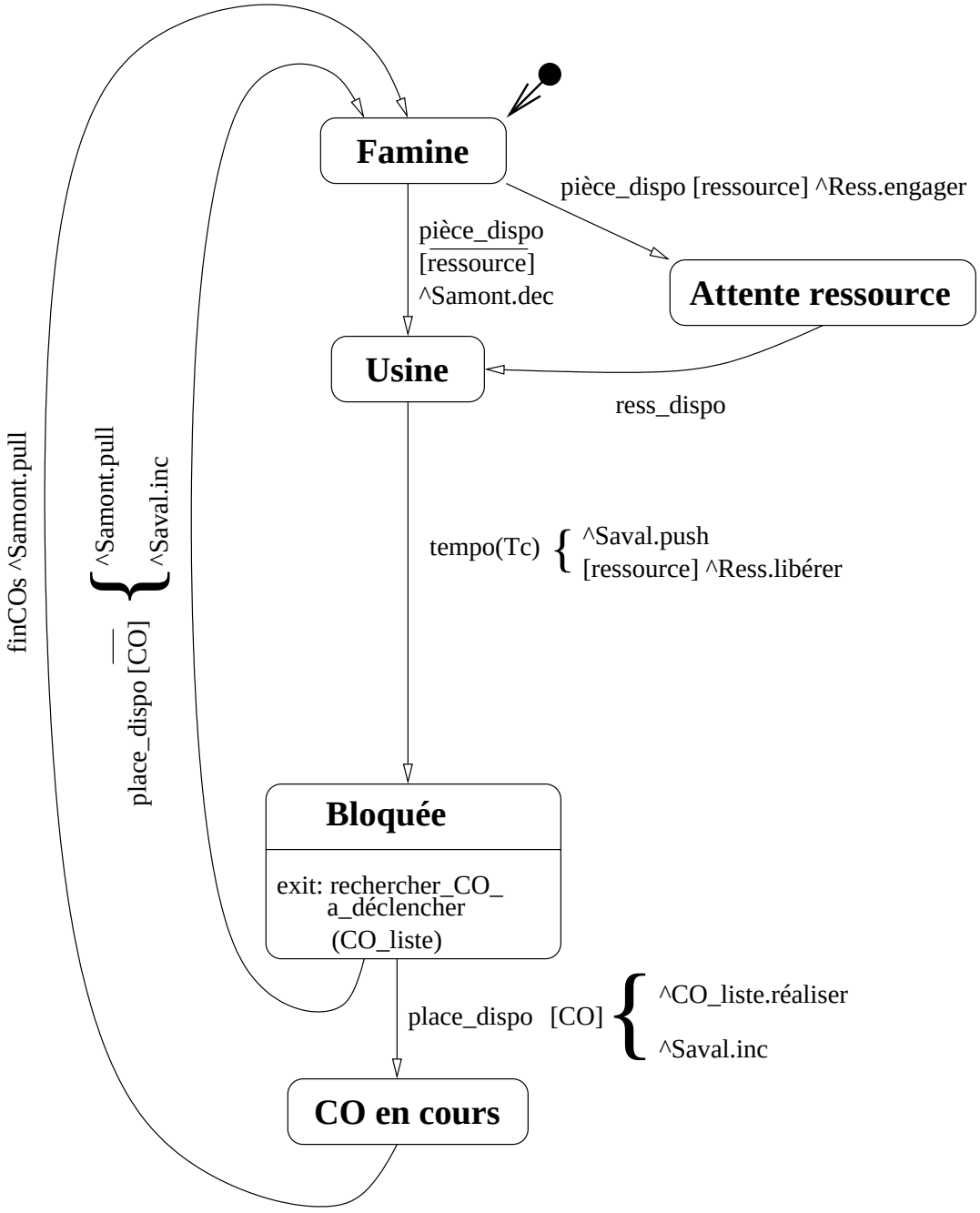

FIG. E.3 – Diagramme état transition d'une machine

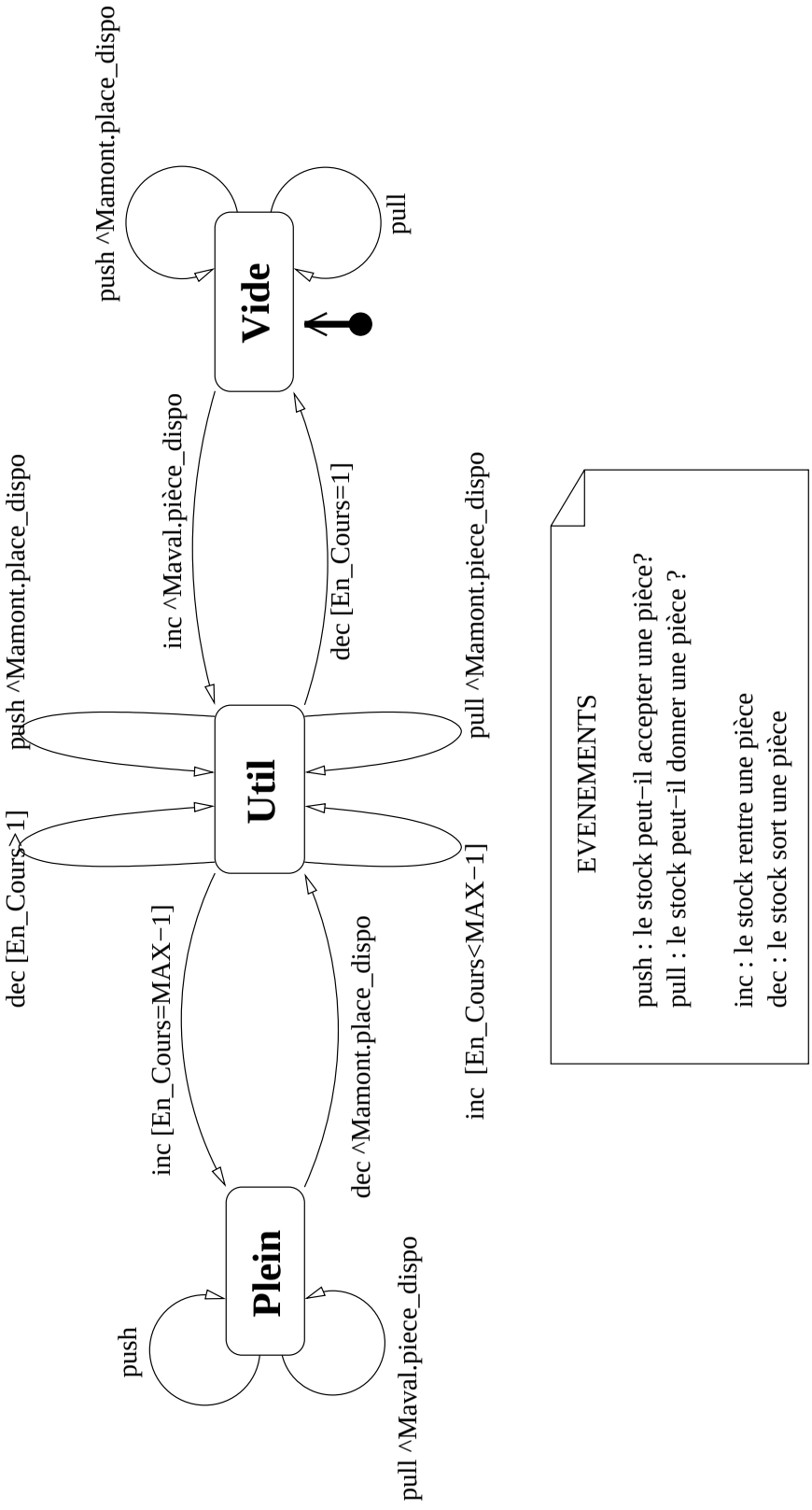

FIG. E.4 – Diagramme état transition d'un stock

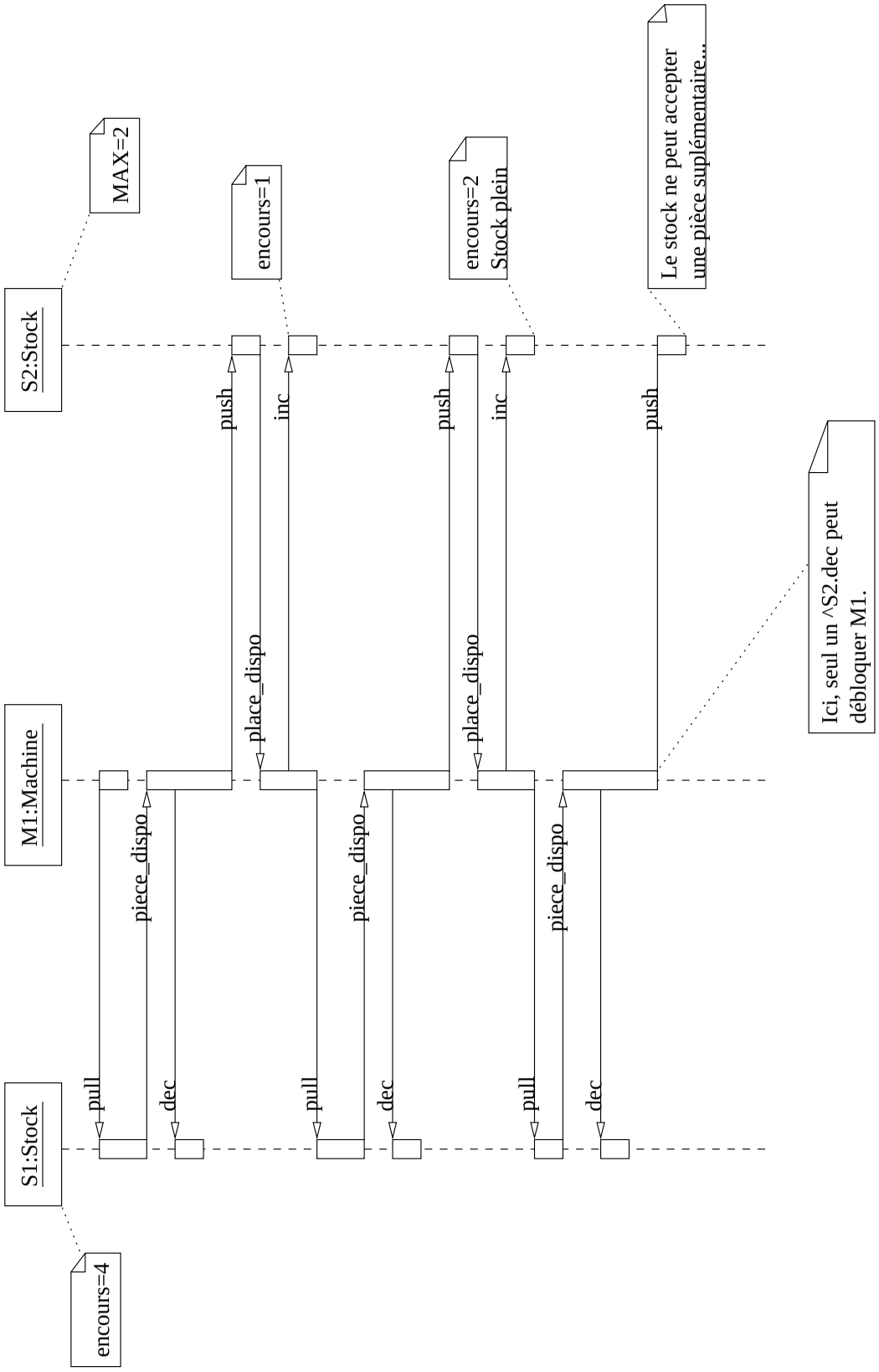

FIG. E.5 – Diagramme de séquences représentant la synchronisation entre une machine et deux stocks

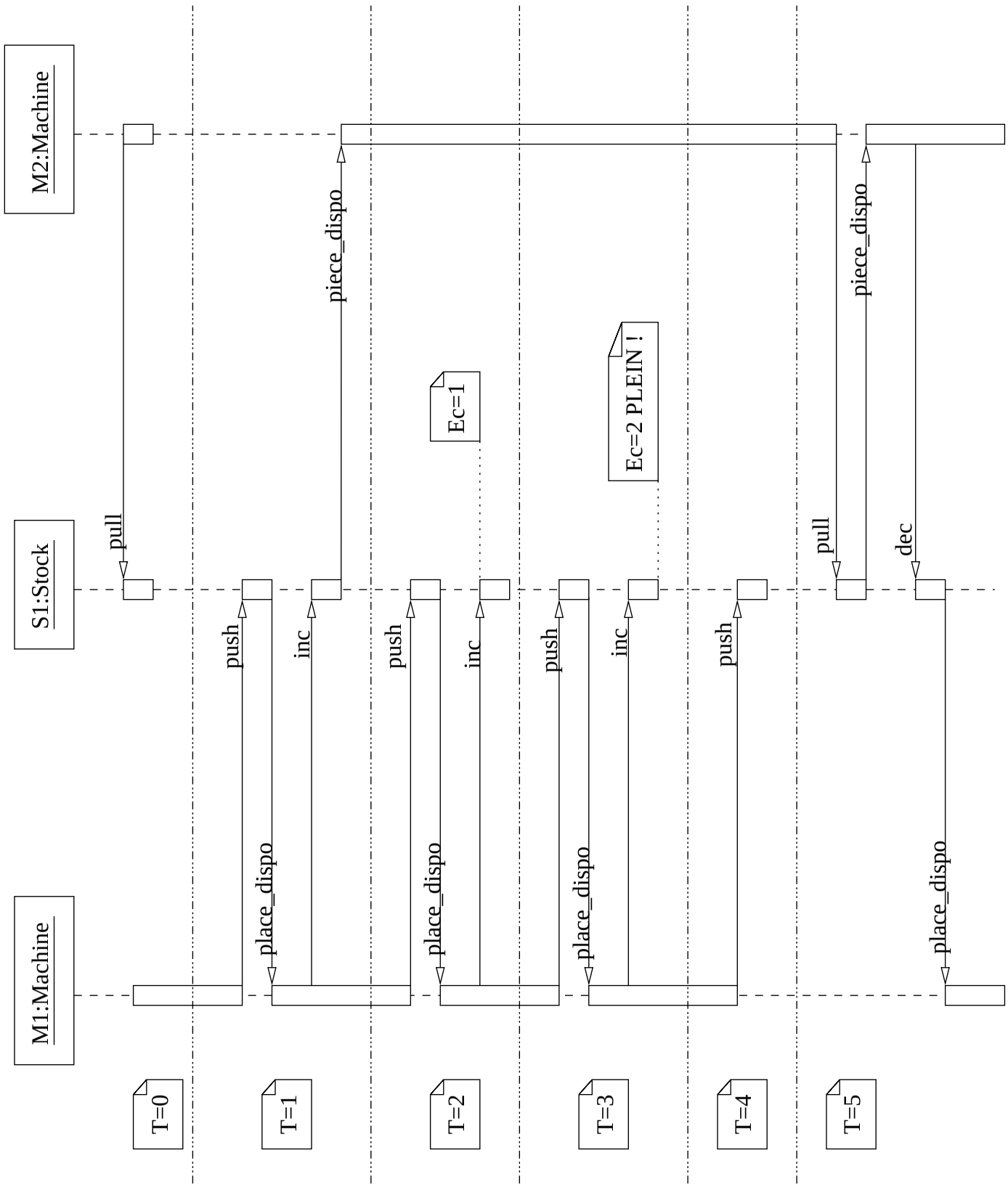

FIG. E.6 – Diagramme de séquences représentant la synchronisation entre deux machines et un stock

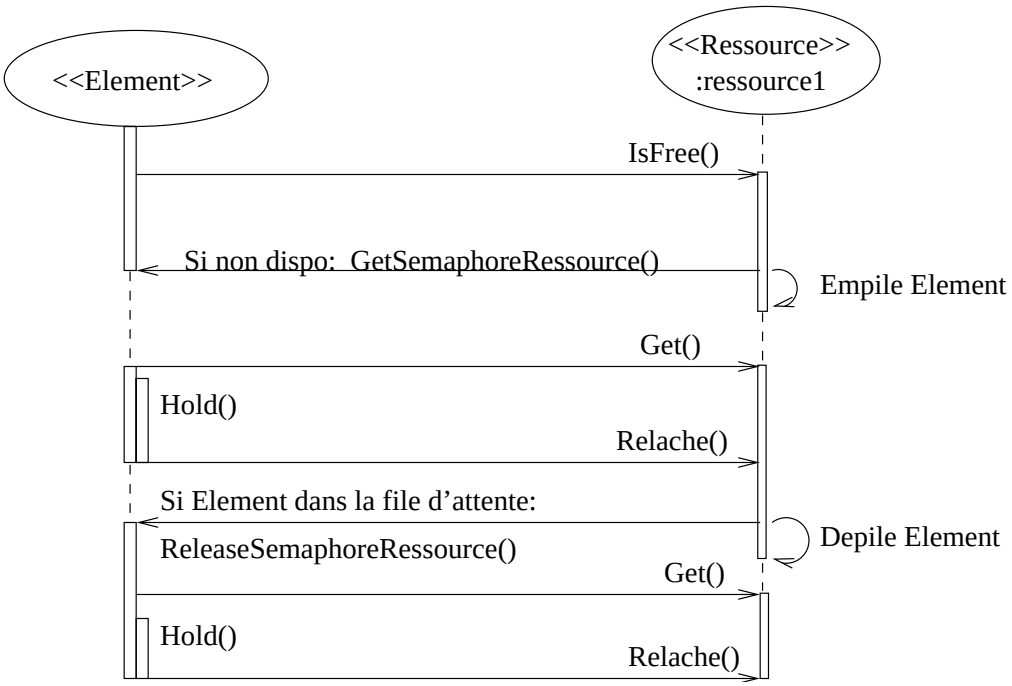

FIG. E.7 – Diagramme de séquences représentant la capture d'une ressource par un objet

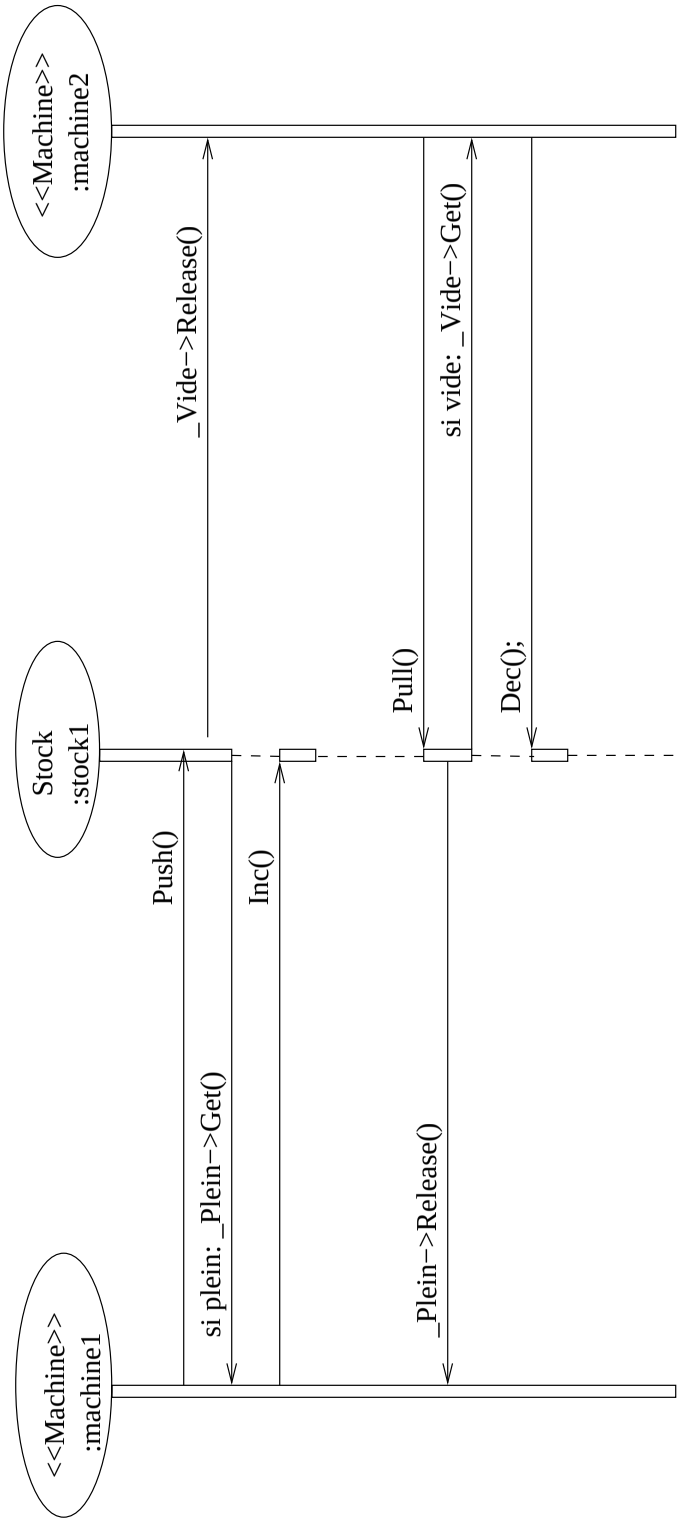

FIG. E.8 – Diagramme de séquences représentant l'utilisation d'un stock par deux machines

FIG. E.9 – Scénarios montrant l'utilisation d'un stock par deux machines

Hypothèses de départ :

Scénario 1 : Le stock S1 est vide, la machine M2 tente de prendre 1 pièce dans le stock S1 et se retrouve bloquée tant que la machine M1 ne dépose pas de pièce dans S1.

Scénario 2 : Le stock S1 est plein, la machine M1 tente de déposer 1 pièce dans le stock S1 et se retrouve bloquée tant que la machine M2 ne prend pas de pièce dans S1.

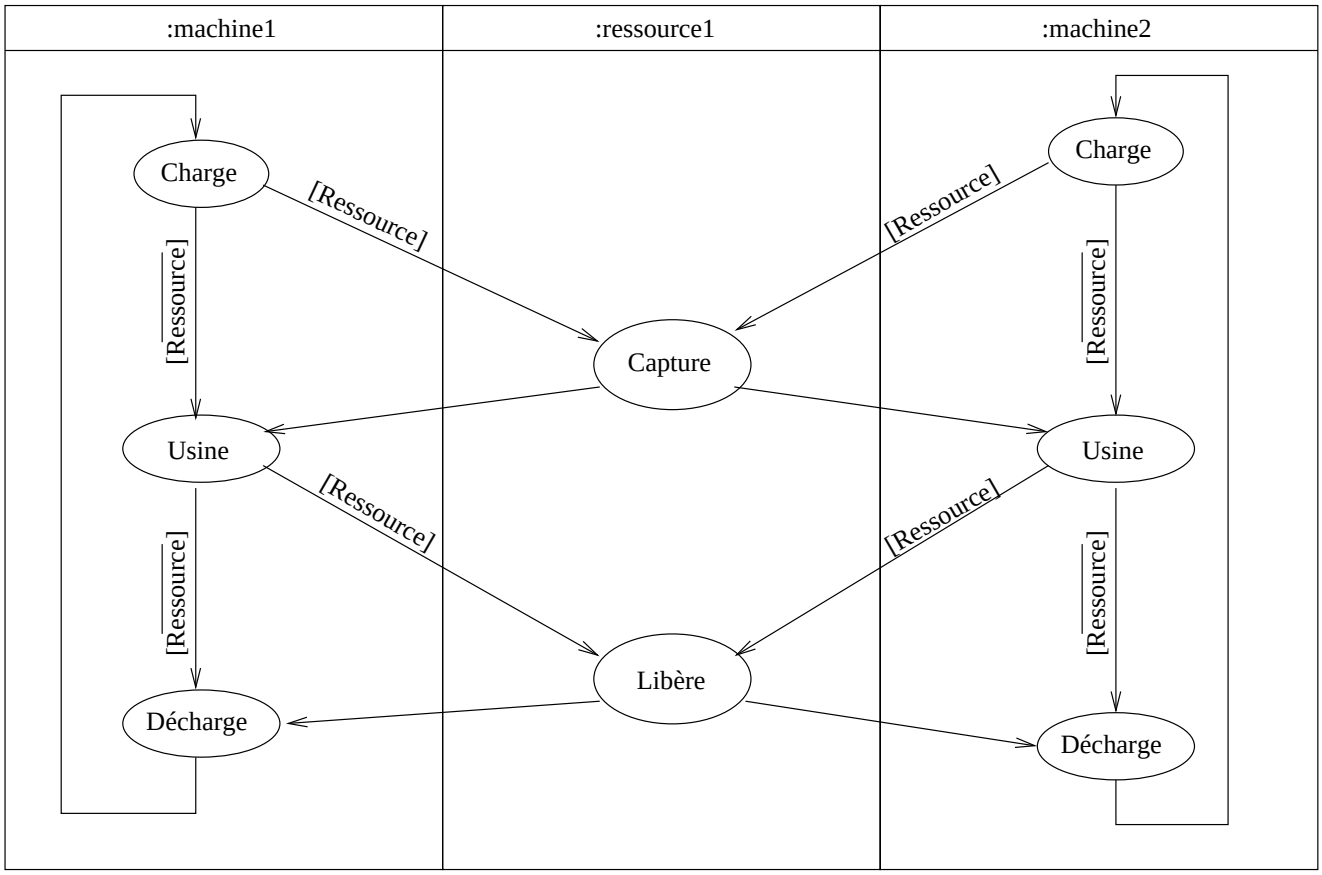

FIG. E.10 – Diagramme d'activité présentant le principe de partage de ressource entre deux machines

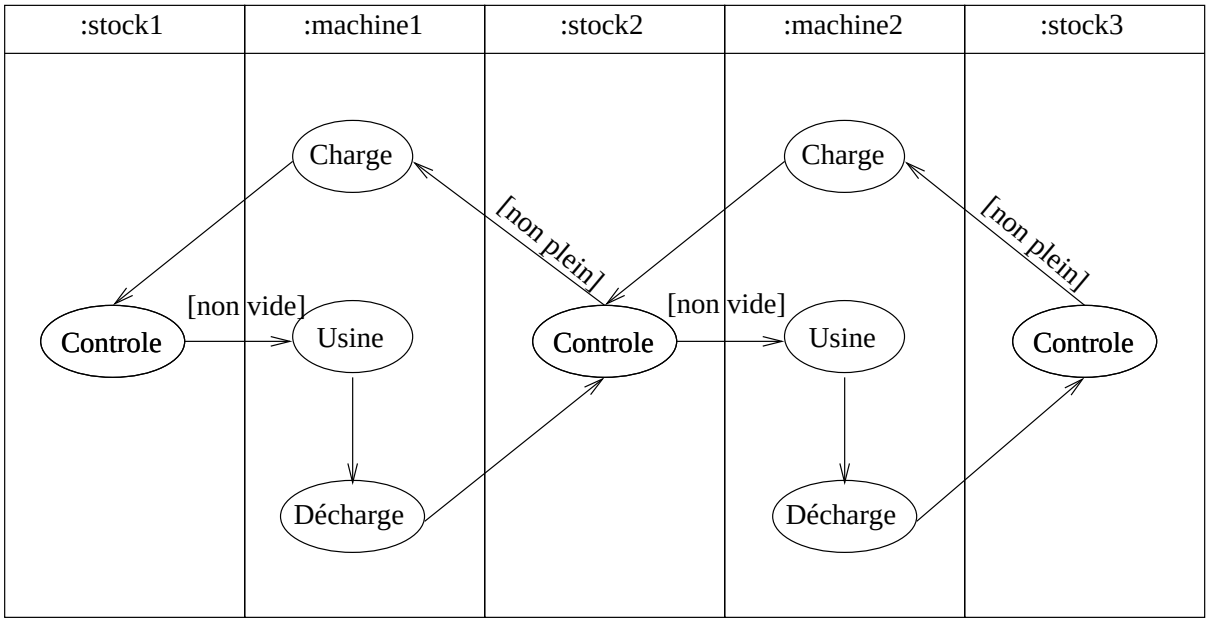

FIG. E.11 – Diagramme d'activités présentant le fonctionnement de la ligne de fabrication

Bibliographie

[arjuna 1994] arjuna . Arjuna project : C++SIM. Dept of computing science, 1994.

[Axsater et al. 1984] S. Axsater and H. Jonsson . « *Aggregation and Dissaggregation in Hierachical Production Planning* », Chapitre 17, pages 338–350. EJOR, 1984.

[Baker 1974] K.R. Baker . *Introduction to sequencing and scheduling.* Harvard University, 1974.

[Beasley 1999] J.E. Beasley . Population heuristics. In Imperial college , editor, *The management school.* London, 1999.

[Bellman et al. 1982] R. Bellman , A.O. Esogbue , and I. Nabeshima . « Mathematical aspects of scheduling and applications ». *International Series in Modern Applied Mathematics and Computer Science*, 4, 1982. Pergamon Press Edition.

[Bierwirth et al. 1999] C. Bierwirth and D.C. Mattfeld . « Production Scheduling and Rescheduling with Genetic Algorithms ». *Evolutionary Computation*, 7(1) :1–17, Spring 1999.

[Bierwirth 1995] C. Bierwirth . « A generalized permutation approach to job shop scheduling with genetic algorithms ». *ORspektrum*, pages 87–92, 1995.

[Binder 1978] K. Binder . *Monte Carlo Methods in statistical physics.* Springer, 1978.

[Bitran et al. 1982] G.R. Bitran , E.A. Haas , and A.C. Hax . « Hierarchical Production Planning : a two stage system ». *Opns. Res.*, 30 :232–251, 1982.

[Brucker et al. 1997] P. Brucker , J. Hurink , and F. Werner . « Improving local search heuristics for some scheduling problems part II ». *Discrete Applied Mathematics*, 72 :47–69, 1997.

[Brucker 1995] P. Brucker . *Scheduling Algorithms.* Berlin Springer-Verlag, 1995.

[Carlier et al. 1982] J. Carlier and A.H.G. Rinnooy Kan . « Financing and Scheduling ». *Opns. Res. Letters*, 1 :52–55, avril 1982.

[Carlier et al. 1988] J. Carlier and P. Chretienne . *Problèmes d'ordonnancement (modélisation / complexité / algorithmes).* Masson Edition, 1988.

[Carlier et al. 1989a] J. Carlier and P. Chretienne . « Timed Petri Nets Schedules ». *Advances in Petri Nets*, pages 62–84, 1989. Spinger Verlag.

[Carlier et al. 1989b] J. Carlier and E. Pinson . « A Branch and Bound method for solving the job shop problem ». *Management Sci.*, pages 164–176, Fevrier 1989.

[Carlier 1982] J. Carlier . « The one machine problem ». *EJOR*, pages 42–47, 1982.

[Carlier 1989] J. Carlier . « Scheduling under Finacial Constraints ». *Elsevier Science*, pages 187–224, April 1989. Advances in Project Scheduling (R. Slowinsky et J. Weglarz eds.).

[Cavory et al. 1999] G. Cavory and R. Dupas . « Une plate-forme d'évaluation et d'amélioration des performances des systèmes de production manufacturiers ». In *Modelisation et simulation des flux physiques et informationnels : deuxième conference francophone de Modelisation et Simulation :MOSIM'99*, Annecy (France), Octobre 1999.

[Cavory et al. 2000a] G. Cavory , R. Dupas , D. Fourmaux , and G. Goncalves . « Une approche génétique pour l'ordonnancement de tâches cycliques de maintenance ». In *Troisième Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision. ROADEF 2000*, Nantes, janvier 2000.

[Cavory et al. 2000b] G. Cavory , J.M. Frayret , and S.D'Amours . « Conception des réseaux logistiques : mesure de fiabilité et planification d'accélérateurs ». In *RIRL 2000*, Trois-Rivières - Canada, 2000.

[Cavory 2000] G. Cavory . « Ordonnancement de tâches de maintenance par les algorithmes génétiques ». In *CNRIUT2000 : Colloque National de la Recherche dans les IUT*, volume 1, pages 241–248, Bourges, 2000.

[Cheng et al. 1999] R. Cheng , M. Gen , and Y Tsujimura . « A tutorial survey of job-shop scheduling problems using genetic algorithms, part II : hybrid genetic search strategies ». *Computer and industrial engineering*, pages 343–364, 1999.

[Chu et al. 1989] C. Chu and M.C. Portman . « Minimisation de la somme des retards pour les problèmes d'ordonnancement à une machine ». Technical Report 1023, INRIA, avril 1989.

[Chu et al. 1992] C. Chu and M.C. Portman . « Some new efficient methods to solve the $n, 1 \mid r_i \mid \sum T_i$ scheduling problems ». *EJOR*, 1992.

[Corcoran 1993] A. Corcoran . « LibGA : a user-friendly workbench for order-based genetic algorithm research ». In *Proc. of the 1993 ACM/SIGAPP symposium on applied*, 1993.

[Dannenbring 1977] D.G. Dannenbring . A Evaluation of Flow-Shop Sequencing Heuristics, 1977. Management Sci.

[Davis 1985a] L. Davis . « Applying adaptative algorithms to epistatic domains ». In *Proceeding of the Int. Joint Conference on Artificial Intelligence*, 1985.

[Davis 1985b] L. Davis . « Job Shop Scheduling with genetic algorithms ». In Lawrence Erlbaum Associates , editor, *Proc of 1st Int Conf on Genetic Algorithm*, pages 136–140, Hillsdale, 1985.

[Draper et al. 1999] D. Draper , A.K. Jonsson , D.P. Clements , and D.E. Joslin . « Cyclic scheduling ». In *IJCAI99*, 1999.

[Dupas et al. 1997a] R. Dupas , D. Fourmaux , and G. Goncalves . « Optimizing machine stoppages by a genetic algorithm ». In *7th IFAC on Computer Aided Control Systems Design*, Gent - Belgique, 1997.

[Dupas et al. 1997b] R. Dupas , D. Fourmaux , and G. Goncalves . « Optimizing machine stoppages by a genetic algorithm in the context of multiple machines assigned to one operator ». In *15th IMACS on Scientific Computation, Applied Mathematics and Simulation - Special session on Evolutionary Computation*, Berlin - Allemagne, 1997.

[Dupas et al. 1998a] R. Dupas , D. Fourmaux , and G. Goncalves . « Toward a methodology for optimising a production line ». In *Computer Science in Management. Tempus Project S_ JEP-09139-95*, Krakow - Poland, 1998.

[Dupas et al. 1998b] R. Dupas , D. Fourmaux , and G. Goncalves . « Vers une méthodologie pour l'optimisation ». In *CNR'IUT 98. Colloque national de recherche universitaire ; exemple des IUT*, Fontainebleau - France, 1998.

[Dupas et al. 1998c] R. Dupas , D. Fourmaux , and G. Goncalves . « Vers une méthodologie pour l'optimisation des lignes de production ». 1998.

[Dupas et al. 1999] R. Dupas , G. Cavory , and G. Goncalves . « Scheduling the changes of tools job on a manufacturing production line with a genetic algorithm ». In *International Conference on Industrial Engineering and Production Management, IEPM99*, Glasgow, 1999.

[Dupas et al. 2000a] R. Dupas , G. Cavory , and G. Goncalves . « A genetic approach to the scheduling of preventive maintenance tasks on a manufacturing production line ». *Journal of Production Economics 2000*, 2000. Elsevier.

[Dupas et al. 2000b] R. Dupas , G. Cavory , M.T Kechadi , and G. Goncalves . « A genetic approach to an industrial cyclic scheduling problem ». In *Workshop on Production Planning and Control : WPPC' 2000*, Mons - Belgium, 2000.

[Eiben et al. 1995] A.E. Eiben , C.H.M. van Kemenade , and J.N. Kok . « Orgy in the computer : Multi-parent reproduction in genetic algorithm ». In *ECAL'95*, 1995.

[Eiben 1998] A.E. Eiben . « Parameters control in EA's ». Evonet Summer School on Evolutionary Computation, 1998.

[Esquirol et al. 1999] P. Esquirol and P. Lopez . *L'ordonnancement*. 1999. Economica.

[Falkenauer et al. 1991] E. Falkenauer and S.Bouffouix . « A genetic algorithm for job-shop ». In *the 1991 IEEE International Conference on Robotics and Automation*, pages 824–829, Sacramento, California, 1991.

[Fishwick 1995] P.A. Fishwick . *Simulation model design and execution*. prentice hall, 1995.

[Fourmaux et al. 1998] D. Fourmaux , R. Dupas , and G. Goncalves . Analyse de performance d'une ligne de production : témoignage de la réalité industrielle. 1998.

[Garey et al. 1979] M. Garey and D. Johnson . *Computers and Intractability : A guide to the théory of NP-completeness*. New-York, 1979.

[Ghedjati 97] F. Ghedjati . Genetic Algorithms for the Job-Shop Problem with Parallel Machines and Precedence Constraints : Heuristic Mixing Method. 97.

[Giffler et al. 1959] B. Giffler and G.L. Thompson . « Algorithms for solving production-scheduling problems ». *International Buisiness Machines Corporation*, 1959.

[Glover 1975] F. Glover . « Surrogate Constraint Duality in Mathematical Programming ». *Opns. Res.*, 23 :434–451, 1975.

[Glover 1986] F. Glover . « Future paths for integer programming and links to artificial intelligence ». *Computer and Operations Research*, 13 :533–549, 1986.

[Glover 1989] F. Glover . « Tabu Search : Part 1 ». *ORSA Journal on Computing*, 1 :190–206, 1989.

[Glover 1990] F. Glover . « Tabu Search : Part 2 ». *ORSA Journal on Computing*, 1 :4–32, 1990.

[Goldberg 1989] Goldberg . *Genetic algorithms in search, Optmisation and Machine Learning*. 1989.

[Gonzales et al. 1976] T. Gonzales and S. Sahni . « Open-Shop Scheduling to Minimize Finish Time ». *J.A.C.M.*, 23(4) :665–679, 1976.

[Gonzalez et al. 1978] T. Gonzalez and S.Sahni . Flow-Shop Job-Shop Schedules : Complexity and Approximation. In *Opns. Res.*, pages 36–52. 1978.

[GOTHA 1993] GOTHA . « Les problèmes d'ordonnancement ». *RAIRO-Recherche Opérationnelle*, 27 :77–150, 1993.

[Grabowski et al. 1997] J. Grabowski and J. Pempera . « Scheduling a constrained flow shop using tabu search ». In *International conference on industrial engeneering and production management*, pages 76–84, 1997.

[Graham et al. 1979] R.L. Graham , E.L. Lawer , J.K. Lenstra , and A.H.G Rinnooy . *Optimisation and approximation in deterministic sequencing and scheduling : a survey*, volume 5. Annals Disc. Math., 1979.

[Hanen et al. 1993] C. Hanen and A. Munier . « Cyclic Scheduling on parallel processors : an overview ». *Scheduling Theory and its Application*, 1993.

[Hanen et al. 1995] C. Hanen and A. Munier . *A study of the cyclic scheduling problem on parallel processors*. 1995. to appear in Discrete Applied Mathematics.

[Hanen 1994] C. Hanen . « Study of a NP-Hard cyclic scheduling problem : The recurrent job-shop. ». *European Journal of Operational Research*, pages 82–101, 1994.

[Hart et al. 1999] E. Hart and P. Ross . A heuristic Combination Method for Solving Job-Shop Scheduling Problems. pages 845–854. 1999.

[Hertz et al. 1990] A. Hertz and D. DE Werra . « The Tabu Search Metaheuristic : How to Used It ». *Ann. of Math and Artificial Intelligence*, 1 :111–121, 1990.

[Holland 1975] J.H. Holland . *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Jackson 1955] J.R. Jackson . « Scheduling a Production Line Minimize Maximum Tardiness ». Technical Report, University of California, Los Angeles, 1955.

[Jain et al. 1999] A.S. Jain and S. Meeran . « Deterministic job-shop scheduling : past, present and future ». *EJOR*, pages 390–434, 1999.

[Jonhson 1954] S.M. Jonhson . Optimal Two and Three Stage Production Schedules with Setup Times Included. pages 61–68. 1954.

[Jr 1976] E.G. Coffman Jr . *Computer and job-shop scheduling*, volume 299. New-York, 1976.

[Kan 1976] A.H.G. Rinnooy Kan . *Machine sequencing problems : classification, complexity and computation.* The Hague, 1976.

[Kirkpatrick et al. 1983] S. Kirkpatrick , C.D. Gelatt , and M.P. Vecchi . *Opimization by Simulated Annealing.* Res. Rep. R.C. 9335, 1983.

[Kobayashi et al. 1995] S. Kobayashi , I. Ono , and M. Yamamura . « An efficient genetic algorithm for job-shop scheduling problems ». In *6-th Int. Conf. on Genetic Algorithms*, pages 506–511, 1995.

[Lai 1997] M. Lai . *UML : La notation unifié de modélisation objet.* InterEditions, 1997.

[Lawler et al. 1985] E.L. Lawler , J.K. Lenstra , A.Rinnoy Kan , and D.B. Shmoys . « Sequencing and scheduling : Algorithms and complexity ». Technical Report, Center for Mathematics and Computer Science, Amsterdam, 1985.

[Lawrence 1984] S. Lawrence . Supplement to ressource constrained project scheduling : An experimental investigation of heuristic scheduling techniques. Graduate school of industrial administration, Canergie-Mellon University, Pittsburgh, PA, 1984.

[Lee et al. 1996] K.M. Lee and T. Yamakawa . « A genetic algorithm for general machine scheduling problems ». In *Int. Conf. on Conventional and Knowledge-based Electronics Systems*, volume 2, pages 60–66, Australia, 1996.

[Lee et al. 2000] K.M Lee and T. Yamakawa . « Genetic Algorithm approaches to job-shop scheduling problems : An overview ». *Int. Journal of Knowledge-Based intelligent engineering systems*, pages 72–85, Avril 2000.

[Matsuo et al. 1998] H. Matsuo , C.J. Suh , and R.S. Sullivan . « A controlled search simulated annealing method for the general job-shop scheduling problem ». *Working paper*, 03-04-88, 1998. Graduate School of Business.

[Mattfeld 1995] D.C. Mattfeld . *Evolutionary Search and the Job-Shop.* Springer-Verlag, 1995.

[Metropolis et al. 1953] N. Metropolis , A. Rosenbluth , M. Rosenbluth , A. Teller , and E. Teller . « Equation of state calculations by fast computing machines ». *journal of chemical physic*, 21 :1087–1092, 1953.

[Michalewicz 1996] Z. Michalewicz . *Genetic Algorithm + Data Structures = Evolution Programs.* 1996.

[Moursli 1997] O. Moursli . « Branch bound algorithm for the hybrid flow shop ». In *International conference on industrial engeneering and production management*, pages 179–188, 1997.

[Munier 1996] A. Munier . « The Basic Cyclic Scheduling Problem with Linear Precedence Constraints ». *Discrete Applied Mathematics*, pages 219–238, 1996.

[Nakano et al. 1991] R. Nakano and T. Yamada . « Conventional Genetic Algorithm for Job Shop Problems ». In *4-th Int. Conf. on Genetic Algorithms*, pages 474–479, 1991.

[Nawaz et al. 1983] M. Nawaz , E. Enscore , and I.Ham . *A Heuristic Algorithm for the m machine, n Job FlowShop Sequence Problem.* Omega, 1983.

[Norenkov et al. 1999] I. Norenkov and E. Goodman . Solving Scheduling Problems via Evolutionary Methods for Rule Sequence Optimization. 1999.

[Oliver et al. 1987] I. Oliver , D. Smith , and J. Holland . « A study of permutation crossover operators on the TSP ». In Lawrence Erlbaum Associates , editor, *Proc of 2nd Int Conf on Genetic Algorithm*, pages 224–230, 1987.

[Pierreval 1997] H. Pierreval . « Evolutionary optimization of a generic model to select a pull control strategy ». In *International conference on industrial engeneering and production management*, pages 541–550, 1997.

[Pillet 1992] M. Pillet . *Introduction aux plans d'expériences par la méthode Tagushi.* Les Editions d'organisation, 1992.

[Portmann 1988] M.CM Portmann . « *Méthodes de décomposition spatiales et temporelles en ordonnancement de la production* », Chapitre 22,5, pages 439–451. RAIRO-APII, 1988.

[Portmann 1997] M.C. Portmann . *Scheduling methodology : optimization and compusearch approachs.* Chapman et Hall edition, 1997. dans the planning and scheduling of production system.

[Proust 1992] C. Proust . « Influences des idées de S.M. Jonhson sur la résolution de problème d'ordonnancement de type n/M/F, contraintes diverses/Cmax ». Technical Report, Laboratoire d'informatique E3i, Université de Tours, 1992.

[Ramchandani 1973] C. Ramchandani . « *Analysis of asynchronous systems by timed Petri nets* ». PhD thesis, MIT, 1973.

[Roy 1970] B. Roy . *Algèbre moderne et théorie des graphes*, volume 2. Dunod, 1970.

[Sakanovitch 1984] M. Sakanovitch . *Optimisation Combinatoire : Programmation discrete.* Herman, 1984.

[Sebag et al. 1996] Michèle Sebag and Marc Schoenauer . « Contrôle d'un Alrithme génétique ». *Revue d'intelligence artificiell*, pages 289–428, 1996.

[Slowinski 1978] R. Slowinski . « Scheduling preemptable tasks on unrelated processors with additional ressources to minimize schedule lenght ». *Computer science*, pages 536–66, 1978.

[Slowinski 1982] R. Slowinski . « Multiobjective Network Scheduling with Efficient Use of Renewable and Non-Renewable Ressources ». *EJOR*, 7(3) :265–273, 1982.

[Smith 1956] W.E. Smith . *Various Optimizers for Single-Stage Production.* Nav. Res. Log. Quart., 1956.

[Syswerda 1989] G. Syswerda . « Uniform cross-over in genetic algorithms ». In *Third Int. Conf. on Genetic Algorithms*, pages 2–9, San Manteo, 1989. Morgan Kauffmann.

[Taillard 1994] E. Taillard . « Parallel taboo search techniques for the job-shop scheduling problem ». *ORSA journal of computing*, 16(2) :108–117, 1994.

[Tamaki et al. 1992] H. Tamaki and Y. Nishikawa . « Maintenance of diversity in a genetic algorithm and an application to the jobshop scheduling ». In *IMACS/SICE Int. Symp. on RM*^{2}, pages 869–874, 1992.

[Venturini 1995] Gilles Venturini . « Algorithmes génétiques et apprentissage ». *Revue d'intelligence artificielle*, 10 :345–387, 1995.

[Wera et al. 1989] D. de Wera and A. Hertz . « Tabu search techniques : a tutorial and an application to algorithms ». *Computers & Operations Research*, 20 :707–722, 1989.

[Xuong 1992] N. Xuong . *Mathématiques discrètes et informatique*. Masson, 1992.

[Yamada et al. 1992] T. Yamada and R. Nakano . « A Genetic Algorithm Applicable to Large-scale Job-Shop Problems ». In *5-th Conf. on Parallel Problem Solving from Nature*, pages 281–290, 1992.

[Yamada et al. 1996] T. Yamada and R. Nakano . *Job-shop scheduling by simulated annealing combined with deterministic local saerch*. Metha heuristics : theory and applications, Hingham, MA, 1996.

Index

Résumé

Les problèmes d'ordonnancement cyclique sont très présents dans le milieu industriel. Malheureusement, ces derniers sont souvent abandonnés ou laissés de coté par les industriels à cause de leur complexité.

Cette thèse propose une approche génétique de résolution du problème de Job-Shop cyclique ainsi que d'une application industrielle. Pour cela, les caractéristiques d'un ordonnancement ainsi que la définition des éléments les constituants sont présentés dans un premier temps. Les algorithmes génétiques ainsi que leur fonctionnement sont détaillés dans un second temps. Le troisième point de cette thèse se focalise sur les problèmes cycliques. C'est dans cette partie que l'approche génétique est présentée. Elle consiste à coupler un algorithme génétique avec un simulateur. Ce simulateur permet d'évaluer selon un critère un problème d'ordonnancement modélisé par un graphe de précédences à contraintes linéaires. Pour cela, le graphe de précédences à contraintes linéaires est transformé en réseau de Petri. C'est ce réseau de Petri couplé à un ensemble d'heuristiques de gestion de conflits de ressource qui permettent d'évaluer le problème. Une application est présenté dans un troisième temps. Pour cette application industrielle, deux simulateurs ont été employés. Le premier est basé sur les évènements discrets et le second utilise les graphes de précédences à contraintes linéaires. Le dernier point de cette thèse porte sur une plate-forme d'évaluation et d'amélioration de performance. Cette plate-forme à été développée dans le but de répondre à des problèmes industriels et académiques de type cyclique.

Mots clefs : ordonnancement cyclique, Job-Shop, simulation, algorithme génétique, "ordonnanceur".