

MODÉLISATION ET PLANIFICATION DES OPÉRATIONS DE RÉGLAGE DE MACHINES LORS DE CHANGEMENTS DE SÉRIE

C. PESSAN, E. NÉRON, O. BELLENGUEZ-MORINEAU

Laboratoire d'Informatique de l'Université François-Rabelais de Tours
Département Informatique de Polytech'Tours
64 av Jean Portalis, 37200 Tours, France
cedric.pessan@etu.univ-tours.fr

RÉSUMÉ : *La réduction de la perte de production due aux changements de série est d'une importance capitale pour l'industrie. Cet article décrit une modélisation que l'on peut faire de ce problème sous forme d'un problème à machines parallèles non reliées qui tient compte notamment des compétences des opérateurs et de la structure des lignes de production pour déterminer la productivité à chaque instant, l'objectif étant de maximiser la production pendant le changement de série. Ce problème sera ici modélisé comme un problème d'affectation. La présence de machines en plusieurs exemplaires sur certains étages permet de produire même si toutes les machines ne sont pas réglées. Cette particularité rend l'expression du critère assez complexe. Pour résoudre ce problème, un algorithme génétique et un algorithme de voisinage basé sur une méthode de descente locale sont présentés. Le voisinage utilisé est basé sur des échanges multiples de tâches. Enfin, un algorithme hybride exploitant à la fois les caractéristiques de la descente locale et de l'algorithme génétique est présenté ainsi que des résultats mettant en évidence l'intérêt de cet algorithme hybride par rapport aux deux autres.*

MOTS-CLÉS : *algorithme génétique, descente locale, voisinage à échanges multiples, changement de série, machines parallèles non reliées, multi-compétences*

1 INTRODUCTION

Un changement de série est l'opération pendant laquelle toutes les machines d'une ligne de production vont être réglées afin de produire une nouvelle série. Depuis quelques années, il est devenu capital de réduire le temps perdu dans les changements de série. En effet, des changements de série efficaces signifient une plus grande flexibilité dans la production, ce qui permet de répondre plus rapidement aux demandes des clients et de réduire les stocks. Depuis une vingtaine d'années, pour améliorer l'efficacité des changements de série, les entreprises se concentrent sur la diminution du temps d'arrêt des machines en travaillant sur la technique du réglage (préparer au maximum ce qui peut être préparé avant l'arrêt de la machine...), généralement en utilisant la méthode SMED (Shingo, 1985). Malheureusement, cette méthode n'est pas toujours applicable, notamment pour des machines où il est absolument nécessaire de travailler directement en ligne pour les régler. Dans ce cas, la seule manière d'améliorer l'efficacité du changement de série est d'améliorer la planification des tâches de réglage en fonction des compétences des opérateurs sur l'ensemble de la ligne. Malgré l'importance que donnent les industriels aux changements de série, il existe peu d'études sur la planification des tâches pour les réaliser : voir (Goubergen et al., 2004) par exemple. Les publications mentionnant les changements de séries s'intéressent généralement à l'ordonnancement de tâches en

prenant en compte des temps de réglage entre celles-ci ou au dimensionnement de lot mais pas à l'organisation du changement de série en lui-même. Le but de cet article est de présenter une modélisation du problème de planification des tâches de réglage dans un changement de série sous forme de problème d'affectation et de présenter les méthodes que nous avons développées pour le résoudre.

Cette étude a été réalisée en collaboration avec l'unité MDGBB de l'usine SKF de Saint-Cyr-sur-Loire. Dans ce cadre, l'expression du coût lié à un changement de série est imposée par les contraintes industrielles. Les lignes de production sur lesquelles nous avons travaillé produisent des roulements à bille mais les méthodes utilisées peuvent être adaptées à de nombreuses lignes. Dans la section 2, le problème et sa modélisation seront présentés. Dans les sections 3 et 4, nous verrons quelles méthodes de résolution ont été mises en places et quels sont les résultats que nous avons obtenus. Enfin, en conclusion, nous présenterons quelques perspectives de recherche.

2 PRÉSENTATION DU PROBLÈME

Le problème que l'on cherche à résoudre est un problème d'affectation et d'ordonnancement des tâches de réglage de machines aux opérateurs, de manière à maximiser le nombre de pièces produites pendant une période englobant un changement de série.

2.1 Les données

Les lignes de production sur lesquelles nous avons basé notre étude sont composées de n machines agencées sous la forme d'une ligne série-parallèle complexe telle que présentée sur le premier diagramme de la figure 1. Parmi ces machines, certaines sont définies comme étant prioritaires : il s'agit d'un ensemble de cardinalité minimum de machines nécessaires pour que la ligne produise. Soit n_1 tel que les machines indicées entre 1 et n_1 sont les machines prioritaires et les machines indicées entre $n_1 + 1$ et n sont les machines non prioritaires (en double, non strictement nécessaires à la production).

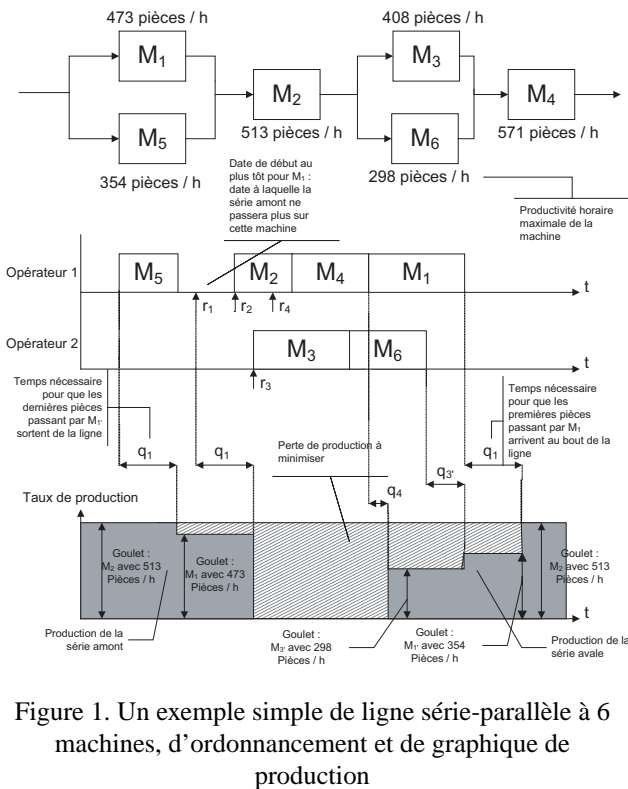


Figure 1. Un exemple simple de ligne série-parallèle à 6 machines, d'ordonnancement et de graphique de production

Sur l'exemple de la figure 1, $n_1 = 4$, les machines prioritaires seront notées $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ et \mathcal{M}_4 . Les machines non prioritaires seront notées \mathcal{M}_5 et \mathcal{M}_6 .

Pour chaque machine \mathcal{M}_i de la ligne, nous connaissons la date de début au plus tôt de l'opération de réglage sur cette machine. Cette date correspond à la sortie de la dernière pièce de la série amont sur la machine (ou temps de vidage). Nous noterons r_i cette date. Dans le cas général, les machines en double n'ont pas besoin de respecter une date de début au plus tôt car il est toujours possible de continuer à produire sans elles. Le deuxième diagramme de la figure 1 illustre un ordonnancement prenant en compte les r_i pour les machines prioritaires : par exemple \mathcal{M}_3 ne peut commencer à être réglée qu'après r_3 .

Nous connaissons également pour chaque machine \mathcal{M}_i , le temps minimal existant entre le début de la production

d'une pièce sur la machine \mathcal{M}_i et la date de sortie de la pièce de la ligne. Nous noterons q_i ce temps. Par définition, les q_i seront toujours égaux pour toutes les machines d'un étage. Sur l'exemple de la figure 1, on aura $q_1 = q_5$ et $q_3 = q_6$. Les deuxième et troisième diagrammes illustrent l'influence des q_i sur la production : par exemple après le réglage de \mathcal{M}_6 il faut attendre un temps q_6 pour que le taux de production soit modifié.

Pour chaque machine \mathcal{M}_i , nous connaissons le temps de cycle en secondes. Il s'agit du temps en secondes mis par la machine pour faire son traitement sur chaque pièce. Plus ce temps est faible, plus la cadence de la machine est élevée. Nous noterons ce temps $t_c(i)$. Nous pouvons en déduire la productivité de la machine (voir section 2.5).

Pour être exécutée, une tâche de réglage nécessite l'immobilisation de la machine et l'intervention d'un opérateur. Il y a dans l'atelier λ opérateurs. Chaque opérateur O_h est plus ou moins apte à effectuer une tâche M_i en fonction de son expérience et nous connaissons le temps moyen qu'il a mis pour effectuer la tâche sur les anciens changements de série des 3 derniers mois. Nous noterons $p_{i,h}$ ce temps. Pour construire l'ordonnancement, nous considérons que l'opérateur mettra ce temps $p_{i,h}$ pour réaliser la tâche. Si l'opérateur O_h n'est pas qualifié pour intervenir sur la machine \mathcal{M}_i , nous considérons que $p_{i,h} = +\infty$.

Pour chaque opérateur O_h nous connaissons ses périodes de disponibilité notées $A(h, t)$. On aura $A(h, t) = 1$ si l'opérateur peut intervenir sur un changement de série à la date t . Il est important de noter que dans le cas de SKF, les périodes de disponibilités sont continues : elles ont un début et une fin et il n'y a aucune pause au milieu.

Nous noterons C_i la date de fin de réglage d'une machine et t_i la date de début de réglage d'une machine. Nous noterons finalement σ_i l'opérateur étant affecté à la machine i .

2.2 Les contraintes

Les contraintes suivantes sont imposées par le contexte industriel de cette étude :

- Les opérations de réglage ne peuvent être préemptées : une fois l'opération commencée, le travail ne peut être interrompu ni par une autre opération ni par une fin de période de travail ou une pause.
- Un opérateur ne peut intervenir que sur une machine à la fois.
- Une opération ne peut-être effectuée que par un opérateur à la fois.
- Le réglage d'une machine prioritaire ne devra commencer que lorsque la production de la série amont sera totalement finie sur la machine (respect de la date de début au plus tôt).

Le deuxième diagramme de la figure 1 montre un exemple d'ordonnancement respectant ces contraintes. Toutes les tâches de réglage commencent après leurs dates de début au plus tôt, sauf M_5 qui est une machine en double et qui peut donc être réglée avant si on le souhaite.

Le troisième diagramme est un graphique de production. La productivité de la ligne est toujours limitée par un étage goulet et celui-ci ne sera pas toujours le même suivant la configuration des machines déjà réglées. L'effet d'un début ou d'une fin de réglage de M_i sur la production n'est visible qu'après un temps q_i qui suit l'événement. Par exemple, avant que la machine M_6 ne soit réglée, la machine goulet était M_3 qui fait partie du même étage que M_6 , on sait donc que lorsque son réglage sera terminé, le goulet changera. Après son réglage, il faut attendre un temps q_6 pour voir le taux de production évoluer : c'est la machine M_5 qui devient machine goulet.

Le critère que nous verrons en section 2.5 exprime analytiquement la surface définie par cette courbe de production (la quantité de pièces produites pendant le changement de séries).

2.3 Identification du problème

Le problème peut s'identifier comme un problème à machines parallèles non identiques et non reliées qui peuvent exécuter divers types de tâches. Les machines représentent les opérateurs et les tâches les machines à régler. De plus, toutes les machines n'ont pas la même période de disponibilité. Le critère représente le nombre de pièces que l'on peut produire sur l'horizon de calcul, cette fonction sera définie dans la section 2.5. Ce problème peut se noter :

$$R, MPM | r_i, q_i | f(C_i, q_i)$$

Les opérateurs multi-compétents sont ici modélisés par des machines à usage multiple telles qu'étudiées par Jurish (Jurish, 1992). Dauzère-Pérès et al. ont également proposé des méthodes de résolution prenant en compte les compétences pour des problèmes d'atelier à ressource flexible (Dauzère-Pérès et al., 1998). Le problème est que toutes ces méthodes se basent sur des critères classiques, or comme nous allons pouvoir le voir en section 2.5, le critère de notre problème est complexe et totalement différent des critères abordés dans la littérature.

2.4 Hypothèses

On fera les hypothèses suivantes propres au cas SKF :

- Dans cet article, nous ne traiterons qu'un cas particulier de ce problème où les machines en double ne peuvent pas être réglées avant la fin de la production du lot amont sur leur étage : ce qui signifie qu'il faut respecter leur date de début au plus tôt et que la production chutera brutalement à 0 une fois

le lot amont terminé sur la dernière machine de la ligne. Cette hypothèse nous a été dictée par les pratiques actuelles de l'entreprise. En fait, régler des machines en double avant la fin du lot précédent signifie que le lot mettra plus de temps à se terminer non seulement à cause des machines que l'on rend inutilisables pour cet ancien lot mais aussi à cause du manque de ressources humaines puisqu'on enlève des opérateurs conduisant la ligne pour leur faire régler des machines : ce serait à priori beaucoup trop pénalisant et rendrait l'organisation humaine difficile en pratique.

La figure 2 illustre un ordonnancement et un graphe de production respectant cette hypothèse.

- On a une priorité définie pour chaque étage : une tâche prioritaire est absolument nécessaire au fonctionnement de la ligne alors qu'une tâche non prioritaire n'est là qu'en renfort pour améliorer la productivité. Pour des raisons pratiques, on considère qu'une machine en double ne pourra rien produire tant que la machine prioritaire de l'étage ne sera pas réglée également.

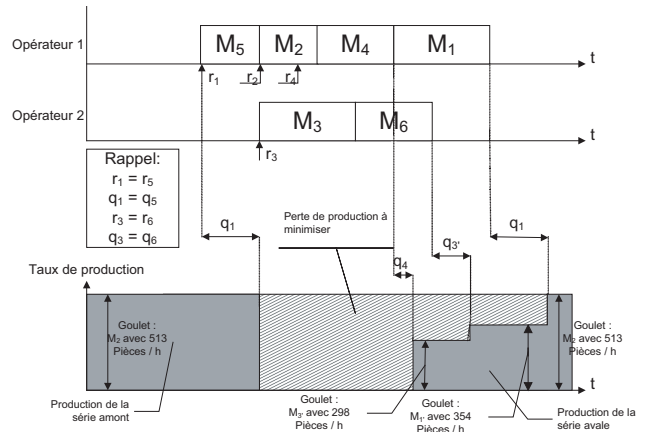


Figure 2. Exemple reprenant la ligne, l'affectation et la séquence de la figure 1 mais respectant les dates de début au plus tôt pour les machines en double

2.5 Critère

L'objectif est de maximiser le nombre de pièces produites pendant le changement de série sachant que la quantité de pièces à produire pour la série amont est connue à l'avance. Cela revient donc à maximiser le nombre de pièces de la série aval que l'on peut produire et c'est cette quantité que le critère $f(C_i, q_i)$ va exprimer.

À un instant donné, suivant la configuration dans laquelle on se trouve, i.e. quelles machines ont déjà été réglées pour la série aval, on peut déterminer quelle est la productivité de la ligne. Au cours du changement de série, cette

configuration va bien sûr évoluer, c'est pourquoi, pour exprimer $f(C_i, q_i)$, il va falloir examiner pour chaque unité de temps, la quantité de pièces qui pourra être produite.

On sait qu'avant d'avoir toutes les machines dites prioritaires réglées, il sera impossible de produire la moindre pièce. Il est donc inutile d'examiner un intervalle avant que toutes les machines prioritaires n'aient été réglées.

Soit :

- T_c : Horizon de calcul de la production pour le critère
- T_g : Date de sortie de la première pièce de la chaîne, c'est à dire la date de fin de réglage de la dernière machine prioritaire à laquelle on ajoute le temps de latence de la machine :

$$T_g = \max_{i \in [1..n_1]} (C_i + q_i)$$

- $Prod(t)$: Quantité de pièces produites entre t et $t + 1$

Le critère s'exprime par :

$$f(C_i, q_i) = \sum_{t=T_g}^{T_c} Prod(t)$$

$Prod(t)$ est déterminé par l'étage le plus lent de la ligne. Soit $ProdEt(i, t)$ la quantité de pièces que peut produire l'étage de la machine i entre t et $t + 1$. Si on considère que chaque étage a exactement une machine prioritaire, on obtient :

$$Prod(t) = \min_{i \in [1..n_1]} ProdEt(i, t)$$

Il reste à exprimer $ProdEt(i, t)$, soit :

$$Et(i, j) = \begin{cases} 1 & \text{si les machines } i \text{ et } j \text{ font} \\ & \text{partie du même étage} \\ 0 & \text{sinon} \end{cases}$$

- $Fini(i, t) = \begin{cases} 1 & \text{si } t > C_i + q_i \\ 0 & \text{sinon} \end{cases}$
indique si une machine est déjà réglée et que les pièces qui passent par celle-ci commencent à sortir de la ligne (ou sont en attente de la fin du réglage d'une machine prioritaire qui bloque la production).

La capacité de production d'un étage est la somme des capacités de production de chacune des machines de cet étage qui sont déjà réglées. Ce qui s'exprime par :

$$ProdEt(i, t) = \sum_{j=1}^n Et(i, j) * Fini(j, t) * \frac{360}{t_c(j)}$$

La constante 360 vient du fait que le temps de cycle est en secondes et que notre unité de temps est le dixième d'heure. $\frac{360}{t_c(j)}$ est donc le nombre de pièces traitées par la machine j lorsque celle-ci fonctionne à plein régime pendant un dixième d'heure.

Il faut maintenant généraliser cette expression au cas (rencontré à SKF) où l'on a dans la ligne de production plusieurs groupes de machines séries en parallèle.

Soit $L(i)$ la longueur de la branche à laquelle appartient la machine i . On parle ici de branches dans un étage : s'il s'agit d'un étage à machines parallèles on aura $L(i) = 1$, s'il s'agit d'un étage où l'on a plusieurs couples de machines en parallèle (couples ébauche/finition par exemple) on aura $L(i) = 2 \dots$

Soit $P(i)$ un booléen indiquant s'il s'agit de la première machine d'une branche.

De plus on considère que toutes les machines d'une branche se suivent dans leurs numéros d'indexation.

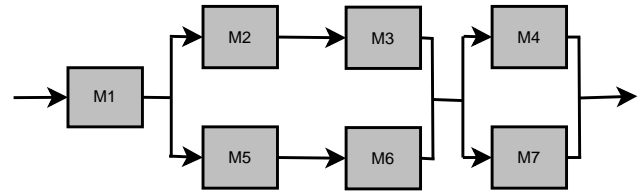


Figure 3. Exemple de ligne avec des branches de longueur 2

Sur la ligne de la figure 3, on a par exemple : $L(1) = 1$, $P(1) = 1$, $L(2) = 2$, $P(2) = 1$, $L(6) = 2$, $P(6) = 0$, $L(7) = 1$, $P(7) = 1 \dots$

L'expression de $ProdEt(i, t)$ devient alors :

$$ProdEt(i, t) = \sum_{j=1}^n Et(i, j) * \prod_{k=j}^{j+L(i)-1} Fini(k, t) * P(i) * \min_{k \in [j, j+L(i)-1]} \frac{360}{t_c(k)}$$

Cette expression tient compte du fait que la productivité de chaque branche est limitée par la machine la plus lente de la branche.

La complexité de calcul du critère ainsi défini est $O(n^2 * T_c)$ et peut être implémenté en $O(n^3)$ si on s'arrange pour ne pas recalculer la productivité de la ligne à chaque instant mais seulement lorsqu'une nouvelle machine est réglée.

2.6 Problèmes d'indisponibilités

Afin de faciliter la mise au point de nos algorithmes, il était nécessaire que le critère pénalise les solutions non réalisables (dans le sens où elles font déborder les charges

de travail des opérateurs de leurs périodes de disponibilité). Ces solutions seront représentées par un critère négatif qui sera d'autant plus petit qu'il y aura plus de tâches prévues hors période de disponibilité :

$$f(C_i, q_i) = - \sum_{i=1}^n \sum_{t=t_i}^{C_i} [1 - A(\sigma_i, t)] \text{ si la solution est non réalisable.}$$

2.7 Priorité sur les opérations

Soit la liste de priorité L_p contenant les tâches dans l'ordre suivant :

1. les machines prioritaires classées par r_i croissants.
2. les machines en double dans l'ordre où elles permettent de débloquent les étages goulets.

Si l'on regarde la séquence sur un opérateur, on sait que L_p serait optimale s'il n'y avait pas de machines en double ou s'il n'y avait pas de r_i . En effet :

- si on n'a pas de machines en double, on cherche simplement à terminer les tâches au plus tôt et cela revient donc à résoudre le problème $1|r_i|C_{max}$ qui est résolu en classant les tâches par r_i croissants.
- si on n'a pas de r_i , entre deux machines, il vaudra toujours mieux régler d'abord la machine qui sera utile en premier pour augmenter la productivité (celle dont l'étage est le plus limitant).

Dans cet article, on fera l'hypothèse que les données ($p_{i,h}$ et r_i) sont telles que toutes les tâches sont exécutées sans temps mort sur un même opérateur après la première tâche. Cette hypothèse est vérifiée dans les lignes de production SKF. Ceci n'est possible que parce que l'on sait que l'on n'a aucune période d'indisponibilité entre deux périodes de disponibilité d'un opérateur. Trouver la séquence optimale sur un opérateur revient donc à déterminer la première tâche puis à placer les autres tâches suivant L_p .

En pratique, il s'avère que mettre une tâche non prioritaire au début de la séquence peut être risqué et n'apporte finalement que très peu par rapport au critère. Nous avons donc choisi dans cet article de nous limiter au cas où l'on séquence toujours les tâches suivant L_p . Notre problème devient donc uniquement un problème d'affectation.

Pour résoudre ce problème, nous nous sommes dirigés vers des algorithmes évolutionnistes basés sur les principes des algorithmes génétiques et vers une méthode de descente locale.

3 MÉTHODES DE RÉOLUTION

3.1 Méthodes de descentes locales

Une descente locale se base sur un opérateur de voisinage permettant de trouver des solutions proches d'une solution

que l'on cherche à améliorer. L'idée de l'opérateur de voisinage que nous avons défini est de faire des échanges de tâches entre opérateurs. Le problème est que chaque opérateur a un profil de compétences particulier, ce qui veut dire que de nombreuses fois, il sera impossible de faire un échange simple de deux tâches entre deux opérateurs. La solution pour ce genre de problème est souvent de faire une rotation de tâches entre 3, 4 opérateurs, voire plus.

Un tel voisinage basé sur un graphe a déjà été étudié par Frangioni (Frangioni et al, 1999) mais leur modélisation ne prenait pas en compte les compétences et ne pouvait marcher que pour minimiser C_{max} .

3.1.1 Opérateur k-voisinage

L'opérateur k-voisinage permet de modéliser les voisins par des chaînes de k réaffectations au maximum. Pour cela, le voisinage a été modélisé par un graphe biparti qui utilise deux types de sommets : opérateurs et tâches. Deux sommets fictifs ont été ajoutés : la source S et le puits P.

Les arcs du graphe sont les suivants :

- Un arc d'un sommet opérateur vers un sommet tâche si la tâche est affectée à cet opérateur dans la solution que l'on cherche à améliorer.
- Un arc d'un sommet tâche vers un sommet opérateur pour chaque tâche que l'on est susceptible d'affecter à un opérateur (c'est à dire, un arc par compétence).
- Un arc de chaque sommet opérateur à P.
- Un arc de S à chaque opérateur qui possède une tâche critique. Les tâches critiques sont celles qui sont susceptibles d'améliorer le critère si elles se terminent plus tôt. Il s'agit d'une part de la tâche prioritaire se terminant le plus tard (celle qui conditionne le début de la production) et d'autre part de chaque tâche non prioritaire qui n'a aucune tâche plus prioritaire (suivant L_p) se terminant après (les machines pour lesquelles on a une marche à la fin de leur réglage sur le graphe de production de la figure 2 par exemple).

Les solutions voisines vont alors être représentées par des chemins de S à P dans ce graphe :

- Lorsque l'on passe d'un sommet opérateur vers un sommet tâche, il s'agit d'une tâche que l'on enlève d'un opérateur.
- Lorsque l'on passe d'un sommet tâche vers un sommet opérateur, il s'agit d'une tâche que l'on affecte à un opérateur.

Le but est donc de trouver des chemins de S à P de longueur $2k + 2$ qui améliorent le critère. L'évaluation d'une

solution étant l'opération la plus coûteuse en temps pour chaque chemin à examiner, le principal problème va donc être de limiter le nombre de chemins que l'on va évaluer dans ce graphe :

- on n'évaluera pas les chemins qui réaffectent les tâches sur les mêmes opérateurs
- on éliminera les chemins qui représentent des solutions identiques à une solution déjà évaluée (par exemple les chemins qui font les réaffectations dans un ordre différent)
- on n'évaluera pas les solutions qui n'améliorent pas la date de fin d'au moins une des tâches critiques.

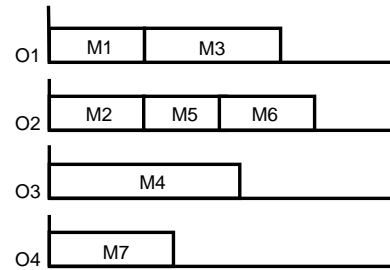
L'ajout de tous ces tests avant d'évaluer réellement la solution a permis de réduire considérablement le temps d'exécution de l'algorithme : pour un 4-voisinage, on est passé d'environ 1h30 de temps de calcul sur les tous premiers essais qui évaluaient tous les chemins à des temps de l'ordre de la minute comme on peut le voir dans les résultats de la section 4.

Si on revient sur la construction du graphe, on peut se demander pourquoi ne relier S qu'aux opérateurs qui ont une tâche critique et non aux autres. En fait, pour améliorer la solution, il faut absolument que le chemin passe par un de ces opérateurs. Or si on place ces opérateurs en premier dans le chemin en s'imposant que si on réaffecte une tâche critique on le fait en premier dans la chemin, on peut essayer de déduire une borne inférieure pour cette tâche critique sur les chemins commençant par les mêmes α premiers sommets. Par exemple, on peut calculer pour un opérateur possédant une tâche critique, ce que l'on pourra enlever au maximum après les α premiers sommets du chemin : si la tâche critique ne se termine toujours pas plus tôt que dans notre solution initiale, tous les chemins commençant par ces mêmes α sommets ne sont pas intéressants. Avec une borne inférieure efficace, il serait possible de couper un certain nombre de chemins à explorer. Malheureusement, nous n'avons pas encore trouvé de borne inférieure suffisamment efficace pour nous faire gagner plus de temps qu'elle nous coûte à calculer, mais cela reste une piste intéressante à explorer pour améliorer la vitesse d'exécution de la descente locale.

3.1.2 Exemple

La figure 4 montre un exemple de graphe construit à partir d'un diagramme de Gantt (la solution courante) et d'une liste de compétences de chaque opérateur. Cet exemple suppose que tous les opérateurs exécutent une tâche critique donc S est relié à tous les opérateurs.

Si on prend par exemple le chemin {S,O1,M1,O3,P} dans le 2-voisinage, la solution représentée par ce graphe est celle où l'on enlève la tâche M1 de l'opérateur O1 et où l'on ajoute cette même tâche M1 à l'opérateur O3.



opérateur :	compétences
O1	M1,M3
O2	M2,M5,M6
O3	M1,M4,M5,M6,M7
O4	M7

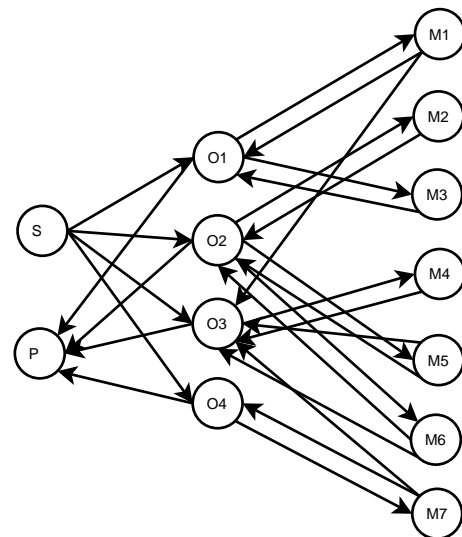


Figure 4. exemple de graphe représentant le k-voisinage d'une solution

3.2 Algorithme génétique

Le principe d'un algorithme génétique est de faire évoluer une population de solutions que l'on appellera individus ou chromosomes, représentés par une méthode de codage. L'évolution se fait par croisement et mutation des individus (Holland,1975). Les performances de l'algorithme dépendent des choix de codage et d'opérateurs de croisement et mutation que l'on fait. Nous avons préféré ce type d'algorithme à une autre méta-heuristique de voisinage car nous avons l'intention d'introduire un deuxième critère dans le futur, ce qui est relativement facile à réaliser dans un algorithme génétique.

3.2.1 Codage

La méthode de codage inspirée de (Liu Min et Wu Cheng, 1999) que nous avons choisie utilise un tableau de nombres entiers : $k_i, i \in [0..n]$ et $k_i \in \{h/h \in$

$[0..\lambda], p_{i,h} \neq +\infty\}$. Le tableau contient une case par tâche (dans notre cas les machines à régler) et chaque case contient un entier indiquant quel opérateur effectuera cette tâche. Le nombre d'opérateurs sélectionnable pour une tâche i dépend bien sûr du nombre d'opérateurs qui possèdent la compétence.

Exemple :

1	2	1	3	2	2	4
---	---	---	---	---	---	---

↑
Le job 4 est placé sur l'opérateur 3.

L'avantage de ce codage est qu'il ne prend en compte que l'affectation, ce qui correspond à ce que l'on souhaite faire pour ce problème car l'ordonnancement est déduit de l'affectation (voir section 2.7).

3.2.2 Croisement

L'opérateur de croisement utilisé ici est une méthode de croisement à deux points dont l'intérêt a été montré par Beasley (Beasley et al., 1993). Soient deux chromosomes. On tire aléatoirement deux nombres p_1 et p_2 compris entre 1 et n et tels que $p_1 \leq p_2$. On peut alors créer deux chromosomes fils :

- Le premier fils recopie les gènes numérotés entre 1 et p_1 et entre p_2 et n du premier chromosome et les gènes numérotés entre p_1 et p_2 du deuxième chromosome.
- Le deuxième fils est construit de la même manière mais en inversant les deux chromosomes parents.

Exemple :

chromosome 1 :						
1	2	1	3	2	2	4
chromosome 2 :						
3	2	1	3	3	3	3
		↑		↑		
		p_1		p_2		
fils 1 :						
1	2	1	3	3	3	4
fils 2 :						
3	2	1	3	2	2	3

Un avantage de cet opérateur de croisement est que l'on est sûr que chaque chromosome fils créé sera valide car la solution générée respecte les compétences des opérateurs.

3.2.3 Mutation

La fonction de mutation modifie simplement un gène : on affecte une tâche choisie au hasard à un opérateur choisi aléatoirement mais différent du précédent et qui a la compétence pour cette tâche (condition nécessaire pour construire une solution valide).

3.2.4 Sélection

A chaque itération de l'algorithme, on génère K individus par croisement / mutation à partir d'individus parents pris totalement au hasard. Comme la taille N de la population est constante après chaque itération, tous les individus ne survivront pas. Pour sélectionner les individus qui survivront, on fait alors à la fin de l'itération une sélection par tournoi sur l'ensemble de la population ($N + K$ individus) jusqu'à ne plus avoir que N individus.

La sélection par tournoi consiste à tirer deux individus au hasard dans la population et à éliminer celui qui est le moins bon au sens du critère.

3.2.5 Diversification

Après un certain nombre d'itérations sans amélioration, on met en place des mécanismes de diversification pour essayer de sortir de l'optimum local vers lequel l'algorithme converge. On fait dans l'ordre les opérations suivantes (entre chaque point on attend un nombre déterminé d'itérations pour voir si on arrive à améliorer la solution ou pas) :

1. Augmentation progressive de la probabilité de mutation.
2. Augmentation du nombre de nouveaux individus créés par itération.
3. Remplacement de l'opérateur de mutation par une génération aléatoire d'individus.
4. Brassage : on régénère aléatoirement tous les individus, sauf le meilleur, et on repasse au niveau 0 des mécanismes de diversification (aucun mécanisme de diversification).

3.3 Techniques d'hybridation

L'hybridation consiste ici à combiner les deux méthodes que nous avons développées (descente locale et algorithme génétique), afin d'obtenir de meilleures performances qu'en utilisant une seule des deux méthodes. Plusieurs approches sont possibles :

- Utiliser la descente locale comme opérateur d'intensification sur les bonnes solutions générées par l'algorithme génétique (sélection par roulette, par exemple, des individus que l'on cherche à améliorer)
- Utiliser la descente locale à chaque fois que l'on trouve une solution améliorant la meilleure solution connue.
- Utiliser l'algorithme génétique pour trouver de nombreuses solutions réalisables, afin de faire du multi-start avec la descente locale (on essaie de faire travailler au maximum la descente locale). Pour cela, il suffit d'appliquer la descente locale sur un individu réalisable pris au hasard dans la

population. Il faut que la probabilité d'appliquer la descente locale à une solution dans une itération de l'algorithme génétique soit réglée de manière à exploiter au mieux les caractéristiques des deux algorithmes.

On peut également se demander s'il vaut mieux utiliser la descente locale à n'importe quel moment ou seulement lorsque l'on n'arrive plus à améliorer la solution par l'algorithme génétique avant de faire un brassage. Toutes ces possibilités ont été testées et les résultats sont présentés dans la section 4.

4 RÉSULTATS EXPÉRIMENTAUX

4.1 Jeu de données

Les jeux de données que nous avons utilisés pour tester les différentes méthodes sont basés sur des données de lignes de production réelles. Certaines instances sont dérivées de données réelles afin de les rendre plus difficiles à résoudre : moins d'opérateurs, des opérateurs peu compétents, des problèmes de disponibilité...

Les instances ont les dimensions suivantes :

- entre 40 et 60 tâches (le nombre de machines à régler sur la ligne où l'on change de série)
- entre 5 et 15 opérateurs
- les durées des tâches sont comprises entre 6 minutes et 3 heures

Nous disposons ainsi de 15 instances qui représentaient la plupart des cas de figure que l'on rencontre dans la réalité surtout au niveau des périodes de disponibilité des opérateurs par rapport au début du changement de série : si certains opérateurs sont indisponibles 1 heure après le début du changement de série, il sera plus difficile de trouver une affectation très performante par exemple. A la date où ce papier est écrit, nous ne disposons pas de borne supérieure efficace pour ce problème, on ne peut donc que comparer les résultats de nos différentes méthodes.

Pour obtenir un logiciel qui reste agréable à utiliser en pratique dans l'entreprise, il serait préférable d'avoir un temps d'exécution de l'algorithme de moins de 5 minutes. Nous nous intéresserons donc plus particulièrement aux performances obtenues en moins de 5 minutes.

4.2 Descente locale seule à partir d'ECT

Afin de tester la descente locale, nous avons besoin de partir d'une solution existante. Pour construire cette solution, nous avons utilisé un algorithme ECT (Earliest completion time) : les tâches sont prises une par une dans l'ordre de L_p (voir 2.7) et placées sur l'opérateur pouvant la terminer au plus tôt.

Les voisinages qui ont été testés sont les k -voisinages pour $k \in [3, 6]$. Les résultats sont présentés sur le tableau 1, les temps sont exprimés en secondes.

	ECT		Desc3		Desc4		Desc5		Desc6	
	Crit	Tps	Crit	Tps	Crit	Tps	Crit	Tps	Crit	Tps
Test 0	3816	0	4200	3	4200	25	4200	188	4200	1560
Test 1	3340	0	3703	3	3703	27	3706	313	3780	2880
Test 2	2852	0	3169	3	3236	28	3236	80	3236	480
Test 3	2888	0	3148	6	3178	46	3188	212	3228	1800
Test 4	1115	0	1820	2	2091	26	2091	77	2091	420
Test 5	0	0	0	0	0	0	0	1	0	9
Test 6	0	0	0	0	0	0	0	1	0	6
Test 7	2031	0	2201	1	2492	25	2492	43	2492	300
Test 8	1710	0	1746	0	1746	1	1746	4	1746	60
Test 9	2178	0	2592	2	2592	9	2592	41	2592	240
Test 10	2349	0	2349	0	2349	1	2403	9	2403	60
Test 11	2934	0	3003	1	3003	9	3003	52	3123	360
Test 12	2455	0	2580	1	2593	14	2593	63	2631	540
Test 13	0	0	0	0	0	0	0	0	0	1
Test 14	2538	0	3727	8	3920	35	4200	210	4200	1140

Tableau 1. Valeurs du critère obtenues et temps mis par l'exécution de ECT + descente locale avec des k -voisinages ($k \in [3, 6]$)

Nous pouvons tout d'abord constater que globalement l'algorithme de descente locale améliore quasiment toujours de manière significative la solution trouvée par ECT du moment que ECT trouve une solution valide (dont le critère est supérieur à 0).

Pour $k < 5$, le temps d'exécution de la descente locale est inférieur à la minute alors que pour des valeurs de k supérieures, le temps est généralement supérieur à la minute et augmente très rapidement avec k . Pour notre algorithme hybride, il ne faudrait donc pas utiliser de k -voisinage avec $k \geq 5$.

Il est intéressant de noter que la performance de ECT suivie de la descente locale avec le 4-voisinage est généralement aussi bien voire meilleure en moins d'une minute que ce qui était fait à la main auparavant dans l'entreprise par des hommes d'expérience en une heure environ.

Le tableau 2 présente les mêmes résultats mais sous forme de pourcentage de déviation par rapport à la meilleure solution connue : obtenue par la descente locale ou au moins une fois par l'algorithme génétique. La moyenne ignore les instances pour lesquelles ECT ne trouve pas de solution initiale afin de mieux se rendre compte de la performance des algorithmes : il s'agit des Tests 5, 6 et 13.

En moyenne, un 3-voisinage améliore déjà la solution trouvée par ECT de manière importante dans un temps raisonnable pour une utilisation dans l'algorithme hybride. Le 4-voisinage reste utilisable du point de vue du temps d'exécution à la fin de l'algorithme hybride pour essayer d'améliorer la solution trouvée.

4.3 Algorithme génétique

Nous avons testé ici 5 variantes de notre algorithme génétique de base qui utilise une population de 50 indivi-

	ECT	Desc3	Desc4	Desc5	Desc6
Test0	9.14 %	0 %	0 %	0 %	0 %
Test1	11.64%	2.04 %	2.04 %	1.96 %	0 %
Test2	14.97%	5.52 %	3.52 %	3.52 %	3.52 %
Test3	10.59%	2.54 %	1.61 %	1.3 %	0.06 %
Test4	62.38%	38.6 %	29.45%	29.45%	29.45%
Test5	100 %	100 %	100 %	100 %	100 %
Test6	100 %	100 %	100 %	100 %	100 %
Test7	19.56%	12.83%	1.31 %	1.31 %	1.31 %
Test8	2.06 %	0 %	0 %	0 %	0 %
Test9	16.84%	1.03 %	1.03 %	1.03 %	1.03 %
Test10	2.25 %	2.25 %	2.25 %	0 %	0 %
Test11	6.05 %	3.84 %	3.84 %	3.84 %	0 %
Test12	8.22 %	3.55 %	3.07 %	3.07 %	1.64 %
Test13	100 %	100 %	100 %	100 %	100 %
Test14	39.57%	11.26%	6.67 %	0 %	0 %
moyenne	14.52%	5.96 %	3.91 %	3.25 %	2.64 %

Tableau 2. Pourcentage de déviation par rapport aux meilleures solutions connues pour ECT + descente locale avec des k-voisinages ($k \in [3, 6]$)

us, qui génère 10 nouveaux individus à chaque itération et a une probabilité de mutation de 10%. Ces valeurs ont été choisies car lors de nos premiers tests, il s'agissait de valeurs qui donnaient globalement de bons résultats en 2 minutes : une population de 100 individus par exemple devenait légèrement meilleure qu'une population à 50 individus mais seulement après 10 minutes en général, ce qui ne nous intéresse pas dans notre cas.

Les variantes testées sont les suivantes :

- POP100 : utilise une population de 100 individus
- MUT35 : utilise une probabilité de mutation de 35%
- RENF : la probabilité de mutation évolue quand il n'y a pas d'amélioration (+0.75% à chaque itération) et après 700 itérations sans amélioration on passe à un mode renforcé où on génère 40 nouveaux individus par itération au lieu de 10
- BRASSAGE : même algorithme que RENF mais on ajoute un brassage lorsqu'il n'y a pas d'amélioration après 300 itérations en mode renforcé

Chaque instance a été testée 10 fois sur chaque algorithme pendant 2 minutes. Un relevé du meilleur ordonnancement trouvé au bout de 15 secondes, 30 secondes, 1 minute et 2 minutes est mémorisé.

Le tableau 3 contient les moyennes de pourcentage de déviation de chaque variante de l'algorithme génétique. Ces moyennes ne prennent pas en compte le test 6 car nous n'avons aucune solution valide connue et donc le résultat serait biaisé.

	15s	30s	1min	2min
Normal	23.85%	20.00%	14.69%	10.48%
POP100	24.78%	20.92%	16.49%	11.92%
MUT35	15.06%	10.63%	8.03 %	6.02 %
RENF	13.39%	9.03 %	6.93 %	4.58 %
BRASSAGE	14.62%	9.68 %	5.98 %	4.23 %

Tableau 3. Pourcentage de déviation par rapport aux meilleures solutions connues pour chaque variante de l'algorithme génétique

A la vue de ces résultats, on peut confirmer ce qui a été dit plus haut pour la taille de la population : une population de 100 individus au lieu de 50 est légèrement moins performante.

Par contre augmenter la probabilité de mutation améliore de manière importante les performances. Ceci est dû à nos opérateurs de croisement et de mutation. En effet, l'opérateur de croisement ne fait qu'explorer les différentes combinaisons d'affectations qui existent déjà dans la population, il est donc important d'avoir une forte probabilité de mutation pour insérer de nouvelles affectations dans la population.

Une forte probabilité de mutation est intéressante mais pourrait empêcher d'exploiter totalement les caractéristiques intéressantes de bonnes solutions. C'est pour remédier à ce problème que la variante RENF a été créée : après une amélioration du critère, on passe à une probabilité de mutation faible pour exploiter les caractéristiques de ce nouvel individu puis on l'augmente progressivement. Cette variante augmente également le nombre d'individus créés par itération (mode dit renforcé) lorsqu'on n'arrive plus à améliorer la solution depuis trop longtemps. Le tableau 3 nous indique que cette variante est plus efficace que les autres, exceptée la variante BRASSAGE.

Les performances de BRASSAGE et RENF sont très proches. On a un léger avantage pour RENF à 15s et pour BRASSAGE à 2min : en fait, il est normal que l'avantage de BRASSAGE ne se voit pas au début de l'exécution de l'algorithme puisque le brassage n'intervient qu'en dernier recours quand on n'arrive pas à améliorer le critère.

Les variantes BRASSAGE et RENF sont très légèrement moins performantes en moyenne que ECT + descente locale avec un 4-voisinage. Par contre, l'algorithme génétique a l'avantage de trouver des solutions pour toutes les instances exceptée Test6 (nous ne connaissons aucune solution valide pour cette instance).

4.4 Algorithme hybride

L'algorithme hybride se base sur la variante BRASSAGE de l'algorithme génétique. Les variantes de l'algorithme hybride testées sont les suivantes :

- Intens : utilise la descente locale avec 3-voisinage sur chaque individu améliorant le critère et éventuellement (probabilité de 2%) sur un individu pris aléatoirement : en utilisant aléatoirement soit un 2-voisinage, soit un 3-voisinage.
- Intens_roul : identique à Intens mais la descente locale est appliquée sur des individus sélectionnés par roulette (plus grande probabilité pour les meilleurs individus).
- Intens_si_renf : la descente locale n'est appliquée que si on est en mode renforcé
- Intens_5 : probabilité d'appliquer la descente locale de 5% au lieu de 2%.
- Intens_v : pas de brassage tant qu'on n'a pas de solution valide.

	15s	30s	1min	2min
Intens	7.04 %	5.1 %	3.04 %	1.57 %
Intens_roul	6.46 %	5.13 %	3.59 %	2.15 %
Intens_si_renf	9.27 %	7.12 %	5.49 %	3.12 %
Intens5	8.73 %	5.44 %	3.76 %	2.88 %
Intens_v	6.72 %	5.09 %	2.33 %	1.59 %

Tableau 4. Pourcentage de déviation par rapport aux meilleures solutions connues pour chaque variante de l'algorithme hybride

Sur le tableau 4, nous pouvons voir que l'intérêt de cet algorithme hybride est très clair : on obtient avec toutes les variantes de meilleures performances qu'avec l'algorithme génétique ou la descente locale seule.

L'augmentation au dessus de 2% de la probabilité d'appliquer la descente locale à un individu semble être mauvais pour le critère. L'utilisation de la roulette ne donne pas d'aussi bonnes performances que si l'on prend des individus au hasard pour appliquer la descente locale. Appliquer la descente locale seulement en mode renforcé semble être la plus mauvaise option. En effet, la variante Intens_si_renf tend à faire travailler en priorité l'algorithme génétique puis la descente locale alors que toutes les autres tendent à faire travailler au maximum la descente locale : l'algorithme génétique sert pour ces variantes à générer de nombreuses solutions valides dont on se sert pour faire du multi-start sur la descente locale.

Enfin, ne pas faire de brassage tant que l'on n'a pas de solution valide semble permettre de trouver des solutions valides plus rapidement, ce qui expliquerait les meilleures performances de Intens_v à 15s et 30s.

5 CONCLUSION

Dans cet article, nous avons présenté une modélisation du problème d'affectation et d'ordonnancement des opérateurs lors des changements de séries simplifié en problème

d'affectation. Pour résoudre ce problème, nous avons utilisé une descente locale basée sur un voisinage qui représente des chaînes de réaffectations. Nous avons également mis au point un algorithme génétique. Ces deux algorithmes sont de performances quasiment égales en moyenne mais chacun a ses points forts et ses points faibles. Nous avons finalement mis au point un algorithme hybride qui utilise la descente locale et l'algorithme génétique. Cet algorithme hybride est bien plus performant que les deux autres à temps d'exécution égal. Ces algorithmes pourront servir de base à la résolution du problème réel qui possède un deuxième critère : l'entraînement des opérateurs sur toutes les machines.

REMERCIEMENTS

Les auteurs tiennent à remercier M.Bruno Valenti et M.Thierry Dufour de l'entreprise SKF pour nous avoir permis de travailler sur ce problème industriel et pour avoir mis à notre disposition toutes les informations dont nous avons eu besoin.

Références

- [1] BEASLEY D., BULL D.R. ET MARTIN R.R., *An overview of genetic algorithms : Part2, Research Topics*, University Computing, vol. 15, n°4, p.170-181, 1993.
- [2] DAUZÈRE-PÉRÈS S., ROUX W. ET LASERRE J.B. 1998, *Multi-resource shop scheduling with resource flexibility* European Journal of Operational Research, 107(1998)289-305.
- [3] ANTONIO FRANGIONI, EMILIANO NECCIARI, MARIO GRAZIA SCUTELLÀ *A multi-exchange neighborhood for minimum makespan machine scheduling problems*, TR-99-22, University of Pisa
- [4] DIRK VAN GOUBERGEN, HENDRIK VAN LANDEGHEM AND HANIF D.SHERALI *A quantitative Approach for Set-Up Reduction of Machine Lines IIE Research Conference 2004 conference in Huston, TX*
- [5] HOLLAND JOHN *Adaptation in Natural and Artificial Systems : An introductory Analysis with Applications to biology, Control, and Artificial Intelligence*. 1975 Ann Arbor MI : University of Michigan Press.
- [6] JURISH B., *Scheduling Jobs in Shops with Multi-purpose Machines*, PhD dissertation, University of Osnabruck (1992).
- [7] LIU MIN, WU CHENG *A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines*, Artificial Intelligence in Engineering, 13(1999)399-403
- [8] SHINGO, S. *A revolution in Manufacturing : the SMED system*. 1985, Cambridge, MA :Productivity Press.