Université de Montréal

La résolution du problème de formation de cellules dans un contexte multicritère

par
Mohamed Zaki AHADRI

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Janvier 2013

Résumé

Les techniques de groupement technologique sont aujourd'hui utilisées dans de nombreux ateliers de fabrication; elles consistent à décomposer les systèmes industriels en sous-systèmes ou cellules constitués de pièces et de machines. Trouver le groupement technologique le plus efficace est formulé en recherche opérationnelle comme un problème de formation de cellules. La résolution de ce problème permet de tirer plusieurs avantages tels que la réduction des stocks et la simplification de la programmation. Plusieurs critères peuvent être définis au niveau des contraintes du problème tel que le flot intercellulaire, l'équilibrage de charges intracellulaires, les coûts de sous-traitance, les coûts de duplication des machines, etc.

Le problème de formation de cellules est un problème d'optimisation NP-difficile. Par conséquent les méthodes exactes ne peuvent être utilisées pour résoudre des problèmes de grande dimension dans un délai raisonnable. Par contre des méthodes heuristiques peuvent générer des solutions de qualité inférieure, mais dans un temps d'exécution raisonnable.

Dans ce mémoire, nous considérons ce problème dans un contexte bi-objectif spécifié en termes d'un facteur d'autonomie et de l'équilibre de charge entre les cellules. Nous présentons trois types de méthodes métaheuristiques pour sa résolution et nous comparons numériquement ces métaheuristiques. De plus, pour des problèmes de petite dimension qui peuvent être résolus de façon exacte avec CPLEX, nous vérifions que ces métaheuristiques génèrent des solutions optimales.

Mots clés : Groupement technologique, optimisation multicritères, optimisation Pareto, problème de formation de cellules, heuristique.

Abstract

Group technology techniques are now widely used in many manufacturing systems. Those techniques aim to decompose industrial systems into subsystems or cells of parts and machines. The problem of finding the most effective group technology is formulated in operations research as the Cell Formation Problem. Several criteria can be used to specify the optimal solution such as flood intercellular, intracellular load balancing, etc. Solving this problem leads to several advantages such as reducing inventory and simplifying programming.

The Cell Formation Problem is an NP-hard problem; therefore, exact methods cannot be used to solve large problems within a reasonable time, whereas heuristics can generate solutions of lower quality, but in a reasonable execution time. We suggest in this work, three different metaheuristics to solve the cell formation problem having two objectives functions: cell autonomy and load balancing between the cells. We compare numerically these metaheuristics. Furthermore, for problems of smaller dimension that can be solved exactly with CPLEX, we verify that the metaheuristics can reach the optimal value.

Keywords : Group technology, multi-objective optimization, cell formation problem, heuristics, Pareto optimization.

Table des matières

Liste des tableaux

Liste des figures

Notation

m le nombre de machines

n le nombre de pièces

K le nombre de cellules

C une partition de machines sur les cellules

F une partition de pièces sur les cellules

(C,F) une solution du problème de formation de cellules

C_k le k-ème groupe de machines

F_k la k-ème famille de pièces

a_{ij} une entrée de la matrice d'incidence du problème de formation de cellules représentant la dépendance entre la i-ème machine et la j-ème pièce

(C°, F°) solution initiale du problème de formation de cellules

Eff la valeur objective du problème de formation de cellules

Introduction

Un circuit intégré à plusieurs nœuds est un composant informatique avec deux ou plusieurs unités centrales de traitement (appelés «noyaux») indépendants. Ces unités lisent et exécutent en parallèle des instructions ordinaires comme "add", "move", et "branch", et cela augmente la vitesse d'ensemble des programmes lancés sur le processeur.

Les fabricants intègrent généralement les noyaux sur une ou plusieurs puces de circuits intégrés appelés "chip multiprocessor"; c'est IBM qui a commercialisé le premier processeur multi-cœurs en 2001. Avant, les constructeurs de processeurs augmentaient la puissance de leurs produits en élevant la fréquence de calcul de leurs processeurs monocœurs; mais cette méthode a fini par atteindre ses limites, car cette augmentation cause des problèmes de surchauffe.

Le traitement multi-cœurs contribue à relever ce défi en augmentant les performances et la productivité des ordinateurs et des machines de petite taille tel que les appareils numériques. Ces systèmes deviennent capables d'exécuter simultanément plusieurs applications complexes et de réaliser davantage de tâches en moins de temps.

La conception d'un circuit intégré à plusieurs nœuds sur la même puce peut être considérée comme une intégration des technologies de base, d'architecture, de compilation, de vérification et d'affectation. Comme dans le cas de tout circuit intégré, trois composantes principales sont nécessaires pour une conception efficace du circuit à plusieurs nœuds: l'expression des besoins de l'utilisateur, la technologie de base sur laquelle repose la fabrication physique et l'architecture du circuit.

Dans ce travail, nous nous attaquons à un problème de conception du circuit à plusieurs nœuds. Le but est de trouver une bonne disposition des processeurs et unités de mémoires sur les nœuds de l'architecture pour garantir une bonne performance de calcul du circuit. La démarche consiste à partitionner l'ensemble des processeurs en groupes et l'ensemble des unités de mémoires en famille, pour ensuite les affecter aux nœuds du circuit.

Actuellement, la conception des puces multiprocesseurs se fait de façon expérimentale en se fiant à l'expérience du concepteur de puces. Ce mémoire se situe dans le cadre d'un projet visant à développer un système d'aide à la décision pour les concepteurs de circuits à

plusieurs nœuds. Ce système de support essaye de trouver la meilleure composition des nœuds, selon l'approche suivante :

1. Partition des processeurs et des unités de mémoires en groupes impliquant le plus d'interaction.
2. Affectation de ces groupes aux sommets du graphe représentant l'architecture de la puce.
3. Spécification de la capacité des liens reliant les sommets du graphe de l'architecture de la puce.

Nous étudions dans ce mémoire la première étape de cette approche: comment regrouper les processeurs et les mémoires qui doivent traiter les mêmes composants des applications. Il s'agit donc de décomposer la puce multiprocesseur en cellules (constituées de processeurs et mémoires) afin de transformer les applications complexes en sous applications moins complexes et faciles à résoudre, dans le but de maximiser l'efficacité de la puce. Ce problème est formulé en un problème de formation de cellules, bien connu en recherche opérationnelle.

Le problème de formation de cellules est un problème d'optimisation NP-difficile. Par conséquent, les méthodes exactes ne peuvent être utilisées pour résoudre des problèmes de grande dimension dans un délai raisonnable. Par contre, des méthodes heuristiques peuvent générer des solutions de qualité inférieure, mais dans un temps d'exécution raisonnable.

Dans ce mémoire, nous considérons ce problème dans un contexte bi-objectif spécifié en termes d'un facteur d'autonomie et de l'équilibre de charge entre les cellules. Nous présentons trois types de méthodes métaheuristiques pour sa résolution et nous comparons numériquement ces métaheuristiques. De plus, pour des problèmes de petite dimension qui peuvent être résolus de façon exacte avec CPLEX, nous vérifions que ces métaheuristiques génèrent des solutions optimales.

Nous débutons ce mémoire en présentant au premier chapitre la formulation du notre problème et les méthodes de résolution proposées dans la littérature. Ensuite, nous décrivons trois méthodes de résolution au niveau du deuxième chapitre. Ces méthodes seront analysées et comparées à l'aide de résultats numériques au troisième chapitre.

Chapitre 1 : Formulation du problème de formation de cellules

Dans cette section, nous présentons le problème de formation de cellules. Nous allons d'abord présenter la version binaire utilisée dans le cadre des circuits de production. Par la suite, nous élaborons la version entière comportant plusieurs objectifs. Nos méthodes de résolution s'inspirent de celles utilisées pour la version binaire.

1.1 Formulation du problème de formation de cellules binaire

Le problème de formation de cellules est apparu principalement dans les métiers de la fabrication mécanique tel que l'usinage, la tôlerie et l'assemblage. Les circuits de production peuvent engendrer un grand nombre de passages de pièces d'un atelier à un autre, engendrant des en-cours importants, des délais plus longs et une gestion plus complexe. D'où venue l'idée de former des cellules ou des îlots associant plusieurs ateliers à une famille de pièces. Il suffisait donc d'identifier ces familles en regroupant un nombre d'ateliers et de postes en un même lieu pour définir une cellule. Dans notre présentation, nous utiliserons les concepts de machines et de pièces pour décrire le problème.

Pour formuler le problème de formation de cellules, nous utilisons la notation suivante :

I: ensemble de *m* machines (*i* = 1, ..., *m*),
J: ensemble de *n* pièces (*j* = 1, ..., *n*),
K: ensemble de *K* cellules (*k* = 1, ..., *K*).
Dans notre problème, le nombre de cellules est fixé a priori.

La matrice d'incidence $A = \left[a_{ij} \right]$ indique les interactions entre les machines et les pièces de production avec $a_{ij} = 1$ si la machine *i* traite la pièce *j*, sinon $a_{ij} = 0$. Notons qu'une pièce *j* peut être traitée par plusieurs machines.

Une cellule de production *k* contient un sous ensemble (groupe) de machines $C_k \subset I$ et un sous ensemble (famille) de pièces $F_k \subset J$. Le problème est de déterminer un ensemble de cellules $(C,F) = \left\{ (C_1,F_1),\ldots,(C_K,F_K) \right\}$ en visant de rendre chaque cellule le plus

autonome possible. Notons qu'une solution définit aussi une partition des machines et des pièces :

$$C_1 \cup \ldots \cup C_K = I \quad et \quad F_1 \cup \ldots \cup F_K = J$$
$$\text{pour tout } k_1 \neq k_2 \in \{1,\ldots,K\} \qquad C_{k_1} \cap C_{k_2} = \phi \quad \text{et} \quad F_{k_1} \cap F_{k_2} = \phi.$$

Pour illustrer le concept de production des cellules, considérons la matrice d'incidence présentée ci dessous au tableau 1.

Pièces		1	2	3	4	5	6	7	8
Machines	1	0	1	0	1	1	1	0	1
	2	1	0	1	0	1	0	0	0
	3	1	0	1	0	0	0	1	0
	4	0	1	0	1	0	1	0	1
	5	1	0	0	0	0	0	1	1
	6	1	1	0	0	0	1	1	1

Tableau 1 : Matrice d'incidence

Le tableau 2 indique une solution qui définit une partition en trois différentes cellules illustrées dans les zones grises. La solution comprend les trois groupes de machines {(1, 4, 6), (3, 5), (2)} et les trois familles de pièces {(2, 4, 6, 8), (1, 7), (3, 5)}. Les éléments exceptionnels (1, 5), (6, 1), (6, 7), (3, 3), (5, 8) et (2, 1) correspondent à des entrées dans le tableau 2 ayant une valeur non nulle qui se trouve à l'extérieur des blocs gris.

Pièces		2	4	6	8	1	7	3	5
Machines	1	1	1	1	1	0	0	0	1
	4	1	1	1	1	0	0	0	0
	6	1	0	1	1	1	1	0	0
	3	0	0	0	0	1	1	1	0
	5	0	0	0	1	1	1	0	0
	2	0	0	0	0	1	0	1	1

Tableau 2 : Solution avec C =3

Pour mesurer l'efficacité d'une solution, différentes mesures ont été proposées. Dans l'article [33], les auteurs ont réalisé une étude comparative des différentes métriques possibles. Par la suite nous adopterons la formule utilisée par [13] :

$$Eff = \frac{a_1^{In}}{a + a_0^{In}}$$

Ici $a = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}$ représente la somme des entrées dans la matrice d'incidence, a_1^{In} représente la somme des entrées dans les cellules de la solution et a_0^{In} représente le nombre des entrées nulles dans les cellules.

Introduisons les variables binaires suivantes :

$$\text{pour tout } i = 1,\ldots,m;\ k = 1,\ldots,K$$
$$x_{ik} = \begin{cases} 1 & \text{si la machine } i \text{ appartient à la cellule } k, \\ 0 & \text{sinon} \end{cases}$$
$$\text{pour tout } j = 1,\ldots,n;\ k = 1,\ldots,K$$
$$y_{jk} = \begin{cases} 1 & \text{si la pièce } j \text{ appartient à la cellule } k, \\ 0 & \text{sinon} \end{cases}$$

Ces variables nous permettent de formuler a_1^{In} et a_0^{In} comme suit :

$$a_1^{In} = \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ik} y_{jk},$$
$$a_0^{In} = \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(1 - a_{ij}\right) x_{ik} y_{jk}.$$

Cela nous emmène au modèle suivant formulé dans [13] :

$$\text{Max} \quad \frac{\sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ik} y_{jk}}{a + \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} (1 - a_{ij}) x_{ik} y_{jk}}$$

$$\text{Sujet à} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,m \qquad (1)$$
$$\sum_{k=1}^{K} y_{jk} = 1 \qquad j = 1,\ldots,n \qquad (2)$$
$$\sum_{i=1}^{m} x_{ik} \geq 1 \qquad k = 1,\ldots,K \qquad (3)$$
$$\sum_{j=1}^{n} y_{jk} \geq 1 \qquad k = 1,\ldots,K \qquad (4)$$
$$x_{ik} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; k = 1,\ldots,K \qquad (5)$$
$$y_{jk} = 0 \text{ or } 1 \qquad j = 1,\ldots,n; k = 1,\ldots,K \qquad (6)$$

Les contraintes (1) et (2) nous assurent que chaque machine et chaque pièce sont affectées à une seule cellule; les contraintes (3) et (4) nous garantissent que chaque cellule comporte au

moins une machine et une pièce et les variables sont binaires pour respecter les contraintes (5) et (6).

Le problème de formation de cellules binaire est connu pour être NP-difficile [32]; la plupart des méthodes qui ont été proposées pour sa résolution sont basées sur des heuristiques ou des métaheuristiques pour générer de bonnes solutions avec un temps de calcul raisonnable. Pour en savoir plus sur les différentes méthodes, nous nous référons aux articles 19, 35, 18, où les auteurs passent en revue les différentes techniques de résolution classées comme suit:

- Le partitionnement de données : des techniques pour reconnaître la structure d'un ensemble de données.
- Les approches de partitionnement de graphe, où un graphe est utilisé pour formuler le problème de formation de cellules.
- Les méthodes de programmation mathématique: le problème de formation de cellules est formulé comme un problème de programmation non linéaire ou linéaire en nombres entiers.
- Heuristiques et métaheuristiques : les méthodes les plus populaires sont le recuit simulé [49], la recherche tabou, l'algorithme génétique [36], l'algorithme de colonies de fourmis [30].

Dans l'article [45], trois méthodes de résolution sont comparées. La première est une implémentation du recuit simulé, la deuxième est une implémentation du recuit simulé adapté, alors que la troisième est une méthode hybride qui combine un algorithme de recherche locale avec un algorithme génétique. Nous allons nous inspirer de ces méthodes pour résoudre la version entière du problème.

1.2 Le problème de formation de cellules entier multi objectif

Dans le contexte de puce multiprocesseur, le problème de trouver la partition optimale de processeurs et de mémoires est formulé comme un problème de formation de cellules entier. Ce problème diffère du problème de formation de cellules binaire par sa matrice d'incidence, puisque les éléments a_{ij} de cette matrice prennent des valeurs entières représentant le niveau d'interaction entre le processeur et la mémoire. Une deuxième différence se trouve au niveau des objectifs, puisque l'efficacité d'une solution tient compte de plusieurs facteurs.

Au niveau de notre présentation, nous considérons les deux facteurs suivants :

- Facteur de l'autonomie : $f_1(x,y) = \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ik} y_{jk}$

- Facteur du l'équilibrage de charge : $f_2(x,y)=\sum_{k=1}^{K}\left|T-\sum_{i=1}^{m}\sum_{j=1}^{n}a_{ij}x_{ik}y_{jk}\right|$ où $T=\dfrac{\sum_{i=1}^{m}\sum_{j=1}^{n}a_{ij}}{K}$

Le premier facteur représente l'autonomie possible que nous cherchons à maximiser. L'autonomie d'une cellule s'exprime comme le niveau d'interaction inter-cellule entre les processeurs et les unités mémoires la constituant. Le deuxième facteur que nous cherchons à minimiser s'exprime comme la somme des déviations entre les charges de cellules et une moyenne de charge *T* cible. Le but est de pénaliser les cellules surchargées ou sous-chargées pour atteindre l'équilibre entre les cellules.

Nous avons donc un problème comportant deux objectifs qui peuvent être conflictuels. Dans la prochaine section, nous résumons les grandes lignes de l'optimisation multi objectif.

1.3 L'optimisation multi-objectif

Les problèmes multi-objectifs ont un intérêt grandissant dans l'industrie, où les responsables sont contraints à tenter d'optimiser des objectifs contradictoires dans le sens où la solution optimale pour un objectif ne l'est pas pour les autres. Cela implique la nécessité de définir la notion qui correspond à celle d'optimalité et les méthodes de résolution pour identifier les solutions ayant cette propriété.

Nous considérons la formulation suivante d'un problème multi objectif :

$$(MP)\quad \text{Min}\quad G(x)=\left(g_1(x),g_2(x),\ldots,g_p(x)\right)$$
$$\text{Sujet à}\ \ f_i(x)\le 0 \qquad i=1,\ldots,m$$
$$x\in X\subseteq R^n$$

où $g_l(x)$ est la fonction économique de l'objectif $l=1,\ldots,p.$

Dans le contexte multi-objectif, au lieu de la notion de solution optimale, nous parlons de solution efficace [42]. Le point x^* est efficace s'il satisfait l'une ou l'autre des conditions suivantes pour tout autre point réalisable de (MP) :

$$\left\{\begin{array}{ll}\text{Si} & g_l(x)\le g_l(x^*) \qquad l=1,\ldots,p\\ \text{alors} & g_l(x)=g_l(x^*) \qquad l=1,\ldots,p\end{array}\right\}$$

ou

$$\left\{\begin{array}{l}\text{S'il existe un indice } \tilde{l},\ 1\le\tilde{l}\le p,\ \text{où}\ g_{\tilde{l}}(x)<g_{\tilde{l}}(x^*),\\ \text{alors il existe un indice } \underset{\sim}{l},\ 1\le\underset{\sim}{l}\le p,\ \text{où}\ g_{\underset{\sim}{l}}(x)>g_{\underset{\sim}{l}}(x^*)\end{array}\right\}$$

En somme, un point est efficace s'il est impossible d'améliorer un des objectifs sans en détériorer un autre. C'est le principe de la Pareto-optimalité.

Deux approches classiques de résolution peuvent être utilisées pour la résolution du problème multi objectif (MP). La première est l'approche de pondération, où le décideur spécifie un poids λ_l relatif pour chaque objectif $g_l(x)$ de façon que plus l'objectif est important par rapport aux autres, plus le poids est élevé. Ces poids permettent de résoudre le problème suivant :

$$(MP_\lambda) \qquad \text{Min} \quad \sum_{l=1}^{p} \lambda_l g_l(x)$$
$$\text{Sujet à} \quad f_i(x) \leq 0 \qquad i = 1, \ldots, m$$
$$x \in X \subseteq R^n$$

Les poids choisis par le décideur vérifient $\sum_{l=1}^{p} \lambda_l = 1$.

Le problème (MP_λ) est un problème avec un seul objectif, et sa solution optimale correspond à une solution efficace de (MP).

La deuxième méthode de résolution est la méthode de priorité, où le décideur veut éviter d'améliorer un objectif moins prioritaire aux dépens d'un autre qu'il l'est davantage. Ce qu'il cherche avant tout est d'optimiser l'objectif le plus prioritaire. S'il y a plusieurs solutions optimales pour cet objectif, alors il retient parmi celles-ci, celles qui optimisent le deuxième objectif plus prioritaire, etc.

1.3.1 L'optimisation Pareto

La notion d'optimum de Pareto permet de partitionner le domaine réalisable d'un problème multi-objectif en deux parties : les solutions améliorables et celles qui ne le sont pas. Ainsi, une solution est améliorable si la valeur d'un des objectifs peut être améliorée sans détériorer celles des autres objectifs. Par contre, pour les solutions de la deuxième classe, l'amélioration de la valeur d'un objectif implique toujours la dégradation de celle d'un des autres objectifs. La première classe des solutions sont les solutions efficaces ou non dominées et ce sont les solutions qu'on désigne comme optimales au sens de Pareto ou Pareto-optimales. Ces solutions constituent la frontière de Pareto [1].

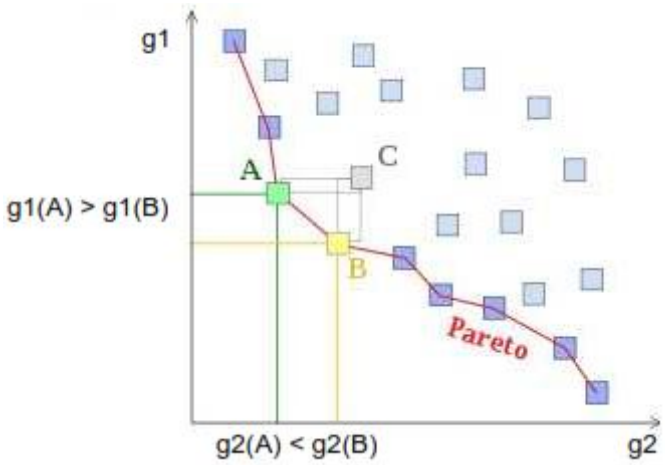

Figure 1 : Exemple de points pareto-optimaux

La figure 1 illustre un problème d'optimisation avec deux objectifs g1 et g2 où l'espace de recherche est représenté par l'ensemble de points carrés sur le graphe. L'ensemble de points pareto-optimaux sont les points qui se situent sur la courbe passant par les points A et B. Le point C n'est pas un point efficace, car il est dominé par le point A puisque g1(A)<g1(C) et g2(A)<g2(C).

Dans la résolution d'un problème multi-objectif, un décideur doit participer à la prise de décision, et pour ce faire, l'une de ces deux approches doit être choisie : optimisation a priori et optimisation a posteriori. Pour l'optimisation a priori, le décideur est consulté d'abord avant de procéder à l'utilisation d'une des deux approches présentées dans la section 1.3. La meilleure solution trouvée avec ce modèle ne requiert plus de nouvelle contribution du décideur par la suite.

L'optimisation a posteriori aborde le problème d'un angle inverse, en commençant par la résolution du problème multi-objectif pour trouver des solutions efficaces afin d'identifier la courbe Pareto. En fait, la notion d'efficacité utilisée dans le contexte multi-objectif correspond à celle d'optimalité pour un problème d'optimisation. Le décideur choisit ensuite parmi les solutions constituantes la courbe Pareto.

1.3.2 L'approche de résolution pour notre problème

Le but de notre projet est la résolution du problème de formation de cellules avec deux objectifs (autonomie et équilibre de charge). Nous adoptons l'approche d'optimisation a posteriori où le paramètre $0 \leq \gamma \leq 1$ définit la priorité du premier objectif et $1-\gamma$ définit celle

du deuxième. Le problème équivalent du problème de formation de cellule entier multi-objectif :

$$\text{Max } Eff = \gamma \sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk} - (1-\gamma)\sum_{k=1}^{K}\left| T - \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk} \right|$$

$$(PE_\gamma) \quad \text{Sujet à} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,m \qquad (1)$$

$$\sum_{k=1}^{K} y_{jk} = 1 \qquad j = 1,\ldots,n \qquad (2)$$

$$\sum_{i=1}^{m} x_{ik} \geq 1 \qquad k = 1,\ldots,K \qquad (3)$$

$$\sum_{j=1}^{n} y_{jk} \geq 1 \qquad k = 1,\ldots,K \qquad (4)$$

$$x_{ik} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; k = 1,\ldots,K \qquad (5)$$

$$y_{jk} = 0 \text{ or } 1 \qquad j = 1,\ldots,n; k = 1,\ldots,K \qquad (6)$$

Toute solution optimale de (PE_γ) est un point efficace du problème de formation de cellules entier multi-objectif pour toute valeur du paramètre γ. Il suffit donc de résoudre (PE_γ) pour obtenir des points efficaces du problème de cellule entier multi-objectif pour diverses valeurs de γ. Le deuxième chapitre présente des méthodes de résolution proposées pour le probleme (PE_γ) ; ce sont des heuristiques qui ne donnent pas forcément des solutions exactes, mais nous testerons l'efficacité des solutions relativement à l'optimalité Pareto. Idéalement, l'ensemble de solutions de (PE_γ) doit definir des points non dominés pour qu'on puisse se rapprocher le mieux possible de solutions Pareto-optimales du problème de formation de cellule entier mutli-objectif.

La figure 2 ci-dessous illustre ce scénario, chaque point de la courbe est une solution de (PE_γ) pour une valeur de γ variant entre 0 à 1 en utilisant un pas de 0,1. Les résultats numériques obtenus en utilisant des méthodes métaheuristiques indiqueront que, pour les problèmes tests, les courbes sont de la forme illustrée à la figure 2, même si les solutions obtenues pour certaines valeurs γ de ne sont pas toujours efficaces, puisque elles sont obtenues avec des métaheuristiques.

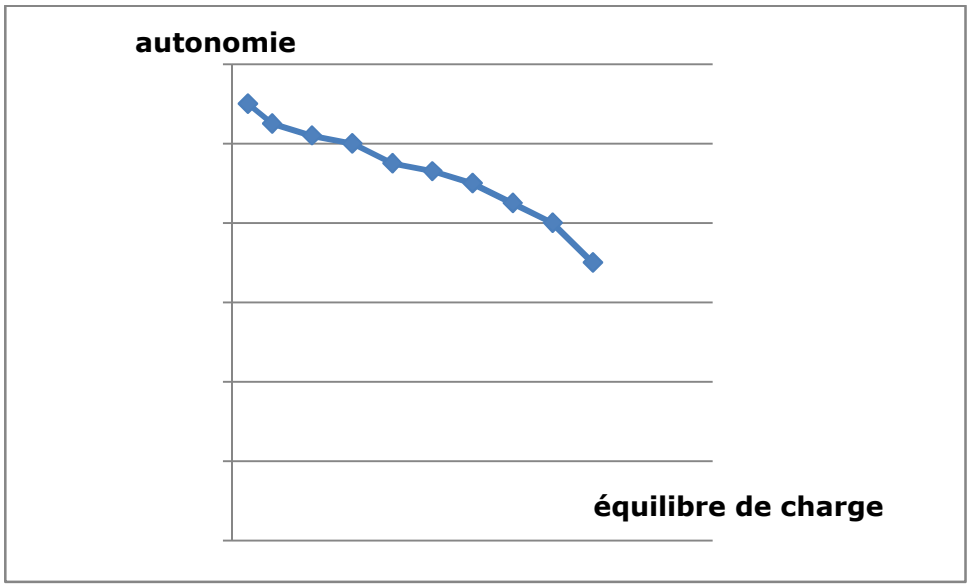

Figure 2 : Résultat idéal obtenu par une méthode de résolution du (PE_γ)

Chapitre 2 : Méthodes de résolution pour le problème de formation de cellules entier multi objectif

Rappelons que nous avons indiqué dans le chapitre précédent que nous utilisons l'approche où la fonction économique est exprimée comme une combinaison linéaire convexe des deux objectifs. Ainsi, nous pouvons identifier une partie des solutions Pareto-optimales en considérant une suite de valeurs $\gamma \in [0,1]$ pour le problème (PE_γ).

$$\text{Max } Eff = \gamma \sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk} - (1-\gamma)\sum_{k=1}^{K}\left| T - \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ik}y_{jk} \right|$$

(PE_γ) Sujet à

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,m \qquad (1)$$

$$\sum_{k=1}^{K} y_{jk} = 1 \qquad j = 1,\ldots,n \qquad (2)$$

$$\sum_{i=1}^{m} x_{ik} \geq 1 \qquad k = 1,\ldots,K \qquad (3)$$

$$\sum_{j=1}^{n} y_{jk} \geq 1 \qquad k = 1,\ldots,K \qquad (4)$$

$$x_{ik} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; k = 1,\ldots,K \qquad (5)$$

$$y_{jk} = 0 \text{ or } 1 \qquad j = 1,\ldots,n; k = 1,\ldots,K \qquad (6)$$

m, *n* et *K* représentent respectivement le nombre de machines, de pièces et de cellules.

Nous avons adopté cette stratégie de résolution pour deux raisons principales. Premièrement, elle permet aux utilisateurs d'avoir plus de flexibilité dans le choix des poids des facteurs (autonomie et équilibre de charge) pour obtenir une solution efficace répondant à leurs besoins. Deuxièmement, nous pouvons baser nos méthodes de résolution pour le problème (PE_γ) sur celles développées dans [45] pour le problème de formation de cellules binaire.

Plusieurs méthodes ont été proposées pour la résolution du problème de formation de cellules entier multi objectif, mais avec des objectifs à optimiser différents des nôtres, tel que la minimisation des éléments exceptionnels et des éléments vides dans les cellules. D'autres méthodes partent d'une formulation différente du problème, en ajoutant par exemple des capacités aux machines et des distances entre les machines. Ces méthodes adoptent une approche de résolution a posteriori, qui consiste en la recherche d'un ensemble de solutions

efficaces pour ensuite laisser au décideur le choix parmi ces solutions. Nous pouvons classer les algorithmes utilisés en deux classes. La première classe utilise des heuristiques pour la recherche locale de la solution efficace telle que la recherche tabou [4] et le recuit simulé [43]. La deuxième classe d'approches de résolution est basée sur les algorithmes évolutionnistes, et plus précisément sur les algorithmes génétiques multi-objectif [48], [2] et [20].

Dans la suite de ce chapitre, nous présentons trois types de méthodes métaheuristiques pour résoudre le problème (PE_γ): une méthode de recherche locale, une implémentation du recuit simulé et une méthode hybride intégrant une recherche locale dans un algorithme génétique. Une comparaison de ces méthodes est ensuite complétée à l'aide de tests numériques dans le troisième chapitre.

2.1 Méthode de la recherche locale

La recherche locale ("local search algorithm" : LSA) est une métaheuristique utilisée pour résoudre des problèmes d'optimisation difficiles. Les algorithmes de recherche locale passent d'une solution à une autre dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution jugée très bonne soit trouvée ou que le temps imparti soit dépassé. La recherche part d'une solution candidate et permet de la déplacer de façon itérative vers une solution voisine.

Dans l'algorithme de recherche locale que nous proposons pour résoudre le problème (PE_γ), le déplacement entre les solutions voisines est basé sur deux procédures qui sont appelées successivement. En effet, pour une solution courante, nous appliquons une stratégie d'intensification qui consiste à modifier successivement les groupes de machines et les familles des pièces jusqu'à ce qu'aucune modification ne soit possible. Ensuite, nous appliquons une stratégie de diversification pour modifier les groupes et les familles de la solution courante. L'approche peut être résumée comme suit :

Étape 1 : Générer une solution réalisable initiale (C°, F°)

Initialiser la solution courante $(C\ , F\ ) \leftarrow (C^{\circ}, F^{\circ})$, aller à l'étape 2a

Étape 2 : Intensification

2a. Modifier les familles de pièces sur la base des groupes de machines
Si aucune modification n'est possible aller à 3a
Sinon aller à 2b

2b. Modifier les groupes de machines sur la base des familles de pièces
Si aucune modification n'est possible aller à 3b
Sinon aller à 2a

Étape 3 : Diversification

3a. Réaffecter aléatoirement un nombre q de pièces, aller à 2b

3b. Réaffecter aléatoirement un nombre q de machines, aller à 2a

Figure 3 : Algorithme du LSA

La meilleure solution est mise à jour chaque fois qu'une nouvelle solution est générée, et l'algorithme s'arrête après l'application de l'étape 3 pour un nombre fixe de fois *numstrat*. Les différentes procédures sont maintenant résumées dans les sections suivantes.

2.1.1 Génération de la solution initiale

Pour générer la solution initiale, nous utilisons un algorithme glouton similaire à celui proposé dans [13]. Nous déterminons les *K* groupes de machines $C_1^{\circ} \ldots C_K^{\circ}$, et après nous déterminons l'ensemble de *K* familles de pièces $F_1^{\circ} \ldots F_K^{\circ}$ sur la base de ces groupes.

Notons $a_{i\bullet} = \sum_{j=1}^{n} a_{ij}$ et $a_{\bullet j} = \sum_{i=1}^{m} a_{ij}$ le nombre de pièces traitées par la machine *i* et le nombre de machines qui traitent la pièce *j*, respectivement. Pour les groupes de machines, on sélectionne les *K* machines ayant les plus grandes valeurs $a_{i\bullet}$ pour les assigner aux groupes C_i°. Pour les *m*-*K* machines restantes, chacune est assignée au groupe traitant le maximum des pièces traitées par cette machine. Plus précisément, notons *INA* l'ensemble des machines restantes. L'affectation de cet ensemble se passe comme suit :

Étape 1 :Pour toute machine $i \in INA$, nous calculons

$\bar{k}(i) = Min_{k=1..K}\{\frac{1}{\left|C_k^\circ\right|}\sum_{j=1}^{n}\sum_{i_k \in C_k^\circ}\left|a_{ij} - a_{i_k j}\right|\}$ // $\left|C_k^\circ\right|$ nombre de machines du groupe C_k°

$gr_i = ArgMin_{k=1..K}\{\frac{1}{\left|C_k^\circ\right|}\sum_{j=1}^{n}\sum_{i_k \in C_k^\circ}\left|a_{ij} - a_{i_k j}\right|\}$

Étape 2 : $\bar{i} = ArgMin_{i \in INA}\{\bar{k}(i)\}$

$C_{gr_{\bar{i}}}^\circ = C_{gr_{\bar{i}}}^\circ \cup \{\bar{i}\}$ //affectation de la machine $\bar{i}$ au groupe de machines $C_{\mathrm{gr}_{\bar{i}}}^\circ$

$INA = INA - \{\bar{i}\}$

si $INA = \varnothing$ *Stop* sinon aller à Étape 1

À ce niveau, les groupes C_1° .....C_K° sont définis, et pour définir les familles de pièces F_1° ...F_C° nous utilisons la même procédure d'intensification 2a décrite dans la section suivante.

2.1.2 Intensification

Intensifier une solution courante dans l'algorithme du LSA consiste à modifier les familles de pièces (étape 2a), puis à modifier les groupes de machines (étape 2b) d'une manière itérative, jusqu'au moment où nous ne pouvons plus modifier la solution courante ou bien arriver à un nombre limite d'itérations *maxrepeat*. Les procédures pour modifier les groupes de machines sur la base des familles de pièces et les familles de pièces sur la base des groupes de machines sont identiques, car les machines et les pièces jouent un rôle similaire au niveau du problème. Nous présentons donc uniquement la procédure affectant les pièces sur la base de groupes de machines. Deux méthodes sont proposées pour trouver les nouvelles affectations des pièces : la méthode GRASP ("Greedy Randomized Adaptive Search Procedure") et la méthode SW ("Squeaky Wheel").

2.1.2.1 Intensification avec GRASP

GRASP est une méthode heuristique introduite dans [16] pour construire une solution réalisable d'un problème combinatoire en combinant des approches gloutonnes et aléatoires. La solution est construite de façon itérative, et à chaque étape la valeur d'une nouvelle variable est fixée. À chaque étape, étant donné les valeurs des variables déjà fixées, la fonction économique est évaluée en fixant chacune des variables restantes. Un sous-ensemble des variables restantes générant les meilleures valeurs de la fonction économique est identifié. Ceci correspond en fait à la partie gloutonne du processus, mais ce n'est pas nécessairement la

variable restante générant la meilleure valeur de la fonction économique qui est fixée. Nous choisissons plutôt aléatoirement une variable du sous ensemble identifié, dont nous fixons la valeur.

Dans le contexte de l'affectation de pièces sur la base des groupes de machines, nous sélectionnons une pièce *j* parmi les pièces non encore affectées pour l'affecter à la cellule *k*. Le choix de la pièce à affecter est basé sur sa contribution ∂_{jk} dans la fonction économique.

A une itération donnée, soit *JA* l'ensemble de pièces déjà affectées. La valeur objective actuelle est: $Eff(JA) = \gamma \sum_{j \in JA} \sum_{k} \sum_{i \in C_k} a_{ij} y_{jk} - (1-\gamma) \sum_{k} | T - \sum_{j \in JA} \sum_{i \in C_k} a_{ij} y_{jk} |$

Notons $l_{jk} = \sum_{i \in C_k} a_{ij}$ et $T_k = \sum_{j \in JA} l_{jk} y_{jk}$, l'expression de la valeur de l'objectif devient:

$Eff(JA) = \gamma \sum_{j \in JA} \sum_{k} l_{jk} y_{jk} - (1-\gamma) \sum_{k} | T - T_k |$.

Si on décide d'affecter la pièce $\bar{j}$ à la cellule $\bar{k}$, la valeur de la fonction économique deviendra: $Eff(JA \cup \{\bar{j}\}) = \gamma(\sum_{j \in JA} \sum_{k} l_{jk} y_{jk} + l_{\bar{j}\bar{k}}) - (1-\gamma)(\sum_{k \neq \bar{k}} | T - T_k | + | T - T_{\bar{k}} - l_{\bar{j}\bar{k}} |)$

Donc la contribution de l'affectation de la pièce *j* à la famille *k* est : $\partial_{jk} = Eff(JA \cup \{j\}) - Eff(JA) = \gamma l_{jk} - (1-\gamma)(| T - T_k - l_{jk} | - | T - T_k |)$

L'approche utilisée par l'algorithme GRASP pour affecter les pièces se base sur les valeurs de ∂_{jk} pour les affecter en choisissant d'une manière itérative une pièce parmi les pièces restantes qui ont la plus grande valeur de ∂_{jk}. L'algorithme est résumé comme suit:

Étape 0 //initialisation

$JNA = \{1..n\}$ // l'ensemble de pièces à affecter

$For k = 1..K$

$T_k = 0$

$For j = 1..n$

$l_{jk} = \sum_{i \in C_k} a_{ij}$ // $C_k =$ le groupe de machines *k*

Étape 1

$For\ j \in JNA$

$For k = 1..K$

$$\partial_{jk} = \gamma l_{jk} - (1-\gamma)\left[|T - T_k - l_{jk}| - |T - T_k| \right]$$

Étape 2

$For\ j \in JNA$

$$k_j = \underset{k=1..K}{ArgMax}\{\partial_{jk}\}$$

$$\partial^* = \underset{j \in JNA}{Max}\ \partial_{jk_j}$$

Étape 3

sélectionner aléatoirement $j \in \left\{ j \in JNA : \partial_{jk_j} >= \alpha\partial^* \right\}$ //$0 \le \alpha \le 1$ paramètre du GRASP

$F_{k_j} = F_{k_j} \cup \{j\}$ // mise à jour de la famille F_{k_j}

$T_{k_j} = T_{k_j} + l_{jk_j}$

$JNA = JNA - \{j\}$

$For\ i \in JNA$

$$\partial_{ik_j} = \gamma l_{ik_j} - (1-\gamma)\left[|T - T_{k_j} - l_{ik_j}| - |T - T_{k_j}| \right]$$

si $JNA = \varnothing$ *Stop* sinon aller à Étape 2

Au niveau de la première étape, nous initialisons la contribution de chaque pièce j lorsqu'elle est affectée à la cellule k et nous calculons ∂^* la meilleure contribution à la fonction économique au cours de la deuxième étape. Nous sélectionnons aléatoirement une pièce j dont la contribution est supérieure à $\alpha\partial^*$ pour l'affecter à la cellule k_j. Enfin, nous mettons à jour la famille de pièces non affectées (JNA) et leur contribution (∂_{jk_j}) lorsqu'affectées à la cellule k_j.

2.1.2.2 Intensification avec SW

L'algorithme SW [21] est une méthode de résolution itérative qui construit les solutions par l'application successive du cycle Construire/Analyser/Définir les priorités, illustré dans la figure 4. La première étape consiste à construire la solution avec une procédure gloutonne en priorisant les variables de la solution ayant les plus grandes priorités.

La solution construite est ensuite analysée pour calculer la pénalité de chaque variable. La pénalité associée à une variable correspond au regret de ne pas utiliser sa pleine contribution dans la fonction économique. La prochaine étape utilise ces regrets pour définir les nouvelles priorités des variables, en mettant à jour les priorités de l'itération précédente. Ce cycle se répète jusqu'à ce qu'une condition d'arrêt se produise, en général basée sur un nombre limite de cycles.

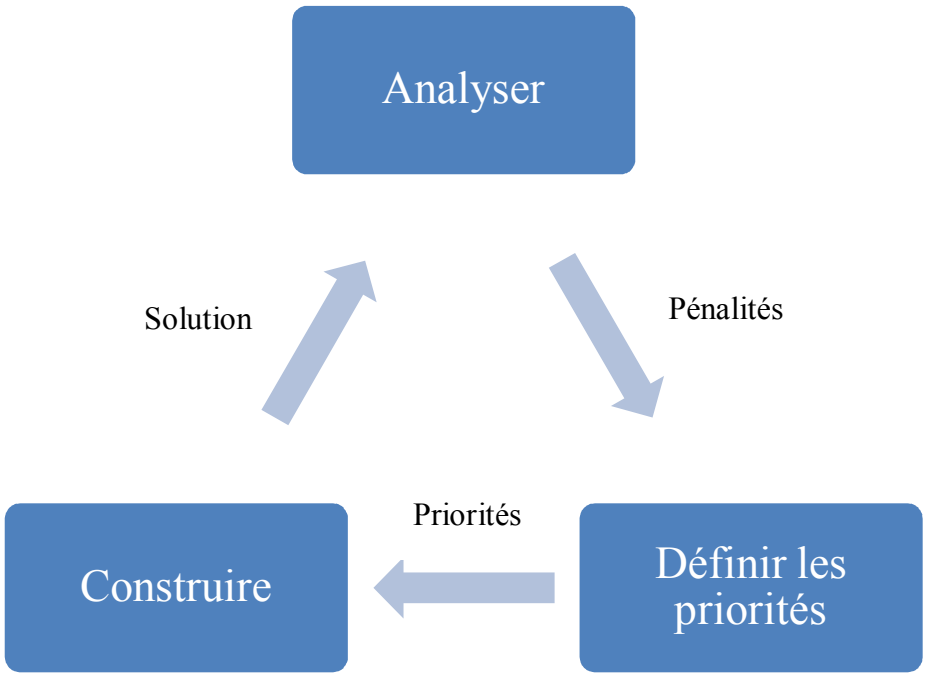

Figure 4 : Cycle de l'algorithme SW

Pour adapter la méthode du SW au contexte de l'affectation de pièces sur la base des groupes de machines, chaque pièce est considérée comme une variable et sa pénalité est égale à la combinaison convexe de son ancienne pénalité et du regret de ne pas utiliser pleinement sa contribution maximale à la fonction économique dans la solution courante. À la base de cette liste de pénalités, on construit la nouvelle solution par l'affectation des variables (pièces) dans l'ordre décroissant de la liste des pénalités. L'algorithme SW est résumé comme suit :

Étape 1:

$For\ j \in 1..n$

$P_j = 0\ ,\ Z_j = 0$ // P_j la pénalite de la pièce j

$For\ k = 1..K$

$For\ j = 1..n$

$l_{jk} = \sum_{i \in C_k} a_{ij}$ // $C_k =$ le groupe de machines k

$For\ k = 1..K$

$For\ j \in 1..n$

$\partial_{jk} = \gamma l_{jk} - (1-\gamma)\left[\,|T - l_{jk}| - T\right]$

$For\ j \in 1..n$

$k_j = \underset{k=1..C}{ArgMax}\{\partial_{jk}\}$ //∂_{jk_j} la contribution maximale de la pièce j

iter = 0

Étape 2

iter ++

$For\ j \in 1..n$

$\bar{k}_j =$ la famille actuelle de la pièce j

$$Z_j = \gamma l_{j\bar{k}_j} - (1-\gamma)\left[\,|T - l_{j\bar{k}_j} - \sum_{h=1/h\neq j}^{n} l_{h\bar{k}_j}| - |T - \sum_{h=1/h\neq j}^{n} l_{h\bar{k}_j}|\right]$$

$P_j := \lambda P_j + (1-\lambda)(\partial_{jk_j} - Z_j)$ //nouvelle pénalite de la pièce j

Étape 3:

$JNA = \{j_1, .. j_n\}$ l'ensemble de pièces classées dans l'ordre décroissant de P_j

Affecter les pièces dans leur ordre dans *JNA*

si *iter* =*numiter Stop* sinon aller à Étape 2

Figure 5 : Algorithme SW

Au niveau de la première étape, nous calculons la contribution maximale ∂_{jk_j} de chaque pièce j à la valeur économique du problème. À l'étape 2, nous déterminons Z_j qui représente la contribution actuelle de la pièce j à la valeur économique du problème dans la solution courante. Ensuite, nous calculons la nouvelle pénalité P_j s'exprimant comme une

combinaison convexe de la pénalité précédente et de $\partial_{jk_j} - Z_j$ représentant le regret de ne pas utiliser la pleine contribution de la pièce j dans la solution courante.

Nous classons les pièces dans l'ordre décroissant de leurs pénalités, pour ensuite les affecter à l'aide de la méthode résumée comme suit :

Étape 1 //initialisation

$JNA = \{j_1, .. j_n\}$ l'ensemble de pièces classées dans l'ordre décroissant de P_j

$For\ k = 1..K$

$T_k = 0$

$For\ j = 1..n$

$l_{jk} = \sum_{i \in C_k} a_{ij}$ // $C_k =$ le groupe de machines k

$For\ j \in JNA$

$For\ k = 1..K$

$\partial_{jk} = \gamma l_{jk} - (1-\gamma)\left[|T - l_{jk}| - T \right]$

$h = 1$ //indexe de la pièce à affecter

Étape 2 // affectation de la pièce j_h

$k_h = ArgMax_{k=1..K}\{\partial_{ik}\}$

$F_{k_h} = F_{k_h} \cup \{j_h\}$ // mise à jour de la famille F_{k_h}

$T_{k_h} = T_{k_h} + l_{j_h k_h}$

$For\ j = h+1..n$

$\partial_{jk_h} = \gamma l_{jk_h} - (1-\gamma)\left[|T - T_{k_h} - l_{jk_h}| - |T - T_{k_h}| \right]$

h++

si $h > n$ *Stop* sinon aller à Étape 2

Figure 6 : Affectation de pièces au niveau du SW

:

Il s'agit d'une procédure gloutonne pour affecter les pièces dans l'ordre pré-calculé au niveau de l'algorithme SW. Cette procédure s'inspire de celle de la méthode GRASP où chaque pièce j doit être affectée à la famille k qui maximise la quantité ∂_{jk} . La seule différence est que nous ne choisissons pas la pièce à affecter d'une manière aléatoire, mais plutôt dans l'ordre de pièces défini par l'ensemble *JNA*.

2.1.3 Diversification

Nous faisons appel à la diversification au moment où nous ne pouvons plus intensifier la solution courante, dans le but de s'éloigner du voisinage de la solution actuelle pour relancer la recherche locale à partir d'une nouvelle solution initiale. Ainsi, nous favorisons une meilleure exploration du domaine réalisable.

La diversification utilisée dans LSA consiste à réaffecter respectivement un nombre q de pièces et de machines aux étapes 3a et 3b. Les pièces ou les machines sont sélectionnées de façon aléatoire, et leur réaffectation aux familles ou aux groupes se fait aussi de façon aléatoire.

2.2 Méthode du recuit simulé

Le recuit simulé (SA : "Simulated Annealing") est une métaheuristique inspirée d'un processus utilisé en métallurgie [23]. Dans cette approche, nous passons d'un cycle à l'autre en refroidissant le système dans le but de minimiser l'énergie du matériau. Cette méthode est transposée en optimisation pour trouver les extrema d'une fonction. La méthode vient du constat que le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide. La configuration la plus stable est atteinte en contrôlant le refroidissement et en le ralentissant par un apport de chaleur externe.

Dans un contexte d'optimisation, nous cherchons à chaque itération une nouvelle solution dans le voisinage de la solution actuelle. Soit que cette nouvelle solution améliore le critère que l'on cherche à optimiser, et qu'elle fait baisser l'énergie du système, soit qu'elle le dégrade. Nous améliorons le critère dans le voisinage de la solution courante si la nouvelle solution est meilleure. Par contre, en acceptant une « mauvaise » solution, ceci permet d'explorer une plus grande partie de l'espace des solutions et d'éviter de s'enfermer trop vite dans le voisinage d'un optimum local.

Nous utilisons l'implémentation que nous retrouvons dans [17], pour résoudre le problème (PE_γ). L'algorithme utilisé peut être résumé comme suit :

Étape 1 :

Générer une solution réalisable initiale (C°, F°)

Initialiser la température initiale TP°

iter=0,TP=TP°, $fcount = 0$

$(C\ , F\ )=(C^{*}, F^{*}) = (C^{\circ}, F^{\circ})$, stop = false

Étape 2 :

changes=0 , trials = 0

Étape 3 :

générer une solution $(C^{'}, F^{'})$ dans le voisinage de $(C\ , F\ )$

$\Delta = Eff(C^{'}, F^{'}) - Eff(C\ , F\ )$ // différence de valeurs de l'objectif

Si Δ >0 alors $(C\ , F\ ) = (C^{'}, F^{'})$, changes++

Sinon générer un nombre aléatoire r $\in (0,1)$

Si r< $\exp(\frac{\Delta}{TP})$ alors $(C\ , F\ ) = (C^{'}, F^{'})$, changes++

Si $Eff(C^{'}, F^{'}) > Eff(C^{*}, F^{*})$ alors $(C^{*}, F^{*}) = (C^{'}, F^{'})$, fcount = 0

trials++

Si trials < SF *and* changes < coff reprendre l'Étape 3 sinon aller à Étape 4

Étape 4 :

TP=βTP , iter++

Si changes/trials<mpc alors fcount++

Si iter ≥ itermax *or* fcount=flimit alors Stop sinon aller à Étape 2

(C^{*}, F^{*}) solution générée

Figure 7 : Algorithme SA

Dans cette adaptation du recuit simulé, nous complétons plusieurs itérations avec la même température *TP* et cette température est modifiée lorsque le nombre des essais (*trials*) ou lorsque le nombre de modifications de la solution actuelle (*changes*) dépassent les seuils *Sf* ou *coff*, respectivement. Le paramètre β est utilisé pour modifier la température.

Deux critères d'arrêt sont utilisés. Le premier est fixé en fonction du nombre de différentes valeurs de la température utilisée (*itermax*). Pour le deuxième critère, nous gardons une trace du nombre de valeurs des températures consécutives (*fcount*), où le nombre de changements par rapport au nombre d'essais est inférieur à une valeur seuil *mpc*. Lorsque *iter* ou *fcount* atteignent les valeurs limites *itermax* ou *flimit* respectivement, la procédure s'arrête.

La méthode de génération de la solution initiale est identique à celle utilisée par le LSA décrite dans la section 2.1.1, alors que la génération de la solution $(C^{'}, F^{'})$ dans le

voisinage de la solution courante (C, F) est basée sur une technique présentée dans [45]. Cette technique utilise les étapes de l'algorithme LSA et consiste à diversifier la solution courante en utilisant la diversification 3a, suivie par la diversification 3b, ensuite nous intensifions la solution diversifiée par l'application successive des intensifications 2a et 2b d'une manière itérative jusqu'au moment où nous ne pouvons plus modifier la solution courante ou bien atteindre un nombre limite d'itérations *maxrepeat*. La solution (C', F') est la meilleure solution trouvée au cours de ce processus.

2.4 Méthode hybride

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes [44], et ils s'inspirent à la fois des mécanismes de la sélection naturelle et de la génétique. Nous identifions le problème à un environnement donné, et les solutions à des individus évoluant dans cet environnement. À chaque génération, on ne retient que les individus les mieux adaptés à l'environnement à se reproduire. Au bout d'un certain nombre de générations, les individus restants sont particulièrement adaptés à l'environnement donné. L'adaptation d'un individu à son milieu est mesurée par la valeur de la fonction économique du problème. Chaque individu est encodé à l'aide de variables qui peuvent être binaires ou réels.

Nous nous inspirons de la méthode utilisée dans [13], qui consiste à hybrider la méthode LSA pour améliorer chaque solution générée avec un algorithme génétique dans le but d'arriver à une exploration améliorée du domaine réalisable. Le pseudo-code de la méthode hybride est résumé à la figure 8:

Étape 1 : :

Générer une population initiale de solutions réalisables S

iter = 0

Étape 2 :

Sélectionnez deux solutions parents de S

Effectuer une opération de croisement pour générer deux nouvelles solutions descedantes

Effectuer une opération de mutation sur chaque solution descendante avec une probabilité pm

Appliquer le LSA sur chaque solution descendante

Mise à jour de la population en gardant les meilleurs |S| solutions de la population actuelle et des deux solutions descendantes

iter++

si iter=nga alors Stop sinon aller à Étape 2

Figure 8 : Algorithme hybride

Chaque individu de la population initiale est généré comme suit. Nous choisissons avec une probabilité de 0,5 de générer soit des groupes de machines ou des familles de pièces. Si, par exemple, nous décidons pour la première alternative, alors chaque machine sera affectée à un groupe d'une manière aléatoire. Ensuite, nous affectons les pièces sur la base de groupes de machines à l'aide la procédure suivie dans le LSA (l'étape d'intensification 2a).

Pour compléter l'algorithme génétique, un codage des solutions est nécessaire. Une solution réalisable $(C,F)=\{(C_1,F_1),\ldots,(C_K,F_K)\}$ est codée par un vecteur $(M_1,...,M_m,P_1,....,P_n)$ où M_k et P_k représentent respectivement l'index du groupe de machines contenant la machine *k* et l'index de la famille de pièces contenant la pièce *k*.

Plusieurs techniques peuvent être utilisées dans le choix des parents et l'opération de croisement. Nous optons dans la cadre de ce projet pour le choix des deux parents par la sélection des deux meilleures solutions parmi quatre solutions choisies aléatoirement parmi la population. L'opération de croisement est réalisée au niveau des vecteurs représentant chaque solution. Plus précisément, supposons que les deux solutions parents sont représentées par les vecteurs $(M_1^1,...,M_m{}^1,P_1^1,....,P_n^1)$ et $(M_1{}^2,...,M_m{}^2,P_1^2,....,P_n{}^2)$, et soit les deux vecteurs représentant les solutions descendantes $(MD_1^1,...,MD_m{}^1,PD_1^1,....,PD_n{}^1)$ et

$(MD_1^2,...,MD_m^2,PD_1^2,....,PD_n^2)$. Le croisement de ces deux solutions revient à appliquer l'algorithme suivant :

Générer aléatoirement un vecteur de masque binaire $(B_1,...,B_m,B_{m+1},....,B_{m+n})$

For $i = 1..m$
si $B_i = 1$ alors $MD_i^1 = M_i^1$ et $MD_i^2 = M_i^2$
sinon $MD_i^1 = M_i^2$ et $MD_i^2 = M_i^1$

For $j = 1..n$
si $B_{m+j} = 1$ alors $PD_j^1 = P_j^1$ et $PD_j^2 = P_j^2$
sinon $PD_j^1 = P_j^2$ et $PD_j^2 = P_j^1$

Figure 9 : Croisement de deux solutions

L'opérateur de mutation est appliqué pour modifier légèrement chaque solution descendante. Il consiste à choisir aléatoirement quatre variables: une machine *i*, une pièce *j*, un groupe de machines *k1*, et une famille de pièces *k2*. Ensuite, nous affectons avec une probabilité *pm* la machine *i* au groupe de machines *k2* et la pièce *j* à la famille de pièces *k1*.

Chapitre 3 : Étude des résultats

Dans ce chapitre, nous présentons les résultats expérimentaux obtenus avec nos méthodes appliquées sur des instances utilisées dans la littérature. Nous comparons ces résultats avec les résultats exacts obtenus avec CPLEX et nous analysons l'optimalité Pareto de nos solutions.

3.1 Instances

Notre expérimentation utilise 34 problèmes largement utilisés dans la littérature par les auteurs pour évaluer l'efficacité de leur méthode pour résoudre des problèmes dont l'objectif ne comporte que le critère d'autonomie et dont les matrices d'incidence sont à valeurs binaires. Nous utilisons également trois autres problèmes de la littérature ayant des matrices d'incidence à valeurs entières. La taille d'une instance est définie par un triplet représentant le nombre de machines *m*, de pièces *n* et le nombre de cellules *K*. Les tableaux 3 et 4 représentent le numéro de chaque problème utilisé, sa taille et sa référence dans la littérature.

Instances	Références	*m*	*n*	*K*
P1	[24]	5	7	2
P2	[47]	5	7	2
P3	[38]	5	18	2
P4	[27]	6	8	2
P5	[28]	7	11	5
P6	[5]	7	11	4
P7	[39]	8	12	4
P8	[10]	8	20	3
P9	[11]	8	20	2
P10	[32]	10	10	5
P11	[8]	10	15	3
P12,P13	[3],[41]	14	24	7
P14	[31]	16	24	8
P15	[40]	16	30	6
P16	[22]	16	43	8
P17	[7]	18	24	9
P18	[33]	20	20	5

Tableau 3 : Instances de tests 1

Instances	Références	*m*	*n*	*K*
P19	[26]	20	23	7
P20,P21	[7],[6]	20	35	5
P22	[9]	24	40	7
P23	[9]	24	40	7
P24	[9]	24	40	7
P25	[9]	24	40	11
P26,P27	[9]	24	40	12
P27	[9]	24	40	12
P28	[31]	27	27	5
P29	[7]	28	46	10
P30	[25]	30	41	14
P31	[41]	30	50	13
P32	[41]	30	50	14
P33	[20]	37	53	3
P34	[9]	40	100	10
P35	[20]	27	15	8
P36	[20]	37	53	4

Tableau 4 : Instances de tests 2

3.2 Code et environnement

Nous avons implémenté les différentes méthodes en C++. Les tests ont été exécutés dans un environnement Linux sous le système d'exploitation Fedora 14, sur un ordinateur ayant un processeur Intel i7-2600 cadencé à 3.40 GHz avec 12 GB de mémoire RAM.

3.3 Choix de paramètres

Chaque méthode de résolution proposée utilise un ensemble de paramètres, et nous décrivons ici une stratégie que nous avons utilisée pour définir leurs valeurs. Au niveau de l'intensification dans la méthode LSA, nous avons identifié grâce à des tests préliminaires les valeurs suivantes de $\alpha = 0,75$ pour GRASP, de $\lambda = 0,75$ pour SW**,** de *numstrat*=*K*+*m*+*n* pour GRASP, *maxrepeat*=10 pour GRASP, de *maxrepeat*=*K*+*m*+*n pour* SA et de $numiter = 50$ pour SW.

Pour choisir le taux de diversification *q*, nous résolvons le problème (PE_γ) où $\gamma = 0,5$ pour cinq instances de tailles différentes P2, P15, P18, P27 et P31 avec les trois valeurs de *q*: 20%, 30% et 50%.

	20%	30%	50%
P2	20	20	20
P15	15,652	15,652	15,652
P18	9,633	9,939	9,939
P27	8,974	8,462	8,462
P31	19,935	19,717	18,845
AVG	14,839	14,754	14,58

Tableau 5 : Choix du taux de diversification

Chaque problème est résolu trois fois et la valeur affichée dans le tableau est la moyenne des trois exécutions. La dernière ligne indique la valeur moyenne pour les cinq problèmes. Le pourcentage 20% semble donner les meilleurs résultats.

Nous procédons de la même façon pour identifier les valeurs des paramètres de la méthode SA, et nous commençons par sélectionner les valeurs de TP^0 et β parmi les valeurs suivantes tout en fixant les valeurs des autres paramètres.

$$TP^0 = 6(K+n+m)\ ,\ \beta = 0.1\ ;\ TP^0 = 12(K+n+m)\ ,\ \beta = 0.1$$
$$TP^0 = 18(K+n+m)\ ,\ \beta = 0.1\ ;\ TP^0 = 6(K+n+m)\ ,\ \beta = 0.2$$
$$TP^0 = 12(K+n+m)\ ,\ \beta = 0.2\ ;\ TP^0 = 18(K+n+m)\ ,\ \beta = 0.2$$
$$TP^0 = 6(K+n+m)\ ,\ \beta = 0.5\ ;\ TP^0 = 12(K+n+m)\ ,\ \beta = 0.5$$
$$TP^0 = 18(K+n+m)\ ,\ \beta = 0.5$$

Le choix entre ces valeurs est effectué de la même façon que pour le choix du taux de diversification, et les meilleures valeurs trouvées sont $TP^0 = 18(K+m+n)\ ,\ \beta = 0.1$.

La deuxième étape est le choix de la valeur des paramètres de *SF* et *coff* parmi les valeurs :

$$SF = \left\lceil \frac{K+n+m}{7} \right\rceil\ ,\ coff = \left\lceil \frac{K+n+m}{14} \right\rceil$$
$$SF = \left\lceil \frac{K+n+m}{5} \right\rceil\ ,\ coff = \left\lceil \frac{K+n+m}{10} \right\rceil$$
$$SF = \left\lceil \frac{K+n+m}{3} \right\rceil\ ,\ coff = \left\lceil \frac{K+n+m}{6} \right\rceil$$

En utilisant toujours le même processus de sélection, nous trouvons que le meilleur choix est $SF = \left\lceil \frac{K+n+m}{5} \right\rceil$, $coff = \left\lceil \frac{K+n+m}{10} \right\rceil$

La dernière étape est le choix de la valeur de *mpc* parmi les valeurs 0.3, 0,5 et 0,8, et les tests montrent que 0,5 est la valeur qui donne les meilleurs résultats. La valeur utilisée pour les variables *flimit* et *itermax* est *K+M+n*.

La méthode hybride utilise un ensemble de paramètres que nous avons listés dans le tableau suivant avec leurs valeurs trouvées dans l'article [13]:

Paramètre	Valeur
S : taille de la population	5*K*
pm: probabilité de mutation	2%
nga : nombre des itérations de l'algorithme génétique	*K+m+n*

Tableau 6 : Paramètres de la méthode hybride

3.4 Comparaison des méthodes

Pour comparer les trois méthodes, nous résolvons le problème (PE_γ) avec $\gamma = 0,5$ pour chacun des 37 instances présentées précédemment. La valeur moyenne de la l'efficacité (Eff) et le temps d'exécution en secondes (TC), sur trois résolutions pour chaque instance est

incluse dans le tableau 7. LSA-GRASP et LSA-SW représentent l'algorithme LSA utilisant respectivement l'intensification GRASP et SW.

Instance	LSA-GRASP		LSA-SW		SA		Hybride	
	Eff	TC	Eff	TC	Eff	TC	Eff	TC
P1	50	0	50	0,013	50	0,003	50	0,017
P2	20	0	20	0,023	20	0,003	20	0,027
P3	32,609	0,003	32,609	0,057	32,609	0,013	32,609	0,123
P4	31,818	0	31,818	0,03	31,818	0,007	31,818	0,033
P5	17,391	0,01	17,391	0,14	17,391	0,093	17,391	0,353
P6	33,333	0,01	33,333	0,067	33,333	0,047	33,333	0,283
P7	20	0,013	20	0,147	20	0,037	20	0,383
P8	31,147	0,013	31,147	0,14	31,147	0,093	31,147	0,427
P9	20,879	0,013	20,879	0,097	20,879	0,087	20,879	0,433
P10	33,333	0,007	33,333	0,093	33,333	0,057	33,333	0,437
P11	46,739	0,003	46,739	0,067	46,739	0,02	46,739	0,183
P12	25,862	0,09	25,862	0,793	25,862	1,5	25,862	8,353
P13	23,497	0,06	24,59	1,357	24,59	2,583	24,59	8,677
P14	5,098	0,14	5,49	1,16	5,882	5,367	5,882	15,813
P15	13,044	0,097	13,044	1,053	13,044	4,88	13,044	13,27
P16	10,318	0,26	11,376	4,263	11,111	26,56	11,905	52,85
P17	5,114	0,127	5,114	1,36	5,114	6,94	5,114	19,623
P18	9,327	0,08	9,633	1,217	9,633	2,063	9,939	8,55
P19	2,95	0,107	3,54	1,7	3,54	5,91	4,13	17,087
P20	31,74	0,08	31,74	1,293	31,985	8,167	31,985	13,423
P21	22,876	0,153	23,965	1,953	23,965	9,987	24,183	20,837
P22	37,405	0,16	37,405	3,193	37,405	12,413	37,405	29,357
P23	34,615	0,257	33,846	3,193	34,615	11,22	34,615	38,537
P24	30,28	0,367	30,534	3,737	30,534	17,863	30,534	52,59
P25	10,305	0,41	10,305	4,067	11,069	47,38	11,069	103,8
P26	6,87	0,463	6,87	5,113	7,634	61,55	7,634	120,313
P27	5,897	0,69	5,641	4,647	6,154	48,807	6,154	121,427
P28	12,253	0,143	12,101	1,74	12,557	10,26	12,557	22,393
P29	3,949	0,717	4,107	6,17	4,265	99,83	4,265	146,417
P30	11,198	0,873	11,719	6,567	12,5	114,58	12,5	209,703

P31		19,063	1,277		18,628	7,383		19,281	146,233		19,281	239,993
P32		9,78	1,07		9,581	7,643		10,18	172,75		10,18	277,043
P33		16,632	0,637		16,632	8,243		16,632	79,3		16,667	91,55
P34		27,302	3,337		27,857	26,707		27,857	1218,537		27,857	1183,403
P35		17,722	0,11		17,81	1,63		18,25	2,33		18,338	16,067
P36		17,183	0,517		17,404	5,103		17,502	84,883		17,822	81,403
P37		31,798	0,147		31,627	1,313		32,045	3,457		32,045	30,967
Moyenne		21,063	0,336		21,180	3,067		21,364	59,616		21,427	79,625

Tableau 7 : Comparaison des résultats

En se fiant aux moyennes que nous retrouvons dans la dernière ligne du tableau 7, la méthode hybride offre des meilleurs résultats que la méthode SA en terme d'efficacité, mais requiert 1,3 fois en temps d'exécution. En revanche, la méthode hybride offre de meilleurs résultats que le LSA, mais toutefois, son temps d'exécution est respectivement 237 fois et 26 fois plus grand que celui du LSA-GRASP et LSA-SW.

De plus, pour analyser la dominance de chaque méthode, nous introduisons le tableau 8, où chaque élément du tableau indique le nombre d'instances pour lesquels la méthode de la ligne est meilleure que la méthode de la colonne. Nous constatons que la méthode hybride domine les autres méthodes, car selon la dernière colonne du tableau, aucune méthode n'arrive à surpasser son résultat sur toutes les instances. En plus, les valeurs de la dernière ligne du tableau correspondant à la méthode hybride sont plus grandes que la valeur de tout le reste des lignes.

	LSA-GRASP	LSA-SW	SA	Hybride
LSA-GRASP		6	0	0
LSA-SW	12		1	0
SA	20	14		0
Hybride	21	19	7	

Tableau 8 : Dominance de méthodes

Afin de vérifier si les résultats de ces méthodes sont statistiquement différents avec un certain niveau de confiance, nous utilisons le test des rangs signés de Wilcoxon. Ce test appartient à la famille des tests non paramétriques qui ne font aucune hypothèse sur la distribution sous-jacente des données. Nous appliquons donc le test de Wilcoxon sur les

résultats de chaque couple de méthodes, avec un niveau de confiance de 5%. Le tableau 9 résume les valeurs trouvées.

	LSA-GRASP	LSA-SW	SA	Hybride
LSA-GRASP		Identique	Différent	Différent
LSA-SW	Identique		Différent	Différent
SA	Différent	Différent		Identique
Hybride	Différent	Différent	Identique	

Tableau 9 : Résultats statistiques du test Wilcoxon

En se fiant aux résultats du test de Wilcoxon, nous pouvons confirmer à 95% que les méthodes LSA-GRASP et LSA-SW donnent statistiquement les mêmes résultats, et également pour les deux méthodes SA et la méthode hybride. Nous pouvons conclure aussi que la méthode hybride donne des résultats statistiquement différents de ceux trouvée par LSA-GRASP et LSA-SW.

Pour vérifier si le temps de résolution accordé aux différentes méthodes peut influencer la valeur de la fonction économique obtenue, nous allongeons ce temps pour LSA-GRASP, LSA-SW et SA en leur accordant un temps d'exécution similaire à celui de la méthode hybride. Pour y arriver, nous modifions le nombre maximal d'itérations *numstrat* dans les méthodes LSA-GRASP et LSA-SW en le multipliant respectivement par 237 et 26. Tandis que pour la méthode SA, nous multiplions les nombres d'itérations *flimit* et *itermax* par 1,3. Les résultats obtenus sont résumés dans le tableau 10.

Instance	LSA-GRASP		LSA-SW		SA		Hybride	
	Eff	TC	Eff	TC	Eff	TC	Eff	TC
P1	50	0,09	50	0,26	50	0	50	0,017
P2	20	0,127	20	0,383	20	0,01	20	0,027
P3	32,609	0,427	32,609	1,27	32,609	0,07	32,609	0,123
P4	31,818	0,153	31,818	0,5	31,818	0,02	31,818	0,033
P5	17,391	1,057	17,391	2,267	17,391	0,38	17,391	0,353
P6	33,333	0,793	33,333	1,52	33,333	0,21	33,333	0,283
P7	20	1,103	20	2,31	20	0,19	20	0,383
P8	31,147	1,083	31,147	2,997	31,148	0,39	31,147	0,427
P9	20,879	1,263	20,879	2,17	20,879	0,33	20,879	0,433
P10	33,333	1,067	33,333	1,78	33,333	0,23	33,333	0,437

P11	46,739	0,553	46,739	1,417	46,739	0,09	46,739	0,183
P12	25,862	12,113	25,862	14,373	25,862	5,02	25,862	8,353
P13	24,59	12,213	24,59	20,057	24,59	9,59	24,59	8,677
P14	5,882	21,23	5,882	31,547	5,882	21,07	5,882	15,813
P15	13,044	20,257	13,044	27,137	13,044	18,57	13,044	13,27
P16	11,376	68,007	11,905	70,113	11,111	99,16	11,905	52,85
P17	5,114	24,78	5,114	27,697	5,114	24,6	5,114	19,623
P18	10,245	19,173	9,633	22,39	9,633	7,41	9,939	8,55
P19	3,835	23,877	3,54	27,567	3,54	21,73	4,13	17,087
P20	31,985	17,573	31,985	27,723	31,985	29,39	31,985	13,423
P21	24,183	38,117	24,183	39,07	23,529	35,74	24,183	20,837
P22	37,405	34,723	37,405	59,693	37,405	50,19	37,405	29,357
P23	34,615	44,3	34,615	54,547	34,615	38,57	34,615	38,537
P24	30,534	50,543	30,534	61,993	30,534	62,22	30,534	52,59
P25	11,069	90,973	11,069	94,293	11,069	180,47	11,069	103,80
P26	7,634	101,367	7,379	89,347	7,634	230,03	7,634	120,313
P27	6,154	101,84	6,154	94,927	6,154	178,98	6,154	121,427
P28	12,557	30,113	12,557	47,577	12,557	32,66	12,557	22,393
P29	4,265	124,45	4,265	115,577	4,265	365,79	4,265	146,417
P30	12,5	171,643	12,5	132,597	12,5	438,5	12,5	209,703
P31	19,281	185,557	19,281	169,37	19,281	535,79	19,281	239,993
P32	10,18	213,61	9,98	185,94	10,18	665,33	10,18	277,043
P33	16,701	108,41	16,735	160,507	16,632	368,87	16,667	91,55
P34	27,857	672,907	27,857	634,453	27,857	4108,84	27,857	1183,403
P35	18,338	18,03	18,338	27,34	18,338	8	18,338	16,067
P36	17,699	133,22	17,576	187,01	17,699	312,03	17,822	81,403
P37	32,045	26,187	32,045	34,853	32,045	13,59	32,045	30,967
Moyenne	21,411	64,133	21,386	66,880	21,36	212,542	21,427	79,625

Tableau 10 : Comparaison de méthodes avec modification du temps d'exécution

Les temps d'exécutions moyens pour LSA-GRASP et LSA-SW sont rapprochés de celui de la méthode hybride, et le temps d'exécution moyen pour SA est multiplié par 3,5 et devient plus grand que celui de la méthode hybride, mais le résultat moyen de celle-ci reste toujours meilleur. Nous pouvons donc conclure qu'à la lumière de cette expérimentation, la

méthode hybride génère de meilleurs résultats pour résoudre le problème (PE_γ) si le temps de résolution que nous sommes prêts à accorder n'est pas identique.

La modification du nombre d'itérations pour la méthode SA n'a pas d'impact sur ses résultats, alors que les résultats moyens de LSA-GRASP et LSA-SW sont améliorées suite à la modification de la valeur de *numstrat*. Pour vérifier si les résultats de ces méthodes demeurent statistiquement différents de la méthode hybride, nous appliquons aussi le test de Wilcoxon, et les résultats obtenus sont résumés dans le tableau 11.

	LSA-GRASP	LSA-SW	Hybride
LSA-GRASP		Identique	Identique
LSA-SW	Identique		Identique

Tableau 11 : Statistiques de Wilcoxon après modification de *numstrat*

Nous pouvons confirmer à 95% que les méthodes ne sont plus statistiquement différentes.

Le tableau 12 représente la dominance de chaque méthode par rapport à l'autre, après la modification de *numstrat*. Les résultats de ce tableau confirment le constat de la non différence statistique de méthodes, puisque les valeurs du tableau sont très rapprochées et la méthode hybride ne domine plus autant les autres méthodes.

	LSA-GRASP	LSA-SW	Hybride
LSA-GRASP		5	2
LSA-SW	2		1
Hybride	3	5	

Tableau 12 : Dominance de méthodes après modification de *numstrat*

Les résultats indiquent alors que les trois méthodes LSA-GRASP, LSA-SW et hybride aboutissent statistiquement aux mêmes résultats en terme d'efficacité. En résolvant les problèmes avec la méthode LSA, nous obtenons de bons résultats en très peu de temps et on peut les améliorer légèrement en allouant plus de temps d'exécution par l'augmentation de nombre des itérations.

3.5 Résolution avec CPLEX

Dans le but de vérifier la qualité des résultats obtenus avec la méthode hybride, nous avons résolu le problème (PE_γ) où $\gamma = 0,5$ avec CPLEX (version 12.4), un outil informatique d'optimisation offrant une bibliothèque de fonctions pouvant s'interfacer avec

notre langage de programmation utilisé C++. Pour la résolution avec CPLEX, nous transformons le problème initial en un problème équivalent (PEL_γ). L'idée est de remplacer le produit *xy* par le vecteur *w* définit comme suit *:*

$$w_{ij}^k = x_{ik} y_{jk} \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K$$
$$w_{ij}^k = 0 \text{ ou } 1 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K.$$

Après cette modification, le problème (PE_γ) devient :

$$\text{Max } Eff = \gamma \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} w_{ij}^k - (1-\gamma) \sum_{k=1}^{K} | T - \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} w_{ij}^k |$$

Sujet à

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,m \qquad (1)$$

$$\sum_{k=1}^{K} y_{jk} = 1 \qquad j = 1,\ldots,n \qquad (2)$$

$$\sum_{i=1}^{m} x_{ik} \geq 1 \qquad k = 1,\ldots,K \qquad (3)$$

$$\sum_{j=1}^{n} y_{jk} \geq 1 \qquad k = 1,\ldots,K \qquad (4)$$

$$x_{ik} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; k = 1,\ldots,K \qquad (5)$$

$$y_{jk} = 0 \text{ or } 1 \qquad j = 1,\ldots,n; k = 1,\ldots,K \qquad (6)$$

$$w_{ij}^k = x_{ik} y_{jk} \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K \qquad (7)$$

$$w_{ij}^k = 0 \text{ or } 1 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K \qquad (8)$$

Ensuite, la première contrainte quadratique (7) est linéarisée en utilisant la forme suivante proposée dans [14] et [29] :

$$-w_{ij}^k + x_{ik} + y_{jk} - 1 \leq 0 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K$$
$$2w_{ij}^k - x_{ik} - y_{jk} \leq 0 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K.$$

Ceci donne lieu au problème équivalent (PEL_γ) :

$$\text{Max } Eff = \gamma \sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} w_{ij}^{k} - (1-\gamma)\sum_{k=1}^{K} | T - \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} w_{ij}^{k} |$$

Sujet à

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1,\ldots,m \qquad (1)$$

$$\sum_{k=1}^{K} y_{jk} = 1 \qquad j = 1,\ldots,n \qquad (2)$$

(PEL_γ)

$$\sum_{i=1}^{m} x_{ik} \geq 1 \qquad k = 1,\ldots,K \qquad (3)$$

$$\sum_{j=1}^{n} y_{jk} \geq 1 \qquad k = 1,\ldots,K \qquad (4)$$

$$x_{ik} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; k = 1,\ldots,K \qquad (5)$$

$$y_{jk} = 0 \text{ or } 1 \qquad j = 1,\ldots,n; k = 1,\ldots,K \qquad (6)$$

$$-w_{ij}^{k} + x_{ik} + y_{jk} - 1 \leq 0 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K \qquad (7)$$

$$2w_{ij}^{k} - x_{ik} - y_{jk} \leq 0 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K \qquad (8)$$

$$w_{ij}^{k} = 0 \text{ or } 1 \qquad i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,K \qquad (9)$$

Le tableau suivant résume les résultats obtenus avec la méthode hybride et la méthode exacte basée sur CPLEX en résolvant le problème (PEL_γ) pour les 13 premières instances.

Instances	Hybride		CPLEX	
	Efficacité	Temps de calcul	Efficacité	Temps de calcul
P1	50	0,017	50	0,01
P2	20	0,027	20	0,07
P3	32,609	0,123	32,609	0,11
P4	31,818	0,033	31,818	0,05
P5	17,391	0,353	17,391	1,78
P6	33,333	0,283	33,333	0,21
P7	20	0,383	20	1,71
P8	31,147	0,427	31,147	0,99
P9	20,879	0,433	20,879	9,33
P10	33,333	0,437	33,333	0,97
P11	46,739	0,183	46,739	0,45
P12	25,862	8,353	25,862	25810
P13	24,59	8,677	24,59	115297,62
Moyenne	29,823	1,52	29,823	10855,64

Tableau 13 : Comparaison entre hybride et CPLEX

Pour les 13 problèmes, la méthode hybride génère la même valeur moyenne des trois résolutions que celle obtenue avec CPLEX. Par contre, le temps de résolution avec l'hybride est plus petit que celui de CPLEX en général, et la différence s'accentue pour les instances de plus grande taille P12 et P13. Pour les instances restantes, CPLEX ne permet pas de les résoudre dans un délai raisonnable. En effet, ces instances requièrent plus de 5 jours et 14,5 GO de mémoire pour permettre d'aboutir une solution optimale. Ainsi, l'utilisation de la méthode exacte CPLEX devient impraticable pour les problèmes de grande taille.

3.6 Tests sur des instances de grandes dimensions

Pour vérifier jusqu'à quel point le temps d'exécution grandit lorsque les instances sont de plus grande taille, nous avons résolu les trois problèmes PL1, PL2 et PL3 avec la méthode LS-GRASP en considérant la valeur $\gamma = 0,5$. La description des problèmes et la moyenne des résultats de trois résolutions sont résumées dans le tableau 14.

P	M	P	C	Efficacité	Temps de calcul(en sec)
PL1	300	500`	20	-26,131	810,637
PL2	350	500	15	-22,39	978,523
PL3	400	500	20	-27,882	1545,39

Tableau 14 : Résultats des instances de grandes dimensions

Notons que même si le nombre d'itérations *numstrat* prend sa valeur initiale de (*m*+*n*+*K*), le temps de résolution augmente grandement par rapport à celui requis pour les 37 problèmes de la section 3.3. De plus, considérant que le rapport entre les temps d'exécution de la méthode hybride et LSA-GRASP est de l'ordre de 237, il s'ensuit que pour résoudre l'instance PL1 avec l'hybride, le temps d'exécution serait de l'ordre de (810x237) secondes, soit plus de 53 heures. Ainsi, l'utilisation de la méthode hybride pour résoudre des problèmes de grande taille s'avère très couteuse en temps de résolution. Donc pour obtenir des résultats dans un délai raisonnable, la méthode LSA-GRASP s'impose d'avantage à cause de son temps de résolution.

3.7 Étude de l'optimalité Pareto des résultats

La fonction objectif du problème (PE_γ) est une combinaison convexe de deux facteurs à maximiser: autonomie (z1) et équilibre de charge (z2). Nous résolvons par la méthode hybride le problème pour toute valeur de γ dans cet ensemble (0, 0,1, 0,2, 0,3, 0,4,

0,5, 0,6, 0,7, 0,8, 0,9, 1) définissant ainsi un couple (z1, z2) pour toute valeur de γ. Les solutions générées ne sont pas forcément des solutions optimales du problème, car nous utilisons une heuristique pour la résolution. Du coup, la courbe définie par ces solutions n'est pas la courbe Pareto du problème constituée par les solutions Pareto efficace, mais plutôt une approximation de celle-ci.

Ainsi, nous vérifions la courbe obtenue dans le but d'avoir une courbe qui n'est constituée que par des solutions non dominées, et nous les appelons des solutions h-efficaces car ce ne sont pas des solutions exactes pour les décrire en tant que solution Pareto efficace. Le test est réalisé sur trois types d'instances de taille variable P14, P30 et P36, les figures ci-dessous sont les diagrammes représentant les points (z1, z2) de chaque instance.

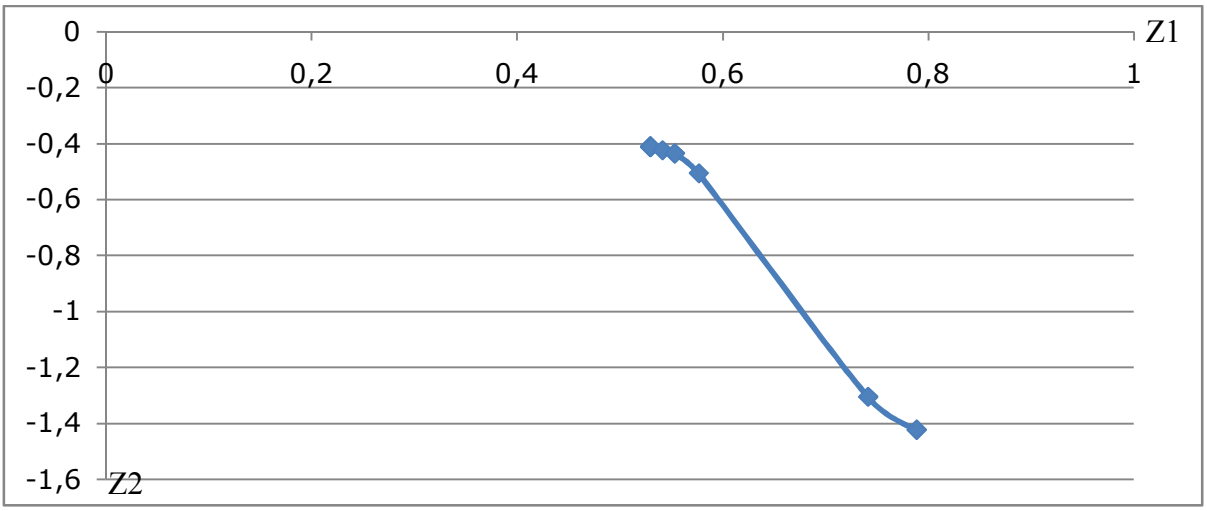

Figure 10 : Diagramme de l'instance P14

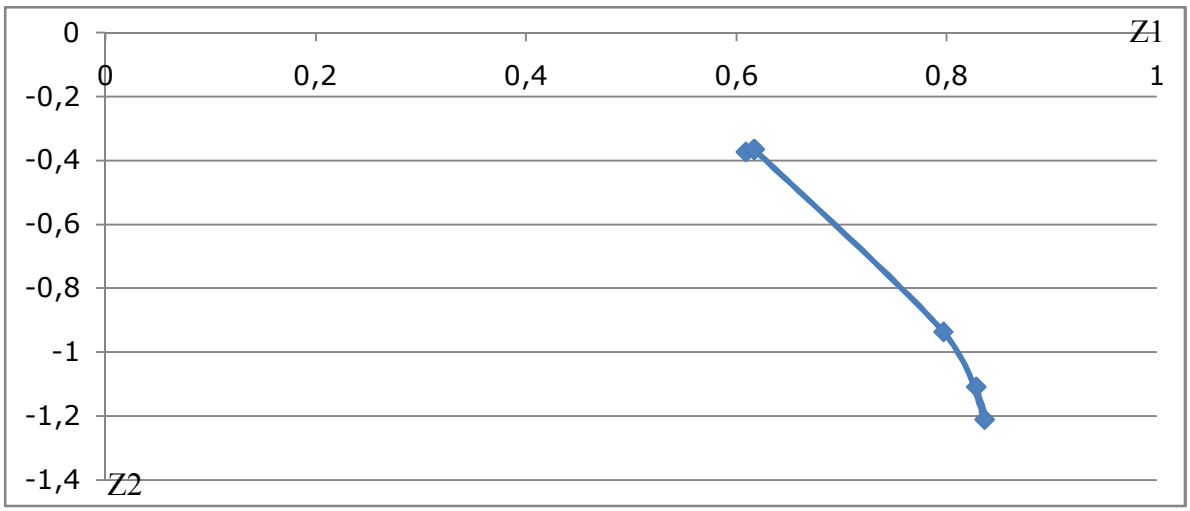

Figure 11 : Diagramme de l'instance P30

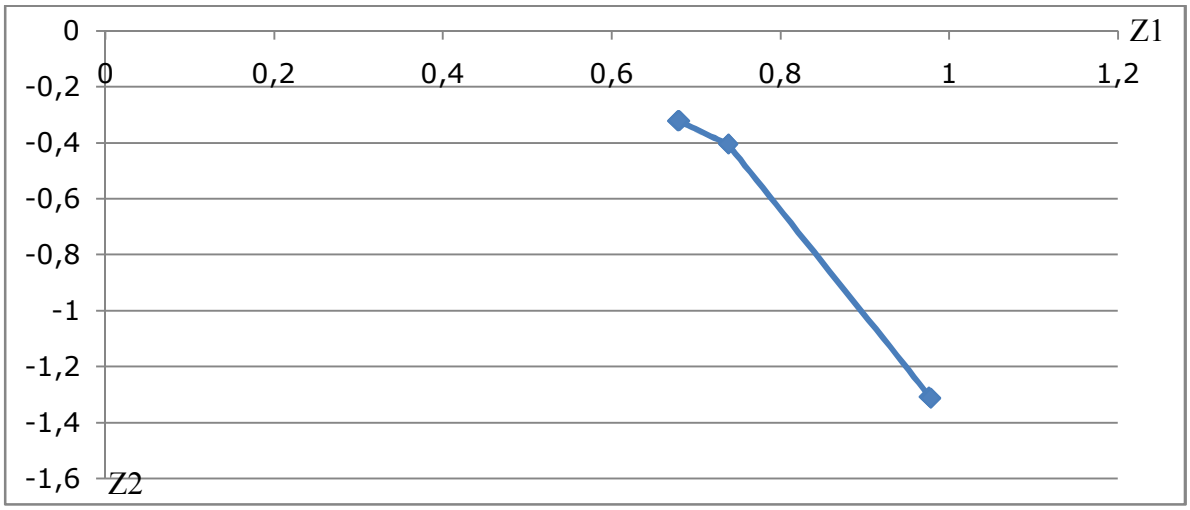

Figure 12 : Diagramme de l'instance P36

Pour la première instance P14, l'ensemble de points générés (z1, z2) sont regroupés en six points h-efficaces : (0,529, -0,411), (0,552, -0,435), (0,541, -0,423), (0,576, -0,505), (0,741, -1,305) et (0,788, -1,423). Par conséquent, nous nous approchons davantage de la courbe Pareto de cette instance. Le même scénario se produit pour l'instance P36 et les points sont regroupés en trois points h-efficaces: (0,632, -0,351), (0,726, -0,523) et (0,978, -1,318). Pour l'instance P30, nous obtenons quatre points h-efficaces : (0,609, -0,375), (0,796, -0,937), (0,835, -1,211) et (0,828, -1,109). Nous obtenons aussi un point non h-efficace, le point (0,609, -0.375), qui est dominé par le point (0.617, -0.367), ce qui explique la nature heuristique et empirique de notre méthode.

Le tableau 15 représente le nombre de points h-efficaces et non h-efficaces parmi les solutions générées par la méthode hybride pour toute instance. Ces deux facteurs donnent une estimation sur la distance entre la courbe générée par notre méthode et la courbe Pareto, de telle sorte qu'on s'approche de celle-ci lorsqu'on a moins de points non h-efficaces.

Instance	Nombre de points h-efficaces	Nombre de points non h-efficaces
P1	1	0
P2	4	0
P3	4	0
P4	2	0
P5	2	1
P6	1	1
P7	3	1
P8	3	0

P9	2	0
P10	2	0
P11	2	0
P12	3	1
P13	4	1
P14	6	0
P15	7	0
P16	6	0
P17	5	0
P18	4	1
P19	4	2
P20	5	0
P21	5	2
P22	4	1
P23	4	0
P24	3	0
P25	4	1
P26	4	0
P27	3	0
P28	6	0
P29	4	0
P30	4	1
P31	5	1
P32	4	0
P33	6	1
P34	4	0
P35	6	0
P36	3	0
P37	3	0
Moyenne	3,8	0,4

Tableau 15 : Nombre de points h-efficaces

Le nombre maximum de points h-efficaces est 7, obtenu au niveau de l'instance P15 et le nombre maximum de points non efficaces est 2 et il est atteint par les deux instances P19 et P21. Il n'y a pas de point non h-efficace pour 24 instances. D'autre part, nous avons une moyenne de 3,8 de solutions h-efficaces et 0,4 de solution non h-efficace par instance, et ceci

par rapport à 11 résolutions du problème, puisque nous résolvons le problème pour toute valeur de γ dans l'ensemble (0, 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9, 1). Ainsi, la méthode hybride nous permet de réaliser un taux de 90,5% de solutions h-efficaces et utilisables par le décideur pour sa conception.

Conclusion

Nous avons présenté dans ce mémoire des méthodes basées sur des algorithmes heuristiques pour résoudre le problème de formation de cellules entier bi-objectif. La comparaison des résultats de ces méthodes, nous permet d'identifier la méthode hybride comme celle générant les meilleurs résultats.

En comparant avec CPLEX (un logiciel d'optimisation mathématique), notre méthode s'est montrée compétitive et rapide, en particulier, pour les instances de taille moyenne et supérieure. En effet, la résolution des problèmes de plus petite taille indique que notre méthode hybride génère des solutions de valeurs identiques à celles obtenues avec CPLEX (12.4), mais dans un temps de résolution beaucoup plus acceptable requérant un espace mémoire moins grand.

Plusieurs possibilités peuvent être envisagées pour des développements futurs de nos travaux et de nos méthodes. Une extension directe de nos méthodes serait d'incorporer d'autres facteurs au niveau de la fonction objectif. De plus si nous arrivons à développer une méthode exacte pour affecter les groupes de machines (les familles de pièces) sur la base des familles de pièces (des groupes de machines), ceci aurait probablement un grand impact au niveau de la résolution en se basant sur les résultats observés pour les problèmes binaires dans [45].

Finalement, bien que les instances sur lesquelles nous avons effectué notre expérimentation soient de tailles variées, il serait intéressant d'appliquer nos méthodes sur des problèmes de taille large ayant des matrices d'incidences entières, afin de mesurer la qualité des résultats.

Bibliographie

[1] Augugliaro, A. et Dusonchet, L. et Sanseverino, E.R, 2001. Evolving non-dominated solutions in multiobjective service restoration for auto-mated distribution networks. *Electric Power Systems Research*, 59(3):185-195.

[2] Arkat, J. et Hosseinabadi, M. et Ahmadizar, F., 2012. Multi-objective genetic algorithm for cell formation problem considering cellular layout and operations scheduling. *International Journal of Computer Integrated Manufacturing*, 25, 625-635.

[3] Askin, R.G. et Subramanian, S.P., 1987. A cost-based heuristic for group technology configuration. *International Journal of Production Research*, 25, 101–113.

[4] Baykasoglu, A. et Gindy, N.Z., 2000. Multiple objective capability based approach to form part-machine groups for cellular manufacturing applications. *International Journal of Production Research*, 38:5, 1133-1161.

[5] Boctor, F.F., 1991. A linear formulation of the machine-part cell formation problem. *International Journal of Production Research*, 29, 343–356.

[6] Boe, W.J. et Cheng, C.H., 1991. A close neighbour algorithm for designing cellular manufacturing systems. *International Journal of Production Research*, 29, 2097–2116.

[7] Carrie, A.S., 1973. Numerical taxonomy applied to group technology and plant layout. *International Journal of Production Research*, 11, 399–416.

[8] Chan, H.M. et Milner, D.A., 1982. Direct clustering algorithm for group formation in cellular manufacture. *Journal of Manufacturing Systems*, 1, 65-75.

[9] Chandrasekharan, M.P. et Rajagopalan, R., 1989. GROUPABILITY: an analysis of the properties of binary data matrices for group technology. *International Journal of Production Research*, 27, 1035–1052.

[10] Chandrasekharan, M.P. et Rajagopalan, R., 1986a. MODROC: an extension of rank order clustering for group technology. *International Journal of Production Research*, 24, 1221–1233.

[11] Chandrasekharan, M.P. et Rajagopalan, R., 1986b. An ideal seed non-hierarchical clustering algorithm for cellular manufacturing. *International Journal of Production Research*, 24, 451–464.

[12] Dimopoulos, C. et Zalzala, A.M.S., 2000. Recent developments in evolutionary computations for manufacturing optimization: problems, solutions, and comparisons. *IEEE Transactions on Evolutionary Computations*, 4, 93–113.

[13] Elbenani, B., Ferland, J.A., Bellemare J., 2012. Genetic algorithm and large neighborhood search to solve the cell formation problem. *Expert Systems with Applications*, 39, 2408-2414.

[14] Elbenani, B., Ferland, J.A., Bellemare J., 2012. Cell formation problem solved exactly. using dinkelbach algorithm. CIRRELT-2012-07, Université de Montréal, Canada.

[15] Farahani, M.H. et Hosseini, L., 2011. An ant colony optimization approach for the machine-part cell formation problem. *International Journal of Computational Intelligence Systems*, 4, 486–496.

[16] Feo, T., Resende, G.C., 1995. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6, 109-133.

[17] Ferland, J.A. et Costa D., 2001. Heuristic search methods for combinatorial programming problems. Publication # 1193, Department of Computer Science et Operations Research, Université de Montréal, Canada.

[18] Ghosh, T., Sengupta, S., Chattopadhyay, M., Dan, P.K., 2011. Meta-heuristic in cellular manufacturing: A state-of-the-art. *International Journal of Industrial Engineering Computations*, 2, 87 – 122.

[19] Goncalves, J. et Resende, M.G.C., 2004. An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering*, 47, 247–273.

[20] Haleh, H. et Iranmaneslr, H. et Kor, H., 2009. A New Hybrid Evolutionary Algorithm for solving Multi Objective Cell Formation Problem, *Computers & Industrial Engineering*, 2009. CIE 2009.

[21] Joslin, D.E. et Clements, D.P., 1999. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research*, 10, 353-373.

[22] King, J.R., 1980. Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm. *International Journal of Production Research*, 18, 213–232.

[23] Kirkpatrick, S., Gelatt, C.D. Jr, Vecchi, M.P., 1983. Optimization by simulated annealing. *Science*, 220, 671 – 680.

[24] King, J.R. et Nakornchai, V., 1982. Machine-component group formation in group technology: review and extension. *International Journal of Production Research*, 20, 117–133.

[25] Kumar, K.R. et Vannelli, A., 1987. Strategic subcontracting for efficient disaggregated manufacturing. *International Journal of Production Research*, 25, 1715–1728.

[26] Kumar, K.R., Kusiak, A., Vannelli, A., 1986. Grouping of parts and components in flexible manufacturing systems. *European Journal of Operational Research*, 24, 387–397.

[27] Kusiak, A., Cho, M., 1992. Similarity coefficient algorithm for solving the group technology problem. *International Journal of Production Research*, 30, 2633–2646.

[28] Kusiak, A., Chow, W.S., 1987. Efficient solving of the group technology problem, *Journal of Manufacturing Systems*, 6, 117–124.

[29] Mahdavi, B., Javadi, F.,Alipour, K., Slomp, K. Designing a new mathematical model for cellular manufacturing system based on cell utilization, *Applied mathematics and Computation*, 190 (2007) 662-670

[30] Manimaran, A., Prabhakaran, G., Venkumar, P., 2011. Manufacturing cell formation using artificial immune system. *International Journal of Computer Aided Engineering and Technology*, 3, 22 – 42.

[31] McCormick, W.T., Schweitzer, P.J., White, T.W., 1972. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20, 993–1009.

[32] Mosier, C.T. et Taube, L., 1985a. The facets of group technology and their impact on implementation. *OMEGA*, 13, 381–391.

[33] Mosier, C. et Taube, L., 1985b. Weighted similarity measure heuristics for the group technology machine clustering problem, *OMEGA*, 13, 577–83.

[34] Ogoubi, E., Ferland, J.A,Hafid, A.,Turcotte, M.,Bellmare, J., 2010. A multiconstaint Cell formation problem for large scale application decomposition. DIRO,Université de Montréal, Canada.

[35] Papaioannou, G. et Wilson, J.N., 2010. The evolution of cell formation problem methodologies based on recent studies (1997-2008): Review et directions for future research, *European Journal of Operational Research*, 206, 509–521.

[36] Sarac, T., Ozcelik F., 2012. A genetic algorithm with proper parameters for manufacturing cell formation problems. *Journal of Intelligent Manufacturing*, 23, 1047 – 1061.

[37] Sarker, B. et Khan, M., 2001. A comparison of existing grouping efficiency measures and a new grouping efficiency measure. *IIE Transactions*, 33, 11–27.

[38] Seifoddini, H., 1989. A note on the similarity coefficient method and the problem of improper machine assignment in group technology applications. *International Journal of Production Research*, 27, 1161–1165.

[39] Seifoddini, H. et Wolfe, P.M., 1986. Application of the similarity coefficient method in group technology, *IIE Transactions*, 18, 271–277.

[40] Srinivasan, G., Narendran, T.T., Mahadevan, B., 1990. An assignment model for the part-families problem in group technology, *International Journal of Production Research*, 28, 145–152.

[41] Stanfel, L.E., 1985. Machine clustering for economic production. *Engineering Costs and Production Economics*, 9, 73–81.

[42] Steuer, R.E., 1989, *Multiple Criteria Optimization: Therory,Computation, and Application*. Krieger Publishing Company, Malabar, Florida.

[43] Su, C.T. et Hsu, C.M., 1998. Multi-objective machine-part cell formation through parallel simulated annealing. *International Journal of Production Research*, 36:8, 2185-2207.

[44] Syswerda, G., 1992. A study of reproduction in generational and steady-state genetic algorithms. *In G. J. E. Rawlings (Ed.)*, Foundations of genetic algorithms, 4–101.

[45] Thanh, L.T., Ferland, J.A., Thuc, N.D., Nguyen, V.H., 2012. Hybrid of Metaheuristic Methods for Solving the Cell Formation Problem. CIRRELT-2012-14, Université de Montréal, Canada.

[46] Tunnukij, T. et Hicks, C., 2009. An Enhanced Genetic Algorithm for solving the cell formation problem. *International Journal of Production Research*, 47, 1989–2007.

[47] Waghodekar, P.H. et Sahu, S., 1984. Machine-component cell formation in group technology MACE. *International Journal of Production Research*, 22, 937–948.

[48] Yasuda, K. et Hu, L. et Yin, Y., 2005. A grouping genetic algorithm for the multi-objective cell formation problem. *International Journal of Production Research*, 43:4, 829-853.

[49] Ying K.C. Lin S.W., Lu C.C. 2011. Cell formation using a simulated annealing algorithm with variable neighbourhood. *European Journal of Industrial Engineering*, 5, 22–42.